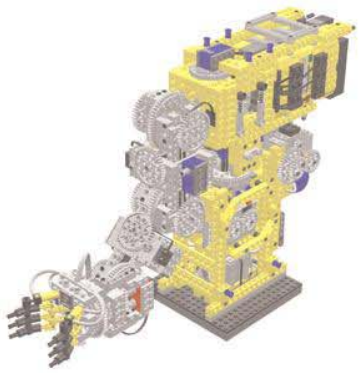
SYNGRESS®

1 YEAR UPGRADE
BUYER PROTECTION PLAN



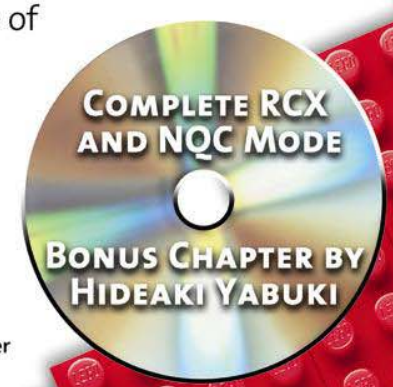
LEGO® Mindstorms™ Masterpieces

Building and Programming Advanced Robots

Learn from the Lego® Masters

- Incredibly sophisticated and technically detailed projects to challenge the most advanced builder
- Designs from world-class masters provide an incredible range of advanced techniques and mechanical solutions
- Meticulously documented building instructions with detailed explanations of methodology and engineering principles

Miguel Agulló
Doug Carlson
Kevin Clague
Giulio Ferrari
Mario Ferrari
Hideaki Yabuki
Ralph Hempel Technical Reviewer



solutions@syngress.com

With more than 1,500,000 copies of our MCSE, MCSD, CompTIA, and Cisco study guides in print, we continue to look for ways we can better serve the information needs of our readers. One way we do that is by listening.

Readers like yourself have been telling us they want an Internet-based service that would extend and enhance the value of our books. Based on reader feedback and our own strategic plan, we have created a Web site that we hope will exceed your expectations.

Solutions@syngress.com is an interactive treasure trove of useful information focusing on our book topics and related technologies. The site offers the following features:

- One-year warranty against content obsolescence due to vendor product upgrades. You can access online updates for any affected chapters.
- "Ask the Author" customer query forms that enable you to post questions to our authors and editors.
- Exclusive monthly mailings in which our experts provide answers to reader queries and clear explanations of complex material.
- Regularly updated links to sites specially selected by our editors for readers desiring additional reliable information on key topics.

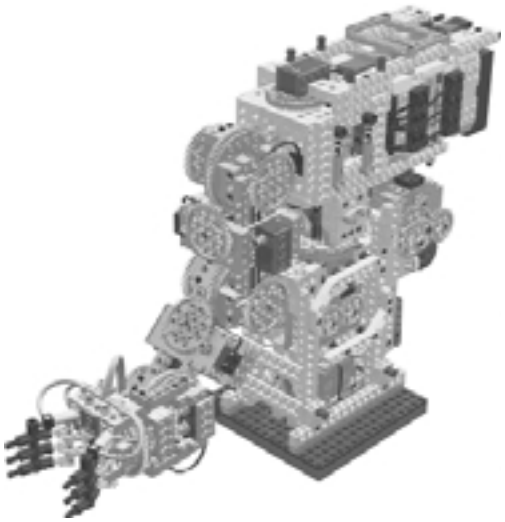
Best of all, the book you're now holding is your key to this amazing site. Just go to **www.syngress.com/solutions**, and keep this book handy when you register to verify your purchase.

Thank you for giving us the opportunity to serve your needs. And be sure to let us know if there's anything else we can do to help you get the maximum value from your investment. We're listening.

www.syngress.com/solutions

SYNGRESS®

SYNGRESS®



LEGO® Mindstorms™ Masterpieces

Building and Programming Advanced Robots

Miguel Agulló
Doug Carlson
Kevin Clague
Giulio Ferrari
Mario Ferrari
Ralph Hempel Technical Reviewer

KEY	SERIAL NUMBER
001	LK9H7GBUNN
002	E2UT6CVN7F
003	GJ9HFMZATT
004	Q2B763QWT2
005	R8MPJAXQU3
006	U6B7ATH3N3
007	L8D4EMUSAK
008	EBKMCUTT3R
009	SW4KGQQSEH
010	HENDQUM39Z

PUBLISHED BY
Syngress Publishing, Inc.
800 Hingham Street
Rockland, MA 02370

LEGO MINDSTORMS Masterpieces Building & Programming Advanced Robots

Printed in the United States of America

1 2 3 4 5 6 7 8 9 0

ISBN: 1-931836-75-2

Technical Reviewer: Ralph Hempel
Acquisitions Editor: Catherine B. Nolan
CD Production: Michael Donovan

Cover Designer: Michael Kavish
Page Layout and Art by: Shannon Tozier & Patricia Lupien
Copy Editor: Cherina Sparks
Indexer: Odessa & Cie

Distributed by Publishers Group West in the United States and Jaguar Book Group in Canada.

Acknowledgments

We would like to acknowledge the following people for their kindness and support in making this book possible.

A special thanks to Matt Gerber at BricksWest for his help and support for our books. Karen Cross, Lance Tilford, Meaghan Cunningham, Kim Wylie, Harry Kirchner, Kevin Votel, Kent Anderson, Frida Yara, Jon Mayes, John Mesjak, Peg O'Donnell, Sandra Patterson, Betty Redmond, Roy Remer, Ron Shapiro, Patricia Kelly, Kristin Keith, Jennifer Pascal, Doug Reil, David Dahl, Janis Carpenter, and Susan Fryer of Publishers Group West for sharing their incredible marketing experience and expertise.

The incredibly hard working team at Elsevier Science, including Jonathan Bunkell, AnnHelen Lindeholm, Duncan Enright, David Burton, Rosanna Ramacciotti, Robert Fairbrother, Miguel Sanchez, Klaus Beran, and Rosie Moss for making certain that our vision remains worldwide in scope.

David Buckland, Wendi Wong, Daniel Loh, Marie Chieng, Lucy Chong, Leslie Lim, Audrey Gan, and Joseph Chan of STP Distributors for the enthusiasm with which they receive our books.

Kwon Sung June at Acorn Publishing for his support.

Jackie Gross, Gayle Voycey, Alexia Penny, Anik Robitaille, Craig Siddall, Darlene Morrow, Iolanda Miller, Jane Mackay, and Marie Skelly at Jackie Gross & Associates for all their help and enthusiasm representing our product in Canada.

Lois Fraser, Connie McMenemy, Shannon Russell, and the rest of the great folks at Jaguar Book Group for their help with distribution of Syngress books in Canada.

David Scott, Tricia Wilden, Marilla Burgess, Annette Scott, Geoff Ebbs, Hedley Partis, Bec Lowe, and Mark Langley of Woodslane for distributing our books throughout Australia, New Zealand, Papua New Guinea, Fiji Tonga, Solomon Islands, and the Cook Islands.

Winston Lim of Global Publishing for his help and support with distribution of Syngress books in the Philippines.

The Masters

Miguel Agulló is a "professional expatriate" in many senses. Born in Spain, he has lived abroad for long periods of time, from the Far East to South America, from central Europe to the U.S. Formally trained as a journalist, his wide range of interests and set of skills have enabled him to work in such diverse industries as finance, media, aeronautics - and even antique trading! As an adult, he picked up his old childhood LEGO hobby, using it as a tool to prototype designs of his invention. His building interests revolve around robotics, and specifically biomechanics: creating mechanisms that mimic the behavior of natural devices such as legs or arms. His creations include biped walkers, robots that jump, as well as more traditional designs like a fully functional (including a brake!) LEGO motorcycle. Miguel is a contributor to Syngress Publishing's *LEGO MINDSTORMS Robotics Invention System Inventor's Guide* (ISBN: 1-931836-59-0) and *LEGO MINDSTORMS Robotics Invention System Constructopedia* (ISBN: 1-931836-76-0).

Doug Carlson is a Lego Master that specializes in mobile configurations. Some of his latest work can be seen at www.visi.com/~dc, including his innovative tri-star wheel design, Killough's mobile robot platforms, synchro drive platforms, pneumatic hexapod. Doug is an active member of the FLL, specifically in Minnesota where he resides with his family.

Kevin Clague is a Senior Staff Engineer at Sun Microsystems, where he does verification work on their Ultra-SPARC V RISC processor. He also worked for Amdahl Corporation for 18 years as a Diagnostic Engineer. Kevin played with LEGO as a child, and got back into LEGO as an adult when his wife, Jan, got him the LEGO MINDSTORMS Dark Side Developer Kit for Christmas three years ago. Kevin soon got himself a LEGO MINDSTORMS Robotics Invention System 1.5 set, and has been having fun inventing LEGO creations ever since. In 2001 Kevin got involved with authoring LEGO instruction books for Syngress Publishing, including *LEGO MINDSTORMS Robotics Invention System Inventor's Guide* (ISBN: 1-931836-59-0), *LEGO MINDSTORMS Robotics Invention System Constructopedia* (ISBN: 1-931836-76-0). In the process, Kevin developed the LPub program for creating professional quality building instructions using MLCad, L3P, and POV-Ray. More recently, Kevin has developed the LSynth program so that bendable LEGO parts can be more easily documented when creating building instructions. Kevin would like to thank his wife, Jan, and children, Aaron, Tony, Allison, and Andrew for "ooohing" and "aaahing" over his LEGO creations.

Giulio Ferrari works as a Software Developer at EDIS, a leader in publishing and finishing solution and promotional packaging. He studied engineering and economics at the University of Modena and Reggio Emilia, and in the past has developed applications, entertainment software, and Web sites for several companies. He is fond of physical and mathematical sciences, as well as of puzzles and games in general (he has a collection of 1500 dice of every kind and shape). Giulio co-authored the best selling LEGO MINDSTORMS Power Programming (Syngress Publishing, ISBN: 1-928994-67-9) with his brother, Mario and Ralph Hempel, a book that has quickly become a fundamental reference and source of ideas for many LEGO robotics fans. Giulio is also a contributor to LEGO MINDSTORMS Masterpieces (Syngress, ISBN: 1-928994-55-5). He has been playing with LEGO bricks since he was very young, and his passion for robotics started in 1998, with the arrival of the MINDSTORMS series. From that moment on, he held an important place in the creation of the Italian LEGO community, ItLUG, now one of the largest and most important LEGO users group worldwide. He works in Modena, Italy, where he lives with his girlfriend, Marina.

Mario Ferrari received his first LEGO box around 1964, when he was four-years-old. LEGO was his favorite toy for many years, until he thought he was too old to play with it. In 1998, the LEGO MINDSTORMS RIS set gave him reason to again have LEGO become his main addiction. Mario believes LEGO is the closest thing to the perfect toy. He is Managing Director at EDIS, a leader in finishing and packaging solutions and promotional packaging. The advent of the MINDSTORMS product line represented for him the perfect opportunity to combine his interest in IT and robotics with his passion for LEGO bricks. Mario has been an active member of the online MINDSTORMS community from the beginning and has pushed LEGO robotics to its limits. Mario holds a bachelor's degree in Business Administration from the University of Turin and has always nourished a strong interest for physics, mathematics, and computer science. He is fluent in many programming languages and his background includes positions as an IT Manager and as a Project Supervisor. With his brother, Giulio Ferrari, Mario is the co-author of the highly successful book LEGO MINDSTORMS Power Programming (Syngress Publishing, ISBN: 1-928994-67-9). Mario estimates he owns over 60,000 LEGO pieces. Mario works in Modena, Italy, where he lives with his wife, Anna, and his children, Sebastiano and Camilla.

Hideaki Yabuki works as a Media Activist, promoting new technologies to the next generation, at the Pioneer Group. Hideaki also gives lectures to high school students in Japan about LEGO. In 1982, after graduation from the College of Science and Technology of Nihon University in Tokyo with a bachelor's degree in Engineering, he was blessed with the results of the MIT Media Lab and worked for the application of advanced technologies using computers such as Laser, Holography, Robot, Multimedia and Internet thus far. To him, robotics is the most important of these technologies and he

is grateful to have the great opportunity of researching robots for his work again from 2000.

In 1985, he was first introduced to LEGO robots by a friend of his who had recently returned from the Media Lab with some LEGO educational products. Influenced also by Dr. Seymour Papert's book, , he feels that LEGOs offer a hands-on approach to learning that is often missing these days in our digital world. His robot in this book, the CyberArm, is the result of much trial and error on his part. Also, he was a contributing author for (Syngress Publishing, ISBN: 1-931836-59-0).

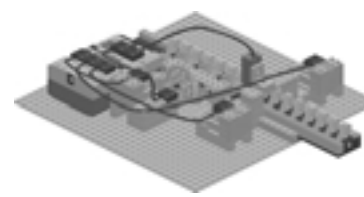
Hideaki would like to thank Catherine Nolan, Cherina Sparks and Luke Ma, because they appreciated his vision and helped him by editing his clumsy writing. He would also like to thank all those people who help his writing and encourage him as follows: Jonathan Babcock, Brian Bagnall, J.P. Brown, Ralph Hempel, Jin Sato, Christopher Smith, Russell Stoll, Edmund Nussbaum, STELARC, Shinichi Kurita, Prof. Yoshikazu Suematsu, Noriko Kageyama, Yoichi Tagi, Masanori Konno, Prof. Masashi Shimizu, Yoshihito Isogawa and all of the co-authors of this book, particularly Mario Ferrari. Lastly, Hideaki would like to give his deepest thanks for the support of his mother, Rei, and his dear wife and son, Keiko and Kei. Hideaki has a dream that one day the people on this planet will be able to join hands with biped robots as friends.

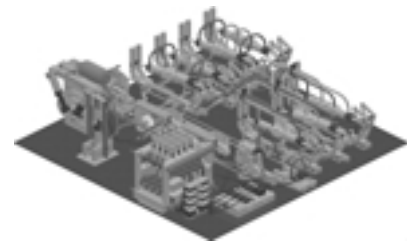
Luke Ma received his bachelor's of Arts degree in Music and Computer Science from Brown University in May 2003. He is currently a starving graduate student studying at the University of California, Santa Barbara. His main field is music theory and thus he continues to spend most of his time analyzing pieces of obscure classical music in even more obscure ways. He has also been known to play the piano as well as sing poorly on occasion. On the technological side of things, Luke has worked for Latitude Communications Inc. as an engineering intern, helping them develop and expand their Web-conferencing platform. He also has extensive experience in designing and publishing Web sites. He is a contributor to Syngress Publishing's (ISBN: 1-931836-61-2). He is fluent in C/C++, JavaScript, HTML/DHTML, Chinese, English, and hopefully French, German, and Japanese sometime in the near future. Luke would like to thank Catherine Nolan of Syngress for all her help (again!) and the opportunity to work with Syngress and to Joda for his input and for writing a wonderful chapter. Luke would also like to thank his parents for their support and his friends for putting up with him and making his life fun and enjoyable.

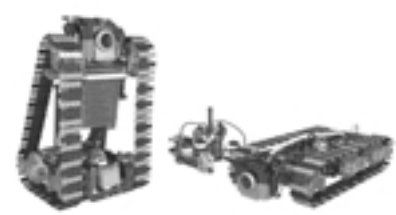
Technical Reviewer

Ralph Hempel (P.Eng) is an Independent Embedded Systems Consultant. He provides systems design services, training, and programming to clients across North America. His specialty is in deeply embedded microcontroller applications, which include alarm systems, automotive controls, and the LEGO RCX system. He is the technical editor of LF P (Syngress Publishing, ISBN: 1-928994-67-9). Ralph provides training and mentoring for software development teams that are new to embedded systems and need an in-depth review of the unique requirements of this type of programming. Ralph holds a degree in Electrical Engineering from the University of Waterloo and is a member of the Ontario Society of Professional Engineers. He lives in Owen Sound, Ontario, Canada, with his family, Christine, Owen, Eric, and Graham.

Contents







Foreword

When LF was finished I felt very satisfied, but completely exhausted. I had been working on the book three to four hours a day — every day — for more than five months, dedicating to it all of my spare time and stealing some additional time from my job and my family. My most recurring thought was: "This has been an incredible opportunity, I'm so glad I did it" immediately followed by "but I'll never do it again".

The months that followed were incredibly exciting. Although the memory of the hard work was still fresh in my mind, it was more than compensated for by the incredible reception that the book received from MINDSTORMS fans. In term of sales — of course — but also in terms of the flattering reviews and very positive comments that many readers expressed directly to me by e-mail.

When Syngress offered me the opportunity to write a second book, I had mixed feelings. On the one hand, I was so proud of the welcome the first book had received that I was tempted to accept; on the other hand, I really didn't feel ready to repeat the effort required to write the first book. More importantly, I didn't want to write a book just to write another book, with nothing special in it.

LF filled a gap in the technical literature about LEGO robotics that had been empty, but the number of books devoted to the LEGO MINDSTORMS system was already rather high before its publication, and many other good titles have been released on the topic since then.

Discussing ideas and possibilities with the Syngress staff, the concept for a new book slowly began to evolve. LF explains scores of techniques about designing and building robots, describing about thirty complete projects and offers suggestions for many others. However, it contains no explanation of these projects with the level of detail to where everybody can reproduce a specific robot, following step-by-step instructions. This fact has been pointed

out by some readers, who said they would have appreciated a book describing challenging projects, each looked at in great detail.

Actually, the shelves of many bookshops offer many excellent books containing step-by-step instructions for building LEGO MINDSTORMS robots. Nevertheless, if you look carefully you will discover that some of these books are aimed at introducing the reader to the basics of the MINDSTORMS world, therefore describing only very simple robots. On the other hand, there exists a second group of books that show step-by-step instructions, but lack substantially detailed explanations about the building techniques used by the authors and the general concepts that are fundamental to the projects.

These reflections are the foundation of *LEGO MINDSTORMS*, whose goal is presenting some very sophisticated projects with the maximum level of detail. In the book you hold in your hands you will find not only high quality step-by-step instructions for building all of the robots, but also a complete description for each of them, including goals, building techniques, programming techniques, and the theory supporting the designer's choices.

In my opinion what makes *LEGO MINDSTORMS* really special is the fact that it is not the work of a single person, but rather the result of the cooperation of a team of authors. The fact is that any designer tends to adopt his or her own building "clichés", a collection of standard solutions to common problems that they use again and again to save design time. This isn't laziness, but rather the normal behavior of the human mind. When faced with a new problem, the first solutions that come to mind are solutions that have already been successful in similar (but perhaps not identical) cases. Additionally, every designer has their own personal style, a set of unexpressed rules that guide building choices toward something that the designer prefers to use, even if another solution exists that would work equally as well or better. For these reasons different creations built by the same person typically show many similarities; a sort of common background that results in what can be thought of as a "family resemblance" among their designs.

The MINDSTORMS Masters that wrote this book brought to it a wide variety of ideas and a wealth of strategies that a single author could simply never hope to offer. This diversity is exactly what we are offering in this book, which we called — with a bit of immodesty — *LEGO MINDSTORMS*. Whether these creations could be considered real masterpieces is not for me to say, but what I am sure of is that we devoted a considerable amount of time to choosing and presenting projects that push the possibilities of the MINDSTORMS RIS system towards its limits,

while at the same time covering a broad range of topics. Within the pages of LEGO you'll find vehicles based on amazing mechanical solutions, robots that introduce you to some basic concepts of Artificial Intelligence, and complex machines that — though not strictly definable "robots" — demonstrate all the power of the LEGO system as a modeling tool.

Each chapter presents a single robot in extreme detail, offering both an introduction to some basic concepts involved in the project and offers cues for many further investigations. To clarify, let me use Kevin Clague's PneumADDic II (Masterpiece 4) as an example. You can build the model just because it's amazing and because you like it, but in the chapter you will also find a detailed introduction to pneumatics, to binary math, to Boolean logic and to the method used in a CPU to add two numbers. During the construction of the machine, you will be presented with a wide collection of techniques, which range from some basic building techniques, to more complex matters such as how to use a single motor for multiple tasks, or how to overcome the limits of the RCX with regards to its number of input ports. Even if you are not interested in the model itself, or if you don't have the large supply of pneumatic parts it requires, there are many other reasons to read Kevin's chapter with attention, because I'm sure you'll find many useful tips to transfer to your own designs.

The other chapters show the same richness, and the same attitude not to take any important concept for granted. I want to invite you to not think of these masterpieces as watertight compartments; because there are many ways you can transfer ideas and solutions from one model to another, or to your own projects. For example, the concept of making a vehicle change its shape, introduced by Miguel Agulló's Shape-Shifting Camera Tank, could be applied to Doug Carlson's Stair-Climber; or the pneumatic logic used in both Kevin Clague's projects could be used to partially automate the movement of the arm in my Learning Brick Sorter; or the "Power Glove" used in Hideaki Yabuki's CyberArm IV could be used as an elegant solution to drive other kind of mechanisms or tele-controlled robots.

I hope I succeed in making you curious about this book. Before letting you dive into its actual content, I'd like to introduce you briefly to each chapter and its author.

Masterpiece 1 Stair-Climber, by Doug Carlson. I met Doug in Toronto, during a robotic event organized by the local rtlToronto LEGO robotics community. Doug attending the contest with an amazing robot, and also showing an impressive collection of LEGO machines of various kinds that were only partially depicted on his Web site (www.visi.com/~dc). One of these machines was the Stair-Climber that

Doug is presenting — in a new, improved version — in this chapter. This vehicle is based on a special kind of triangular wheel, called , that are actually made of three wheels placed at the vertexes of an equilateral triangle. This is a device that you have to see it in action to believe what it's capable of!

Masterpiece 2 Learning Brick Sorter, by Mario Ferrari. For my contribution to this book, I chose a project which admittedly is not an entirely new idea. In fact, I had already published a similar project on my Web page in 1999 (www.mario-ferrari.org). However, apart from being one of my favorite projects, this robot aroused a lot of interest and I received many e-mails asking for details about the building steps and programming instructions. The Learning Brick Sorter has even been used as a starting point for a few graduate theses. This interest, made me think that it would be a good candidate to appear in this book, offering the reader the opportunity to have a look at the intriguing world of Artificial Intelligence. The Learning Brick Sorter is probably the most software-centered model of *LEGO*, but for readers who are already familiar with the robot you'll note that this current iteration has been entirely re-designed and re-programmed.

Masterpiece 3 The LEGO Turing Machine, by Giulio Ferrari. When I described the efforts that writing *LEGO* required at the beginning of this Foreword — I should have used the plural, as my brother Giulio held up half of the strain. In this chapter, Giulio presents a LEGO version of the Turing Machine, a computing machine devised by the famous mathematician Alan Turing and a pillar of modern theory of computation. This device is usually considered a theoretical representation, a sort of mental experiment, to demonstrate what a computing machine can and cannot do. However, it can be actually built, and Giulio's version is the starting point for a captivating journey into the world of calculus and into the mind of a mathematical genius.

Masterpiece 4 PneumADDic II, by Kevin Clague. I got in touch with Kevin by e-mail for the first time in April 2002, about one year before the publication of this book. Though I had been spending many hours browsing the Internet for sites about LEGO robotics, until that moment I had unbelievably missed Kevin's page, a true gold mine of original ideas and clever implementations (www.users.qwest.net/~kclague). I fell in love immediately with his pneumatic adding machine, the same one you'll find in this chapter in a revised and improved shape. As I previously explained, this project is not only very interesting on its own, but also contains a collection of tips and techniques that are fundamental tools for any serious Mindstormer.

Masterpiece 5 Synchropillar, by Kevin Clague. Kevin's second project in LEGO is a pneumatic caterpillar. Synchropillar is a robot which has no RCX and no electric component of any kind. You might be surprised to find a model without an RCX in a book devoted to the MINDSTORMS system. However, this fact is actually what makes the Synchropillar so interesting: It demonstrates that by using pneumatics you can achieve some simple automation characteristics, and that you can build a machine able to iterate through different states without the need of an electronic controller. This chapter shows how pneumatic logic should be considered as an additional design resource which can be used to delegate some simple operations away from the RCX.

Masterpiece 6 Shape-Shifting Camera Tank, by Miguel Agulló. Inside the diversified world of the LEGO hobbyists, Miguel and Kevin share two common interests: Computer Aided Design (CAD) and biped robots. In the first field they can be considered true experts, and their book LEGO (ISBN: 1-931836-76-0) is an unavoidable reference point for whomever wants to approach the creation of professional building instructions, like those you find in this book. Additionally, Miguel is well known for a Web page that contains not only his own projects, but an impressive collection of information and links to everything has been published on the Internet about LEGO bipeds (www.geocities.com/technicpuppy/index.html). In this chapter, however, Miguel sets aside his passion for walking robots and describes a very special tracked vehicle, which is able to change its shape: When completely flattened, it's not taller than its tracks, but it can lift up and let less than half its tracks touching the ground. It's a small masterpiece of engineering, and introduces you to some sophisticated techniques I'm sure will proof useful for many other projects.

Masterpiece 7 CyberArm IV, by Hideaki Yabuki. Hideaki's passion is building robotic arms. He bought the MINDSTORMS System with this precise goal in mind, and his CyberArm series testifies the incredible level he has reached in this specific field. With his never-ending search for perfection, Hideaki demonstrates that robotics can be approached like an art (http://mindstorms.lego.com/eng/community/pioneers/joda/default.asp). In his CyberArm IV even the smallest detail comes from careful thinking and the evaluation of different options, where for every choice complexity is perfectly balanced by aesthetics. This incredible attention shows clearly in the final result, a multiple RCX, with over a thousand pieces, five degrees of freedom robotic arm, which, in my opinion, is a true masterpiece.

These masterpieces are true testaments that almost anything is possible with LEGO, don't feel limited by the constraints you fine in the system, feel inspired. Take our ideas and instructions and create a robot that makes your friends say, "I didn't think it was possible to make such an incredible thing with LEGO" Because you can.

—

Co-author of

LF P

(ISBN: 1-928994-67-9)

About This Book

Each of the masterpieces in this book is presented using a method that makes its construction as easy and intuitive as possible. Each chapter begins with a picture of the completed robot, a bill of materials for the entire robot (so that you can preview all of the parts that are required to construct that specific masterpiece, and a brief introduction to the robot's history, its unique challenges and characteristics, as well as any concerns that the robot's creator wants you to be aware of during construction.

The instructions for building each robot are broken down into several sub-assemblies, which each consist of an integral structural component of the finished robot. You will see a picture of each finished sub-assembly before you begin its construction.

You will be guided through the construction of each sub-assembly by following the individual building steps, beginning with Step 0. Each step shows you two important things—what parts you need, and what to do with them—by using two pictures. The *parts list* picture shows you which LEGO bricks you will need for that particular step, as well as the quantity of parts required, and the color of the parts (if necessary). However that you should note that these colors are suggested only as guidelines, and any parts can be substituted to suit your own preferences. Since this book is printed in black and white, we have used the following key to represent the colors:

- **B** Blue
- **G** Green
- **M** Magenta
- **LB** Light Blue
- **Y** Yellow

- **Ppl** Purple
- **TLG** Transparent Light Green
- **TY** Transparent Yellow
- **R** Red

The *instructional* picture next to the parts list shows how those parts connect to one another. As the robot's construction progresses, it gets harder to see where parts get added, so you'll see we have made the parts that you add in each particular step *darker than* those added in previous steps. Many of the steps also have a few brief lines of text to more fully explain building procedures that may not be obvious from the pictures alone, or to discuss what role this step plays in the larger scheme of the robot's construction.

Once you have finished building all of the separate sub-assemblies, it's time to put them all together to complete the robot. The set of steps at the end of each chapter titled "Putting It All Together" walks you through the process of attaching together the sub-assemblies.

Throughout the chapters you will see three types of sidebars:

- **Bricks & Chips…** These sidebars explain key LEGO building concepts and terminology.
- **Developing & Deploying…** These sidebars explain why certain building techniques were used with a particular robot and what purpose they serve.
- **Inventing…** These sidebars offer suggestions for customizing the robots or modifying the robots using alternate parts should you not have the parts listed in the parts list at your disposal.

About the CD-ROM

Building your robots is, or course, only half the fun! Getting them to run using the RCX brick is what distinguishes MINDSTORMS robots from ordinary models created with LEGO bricks. Some of the robots in this book will require specific programs for these robots to perform as intended. Many of them will use unique programs that the authors have written specifically for their robots. Keep an eye out for the CD-ROM icons scattered throughout the book.

These icons alert you to the fact that there is code for this particular robot available on the companion CD-ROM for the book. The downloadable programs for the

robots in this book are written in two of the most common programming languages used for LEGO MINDSTORMS:

- **RCX** LEGO's official programming language.
- **NQC** Standing for "Not Quite C," NQC is a programming language created by Dave Baum. Very similar in many ways to the C computer programming language, NQC is a text-based language that is more powerful and flexible than RCX.

For instruction on uploading these programs to your RCX brick, refer to the documentation that came with your LEGO MINDSTORMS RIS 2.0 kit.

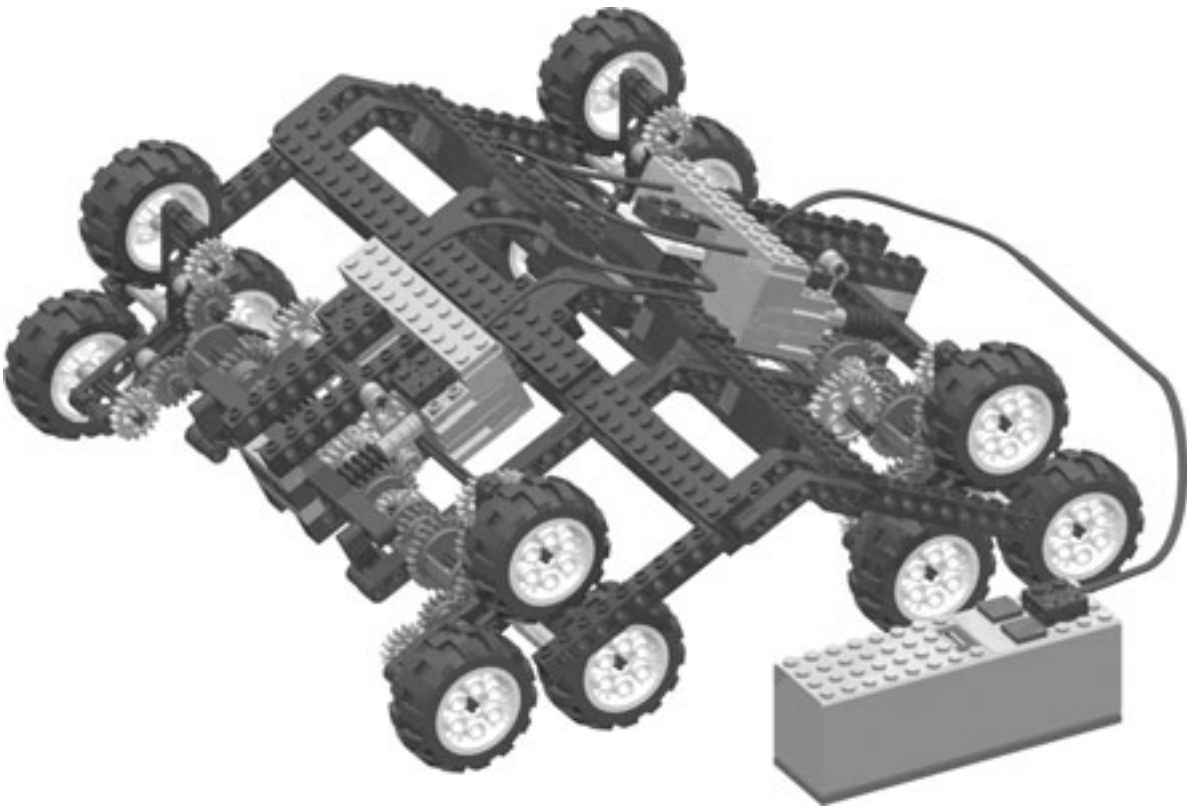
The code files and movies of the robot in action, or additional photos of the robot are located in a *MasterpieceXX* directory. For example, the files for Masterpiece 5 are located in folder Masterpiece05. Any further directory structure depends upon the specific files included for the robot in that particular chapter. In some cases there are movies of the robots in action.

Of special interest is the Appendix *The Origin of My Designs: From Robotics to LEGOs, the Genesis of my Ideas and the CyberArm Series* to Masterpiece 7 "CyberArm IV" by Hideaki Yabuki. This addition takes readers on a journey into the mind of one of the LEGO MINDSTORM pioneers, as Joda discusses the cultural and historical influences that fuel his passion for building robotic arms.

Masterpiece 1

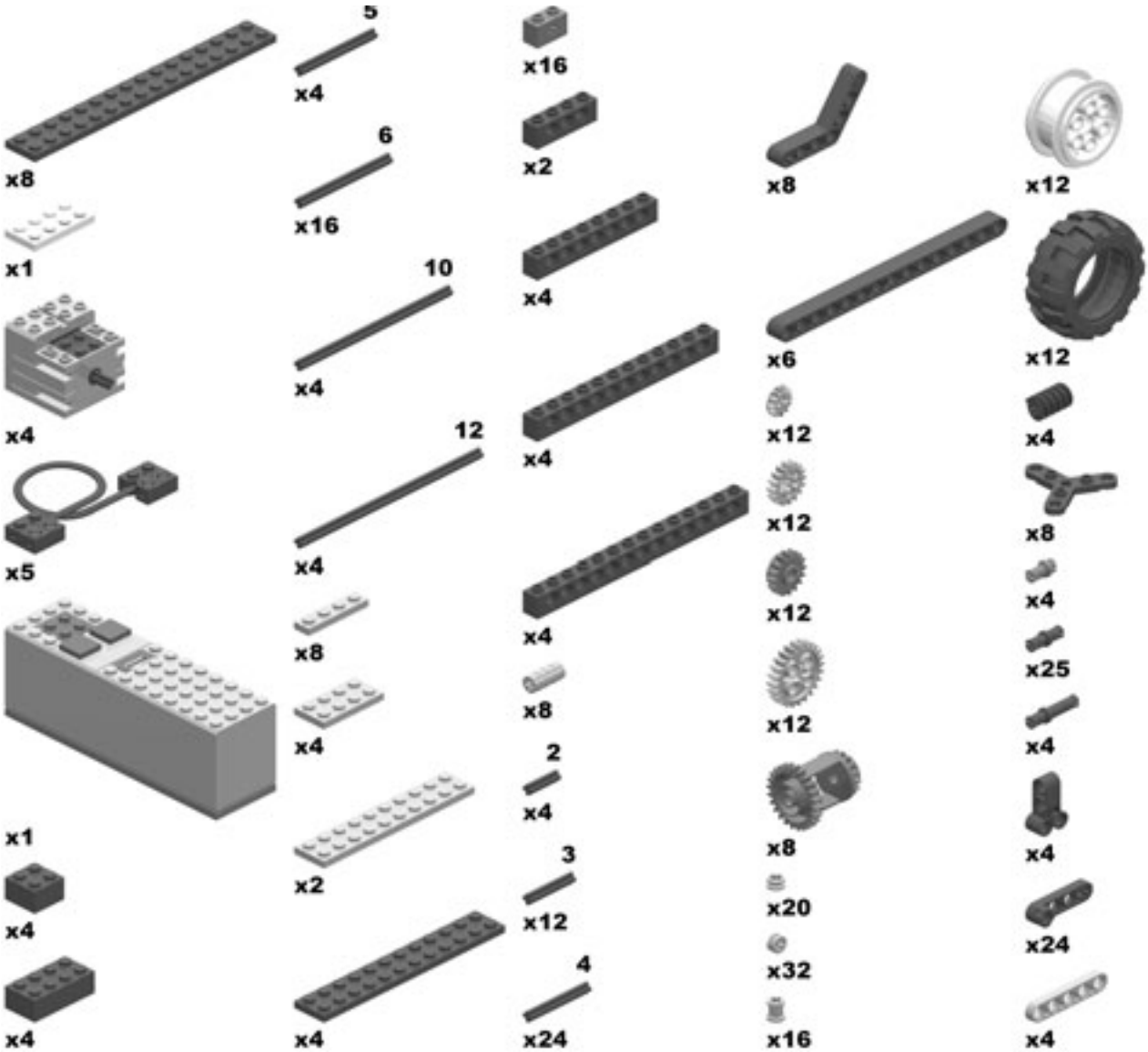
Stair-Climber

Doug Carlson



Bill of Materials

These are the parts you will need to build the Stair-Climber as shown.



Introduction

Stair-Climber is the latest in a series of models based on the tri-star wheel concept. The tri-star wheel is a triangular star-wheel arrangement in which the wheels can be driven in a normal fashion for rolling over flat terrain (Figure 1.1), traversing a terrain gap (Figure 1.2) or climbing over obstacles, as the whole assembly can be rotates (Figure 1.3). One extreme obstacle for many robotic vehicles—but for which the tri-star configuration really excels—is climbing stairs. When was the last time you saw something with wheels actually climb a set of stairs?

Figure 1.1 Rolling Mode

Figure 1.2 Traversing a Terrain Gap

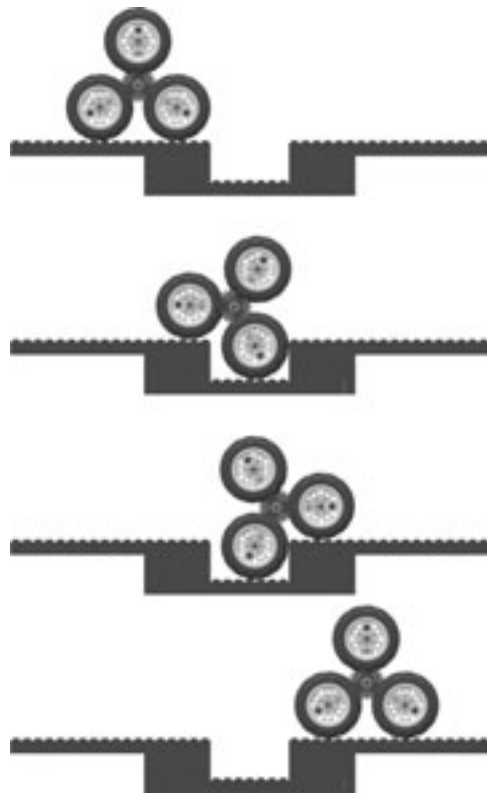
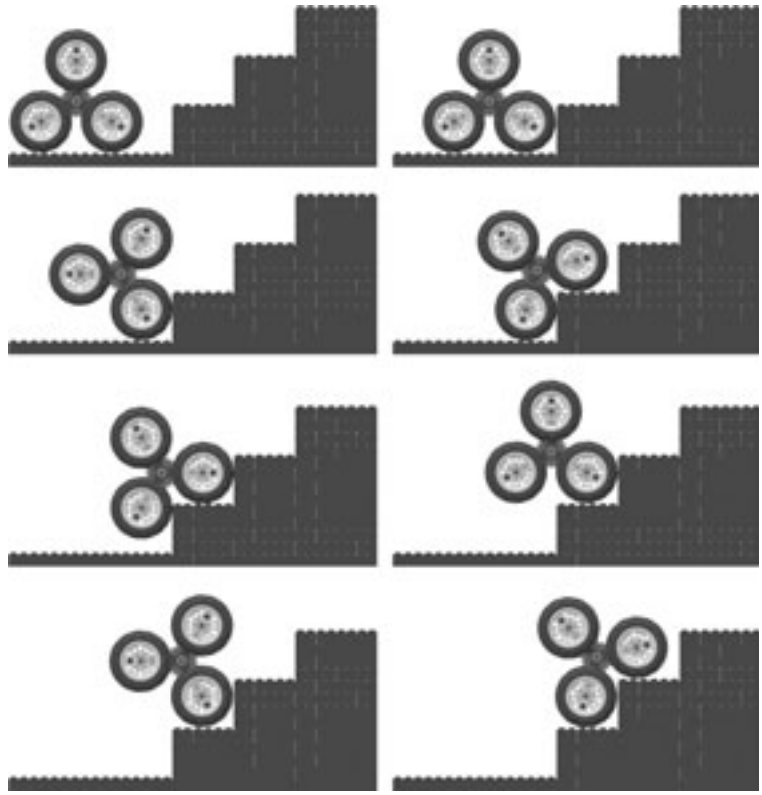


Figure 1.3 Climbing Mode



I came across this concept some years back on Cynde Callera's LEGO Web page: http://tyranny.egregious.net/~khrome/lego. The sketches and information there captured my imagination, and I had soon built several tri-star variations. To review some of my earlier designs, please take a look at: www.visi.com/~dc.

While experimenting with these variations, it became clear to me that for stair climbing, as opposed to minor obstacle avoidance, it was necessary to provide separate drive mechanisms for both the wheel assembly and the wheels. The Stair-Climber model uses a differential to split the drive torque between the two separate drive modes. On a flat surface, the model will roll along like any normal wheeled vehicle. However, as soon as the model encounters enough resistance to start climbing, as when something blocks the wheel from rolling, the drive torque is transferred in order to rotate the wheel assembly that begins the climbing process. Each tri-star wheel has its own drive motor and differential to enable independent wheel action, as well as enough torque to easily climb a set of LEGO sized stairs at a reasonable speed.

Another way to accomplish stair climbing would be to use separate drive motors for the climbing and rolling functions. Many sensors would be needed to determine the vehicle's position and attitude relative to stairs and other terrain. If you had enough computing power and sensors you could possibly use only one pair of star-wheels, program

the robot to balance on two wheels, and climb stairs as well. Recently, Dean Kamen has created an incredible wheelchair (iBOT) with all these capabilities. It has two wheels per side used in a bi-star configuration (Figures 1.4 and Figure 1.5). Check out the following web sites to find out more on this incredible device:

- www.dekaresearch.com
- www.indetech.com
- www.dynopower.freeserve.co.uk/homepages/newchair.htm

Figure 1.4 The iBOT Wheelchair in Elevated Mode

Figure 1.5 iBOT Wheelchair Descending Stairs

Building the Stair Climber

There are two sets of instructions and two corresponding part lists for building the basic tri-star wheel assemblies:

- The first option uses the older style TECHNIC tri-plate and associated toothed bushings.
- The second option uses a newer tri-beam piece available in one of the LEGO Spybotic sets, and possibly other sets as well.

As a third option, it would be possible to mix both types of tri-star wheels within the same model, as they are functionally equivalent. Use whatever combination you find convenient. When completed, the Stair-Climber is symmetrical from side to side and front to rear. Remember this when following the instructions, as many parts being added may be hidden from view. The only exception is the motor wiring, which is all tied to one common point.

Engineering Trade-Offs

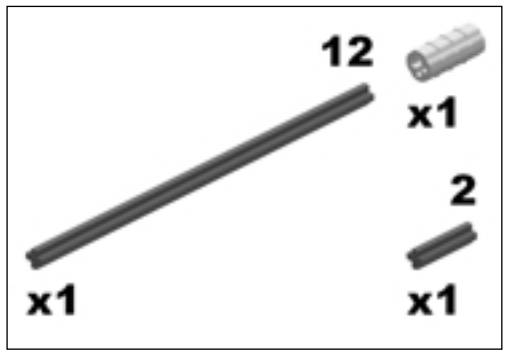
So, by now you may be wondering where is the RCX? Well, as much fun as it would be to make this model autonomous by adding a few sensors and RCX, it really isn't practical. All the extra weight is just too much for the tri-star assemblies of this model to function properly. When attempting to climb with higher loads, the excessive torque on the main tri-star axles leads to breaking axles and gears with higher loads.
Beyond this, the main goal of this model was stair climbing and attaching an RCX to the model would raise the center of gravity (CG) enough to seriously limit vehicle stability, thus causing the model to flip over backwards when climbing steeper inclines.

However, you could use an RCX as a handheld battery pack if you wanted to, and I will show you how this is possible toward the end of the chapter. But then again, the Stair-Climber doesn't need the processing power, as it uses the differentials to shift between modes. There is one more reason for just using a battery pack with this particular model: if you try to turn this model by powering each side separately, you may find the wheels will slide out of place and jam.

The Wheel Set

The tri-star wheel set has a concentric drive arrangement to provide separate power for both rolling and stepping modes of operation. The differential housing used without internal gears provides this concentric drive mechanism and acts to hold the dark gray 16T gears in place. You will need to build **four** of these.

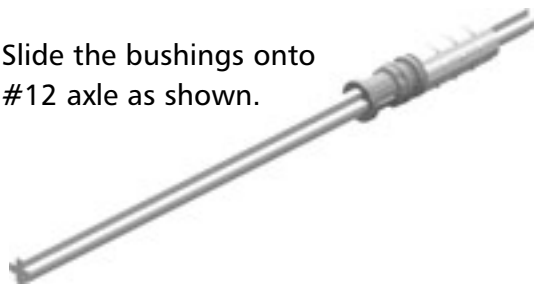
Wheel Set Step 0



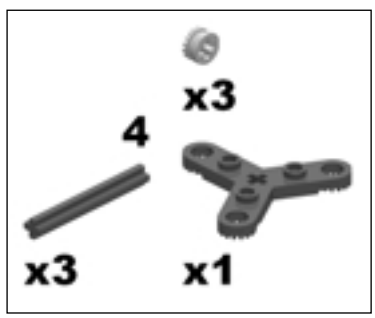
Connect the #12 axle and the #2 axle together using the axle joiner as shown.

Wheel Set Step 1

Slide the bushings onto #12 axle as shown.

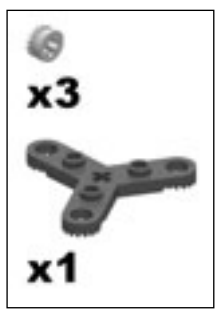


Wheel Set Step 2



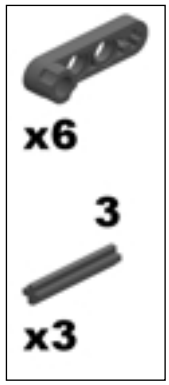
Slide the tri-plate onto the #12 axle with orientation as shown. Install the three #4 axles and half-bushings so that the axle is aligned with its corresponding tri-plate section. There will need to be just enough axle extending behind the tri-plate to attach a half-thickness liftarm.

Wheel Set Step 3

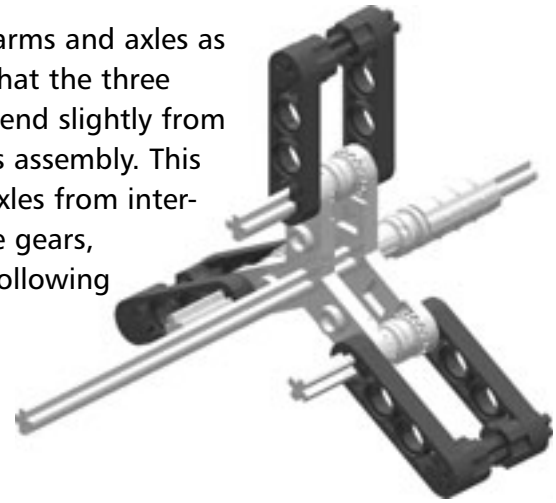


Slide the bushings and a second tri-plate into place.

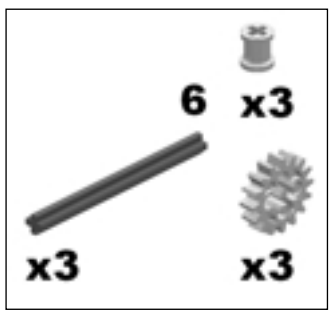
Wheel Set Step 4

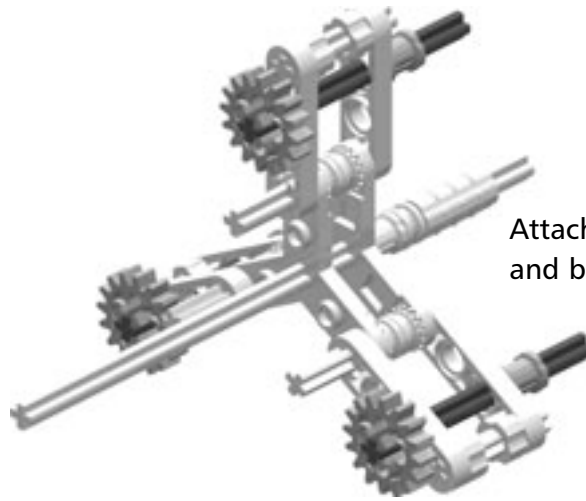


Attach the liftarms and axles as shown. Note that the three outer axles extend slightly from the rear of this assembly. This prevents the axles from interfering with the gears, added in the following step.



Wheel Set Step 5

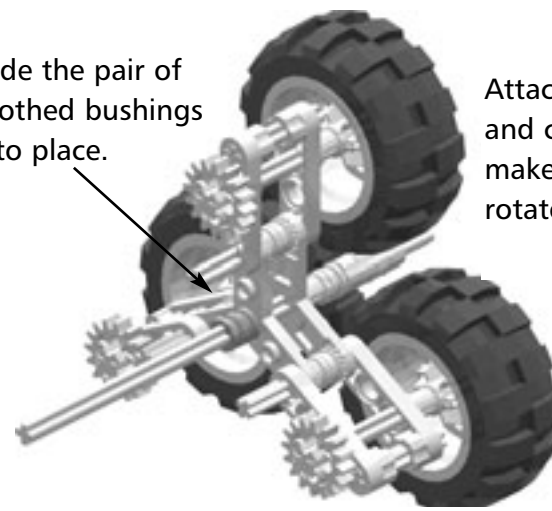




Attach the gears, axles, and bushings as shown.

Wheel Set Step 6

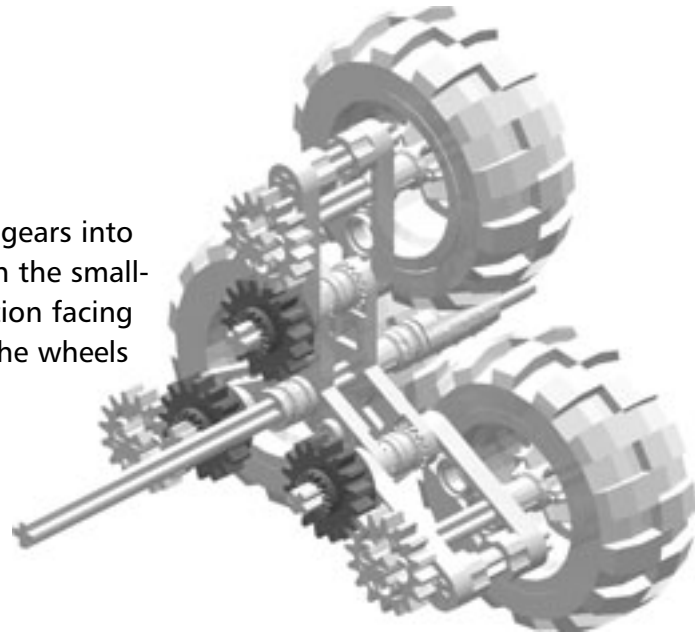
Slide the pair of toothed bushings into place.



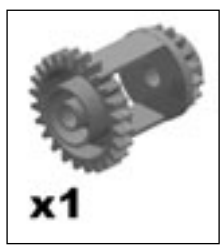
Attach the wheels and check to make sure they rotate easily.

Wheel Set Step 7

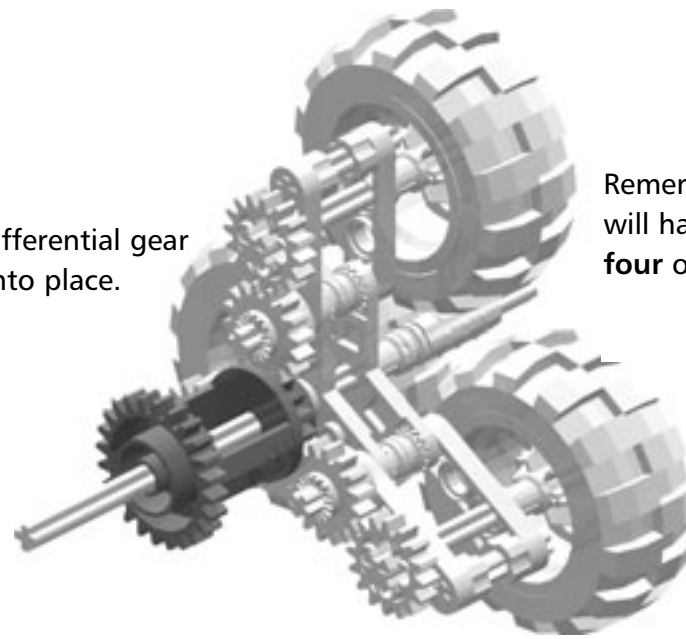
Slip the 16T gears into position with the small-toothed section facing away from the wheels



Wheel Set Step 8

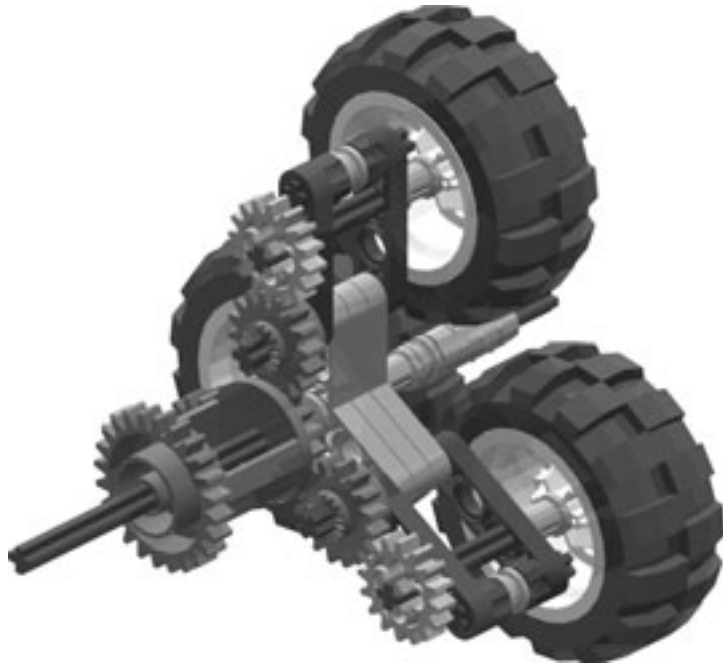


Slip the differential gear housing into place.



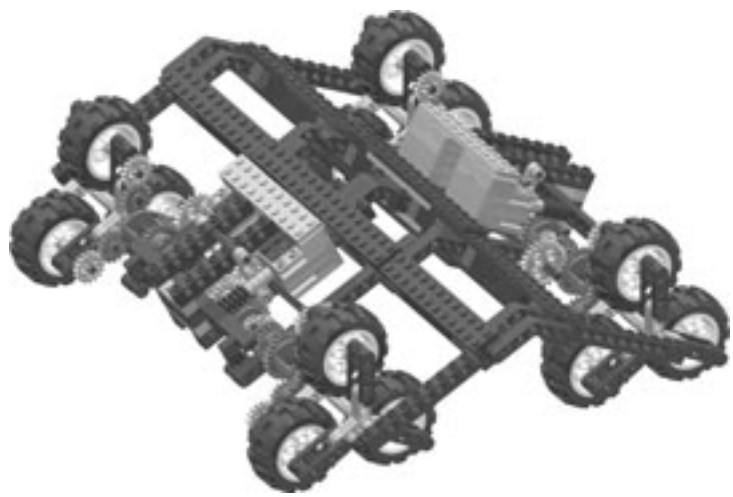
Remember you will have to build **four** of these.

Building an Alternate Wheel Set

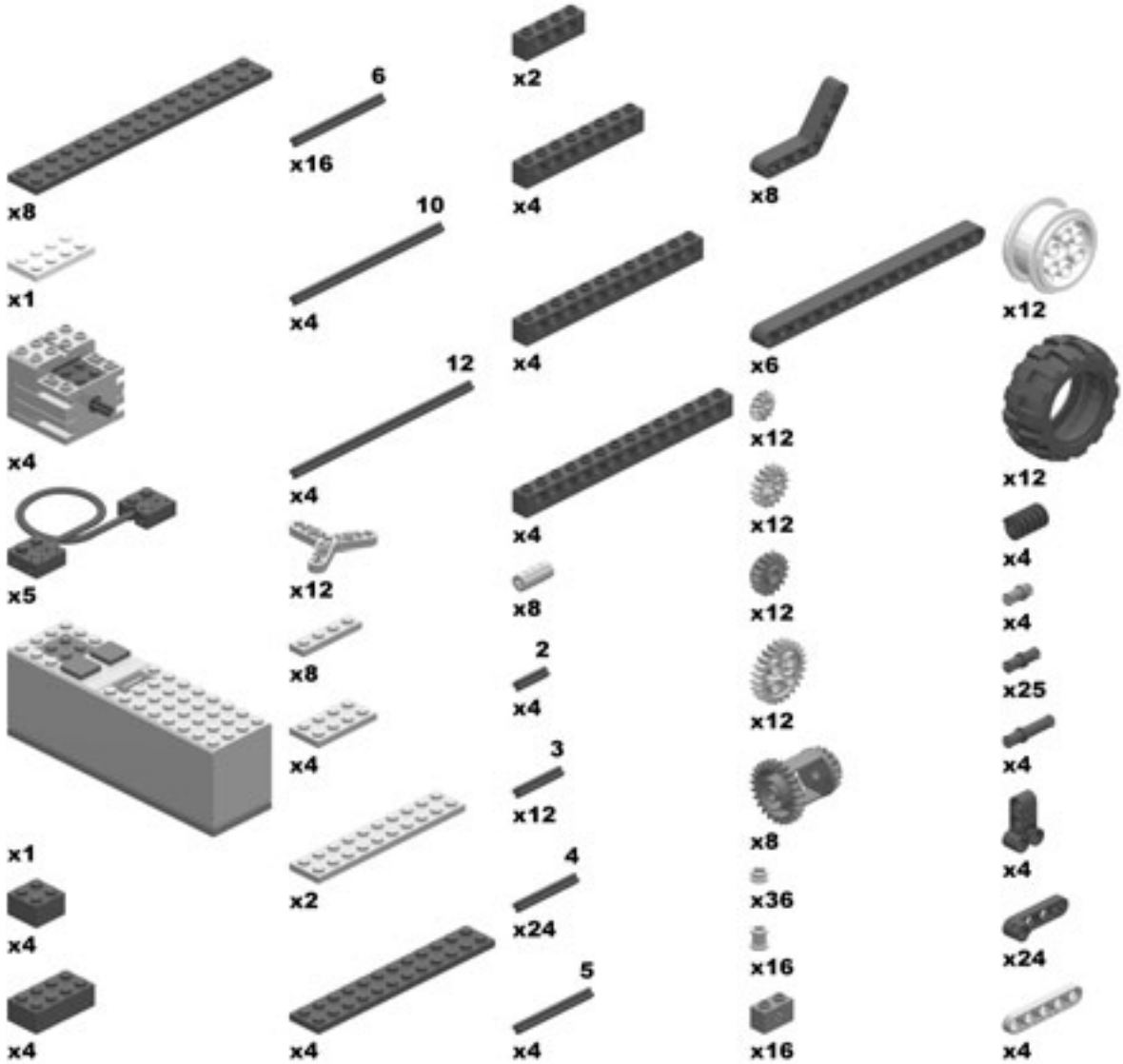


If steering is deemed necessary, one could re-engineer the tri-star wheels by adding tri-plate support on both sides of the wheels instead of just a single drive side. The chassis would have to be modified to accommodate the wider wheel sets, and it would now make sense to use an RCX for a handheld power source. The control inputs could be either from an array of touch sensors or possibly a pair of rotation sensors configured as left and right joystick style inputs. A version of the Stair-Climber built with the Alternate Wheel Set sub-assembly would look like Figure 1.6.

Figure 1.6 A Version of Stair-Climber Built with the Alternate Wheel-Set Sub-assembly



If you opt to build the Stair-Climber using the wider alternate wheel sets, you will need the following parts.

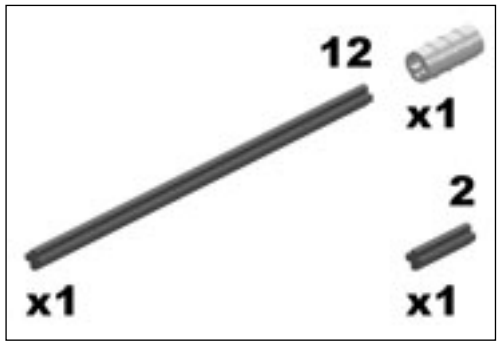


This version is slightly stronger owing to the triple sandwich of the tri-beams. If you opt to build the Alternate Wheel Set sub-assembly, rather than the standard Wheel Set sub-assembly built earlier in the chapter, you will need to build **four** of these.

NOTE

Fear not, if you opt to build the Alternate Wheel Set sub-assembly, there are no changes to any of the other sub-assemblies in the Stair-Climber. The Frame sub-assemblies, and the Final sub-assemblies are compatible with either version of the Wheel Set sub-assemblies.

Alternate Wheel Set Step 0

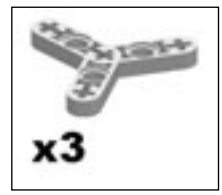


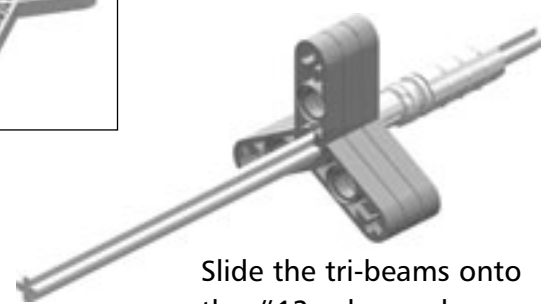
Connect the #12 axle and the #2 axle together using an axle joiner as shown.

Alternate Wheel Set Step 1

Slide bushings onto #12 axle as shown.

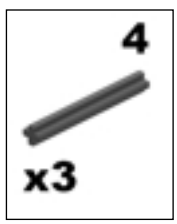
Alternate Wheel Set Step 2

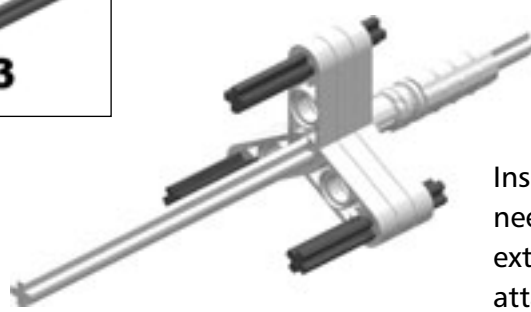




Slide the tri-beams onto the #12 axles as shown.

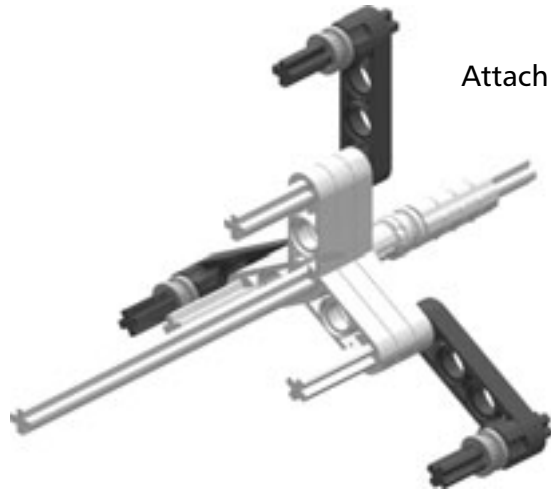
Alternate Wheel Set Step 3





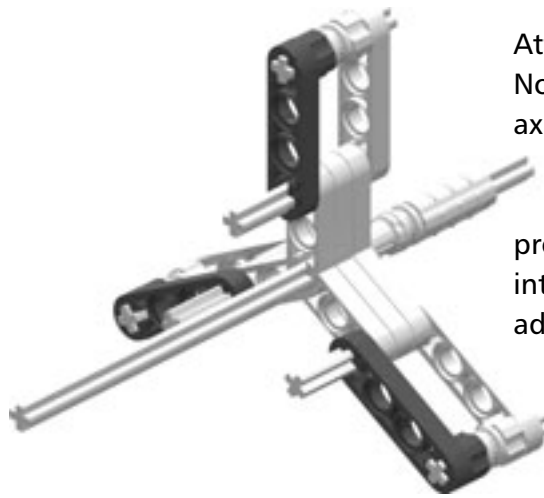
Insert axles as shown. There will need to be just enough axle extending behind the tri-beams to attach a half-thickness liftarm.

Alternate Wheel Set Step 4



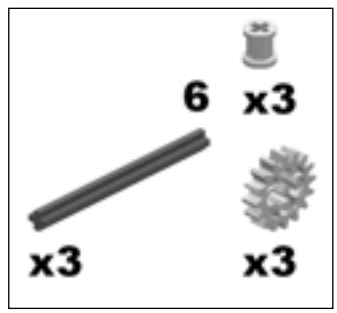
Attach parts as shown.

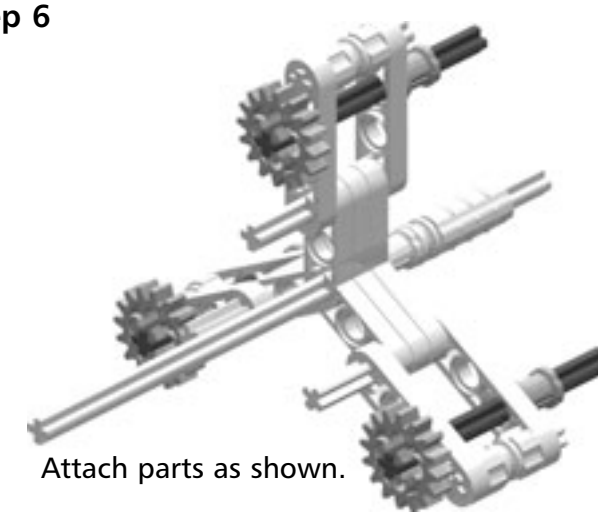
Alternate Wheel Set Step 5



Attach the three liftarms. Note that the three outer axles extend out slightly from the rear of this assembly. This prevents them from interfering with the gears added in the next step.

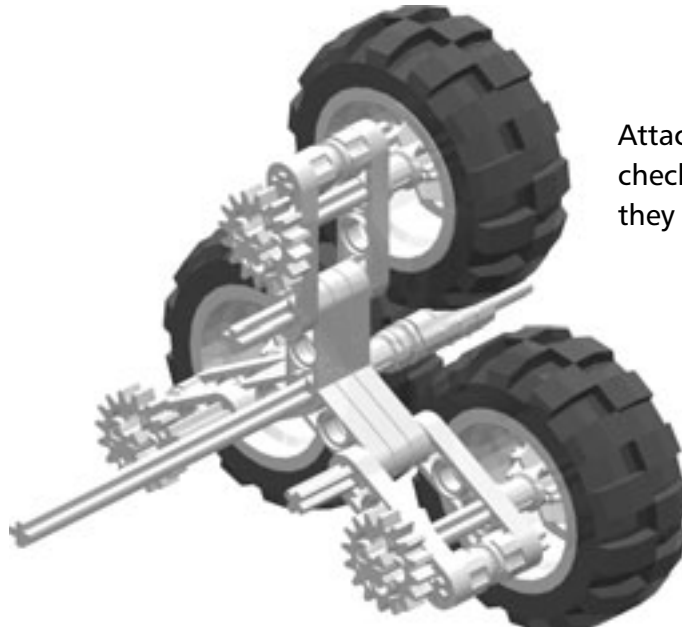
Alternate Wheel Set Step 6





Attach parts as shown.

Alternate Wheel Set Step 7

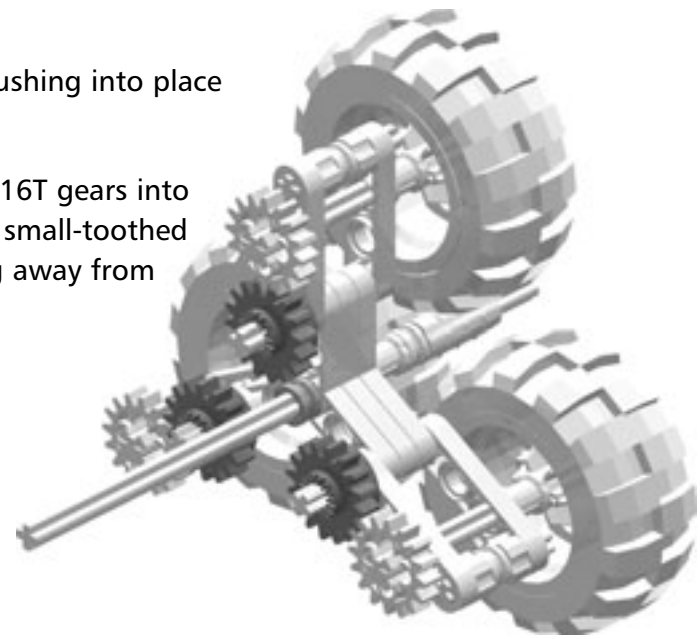


Attach the wheels and check to make sure that they rotate easily.

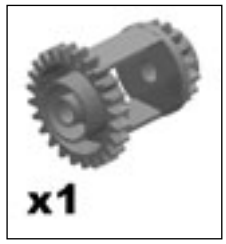
Alternate Wheel Set Step 8

Slide a half-bushing into place on main axle.

Then slip the 16T gears into position with small-toothed section facing away from the wheels.

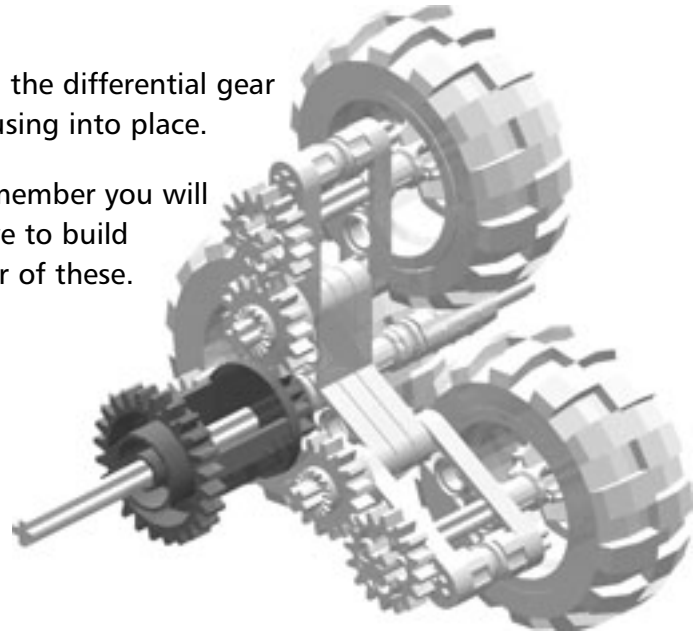


Alternate Wheel Set Step 9



Slip the differential gear housing into place.

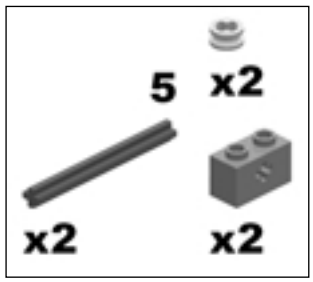
Remember you will have to build four of these.



The Mid-Frame

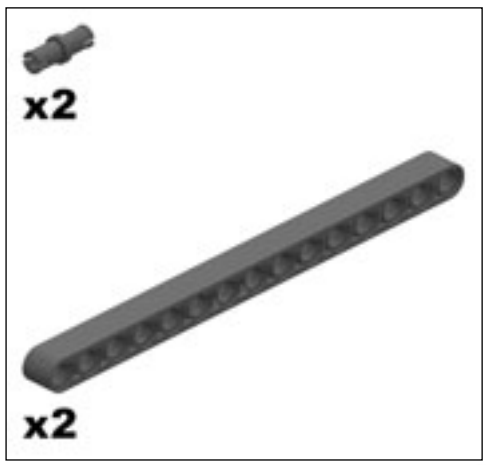
The Mid-Frame sub-assembly is one of the components of the chassis that runs from the front to the rear of the model just inside the tri-star wheels. You will need to build **two** of these.

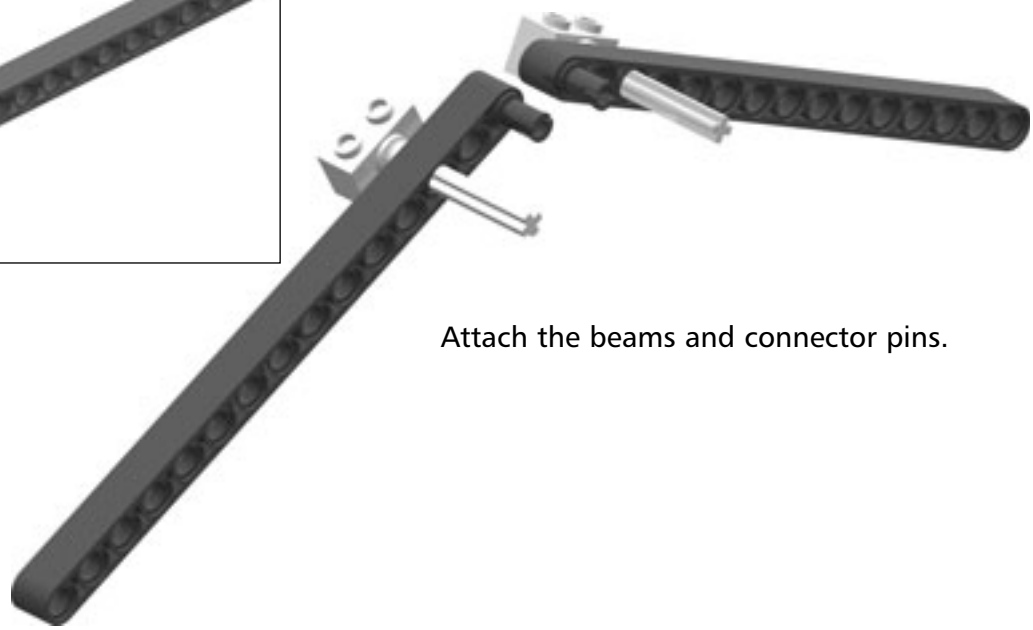
Mid-Frame Step 0



Assemble parts as shown.

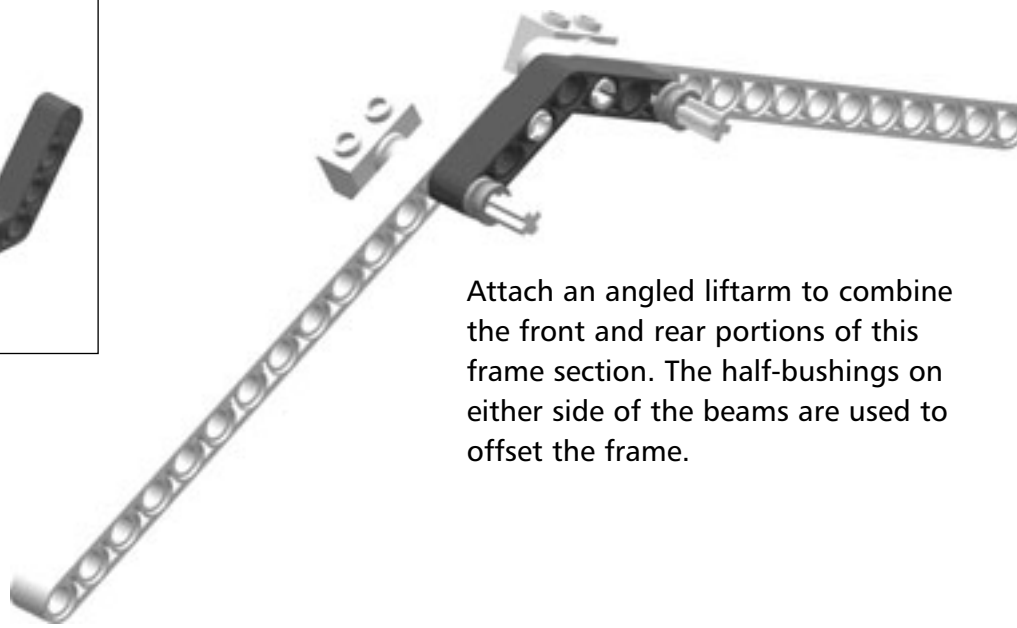
Mid-Frame Step 1





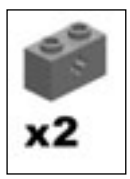
Attach the beams and connector pins.

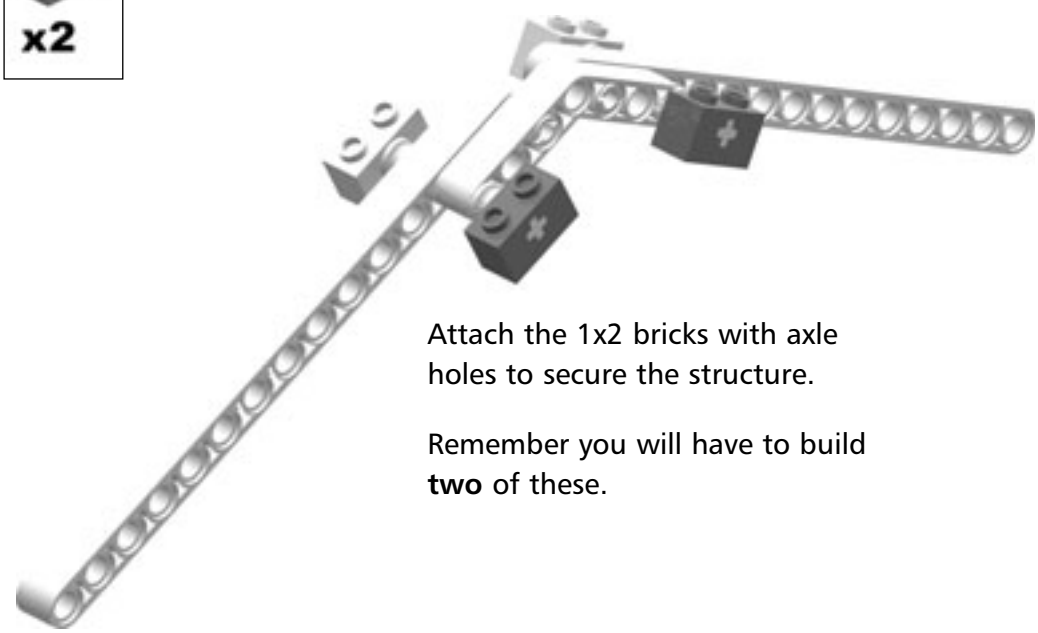
Mid-Frame Step 2



Attach an angled liftarm to combine the front and rear portions of this frame section. The half-bushings on either side of the beams are used to offset the frame.

Mid-Frame Step 3





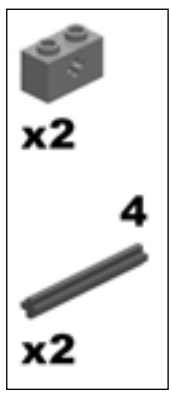
Attach the 1x2 bricks with axle holes to secure the structure.

Remember you will have to build **two** of these.

The Outer-Frame

The Outer-Frame sub-assembly is also a component of the chassis, similar to the Mid-Frame sub-assembly. The difference between the Mid-Frame sub-assembly and the Outer-Frame sub-assembly, is that the Outer-Frame sub-assembly is positioned on the outside of the tri-star wheels. You will also need to build **two** of these.

Outer-Frame Step 0

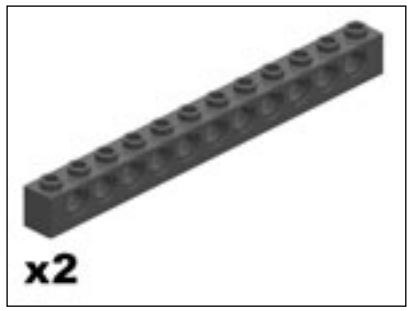


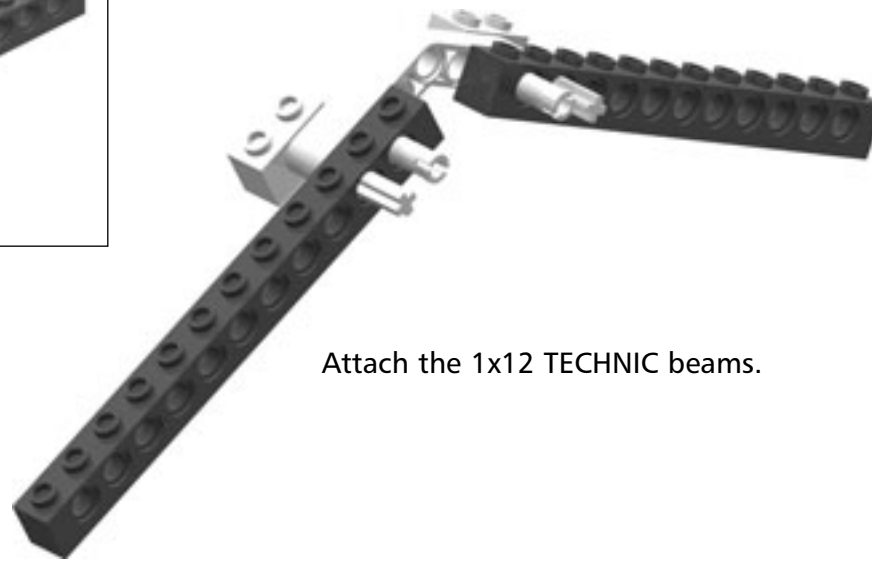
Insert the #4 axles into 1x2 bricks with axle holes.

Outer-Frame Step 1

Attach an angled liftarm to combine the front and rear portions of the -frame section. Insert the full-length pins with stop bushings as shown.

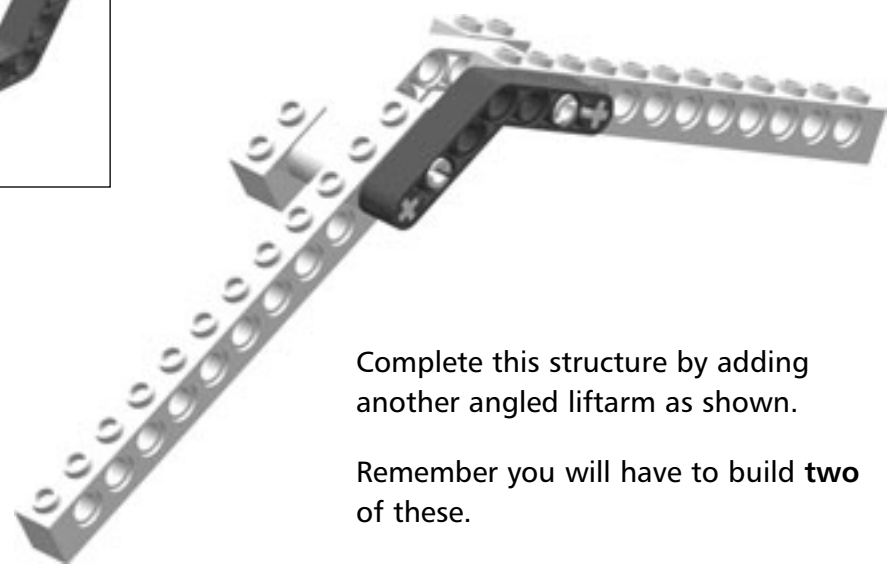
Outer-Frame Step 2





Attach the 1x12 TECHNIC beams.

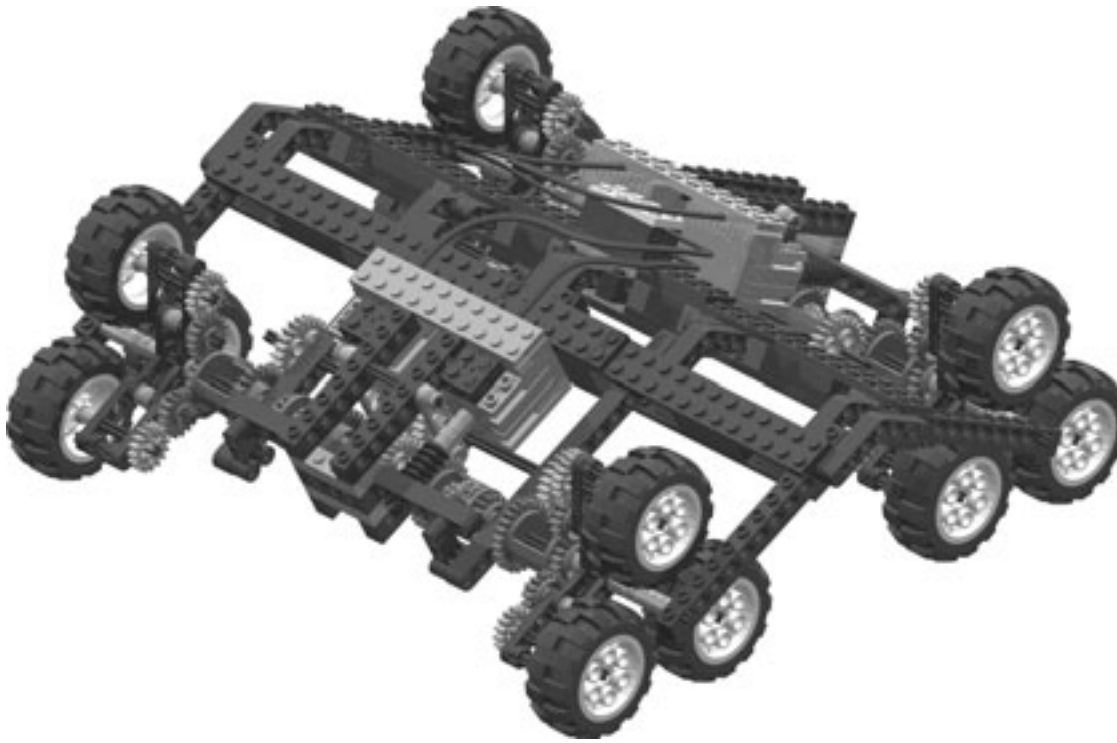
Outer-Frame Step 3



Complete this structure by adding another angled liftarm as shown.

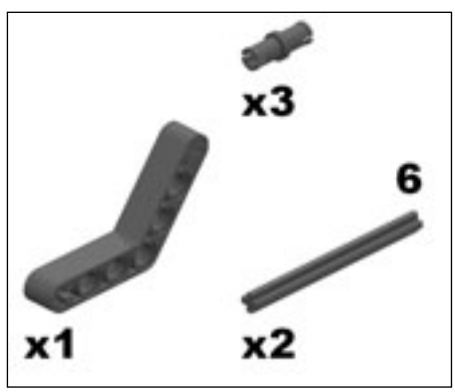
Remember you will have to build **two** of these.

Putting It All Together



Here is where we will complete the Stair-Climber. We will first build the central part of the model, and then attach the previous sub-assemblies in order. The directions show the original style tri-star wheel, but the assembly process is identical regardless of whether you opted to build the Wheel Set sub-assembly or the Alternate Wheel Set sub-assembly. When assembling the model, take care to be sure the parts are aligned exactly as shown. Because the model is symmetrical from side to side and front to rear, it should be easy to see if any parts are missing or misplaced.

Final Step 0

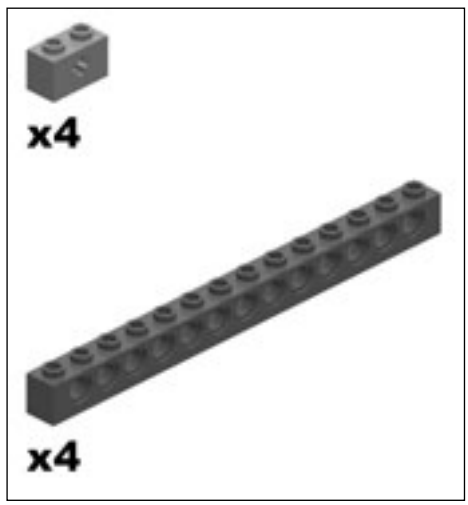


Insert the connector pins and axles into the angled liftarm. There should be equal lengths of axle extending out from either side of the liftarm

Final Step 1

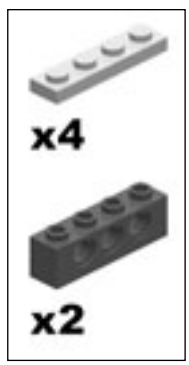
Attach a second angled liftarm and insert the connector pins.

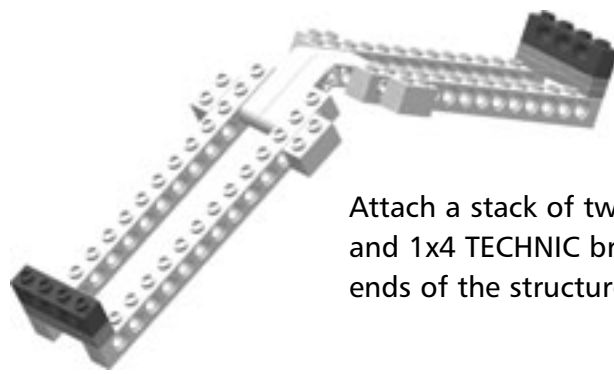
Final Step 2



Attach the 1x2 bricks with axle holes and the 1x14 TECHNIC bricks as shown.

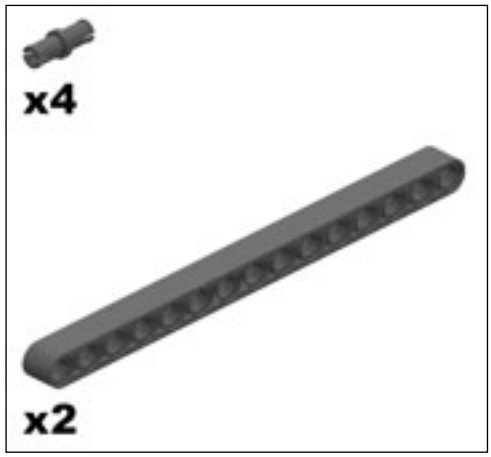
Final Step 3

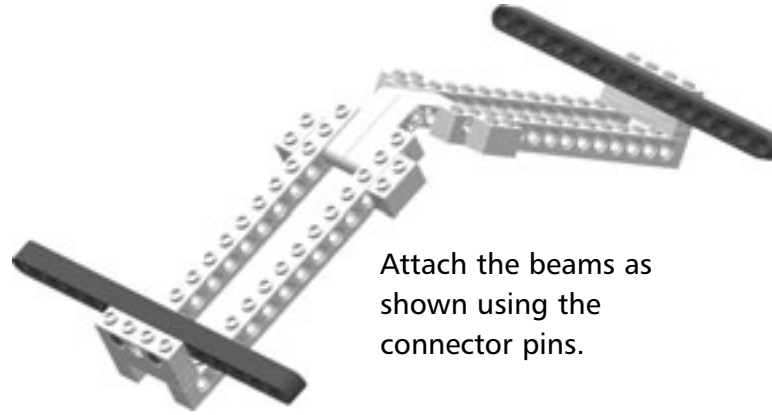




Attach a stack of two 1x4 plates and 1x4 TECHNIC brick on both ends of the structure.

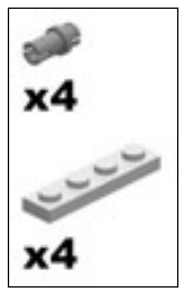
Final Step 4



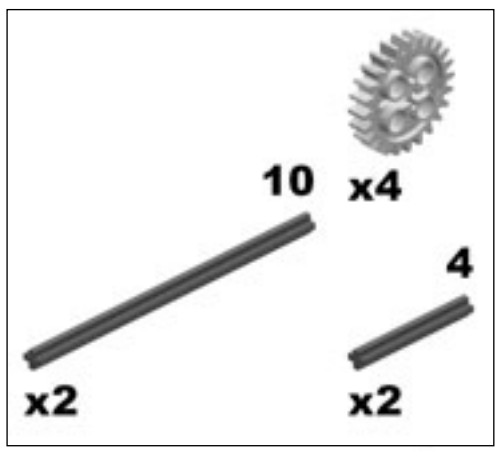


Attach the beams as shown using the connector pins.

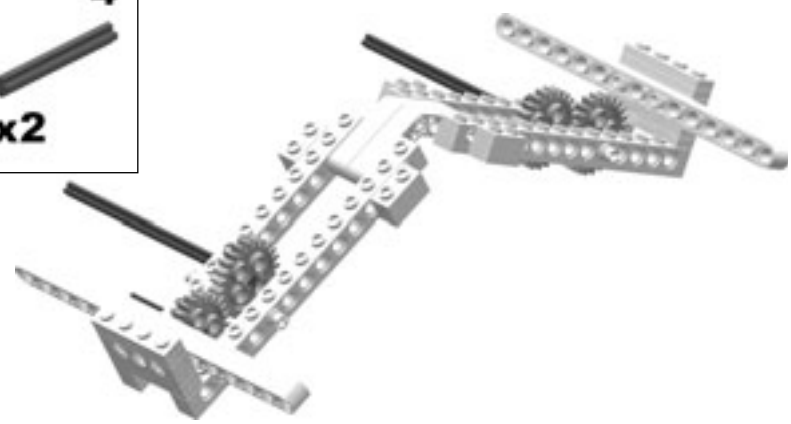
Final Step 5



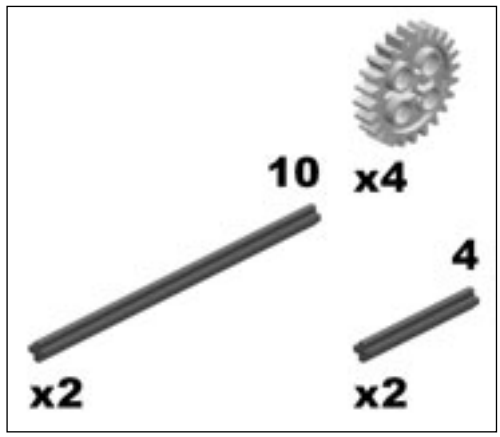
Final Step 6



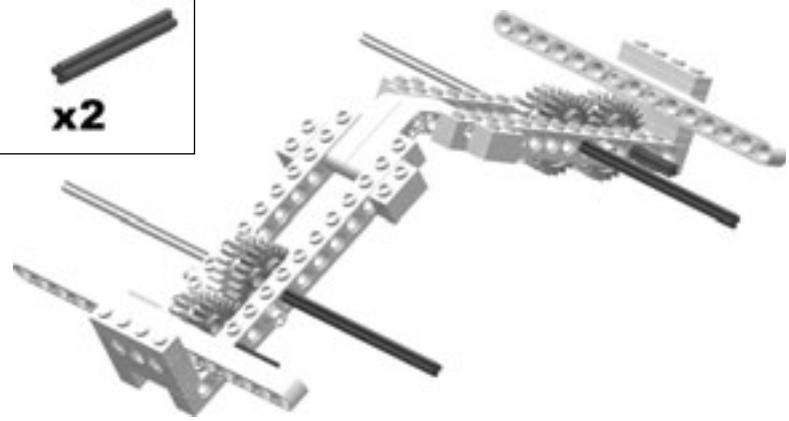
Slide the #4 axles through the four 24T gears and attach these to the beams as shown. Then attach the #10 axles.



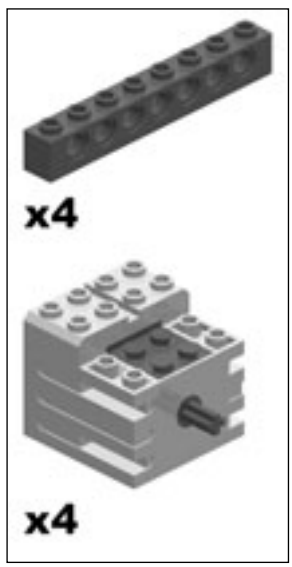
Final Step 7

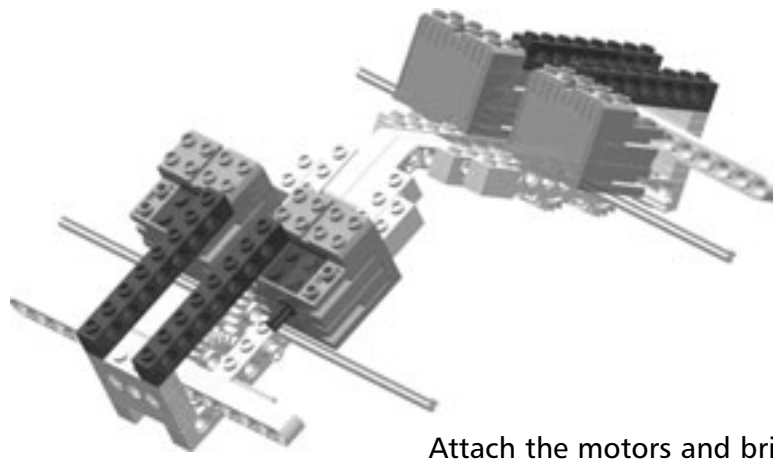


Repeat the installation process performed in **Final Step 6**. The gears on each side should rotate freely without interference from its adjacent side.



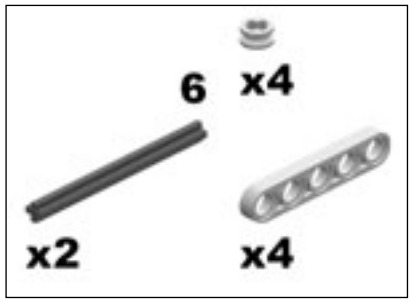
Final Step 8

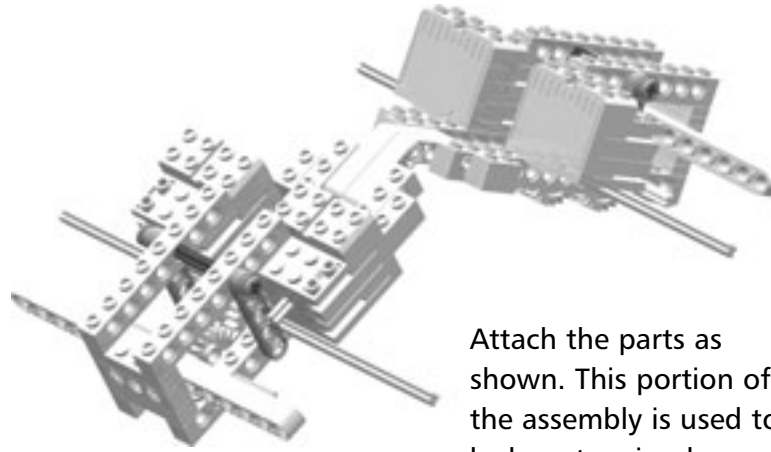




Attach the motors and bricks to each side of the chassis as shown.

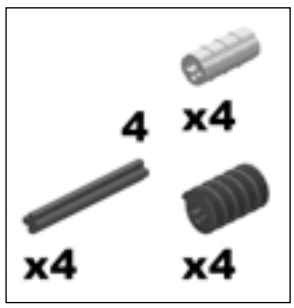
Final Step 9

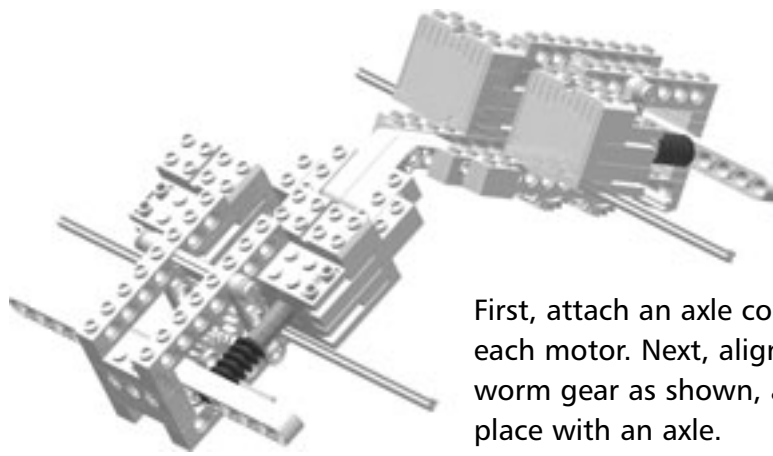




Attach the parts as shown. This portion of the assembly is used to lock motors in place.

Final Step 10



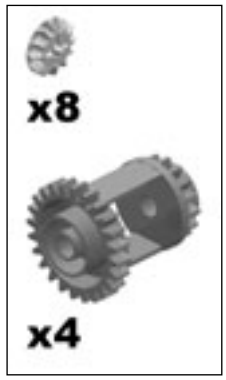


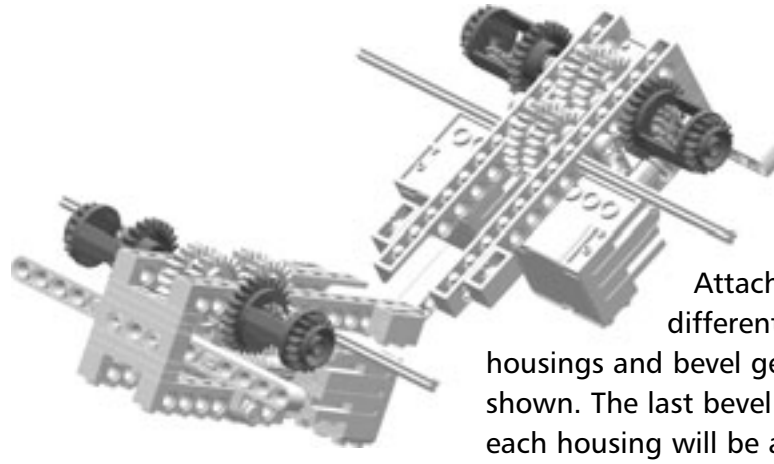
First, attach an axle connector to each motor. Next, align each worm gear as shown, and pin in place with an axle.

Final Step 11

Rotate the model so that you are looking at the bottom side, and place four half-bushings as shown.

Final Step 12

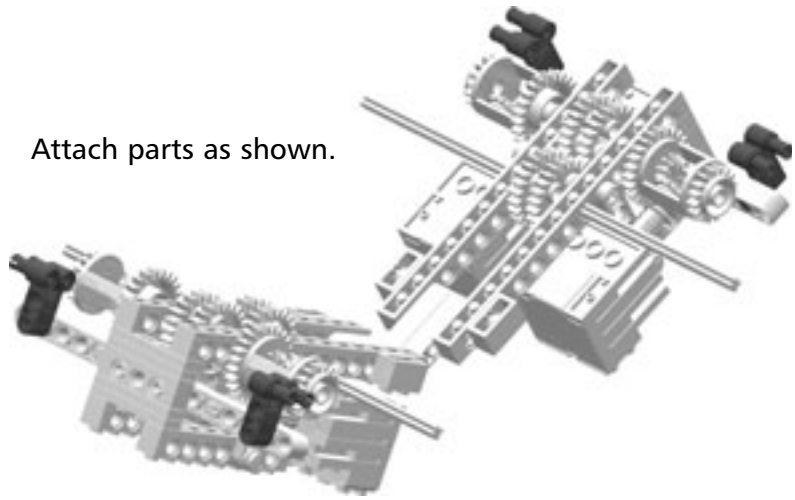




Attach the differential gear housings and bevel gears as shown. The last bevel gear for each housing will be added with the tri-star wheel assemblies in **Final Steps 17** and **20**.

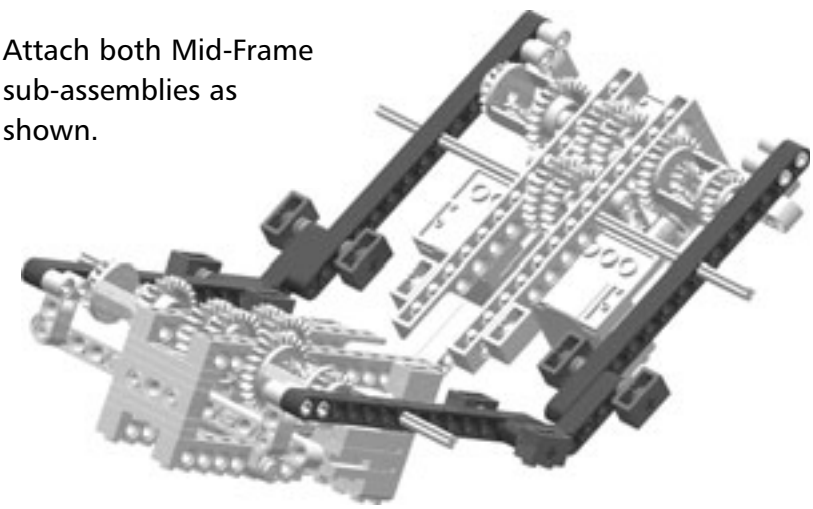
Final Step 13

Attach parts as shown.

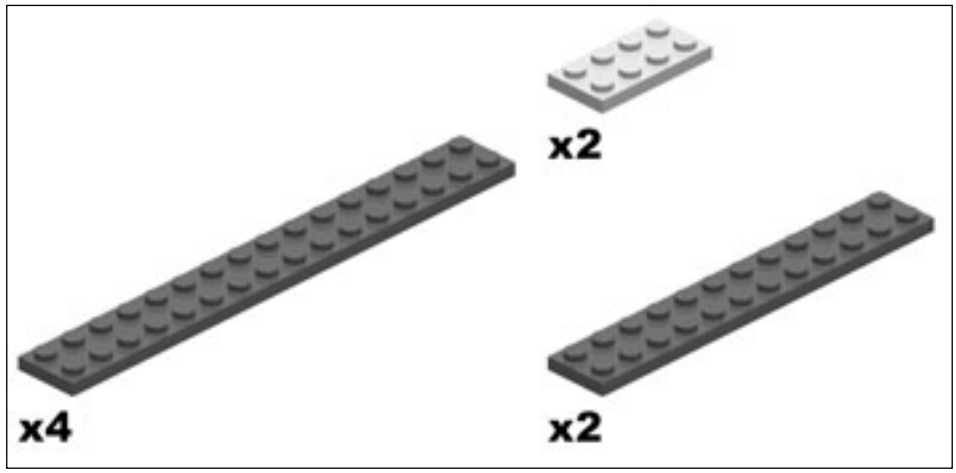


Final Step 14

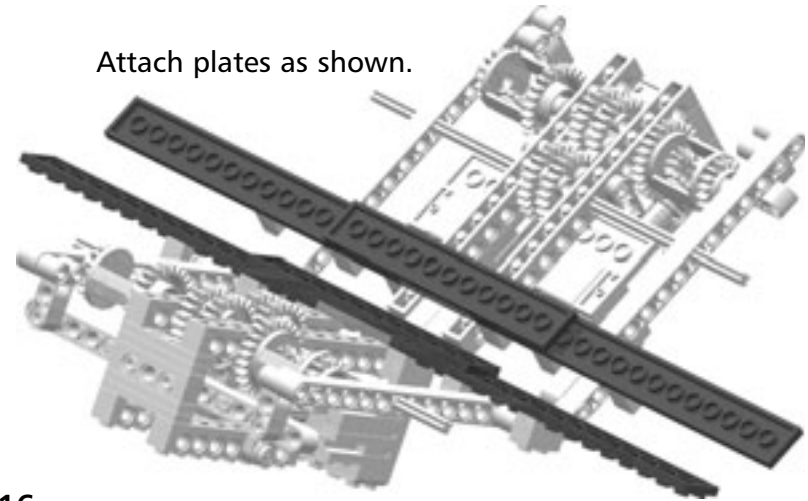
Attach both Mid-Frame sub-assemblies as shown.



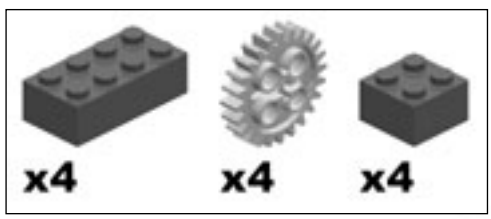
Final Step 15



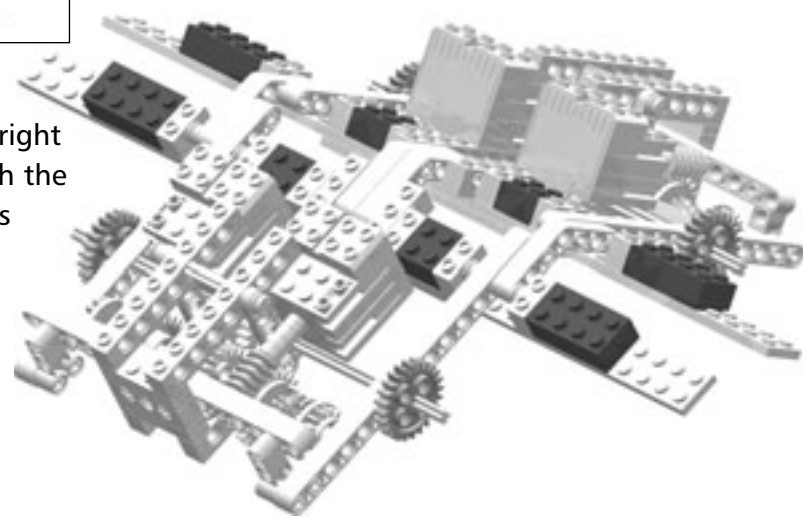
Attach plates as shown.



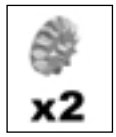
Final Step 16



Rotate the model right side up, and attach the bricks and gears as shown.

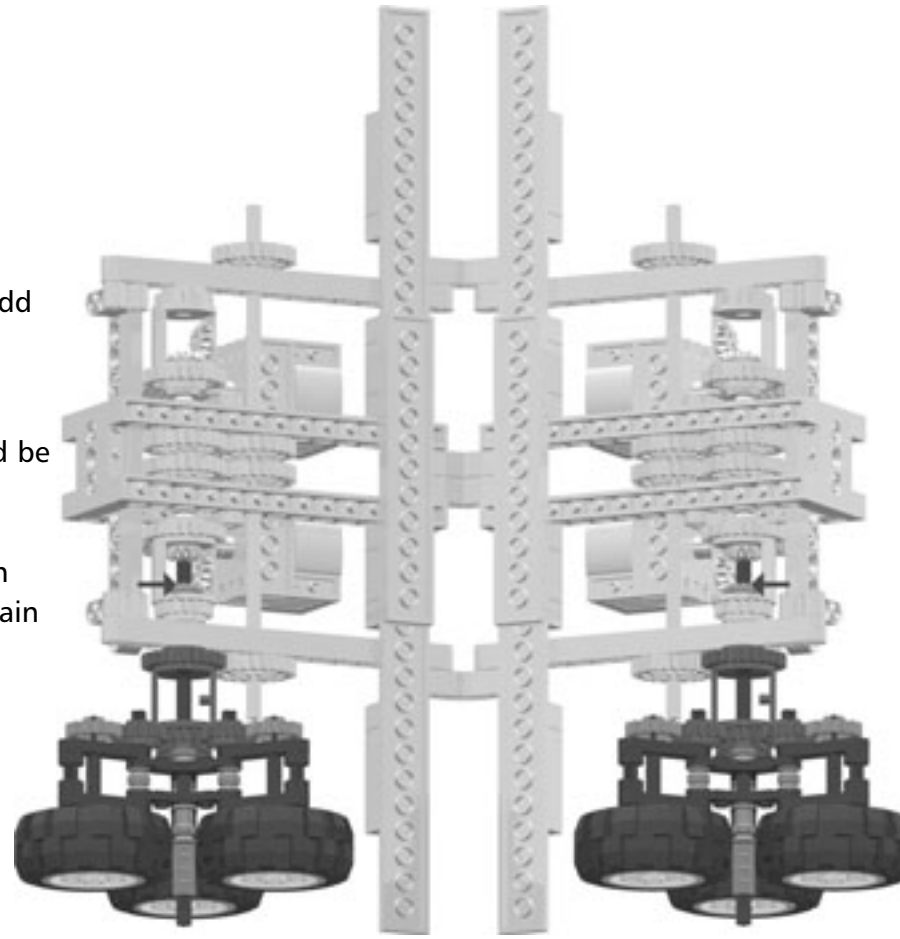


Final Step 17

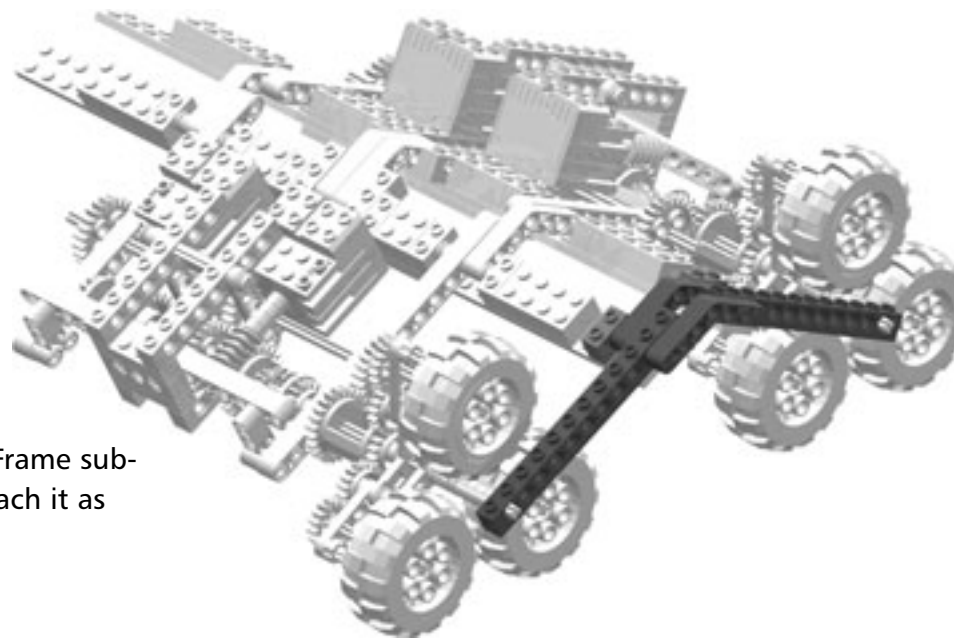


In this step, you will add two of the Wheel Set sub-assemblies.

The bevel gears should be placed within the corresponding drive differential and held in place by the tri-star main axle.

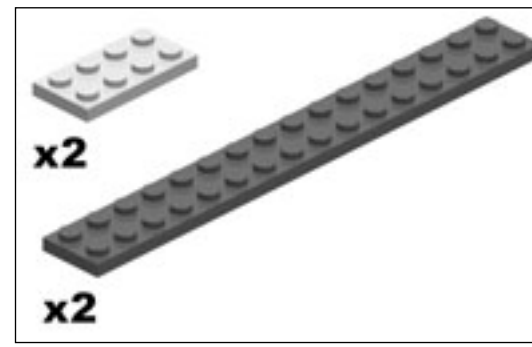


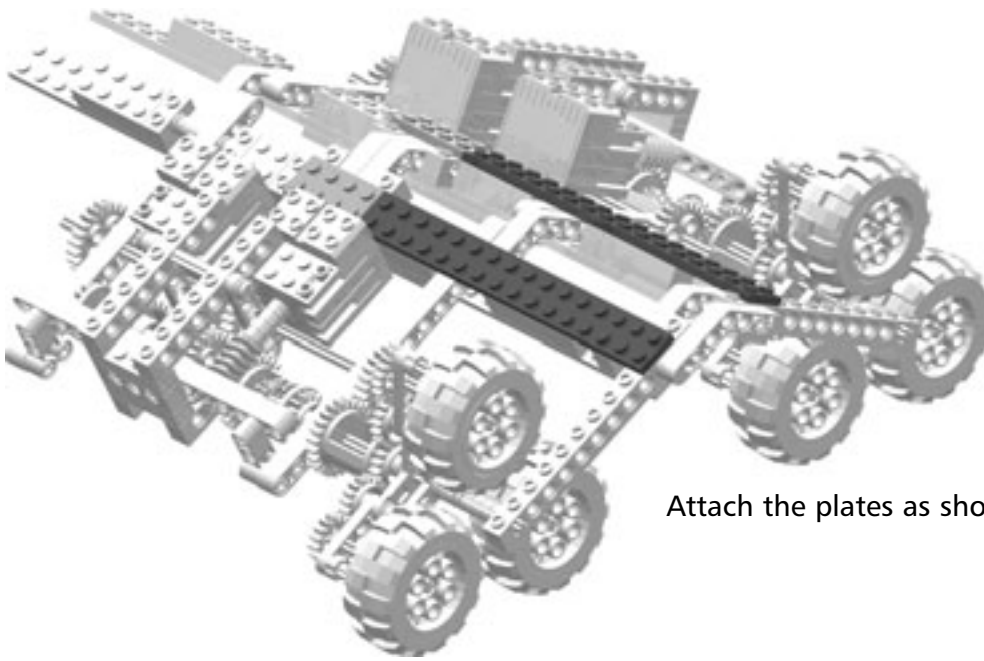
Final Step 18



Locate an Outer-Frame sub-assembly and attach it as shown.

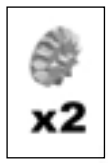
Final Step 19

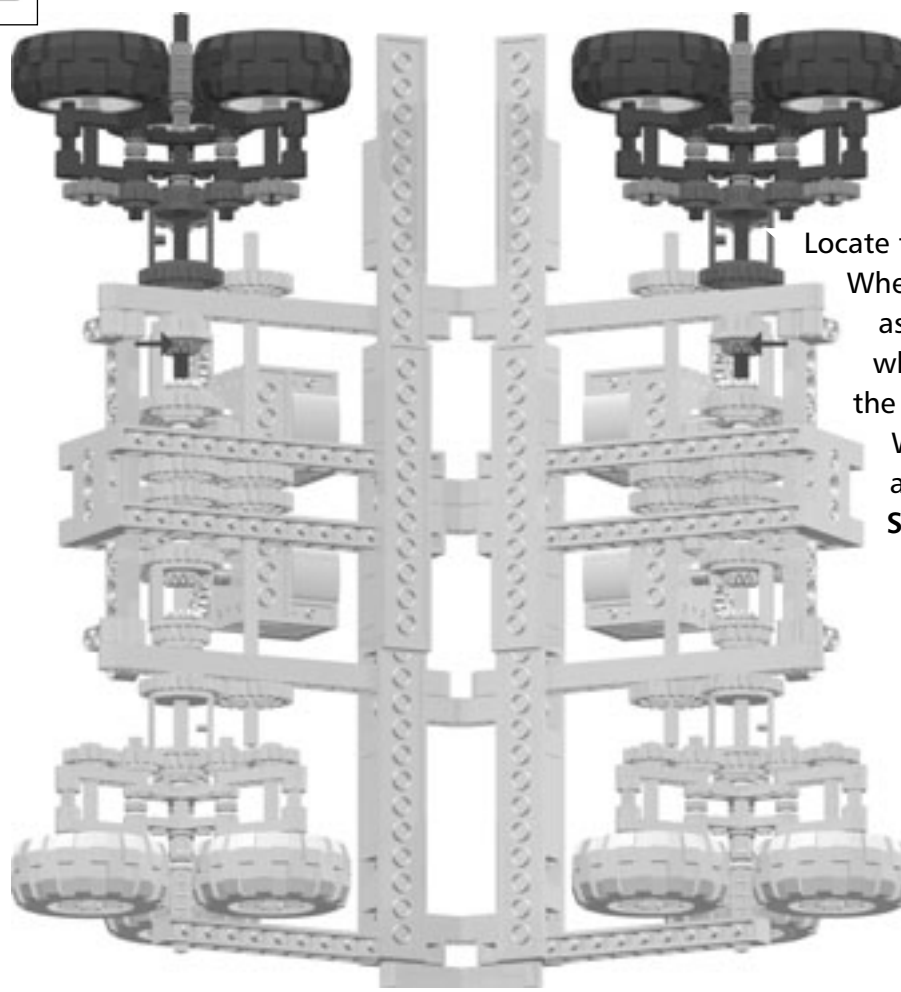




Attach the plates as shown.

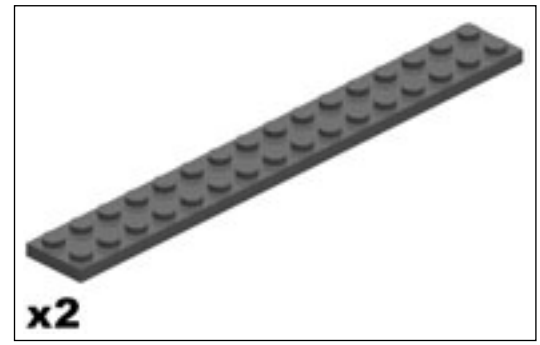
Final Step 20

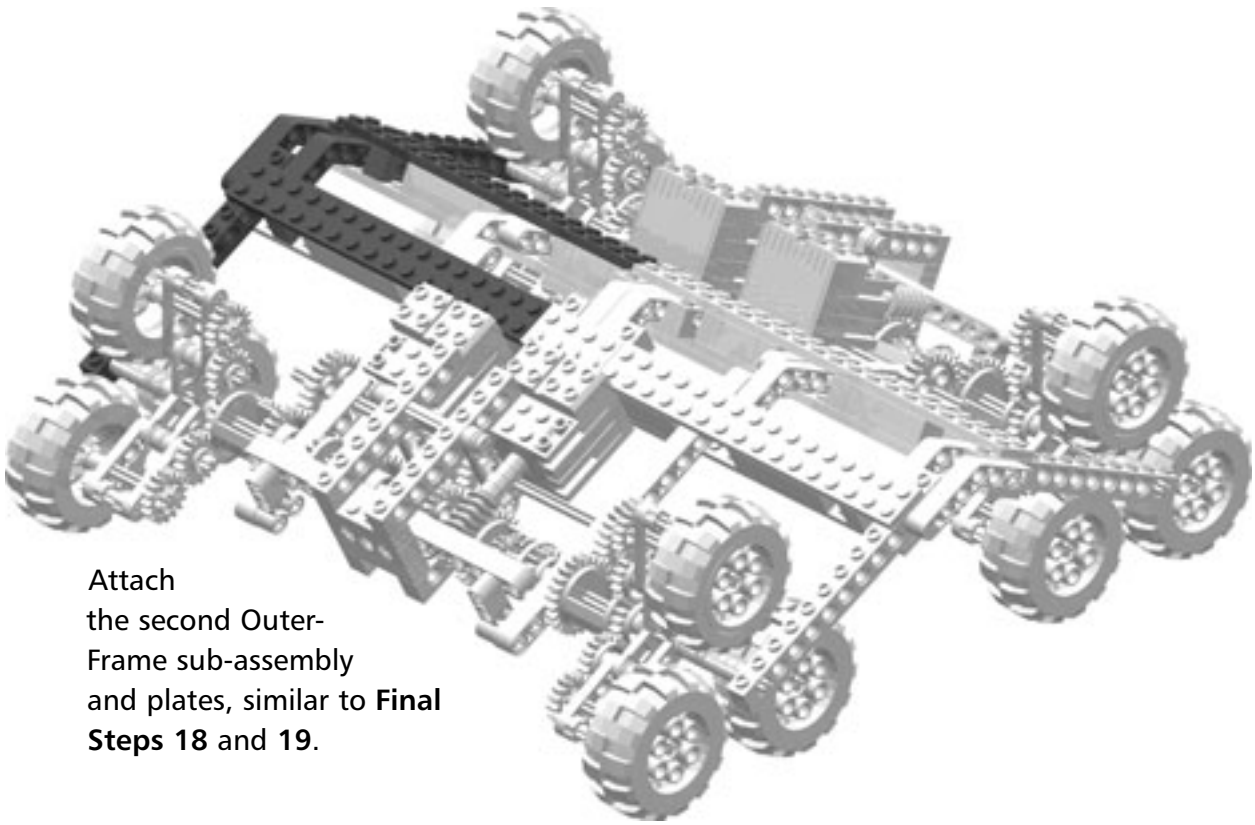




Locate the remaining two Wheel Set sub-assemblies. These wheels are attached in the same manner as the Wheel Set sub-assemblies in **Final Step 17**.

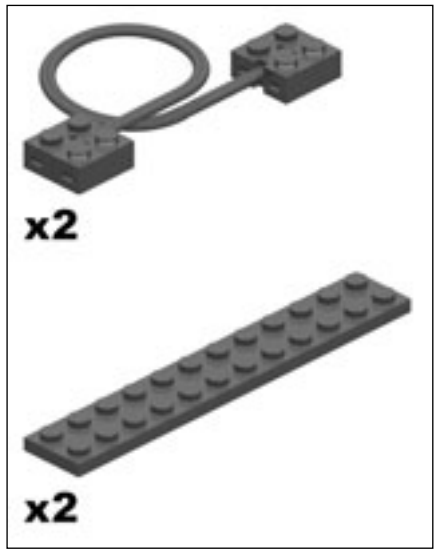
Final Step 21



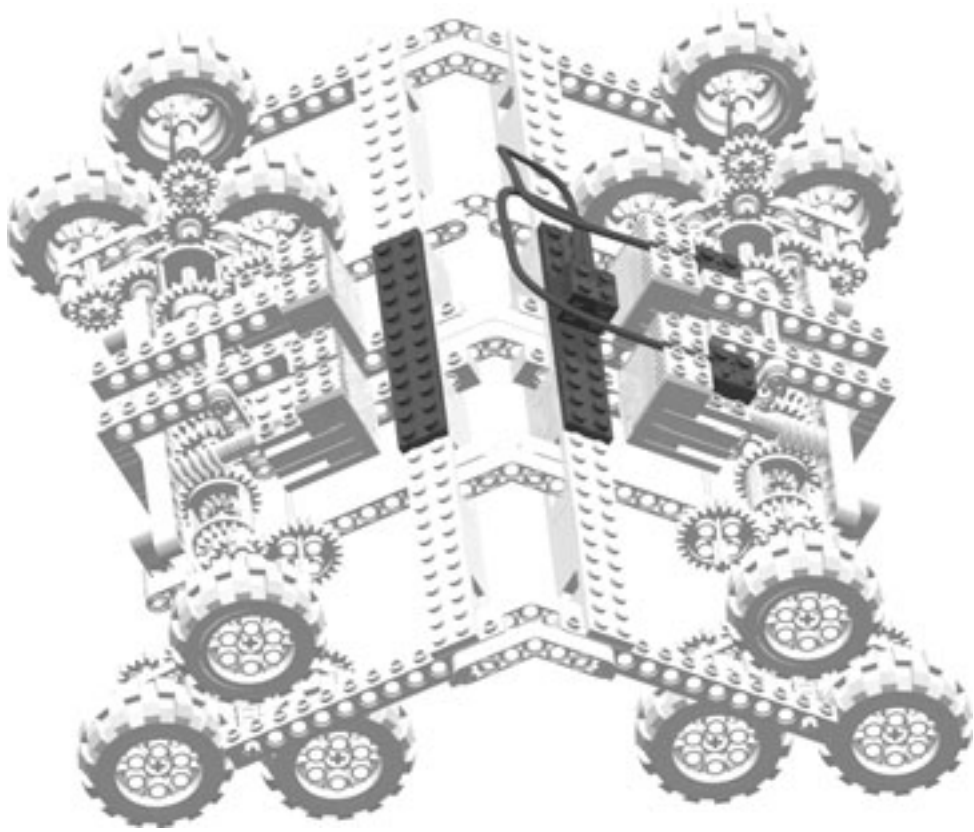


Attach the second Outer-Frame sub-assembly and plates, similar to **Final Steps 18** and **19**.

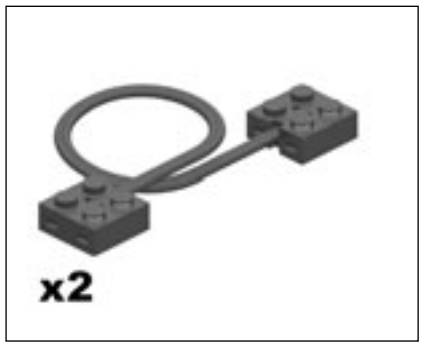
Final Step 22

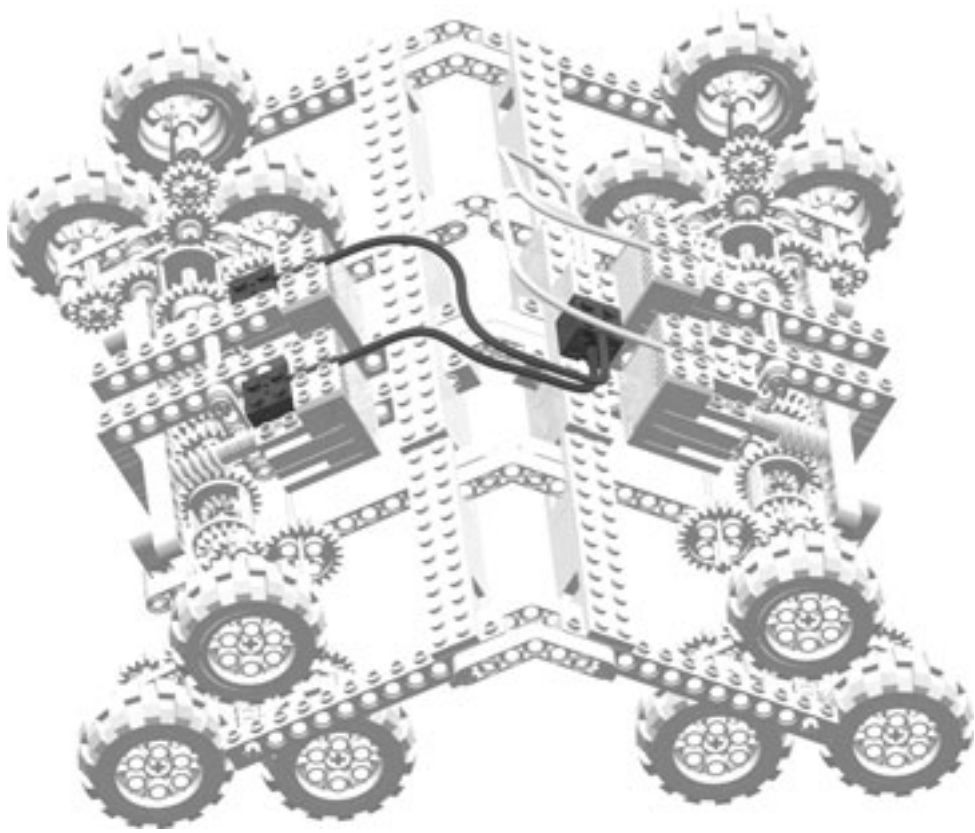


Rotate the model as shown. Attach plates and electric wires as shown.



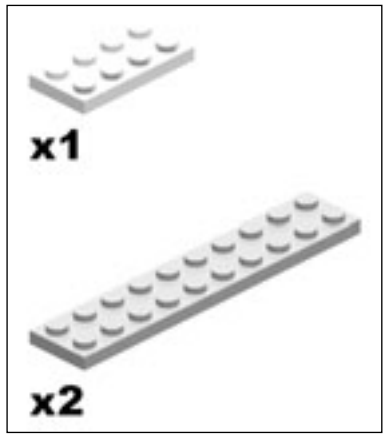
Final Step 23

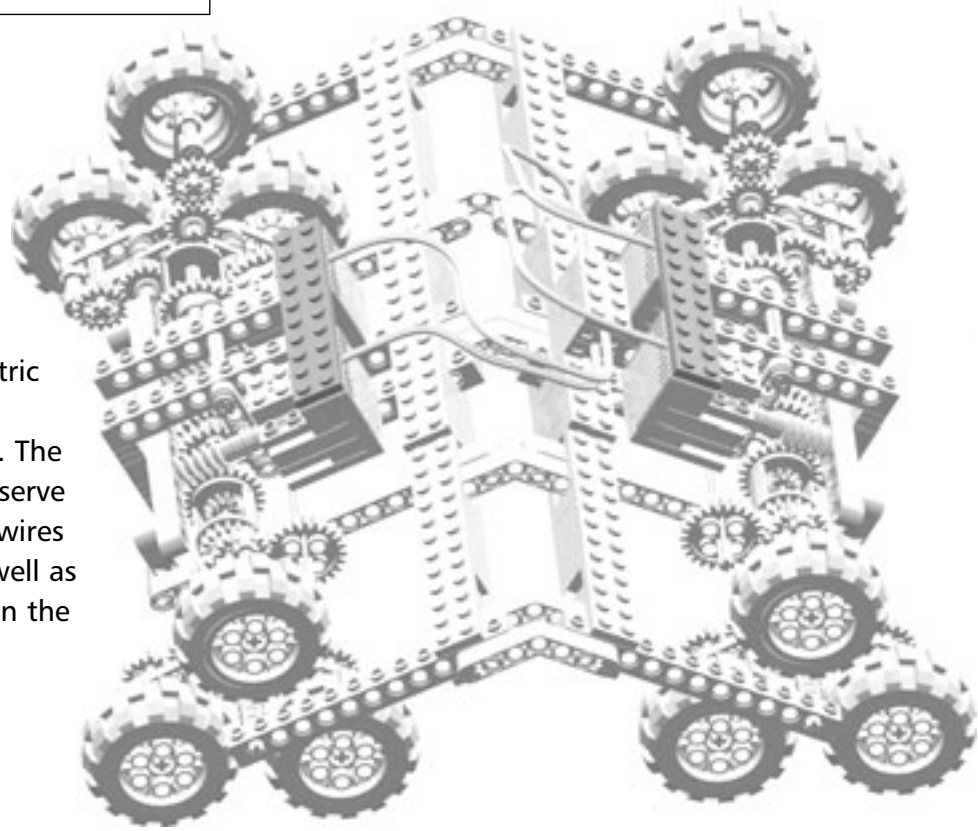




Attach two more electric wires.

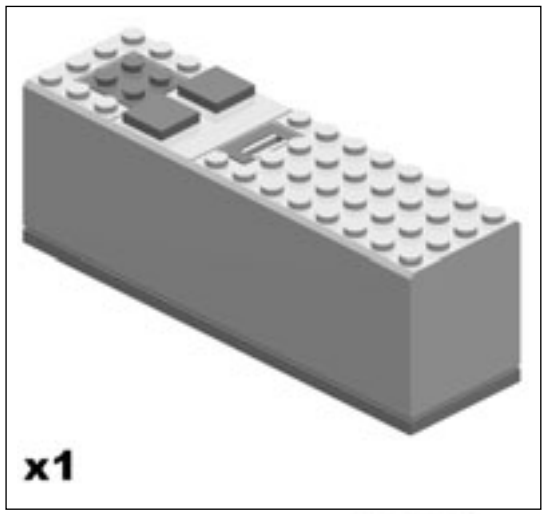
Final Step 24



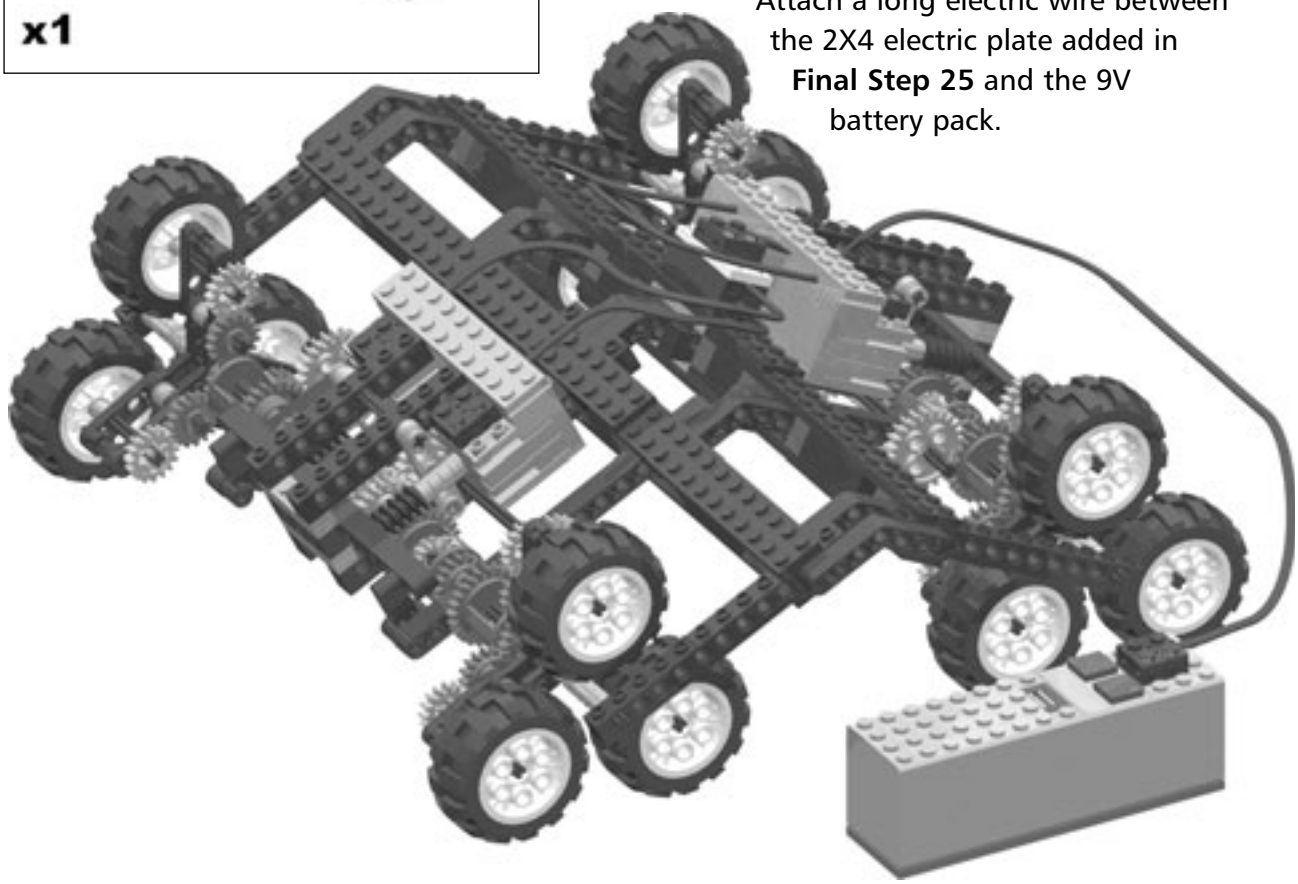


Attach the 2x4 electric plate to the motor wires. The 2x10 plates serve to hold the wires in place as well as to strengthen the assembly.

Final Step 25



Attach a long electric wire between the 2X4 electric plate added in **Final Step 25** and the 9V battery pack.



Operating the Stair-Climber

Operating the Stair-Climber is relatively straightforward. Pushing the buttons on the battery pack should drive all four wheels forward or backward in unison.
I suggest that you experiment and watch how the model drives, steps, and climbs over various obstacles. A pile of LEGO bricks is a perfect, re-configurable obstacle course. Try making stairs of various inclines using whatever is convenient. LEGO bricks work well for this, but books and scrap lumber are good alternatives as well.

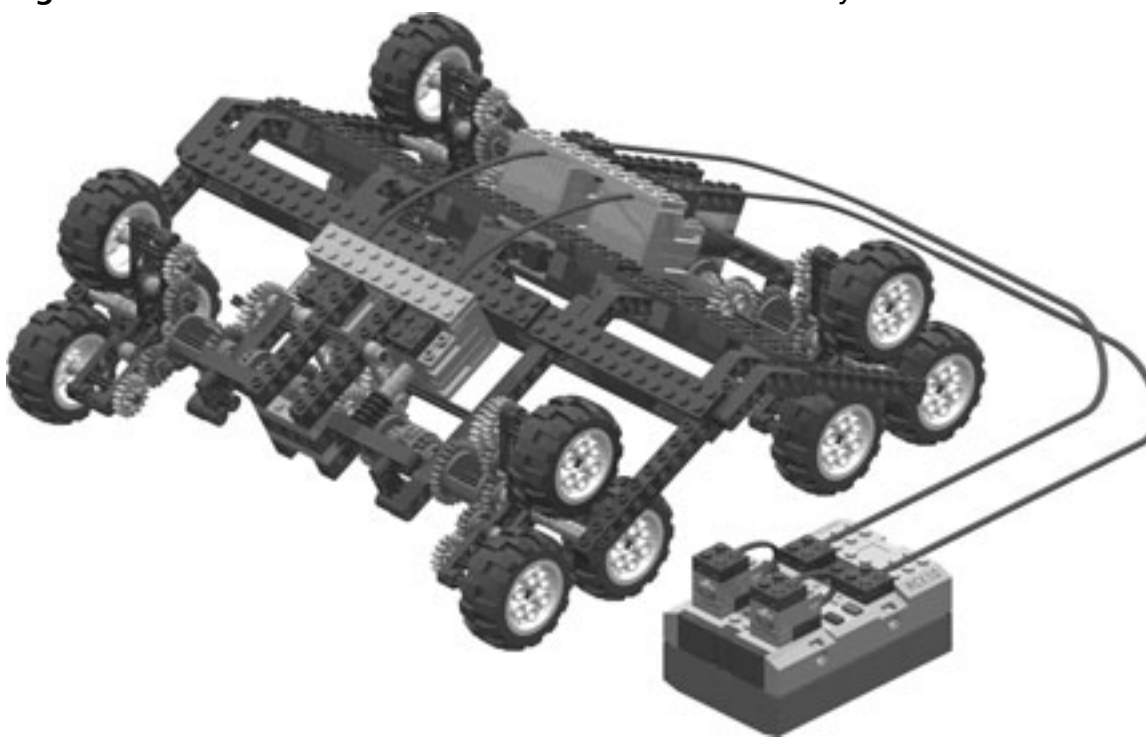
NOTE

The CD-ROM that accompanies this book contains video of the Stair-Climber in action traversing various obstacles.

Using an RCX instead of a Battery Pack

Should you opt to use an RCX instead of the LEGO battery pack, your model might take the form of the Stair-Climber shown in Figure 1.7.

Figure 1.7 Stair-Climber Built with an RCX instead of a Battery Box



It is a fairly simple process to modify your Stair-Climber so it is controllable by an RCX. First, you should attach a touch sensor to Input Port A, and second touch sensor to Input Port C, as shown in Figure 1.7. You will then have to modify the Stair-Climber motor wiring as shown. This change connects the left-side motors to Output Port 1 and the right-side motors to Output Port 3.

NOTE

The wires used for connecting the motors together need to be slightly longer than the ones supplied in the RIS 2.0 set. Use whatever combination of wires you have available to make these connections.

Then, by writing a simple program in the language of your choice, assign the following values:

- Touch Sensor A causes both Output Ports 1 and 3 to be set to 'reverse' and 'on' while pressed.
- Touch Sensor C causes both Output Ports 1 and 3 to be set to 'forward' and 'on' while pressed.
- Both Output Ports 1 and 3 are turned 'off' when neither Touch Sensor is pressed.

A sample program built with the RIS 2.0 language and programming interface would look something like the program seen in Figure 1.8.

Figure 1.8 A Sample Stair-Climber Program Built with RIS 2.0

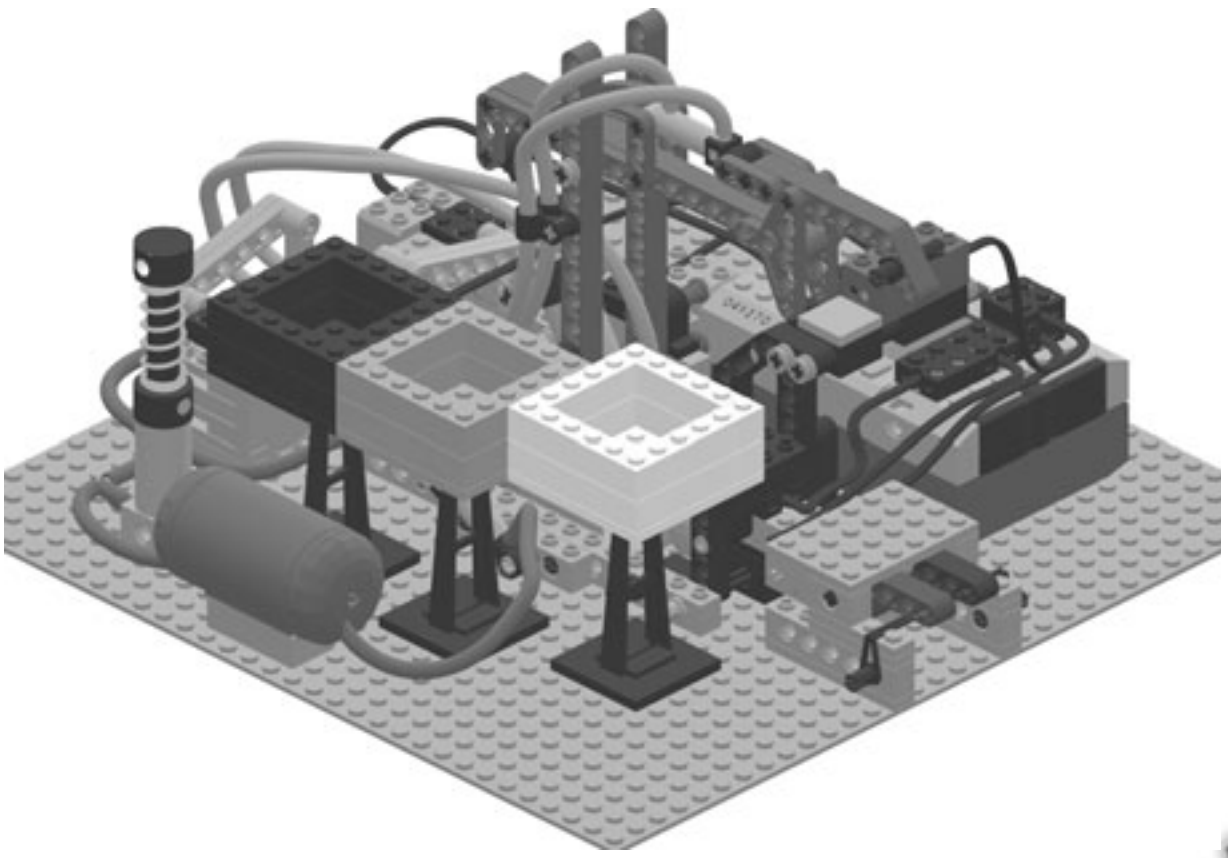
Summary

In this chapter, we have explored the use of a special type of star-wheel configuration designed specifically to overcome severe terrain obstacles including stairs. The model we built demonstrated some of the capabilities of this type of design. Others have used variations of the star-wheel for all terrain vehicles (ATVs) and wheelchairs. Future uses may include autonomous robots that have little difficulty navigating the same environments as we do. If ATVs are of a specific interest to you, I recommend that you jump ahead in the book to Masterpeice 6, and check out the Shape-Shifting Camera Tank built by Miguel Agulló.

Masterpiece 2

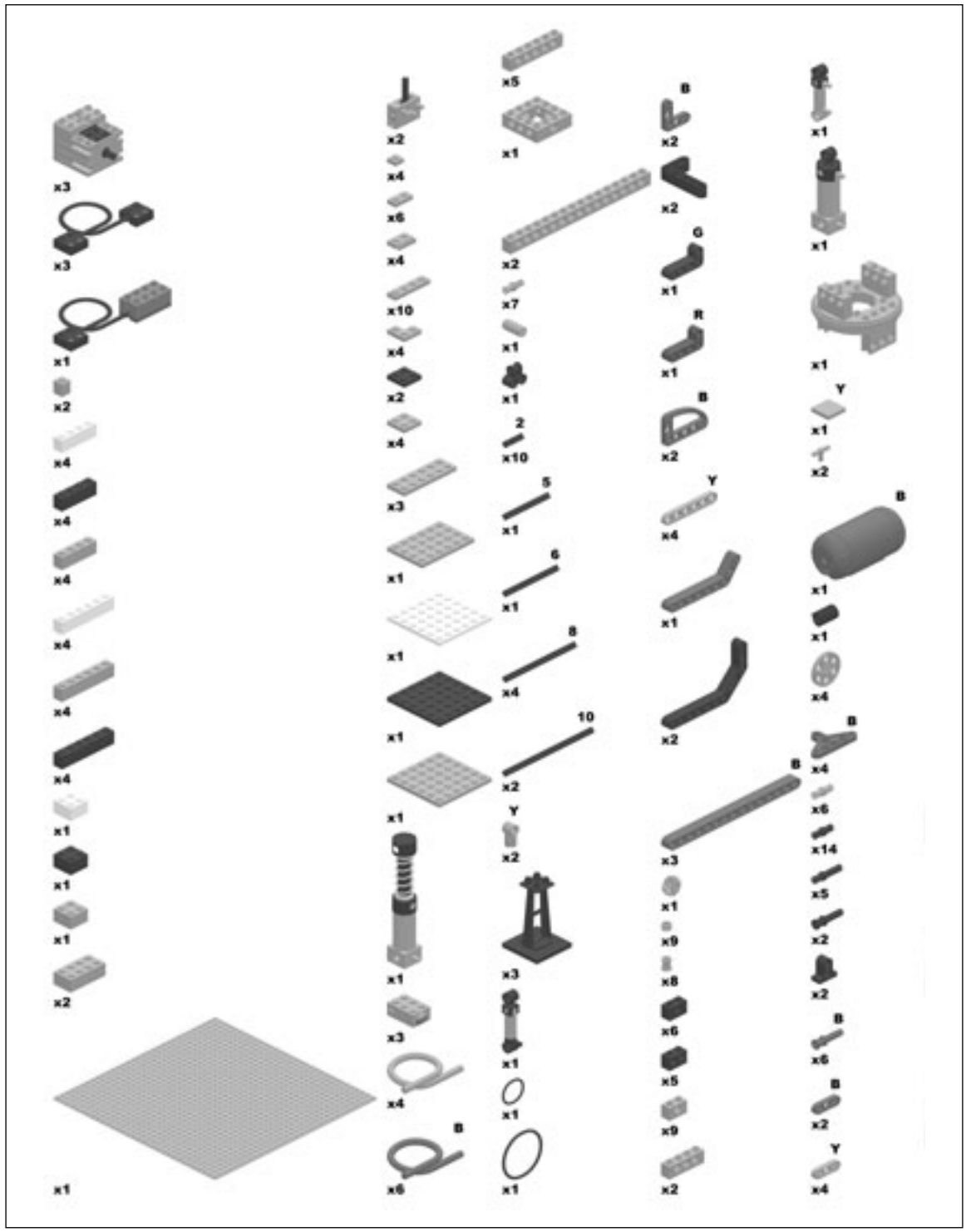
The Learning Brick Sorter

Mario Ferrari



Bill of Materials

These are the parts you will need to build the Learning Brick Sorter as shown.



NOTE

I designed it to be easy to replicate from parts available in LEGO sets in current production. If you own the RIS kit and the Ultimate Accessory Kit, you will need very few additional parts. Moreover, most of the parts can be easily replaced with other ones if you don't have them. During the building steps, I will provide some suggestions for replacing the less common parts.

Introduction

A few weeks after the LEGO MINDSTORMS Robotics Invention System (RIS) was released in October1998, Huw Millington published an Automatic Brick Sorter on his Web page www.brickset.com/huwhomepage/moc/mindstorms/abs1. At that time, I believe this was the first robot of this type. Next, Dave Baum created a second type of brick sorting machine, which can be seen at www.baumfamily.org/lego/creations/sorter.html. In my eyes, this brick sorter was a work of genius as David's interpretation of a brick sorting robot solves the task with minimal configuration.

These examples are only the first two milestones; however, since the release of the RIS, the theme of placing LEGO bricks into different bins according to their color has been explored quite thoroughly through a variety of different of robotic creations.

That said, you might wonder what makes this particular Brick Sorter worthy of this book? Although it shows some interesting building techniques and is a pleasant machine to see in action, what sets this robot apart from other brick sorter is not its ability to sort bricks by color, but rather its ability to learn how. Why is this feature so important? To understand the point we must step back a bit from our brick sorter and consider a wider perspective.

Machines that Learn

Let's jump back to the time when you were a toddler and your parents taught you how to walk. Did they really *teach* you how to walk in the same way a teacher explains his/her pupils how to solve quadratic equations? I bet they didn't. Parents don't *explain* their children how to walk, they *show* them how to walk. The difference is very important: if parents had to explain the processes behind the apparently simple action of walking, they should wait for their children to be old enough to understand the many principle of physics which are involved, and would be a very challenging task for both the master and the apprentice. In fact, if the aim of physics is to explain the world, then, from the learner's point of view, the contrary is equally true: the world is the best possible explanation for physics.

Think of learning how to ski using a book with text and schematics. Would it be easy? Surely humorous, but not easy. Having a teacher that shows you what to do makes the learning process much easier: as any of us know from direct experience, an example is worth a thousand words. This applies to many other activities, especially those abilities that involve learning how to use our arms, legs or body for a specific task. However, the mechanism of learning by example is so powerful that even when we approach conceptual matters we tend to consolidate the theory with analogies taken from the real world. Unfortunately, examples are by themselves not sufficient. If this were the case, we could learn how to ski by simply watching other skiers, Which, as anyone who has skied is aware, is a sport not as easy at it looks. We need a way to understand how close we are to the behavior we want to mimic to distinguish what's right and what's wrong in our actions. In other words, we need *feedback*. Sometimes the feedback comes from a person who helps us to learn a new skill, other times the feedback is the environment, as a skier discovers by falling a few times.

This learning mechanism has another big advantage: We not only learn, we also learn *how to* learn. In other words, when we acquire a new ability, we also acquire a model that we can apply—with a few changes—to different situations, a property called *adaptability*. Adaptability is what allows an experienced skier to perform relatively well at his first time water skiing: He just has to learn the differences between the two activities instead of having to learn from scratch. Adaptability is what saves parents and teachers from explaining every single detail of the universe.

As feedback and adaptability work for human beings, they seem a desirable and promising approach to use with robots too. A robot learns to solve a task without having to be programmed, and this makes it suitable for a wide range of applications. Additionally, it allows people with no programming experience to instruct it through simple demonstrations and feedback. Is this a dream? No, it's reality. Many industrial robots can already be "programmed" this way, where during a learning session, a human trainer manually activates the robot, forcing it to perform the movements it will need to perform during the production cycle; the robot will then be able to replicate the same movements an unlimited number of times. Some artificial vision systems include a more sophisticated learning mechanism: the operator shows the vision system a variety of objects, each one at a variety of angles, and every time supplies the software with the name — or code — of the shown object. When the training procedure is finished, the system is able to recognize the different objects and tell their names by just "looking" at them. Finally, an increasing number of software applications—including computer games—incorporate Artificial Intelligence (AI) techniques to learn from their users' behavior and become more responsive or more competitive. Actually, learning and adaptability have been part of AI research since the beginning, because we cannot even conceive a form of intelligence that does not incorporate these concepts.

On the subject of computer programs, we are used to thinking of computer games as sequences of simple instructions, and this is what they ultimately are: add the number X

to the number Y, if the result is greater than Z then do this, otherwise do that, and so forth. This process is defined as an *algorithm*. A more formal definition of an algorithm is a step-by-step problem-solving procedure, especially an established, recursive computational procedure for solving a problem in a finite number of steps. How can we turn this rigid structure of equation-based logic into a process that is flexible enough to learn from experience? The answer is *data*. In fact, data is the key to storing knowledge that comes from experience. In most of AI programs, the algorithm remains the same, but exhibits different behaviors according to the data it finds in its knowledge base.

AI is a very complex matter, being at the top of computer science hierarchy for both for the ambitiousness of its goals and for the sophistication of the techniques involved. Genetic algorithms, genetic programming, artificial neural networks… all these techniques lead to amazing results, but require the mastery of many mathematical instruments to be fully understood and used. The Learning Brick Sorter uses a much simpler technique, but it shows us how even an elementary learning mechanism greatly expands the flexibility of a robot. In fact, it can be trained to distinguish any three different colors (provided that they return well-separated light readings) and to place them in three bins according to your preferences. You can change the colors, or the order of the bins, and after a short training session the Brick Sorter will be ready to work with the new specifications.

Building the Learning Brick Sorter

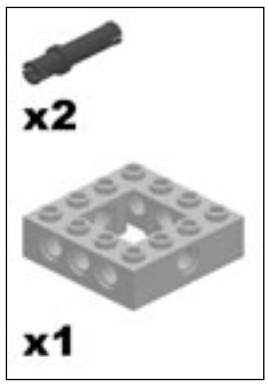
Without further ado, let us begin with the building instructions for the Learning Brick Sorter.

The Body

First, we are going to build the main body of the Brick Sorter. The Body sub-assembly supports the Arm sub-assembly (built later on) and provides the Arm sub-assembly with the capability to perform rotary motions in order to reach the bins where the bricks are sorted. The double pulley with pins, visible on the right of the model, will interface with

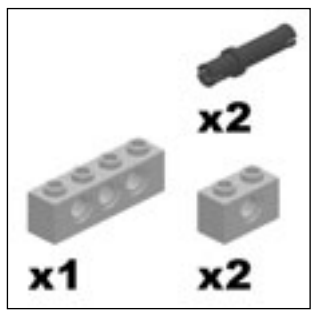
a touch sensor added later on. This combination is what allows the Brick Sorter to track the position of the Arm sub-assembly. The small pump, visible on the left of the model, provides airflow for the pneumatic pistons that operate the Arm sub-assembly.

Body Step 0a



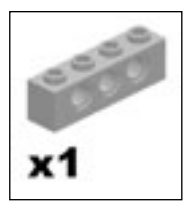
The TECHNIC Brick x4 with an open center is used to hold the motor. If you don't have it, you can replace it with standard TECHNIC bricks. This alternate sub-assembly is shown in **Body Step 0b**.

Body Step 0b

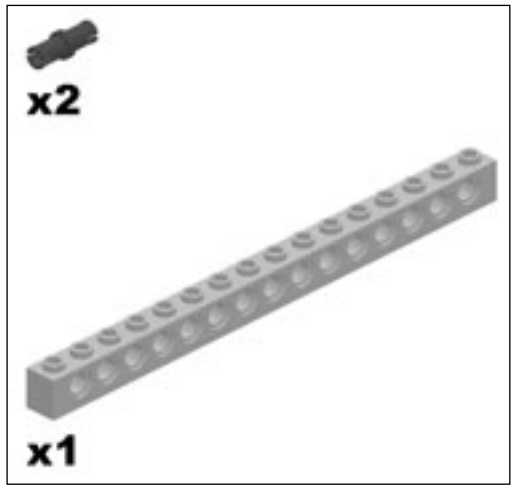


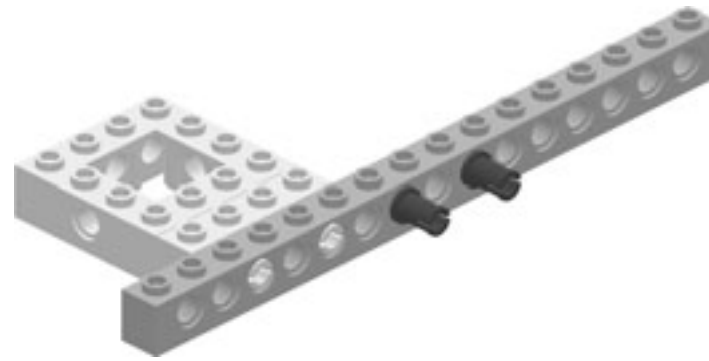
This is the alternate sub-assembly for **Body Step 0a**.

Body Step 1

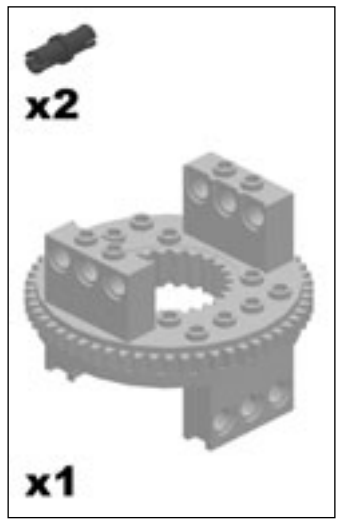


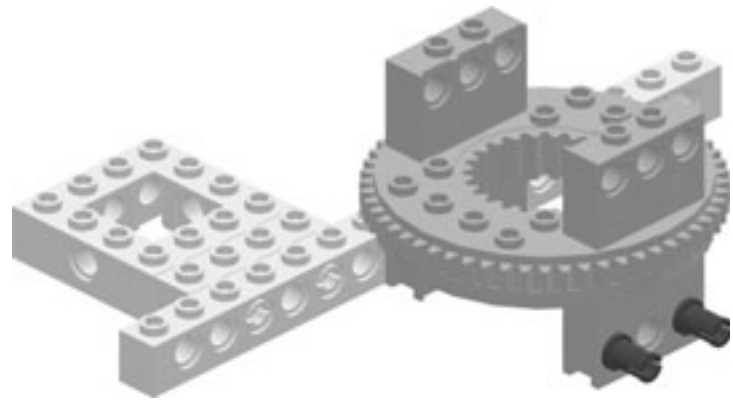
Body Step 2



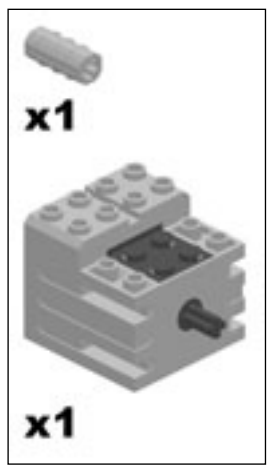


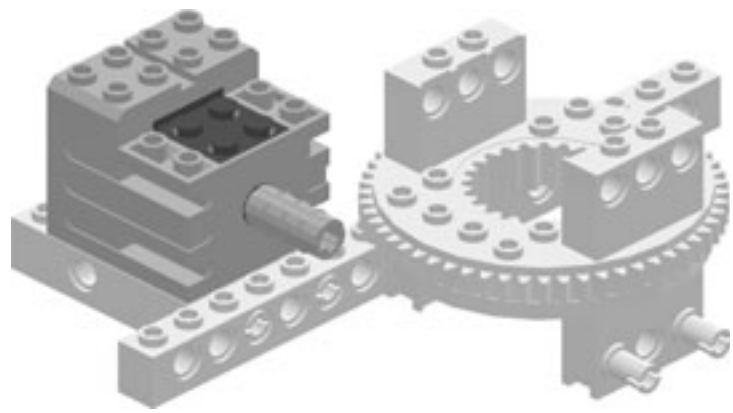
Body Step 3



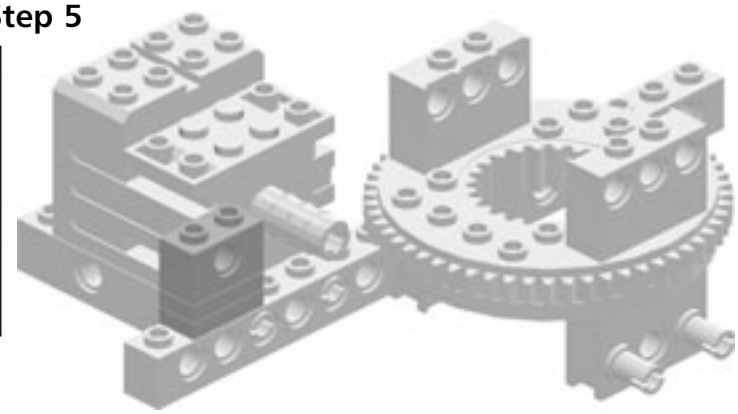


Body Step 4

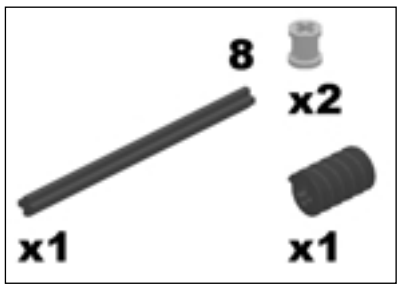


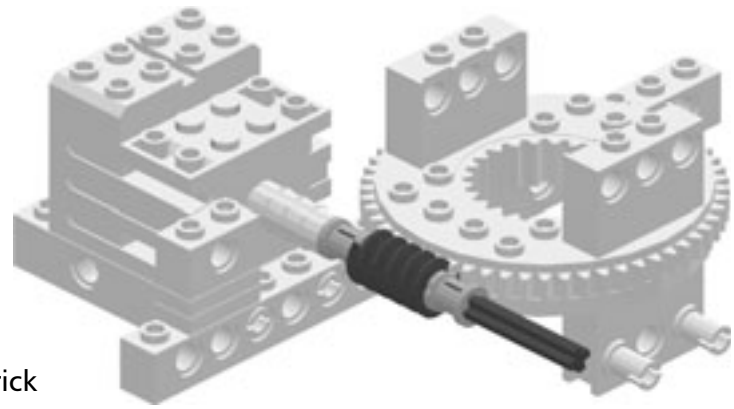


Body Step 5



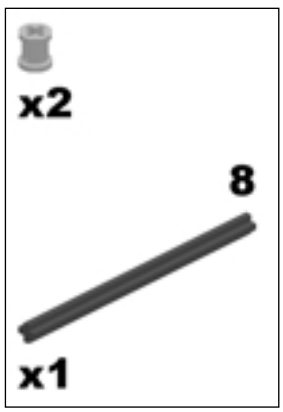
Body Step 6

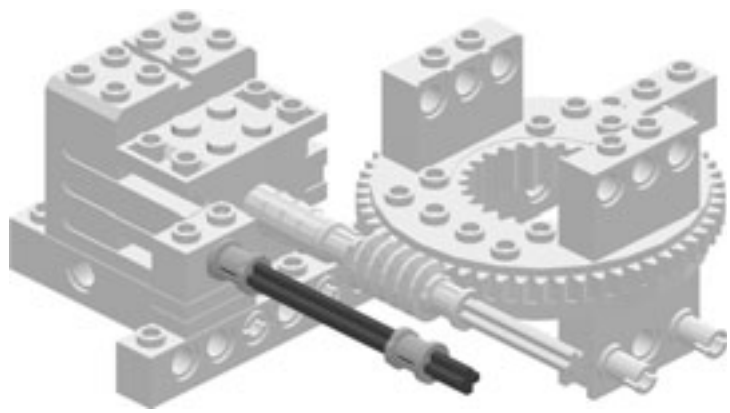




Although we will tune the mechanical phase of the Brick Sorter in the section "Tuning and Testing the Learning Brick Sorter" later in the chapter. I suggest keeping the upper portion of the turntable parallel to the lower part, as shown here.

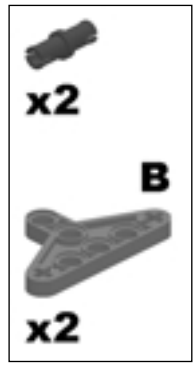
Body Step 7

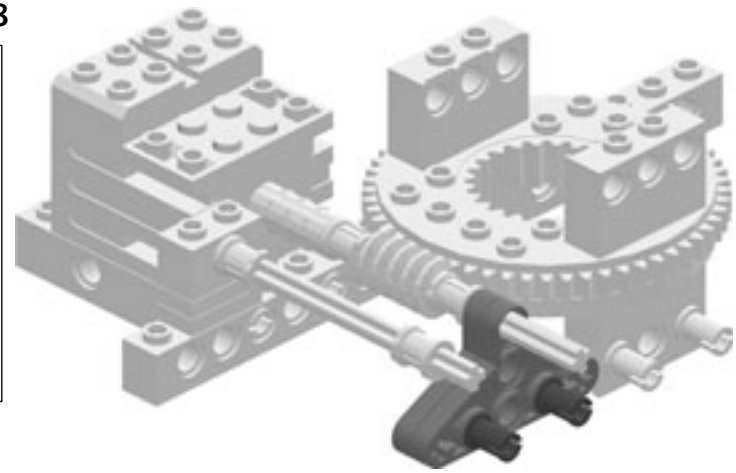




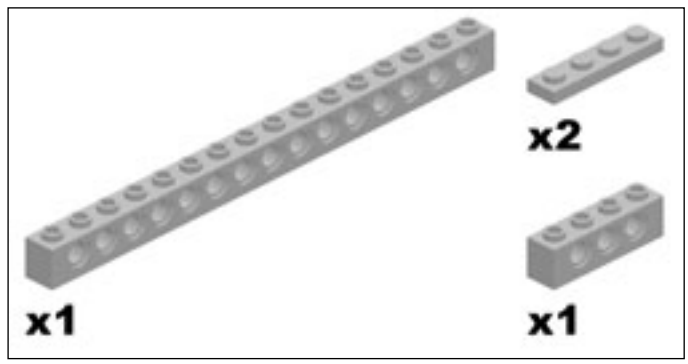
The sole purpose of this axle is to prevent the pneumatic tubes from touching the gears.

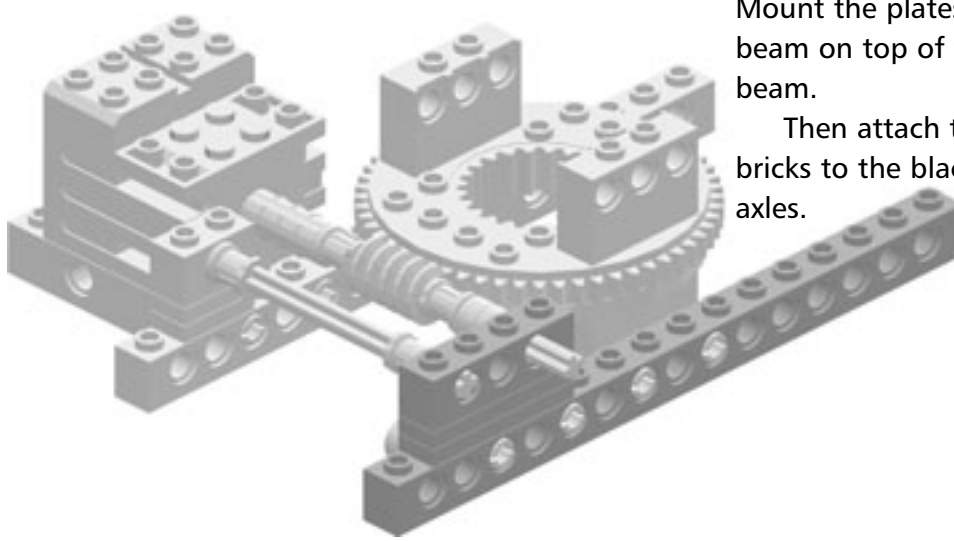
Body Step 8





Body Step 9

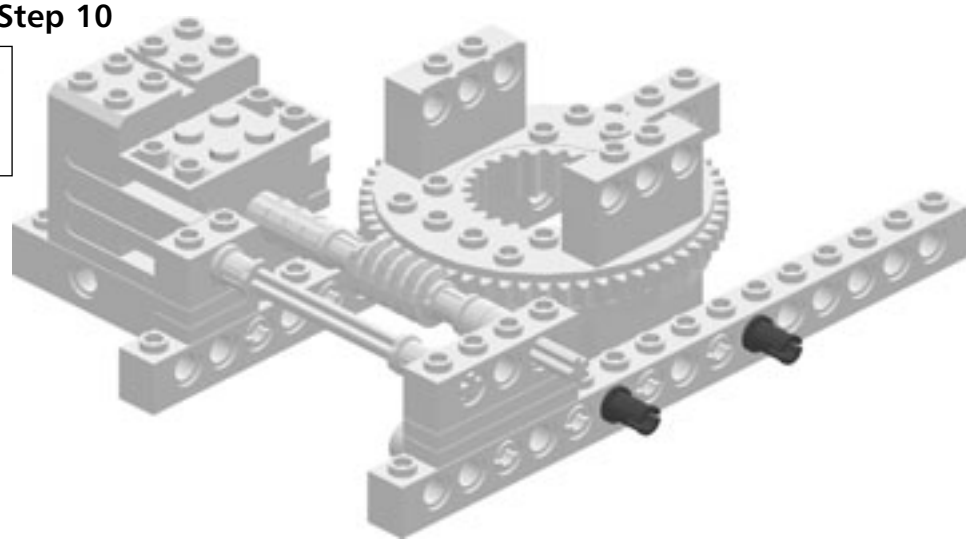




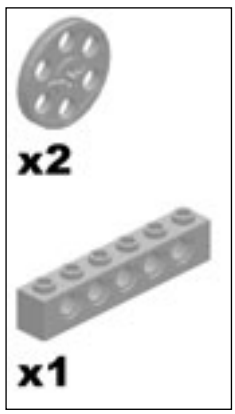
Mount the plates and the 1x4 beam on top of the 1x16 beam.

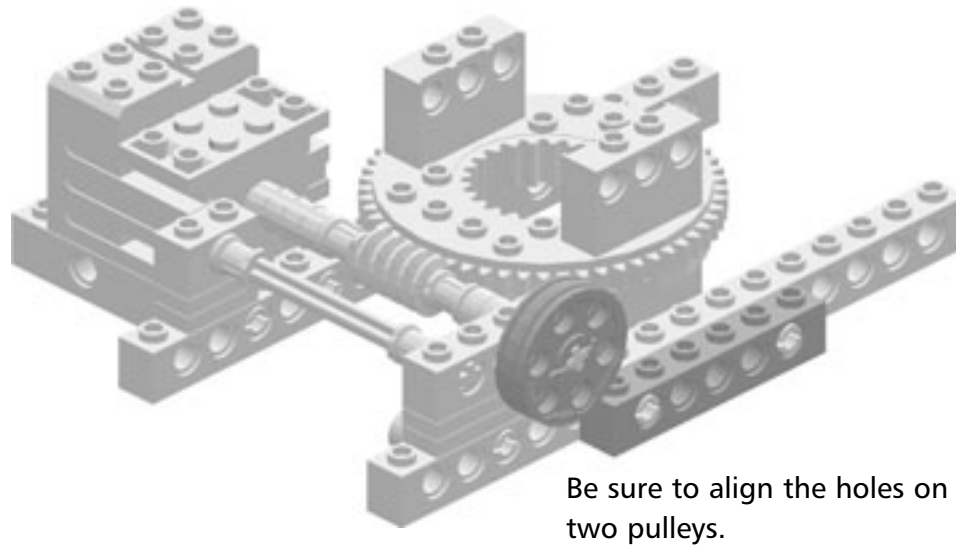
Then attach the plates and bricks to the black pins and axles.

Body Step 10



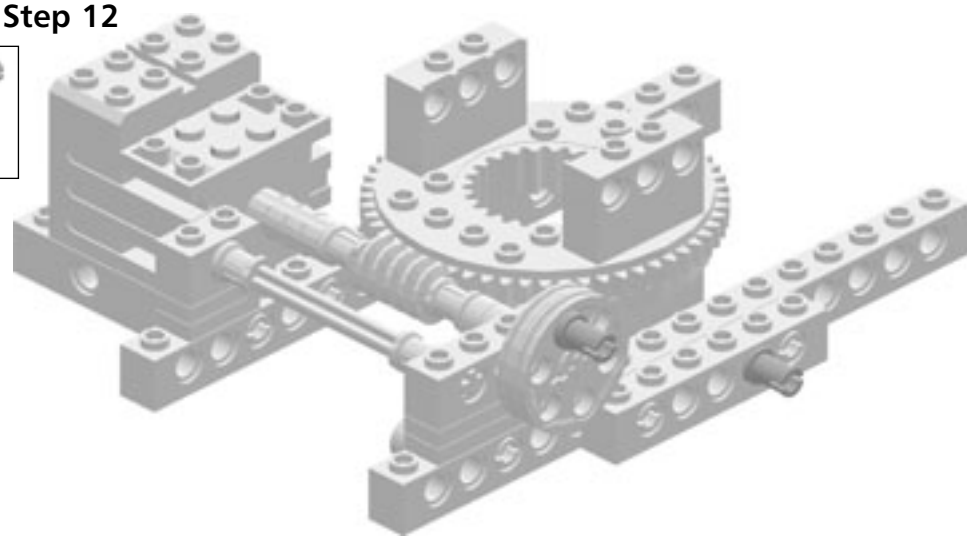
Body Step 11



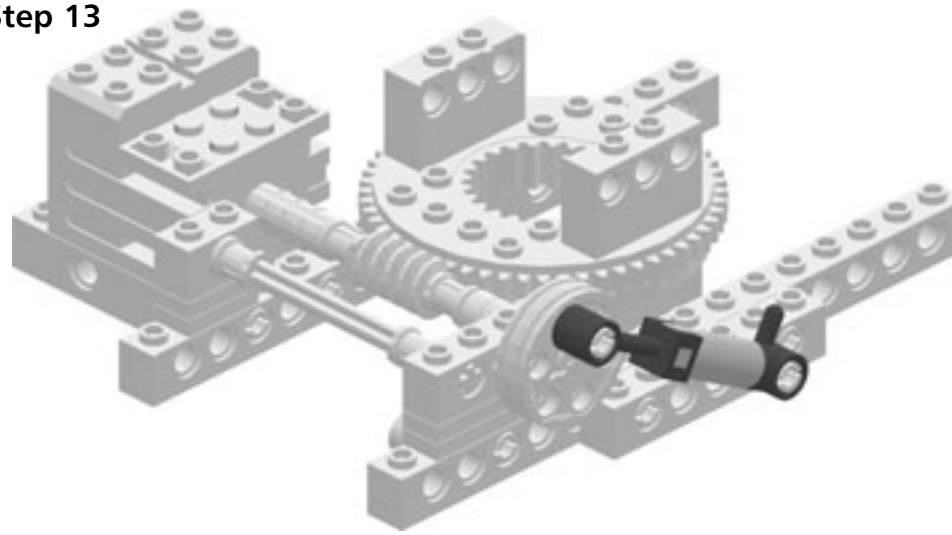


Be sure to align the holes on the two pulleys.

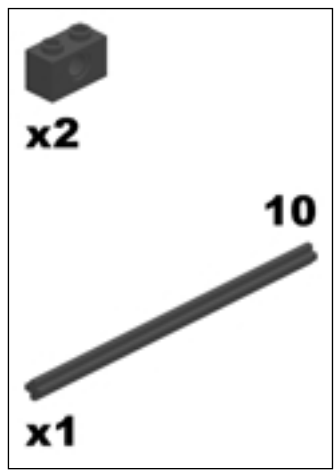
Body Step 12



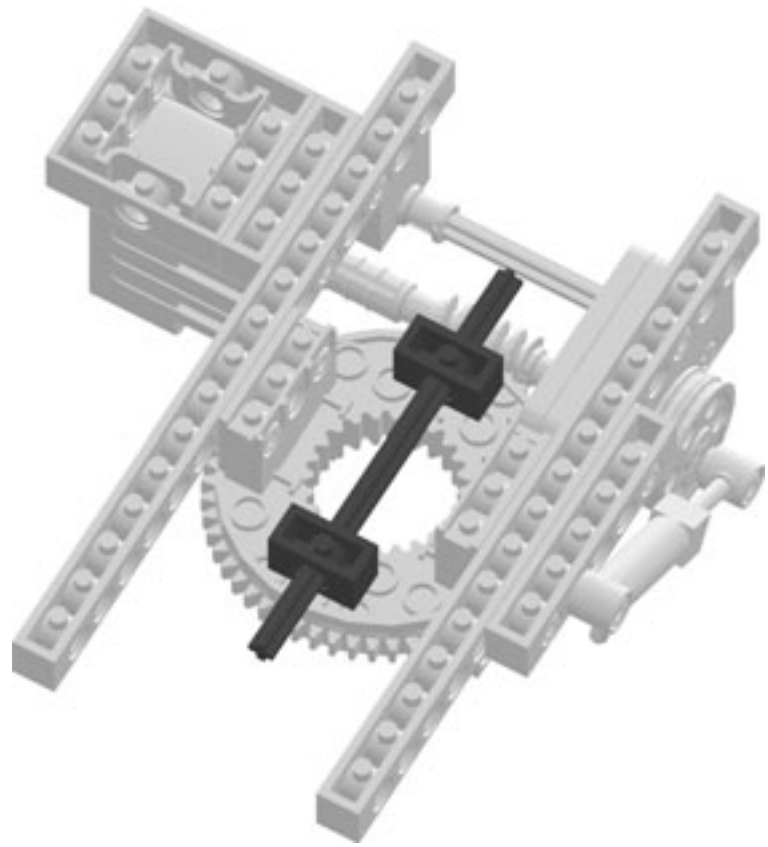
Body Step 13



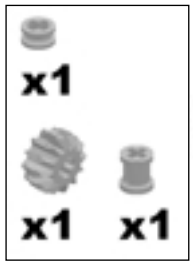
Body Step 14

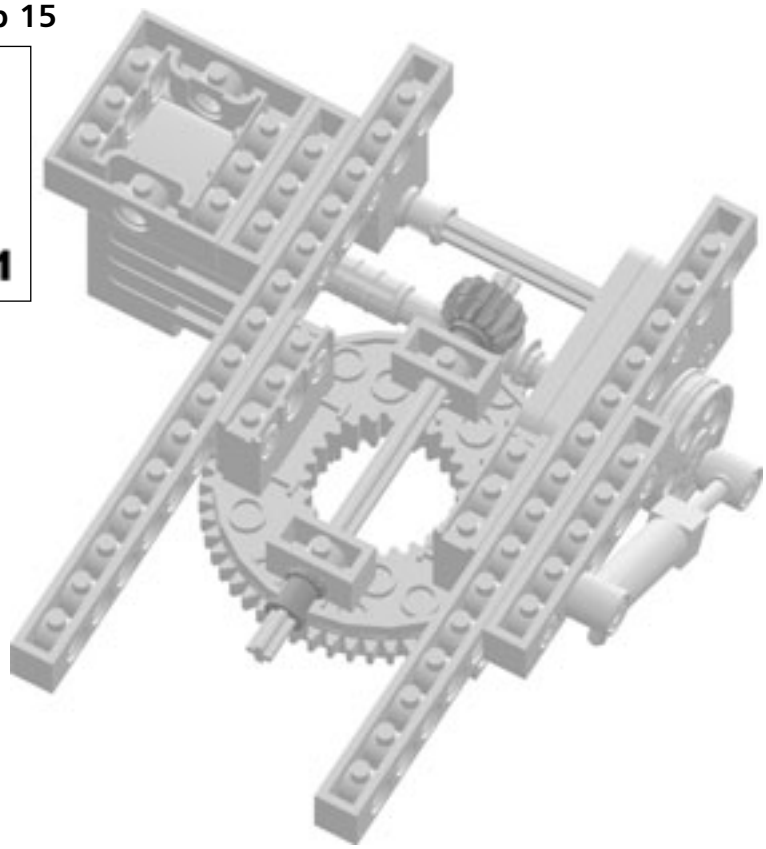


At this point, turn the assembly upside down to mount the parts as shown. These bricks and axle form the tracking system.

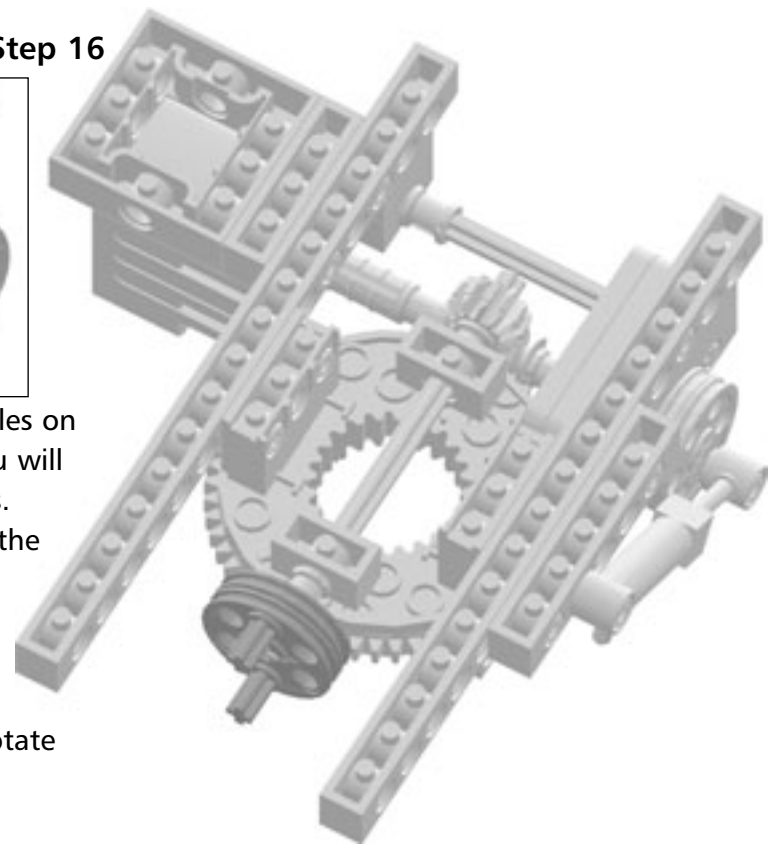


Body Step 15



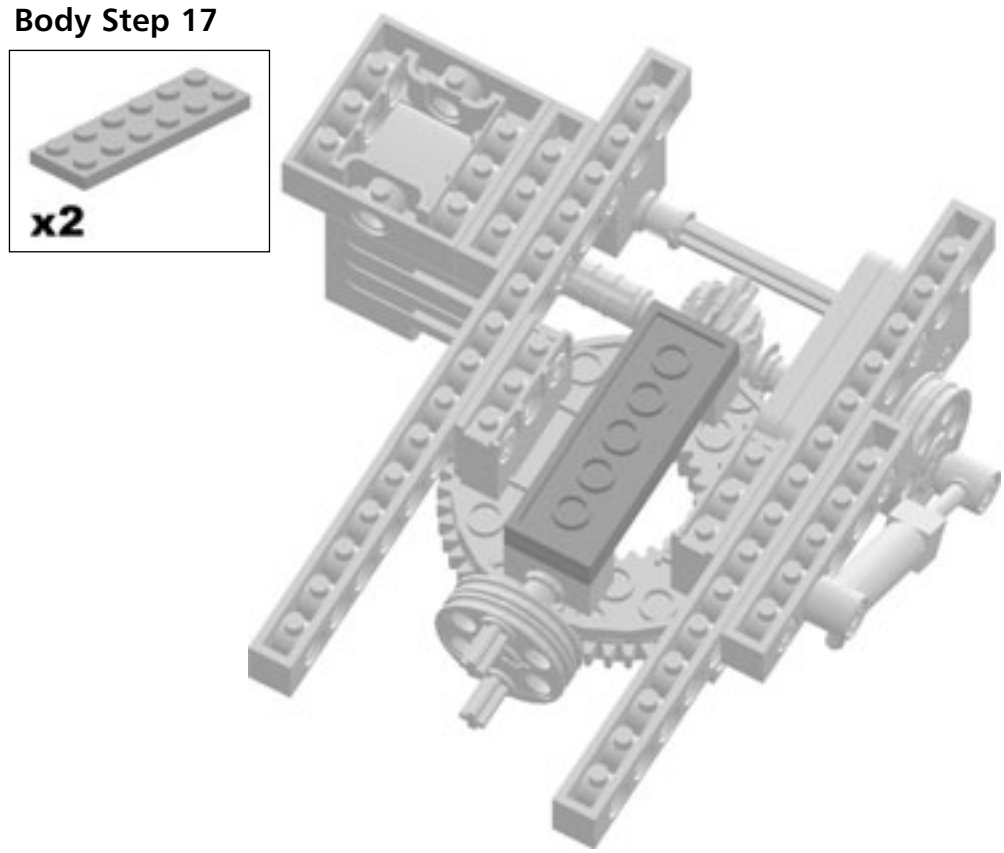


Body Step 16

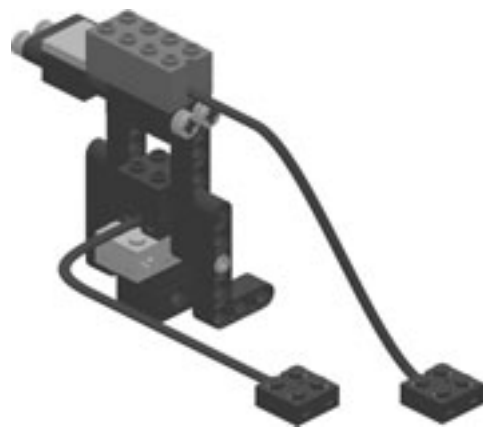


Again, be sure to align the holes on the two pulleys, otherwise you will not be able to mount the pins.

The correct orientation of the pulleys is that shown in this image. If the pins of your model do not align as shown, slide the gear added in **Body Step 15** off the worm gear, rotate the pulley and reassemble.



The Pickup Stand

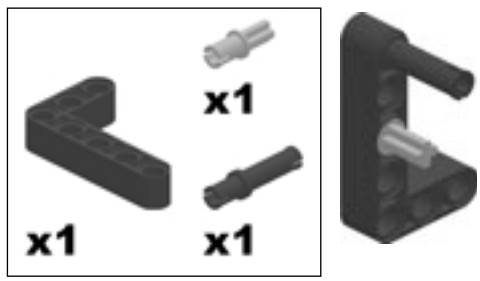


The Pickup Stand sub-assembly may be small, but it performs three extremely important actions:

- It holds the brick waiting to be sorted.
- It decodes the color of the held brick through the light sensor.
- It tracks the position of the rotating arm through a touch sensor.

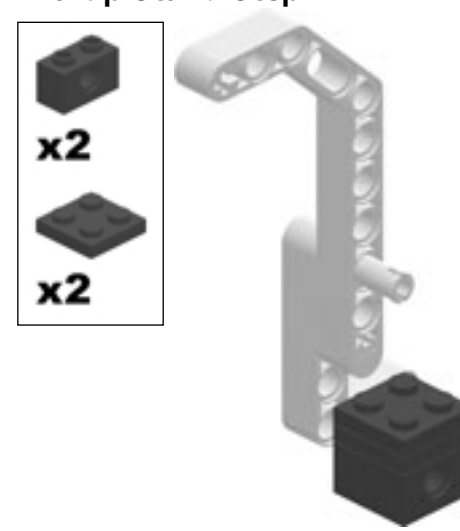
The LEGO light sensor is actually unable to distinguish colors. Its ability is limited to detecting light and measuring its intensity. Thus, we are going to distinguish the color of the bricks through the amount of light that they reflect back into the sensor. For this reason some colors, which reflect a similar amount of light, are nearly indistinguishable. For example, it's very hard to tell a black brick from a blue one, or a gray brick from a green one.

Pickup Stand Step 0

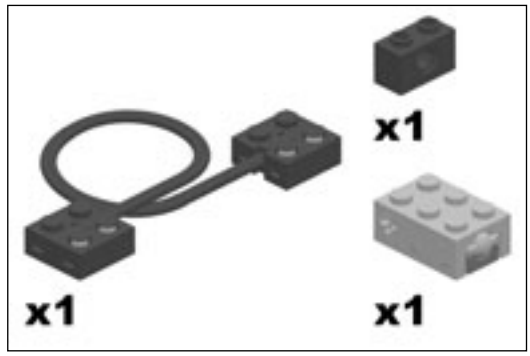


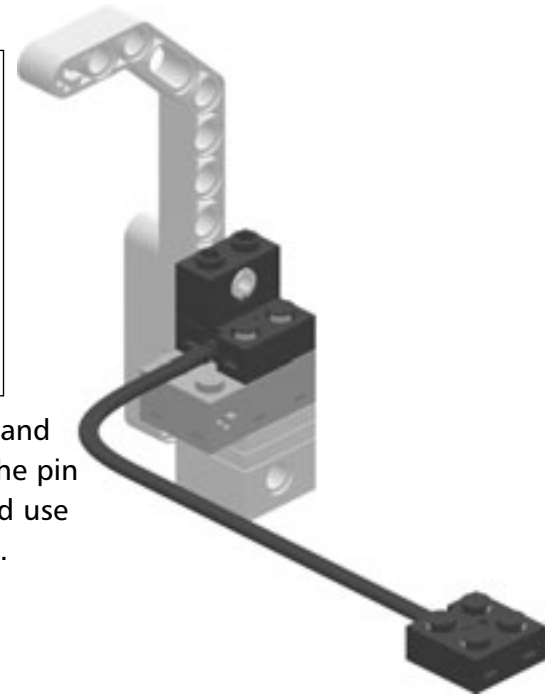
Pickup Stand Step 1

Pickup Stand Step 2



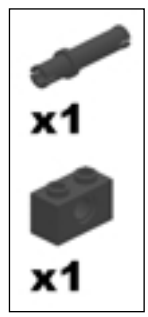
Pickup Stand Step 3

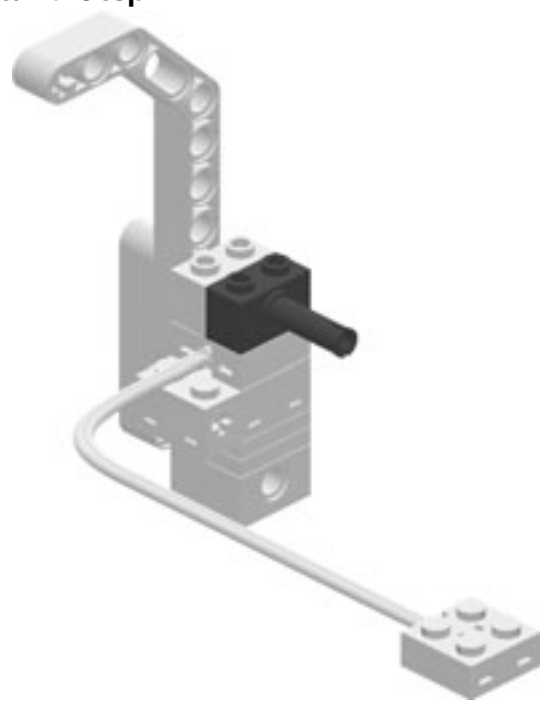




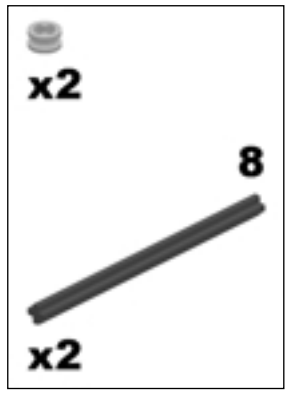
First, stack the touch sensor, the cable and the brick. Then connect the group to the pin protruding from the liftarm. You should use a short electrical connector in this step.

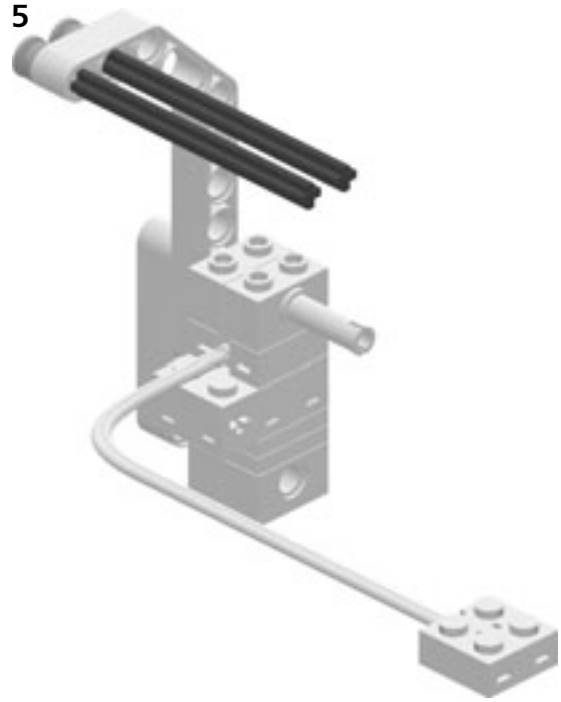
Pickup Stand Step 4



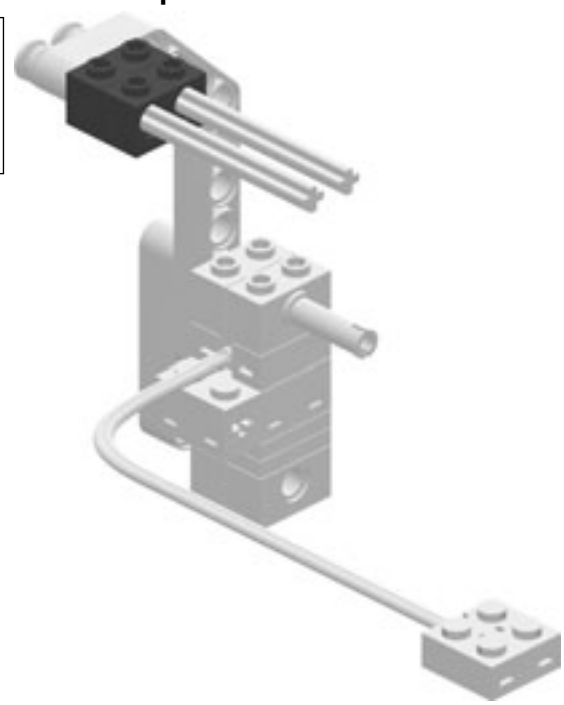


Pickup Stand Step 5

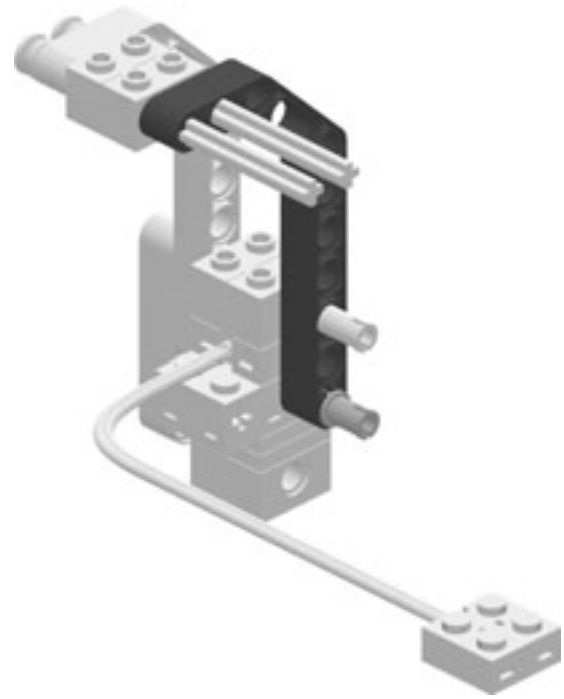




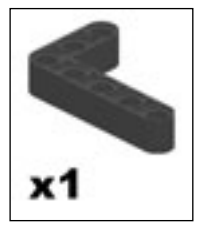
Pickup Stand Step 6

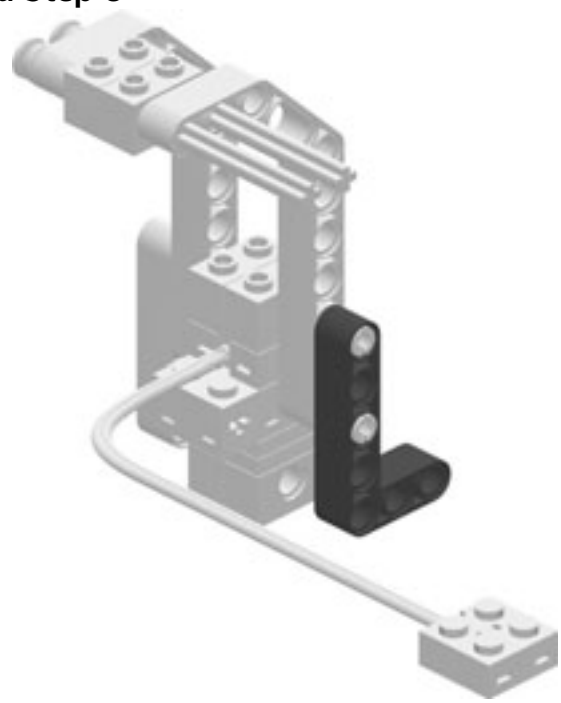


Pickup Stand Step 7

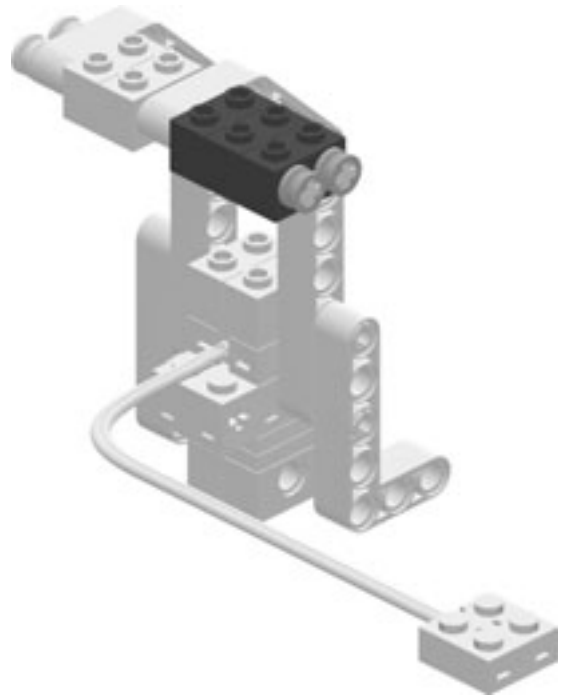


Pickup Stand Step 8

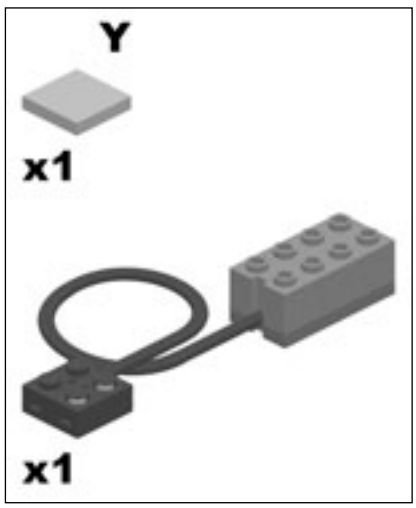


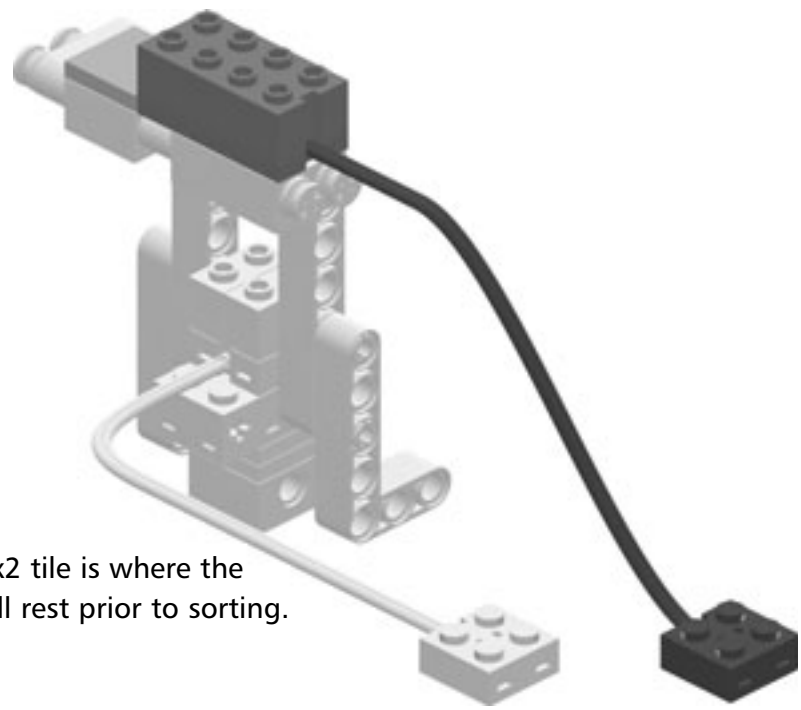


Pickup Stand Step 9



Pickup Stand Step 10





The yellow 2x2 tile is where the 2x2 bricks will rest prior to sorting.

The Arm

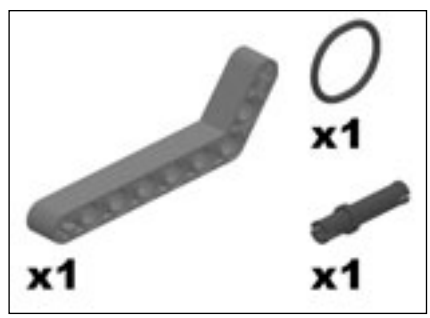
The Arm sub-assembly will be mounted on top of the turntable, which is part of the Body sub-assembly. The Arm sub-assembly features two pneumatic pistons:

- A large pneumatic piston used to lift and lower the hand
- A small pneumatic piston used to open and close the hand that will grab the brick

NOTE

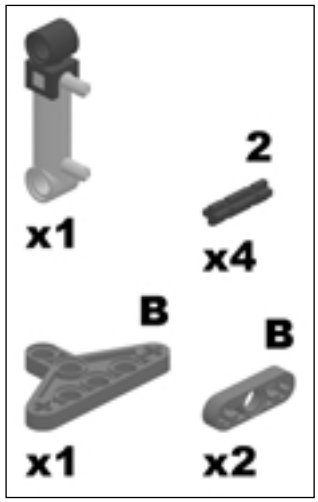
For more information on the mechanics of pneumatics, please refer to Masterpeice 4 "PneumADDic II" by Kevin Clague.

Arm Step 0

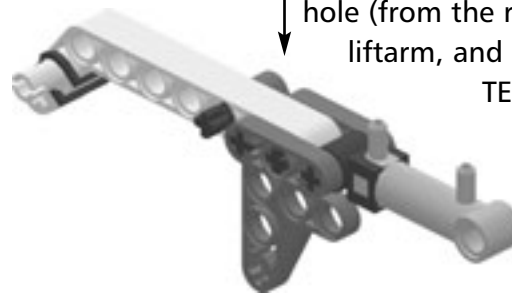


The short rubber band is used to increase the grip that the hand has on the brick. Thread the rubber band over the pin on one side of the liftarm, then pull it gently below the liftarm, and thread it over the pin on the other side.

Arm Step 1

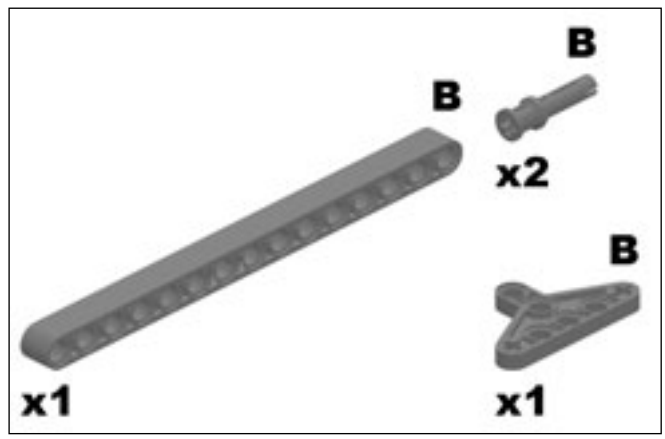


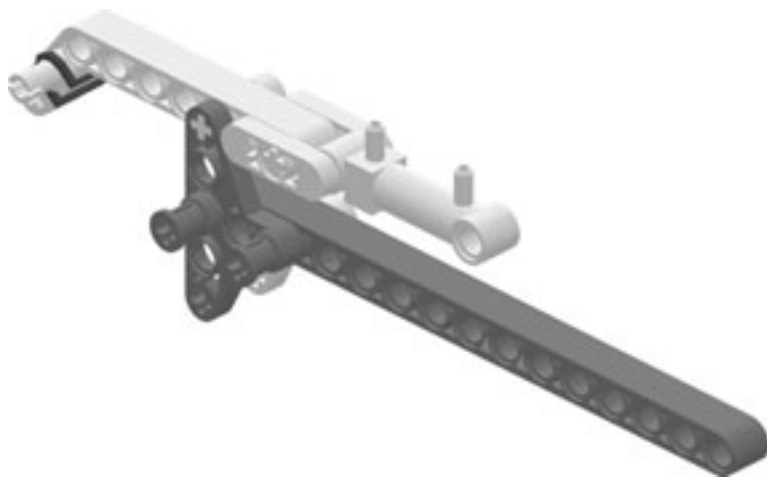
Slide the fourth #2 axle into the third hole (from the right) of the gray liftarm, and use it to connect the TECHNIC triangle.



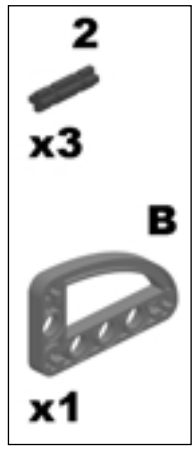
Attach the small pneumatic piston to the end of the liftarm in **Arm Step 0**, securing it with two 1x3 liftarms and three #2 axles.

Arm Step 2

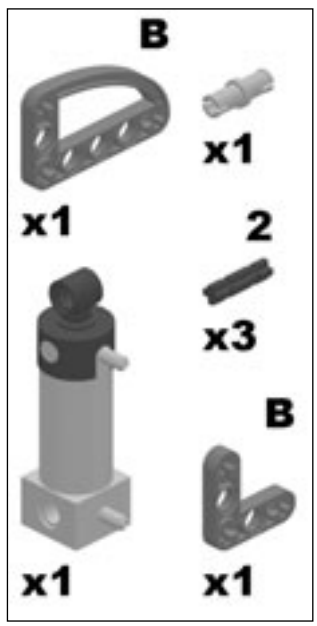




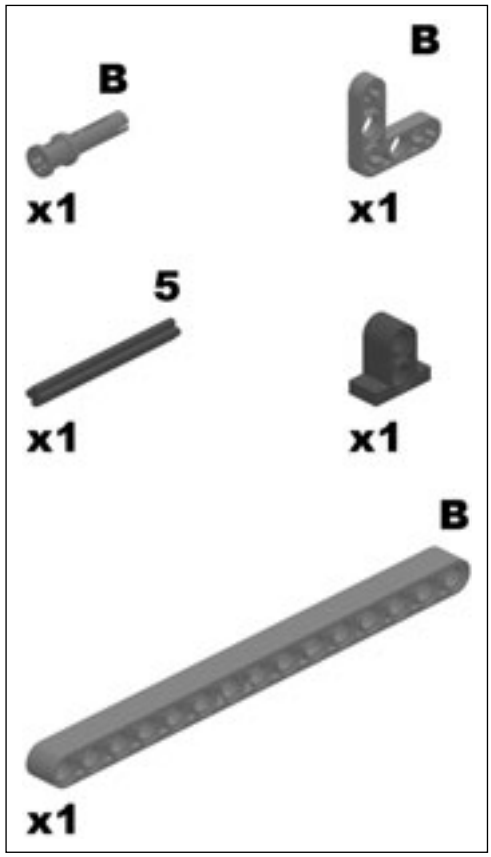
Arm Step 3



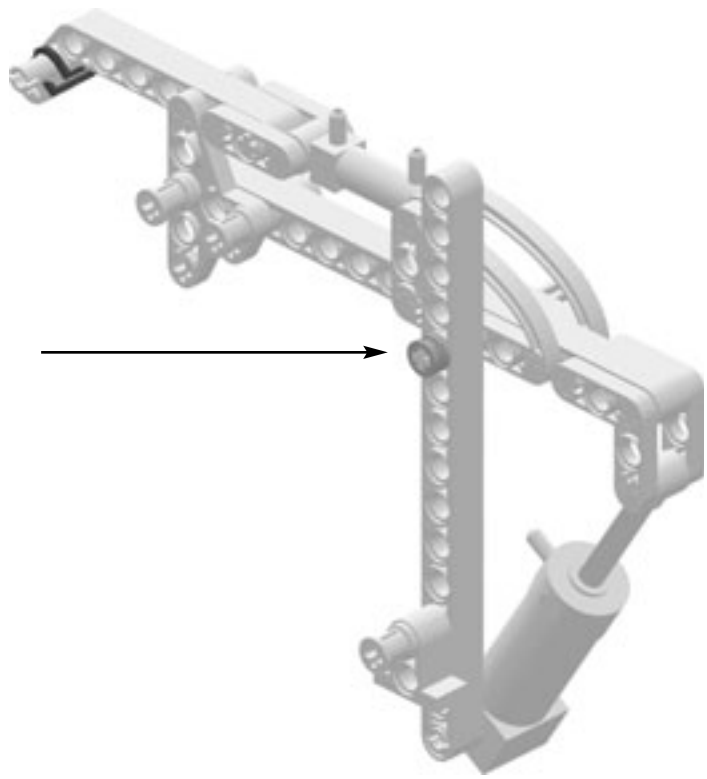
Arm Step 4



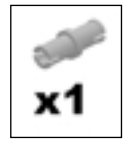
Arm Step 5



Arm Step 6

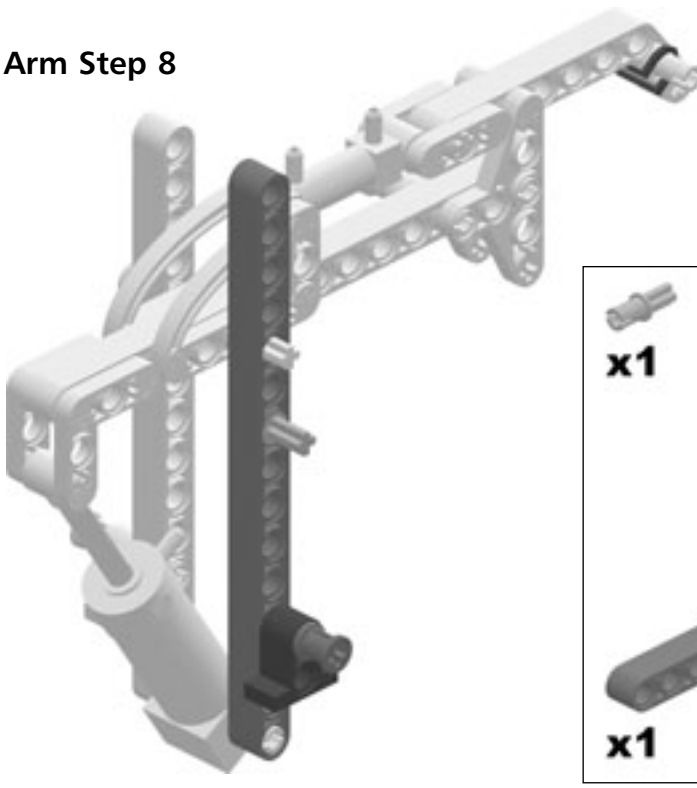


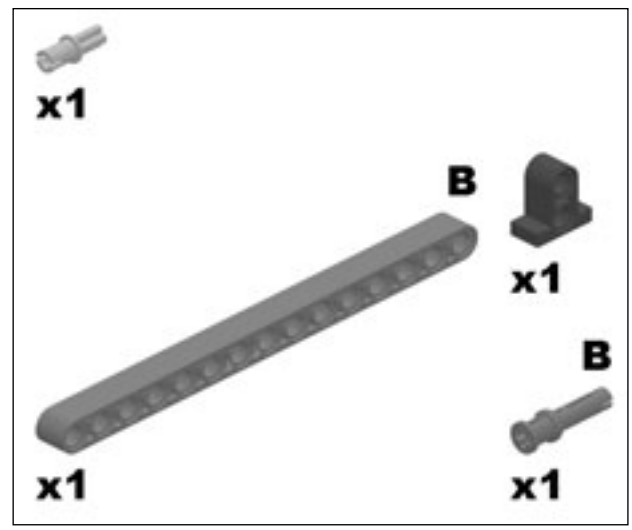
Arm Step 7



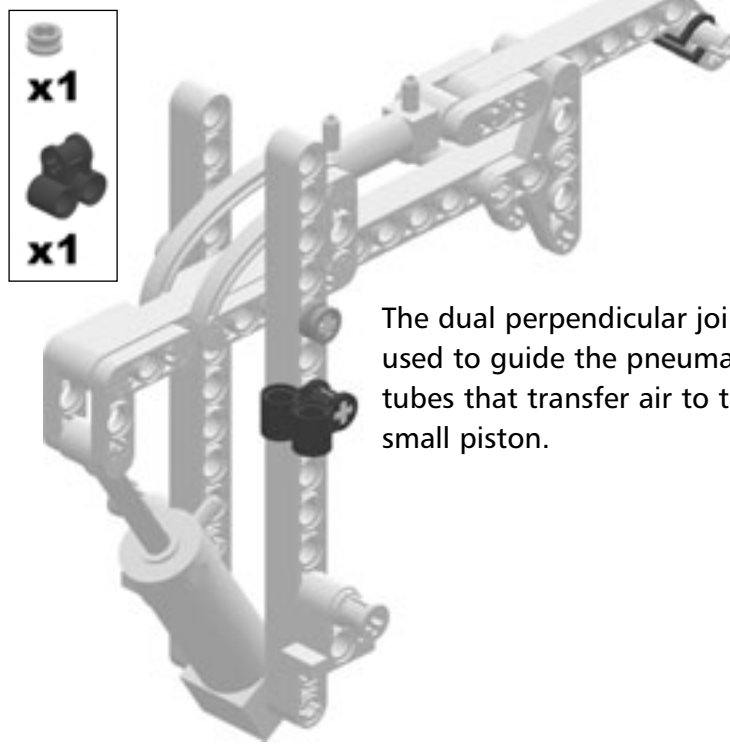
Rotate the model so that you are looking at the other side. Insert the pin into the bottom of the piston.

Arm Step 8





Arm Step 9

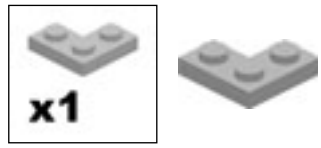


The dual perpendicular joiner is used to guide the pneumatic tubes that transfer air to the small piston.

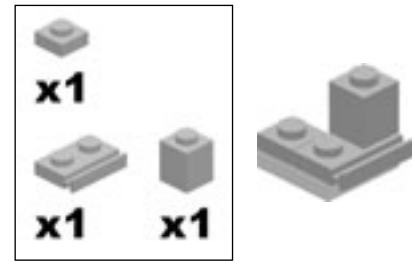
The Valve

The Valve sub-assembly is a motorized valve switch. As the RCX can control only electric devices (like motors and lamps), we need an interface to make it able to control the pneumatic pistons of the Arm sub-assembly. This Valve sub-assembly couples an electric motor with a pneumatic switch, thus allowing the program to transform the airflow into short electric impulses to the motor. You will need to create **two** of these sub-assemblies, one to control the small pneumatic piston that operates the hand, and another to control the large pneumatic pistons that raise and lower the arm.

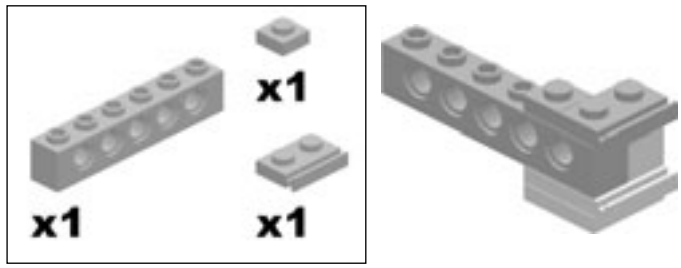
Valve Step 0



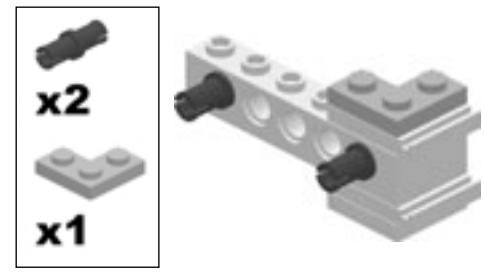
Valve Step 1



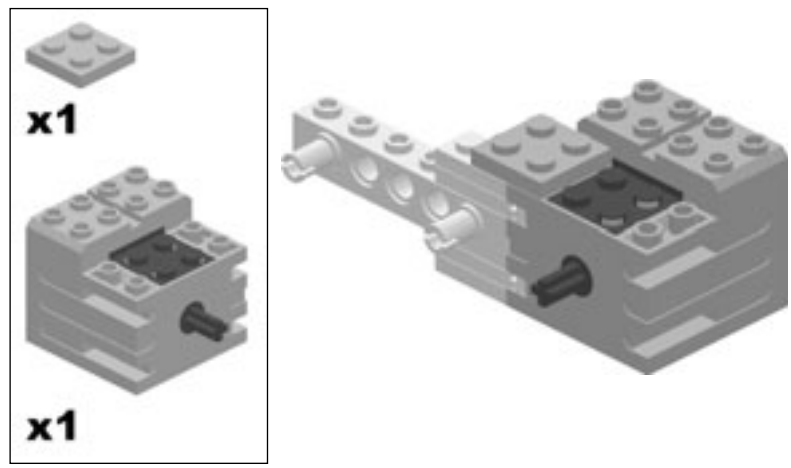
Valve Step 2



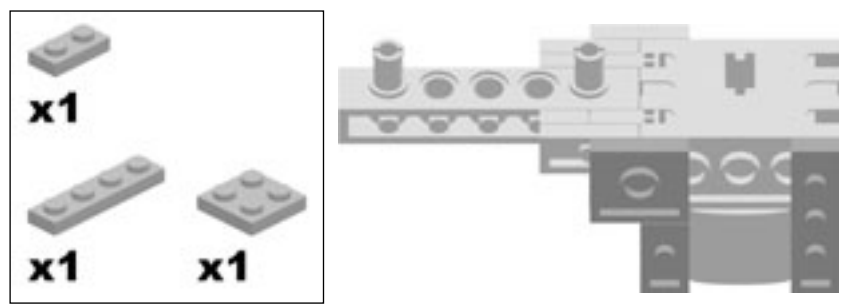
Valve Step 3



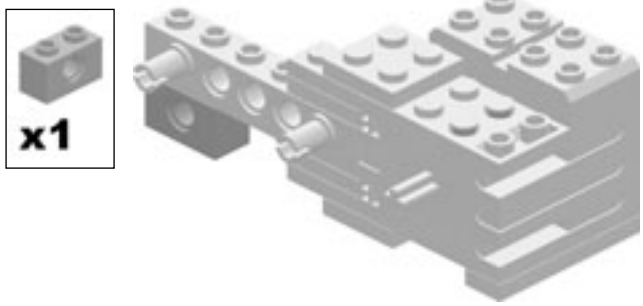
Valve Step 4



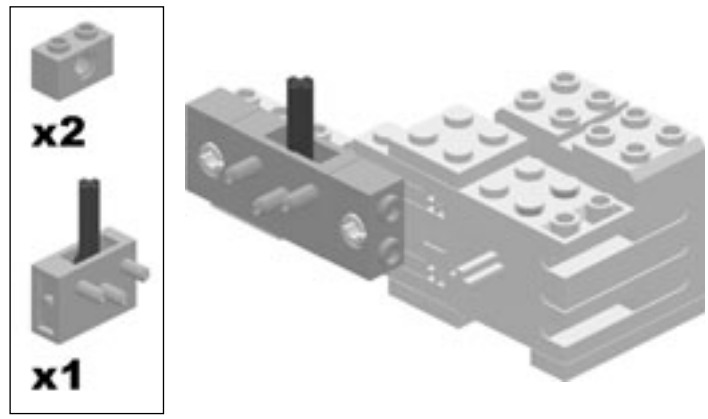
Valve Step 5



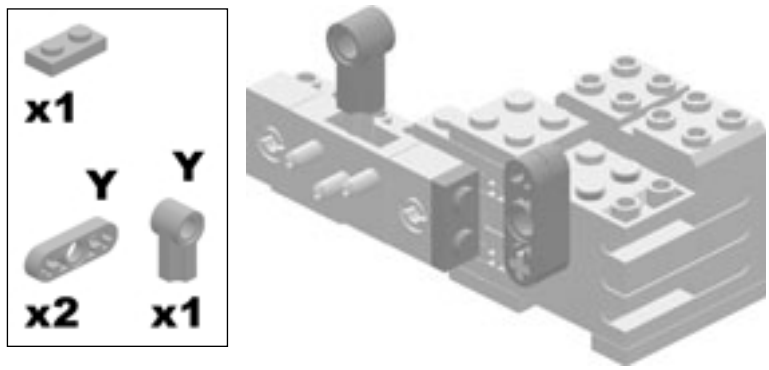
Valve Step 6



Valve Step 7

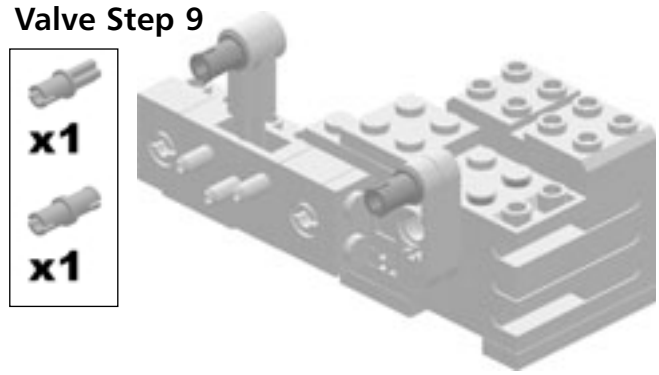


Valve Step 8

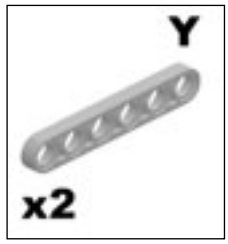


The 1x2 plate serves as stop point for the liftarms attached to the motor.

Valve Step 9



Valve Step 10

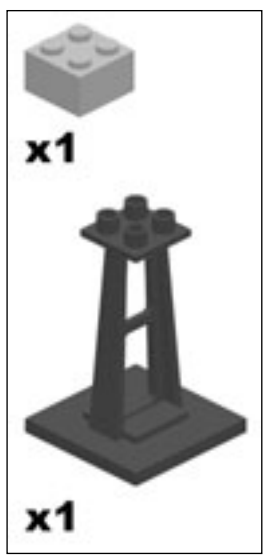


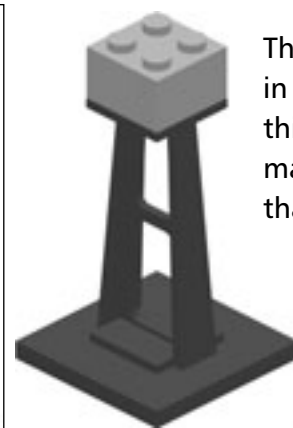
The valve switch is now finished. Don't forget you will need to build **two** of these.

The Bins

The Bin sub-assemblies are the receptacles for the sorted bricks. You will need to create **three** Bin sub-assemblies. In this model, I have chosen to make one black, one light gray and one white. You are free to build these using any color that you prefer; however, remember that you should choose three colors that reflect significantly different light values.

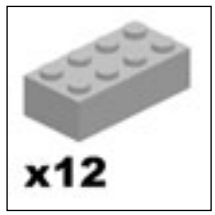
Bin Step 0a



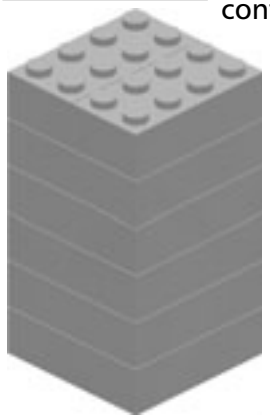


This support is probably the least common part in the Brick Sorter. If you do not have access to this support, it can be replaced with a support made of standard bricks. It must be not larger than 4x4 and four bricks tall (**Bin Step 0b**).

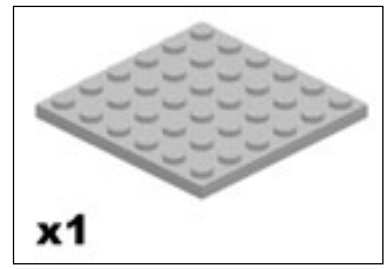
Bin Step 0b



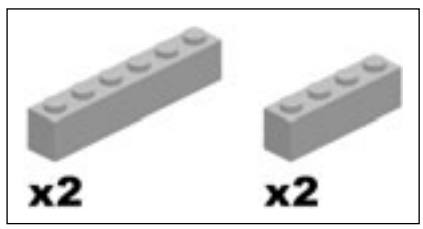
This is the alternate sub-assembly for the support which will hold the container.



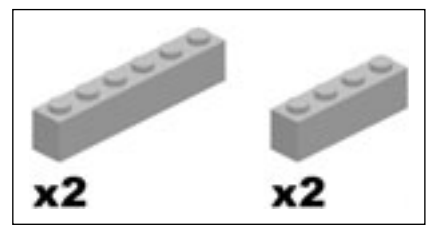
Bin Step 1

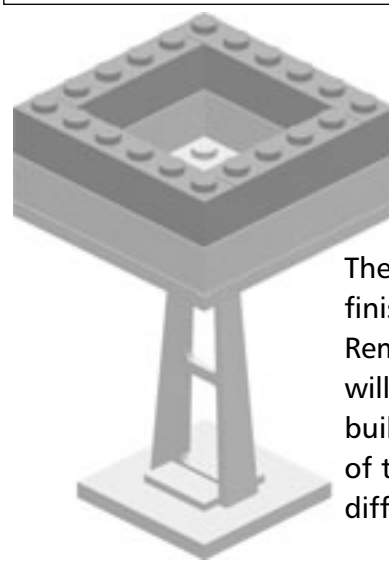


Bin Step 2



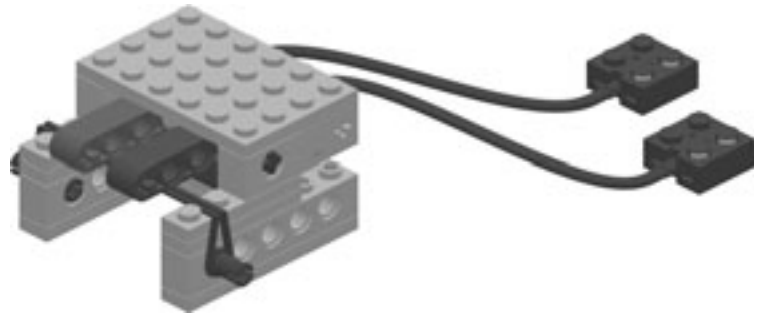
Bin Step 3





The gray bin is finished. Remember you will need to build **two** more of these using different colors.

The Switchbox

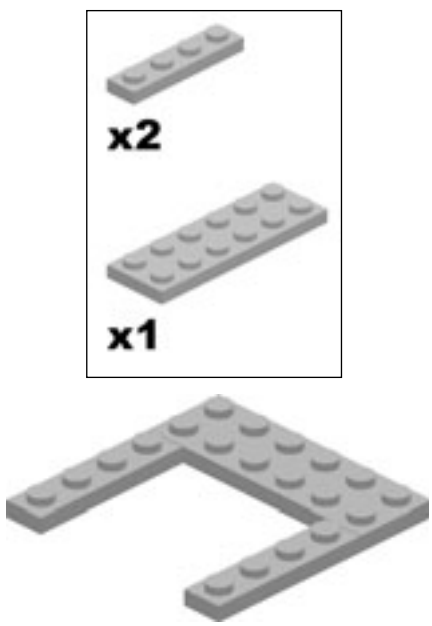


The Switchbox sub-assembly is the means through which the user will train the Learning Brick Sorter by providing the necessary feedback. The Switchbox sub-assembly is a small self-contained unit that has no mechanical linkage to any other part of the robot, except the two electrical connections to the RCX. For this reason, the position of this unit is not overly critical. Provided that it doesn't interfere with the movement of the Arm sub-assembly, and provided that you keep it attached to the RCX, you can place it wherever you prefer.

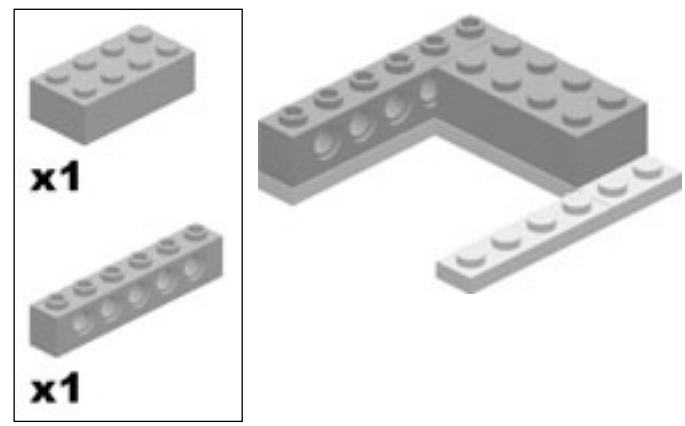
All in all, this Switchbox is nothing more than a pair of touch sensors assigned to define actions as either right or wrong. After having placed a brick in one of the bins, the robot will stop and wait for the user to press one of these sensors, according to whether the bin was the right one or a wrong one. The program will then use this information to "learn" from this feedback.

Instead of presenting the user two bare touch sensors to use as push buttons, I chose to encapsulate the sensors into small structures, so they can be activated by using two different colored levers.

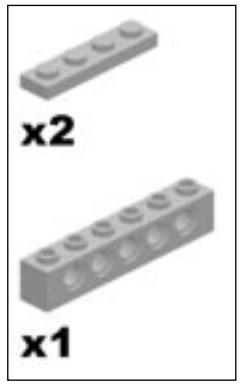
Switchbox Step 0

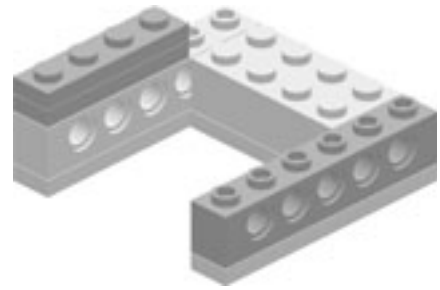


Switchbox Step 1

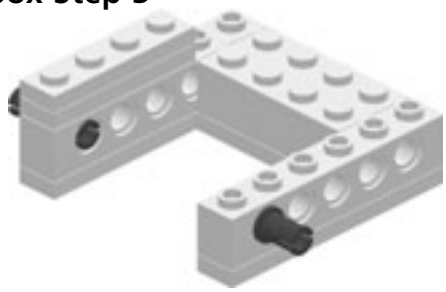


Switchbox Step 2

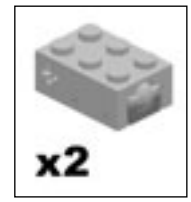


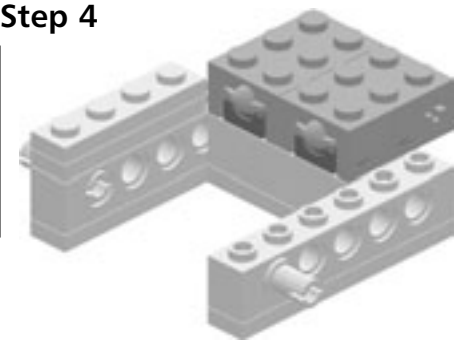


Switchbox Step 3

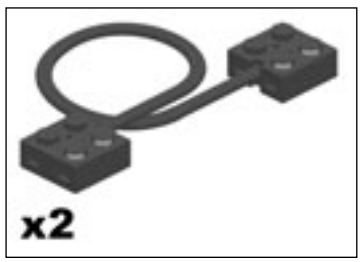


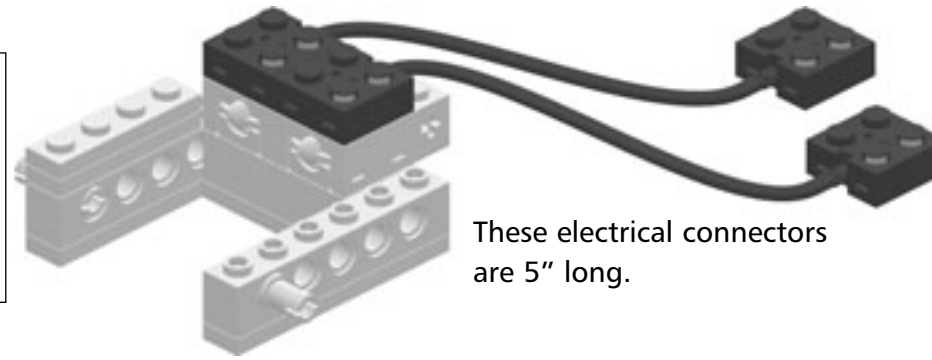
Switchbox Step 4





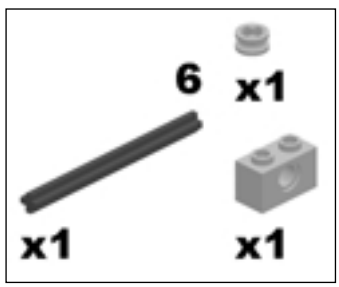
Switchbox Step 5

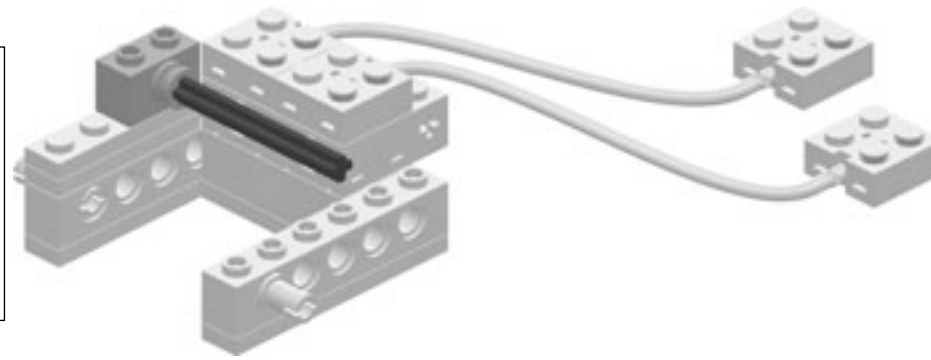




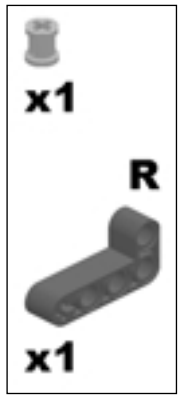
These electrical connectors are 5″ long.

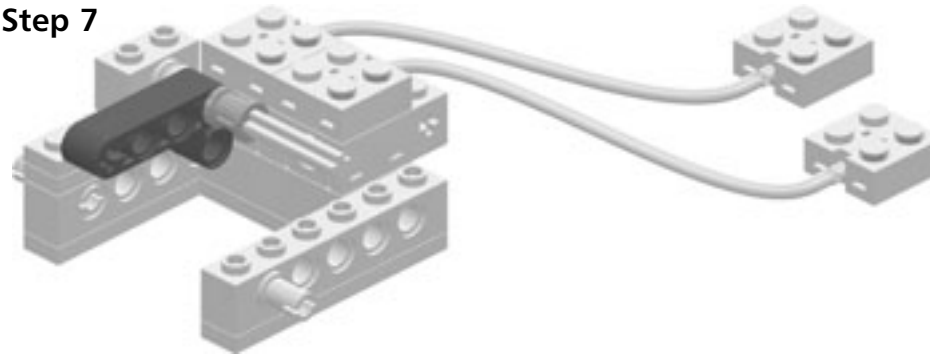
Switchbox Step 6



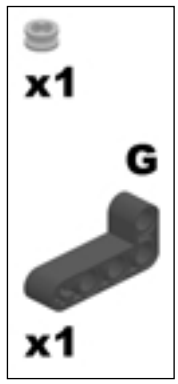


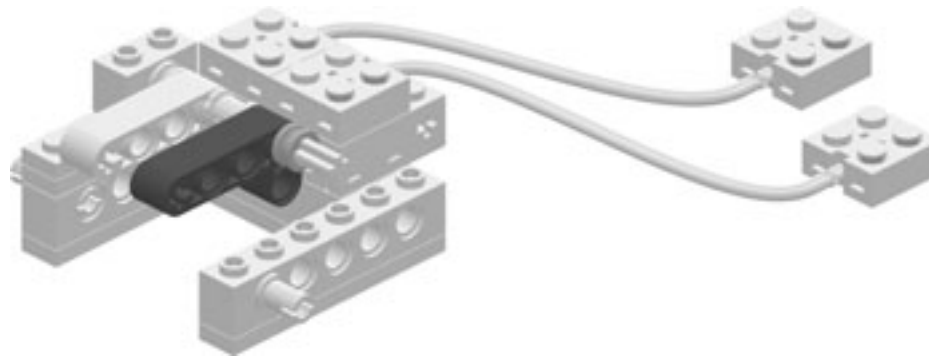
Switchbox Step 7





Switchbox Step 8





Bricks & Chips…

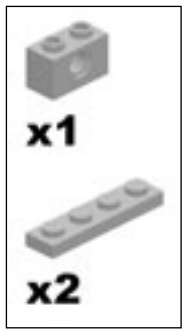
Lever Color Considerations

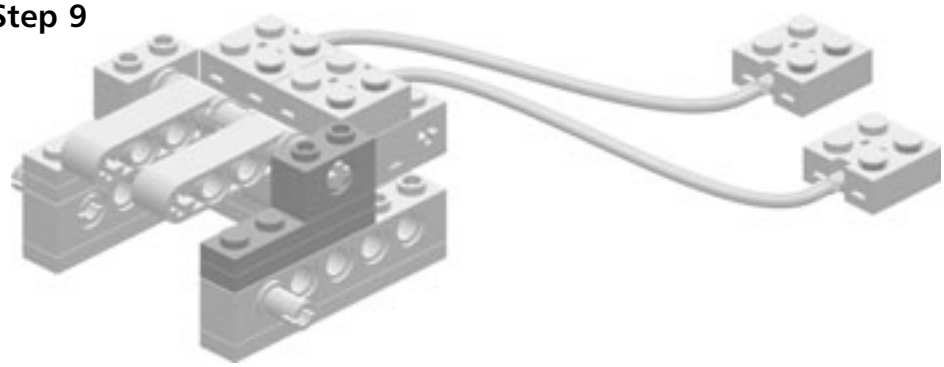
The colors of the levers were chosen to reflect their meaning:

- Red for the wrong bin
- Green for the right bin

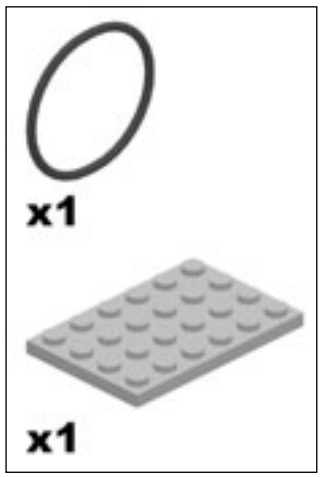
However, if you do not have access to these liftarms in red and green, you can choose any other color that you can easily associate with these meanings, or place two differently colored plates on top of the 1x6 plate to help you differentiate between the two levers.

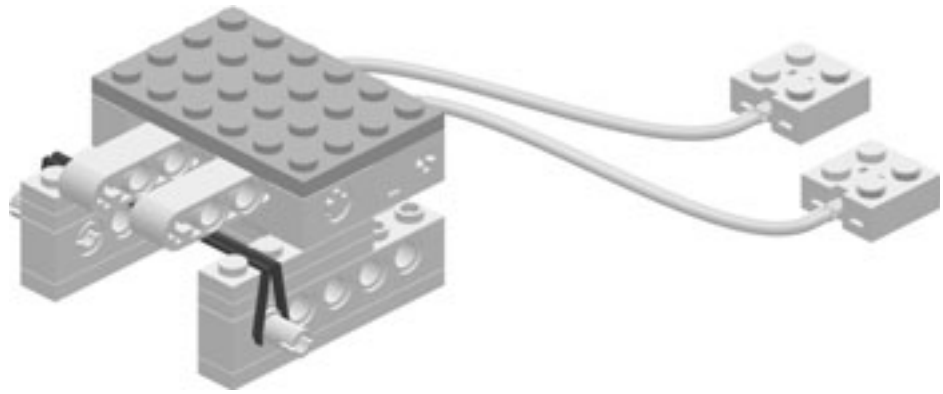
Switchbox Step 9



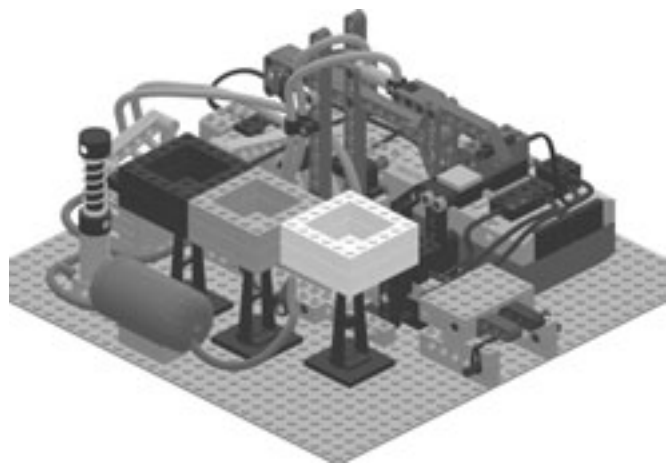


Switchbox Step 10



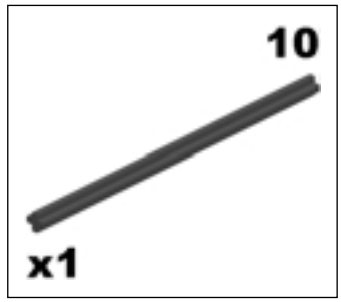


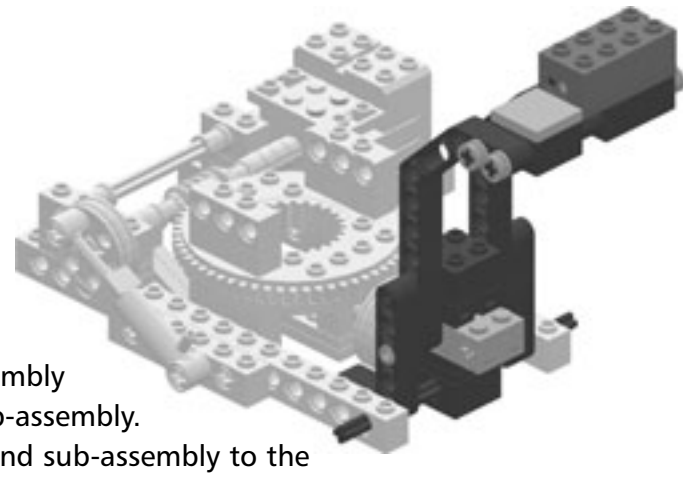
Putting It All Together



Now it is time to complete the assembly of the Learning Brick Sorter. We are going to place all of the various sub-assemblies on a large baseplate, as well as add a few additional parts, cables and pneumatic tubing.

Final Step 0





Locate the Body sub-assembly and the Pickup Stand sub-assembly.

Attach the Pickup Stand sub-assembly to the Body sub-assembly by sliding the #10 axle through the first hole of the gray 1x16 beams protruding from the Body sub-assembly and the bottom holes of the Pickup Stand sub-assembly.

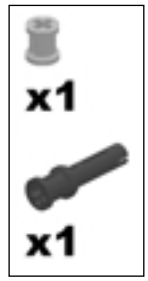
Bricks & Chips…

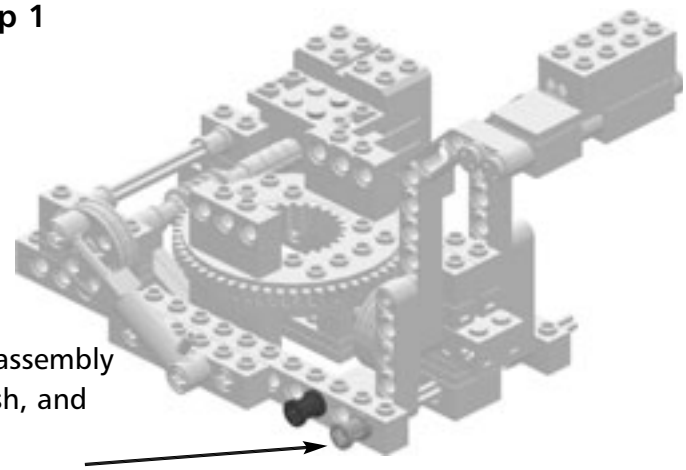
To Use Or Not To Use A Rotation Sensor?

The combination of the rotating pulleys with pins attached to the Body sub-assembly, paired with the touch sensors mounted on the Pickup Stand sub-assembly, creates a model similar in function to LEGO rotation sensor. You might wonder why I did not use the original LEGO rotation sensor? Unfortunately, the reliability of LEGO rotation sensor is directly related to the speed of the rotation. If the rotation sensor is operating at a low rotation, then the reliability of the rotation sensor is adversely affected. There are both hardware and software solutions for this; however, for the purpose of this project, I did not wish to force readers to modify their rotation sensors or use some specific alternative programming language. This building process solves this problem, and has the benefit of learning a technique to emulate a rotation sensor when that solution is not available for use.

If you want to know more about the previously mentioned problem that affects the rotation sensor, please refer to Philippe Hurbain's very well-documented Web page: www.philohome.com/sensors/legorot.htm

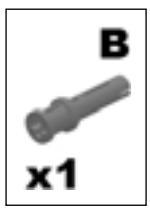
Final Step 1



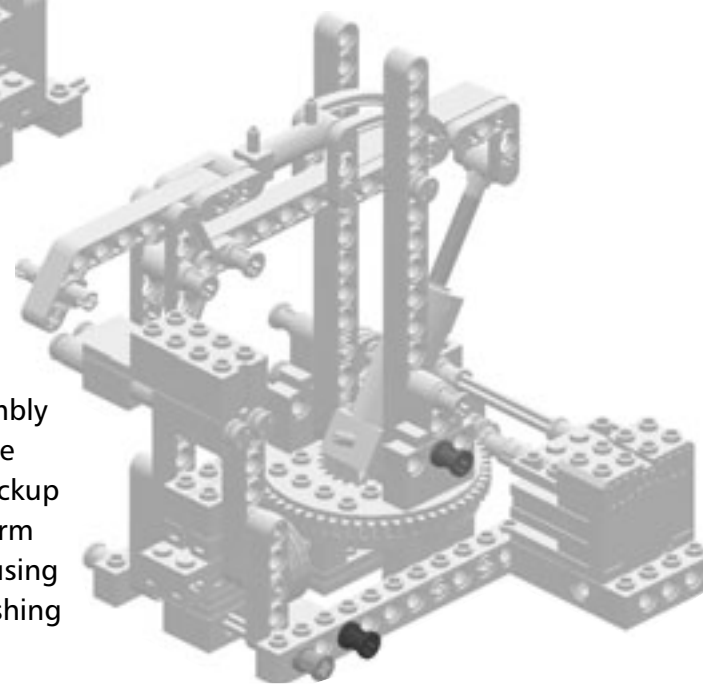


Secure the Pickup Stand sub-assembly with a long pin with stop bush, and lock the axle with a bushing.

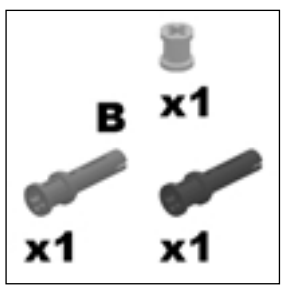
Final Step 2



Locate the previously built Arm sub-assembly and place it on top of the turntable. Lock the Arm sub-assembly into place using the pin with stop bushing as shown.

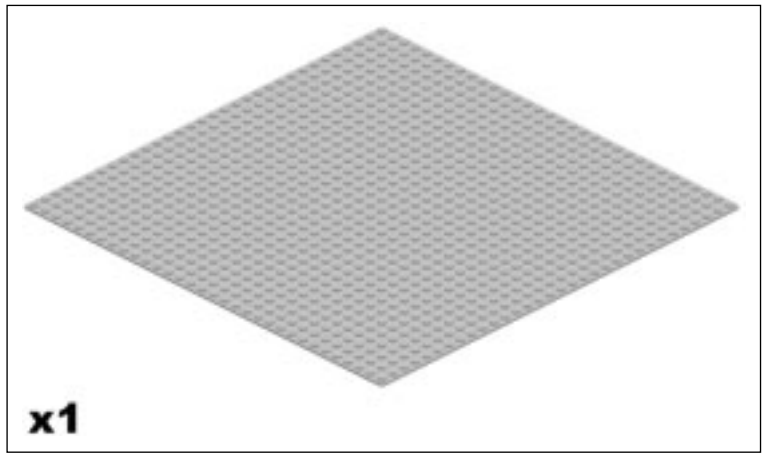


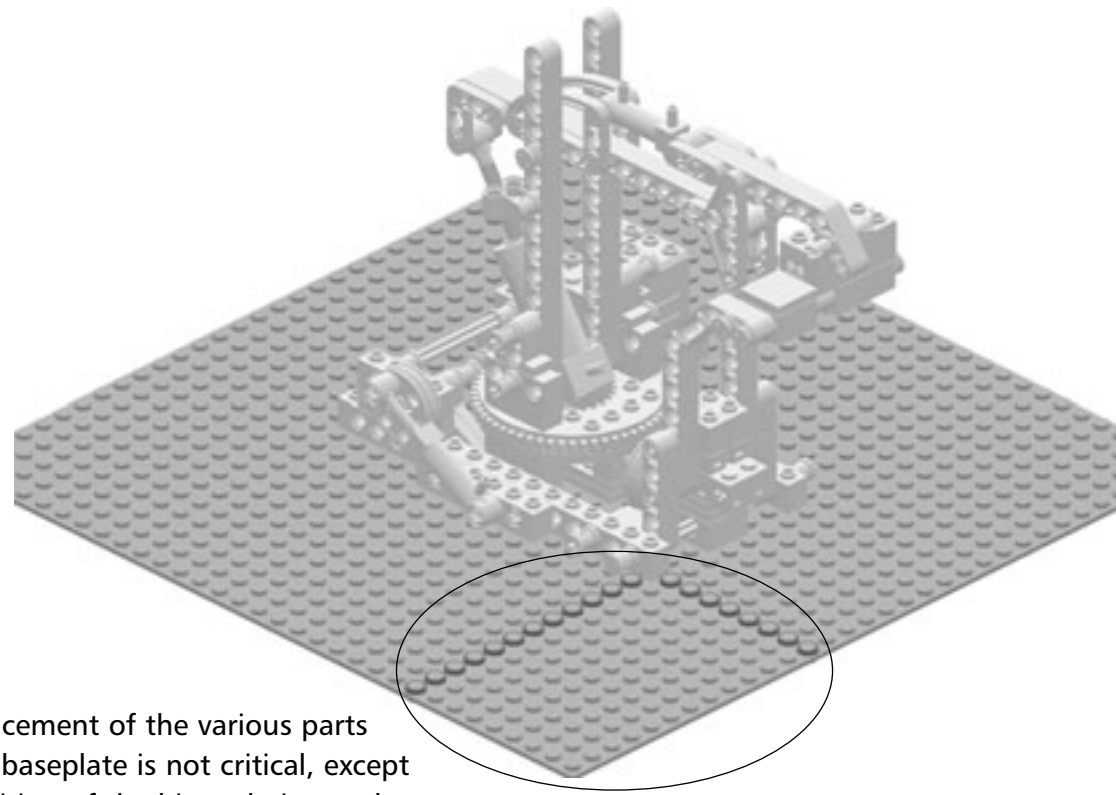
Final Step 3



Rotate the assembly and complete the locking of the Pickup Stand and the Arm sub-assemblies using the pins and bushing as shown.

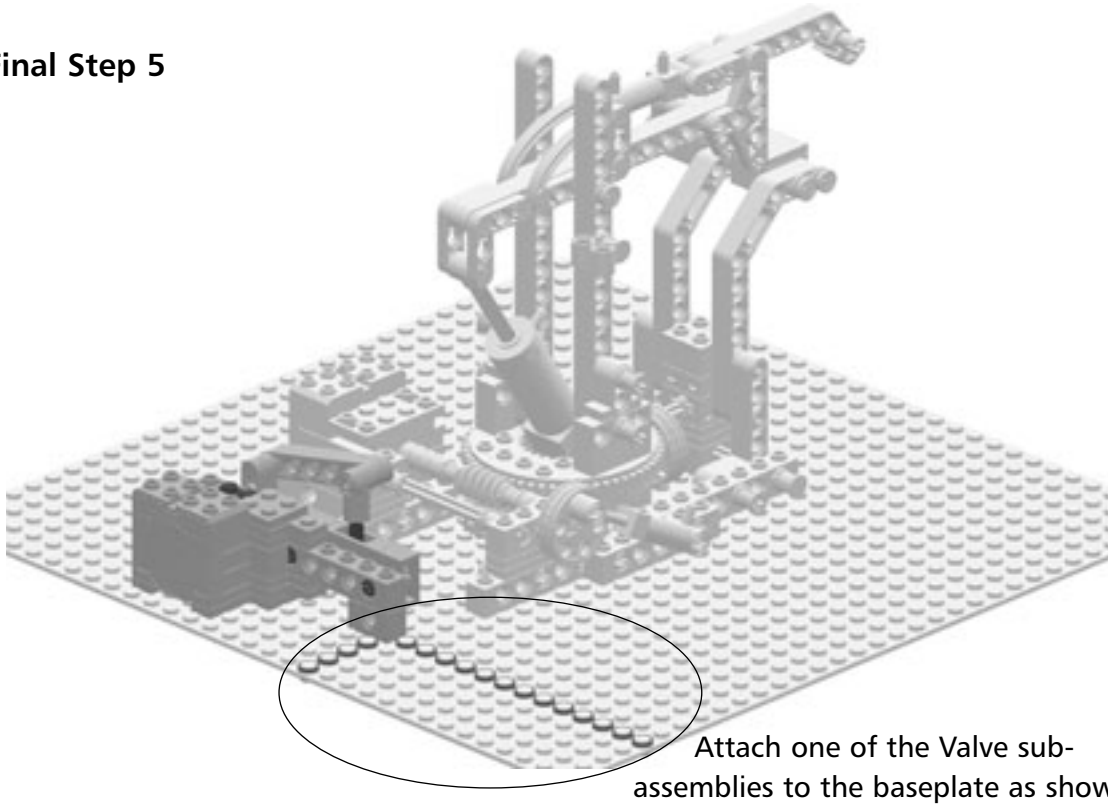
Final Step 4





The placement of the various parts on the baseplate is not critical, except for position of the bins relative to the body. To make positioning easier, I circled some the studs on the baseplate to mimic a system of orthogonal coordinates.

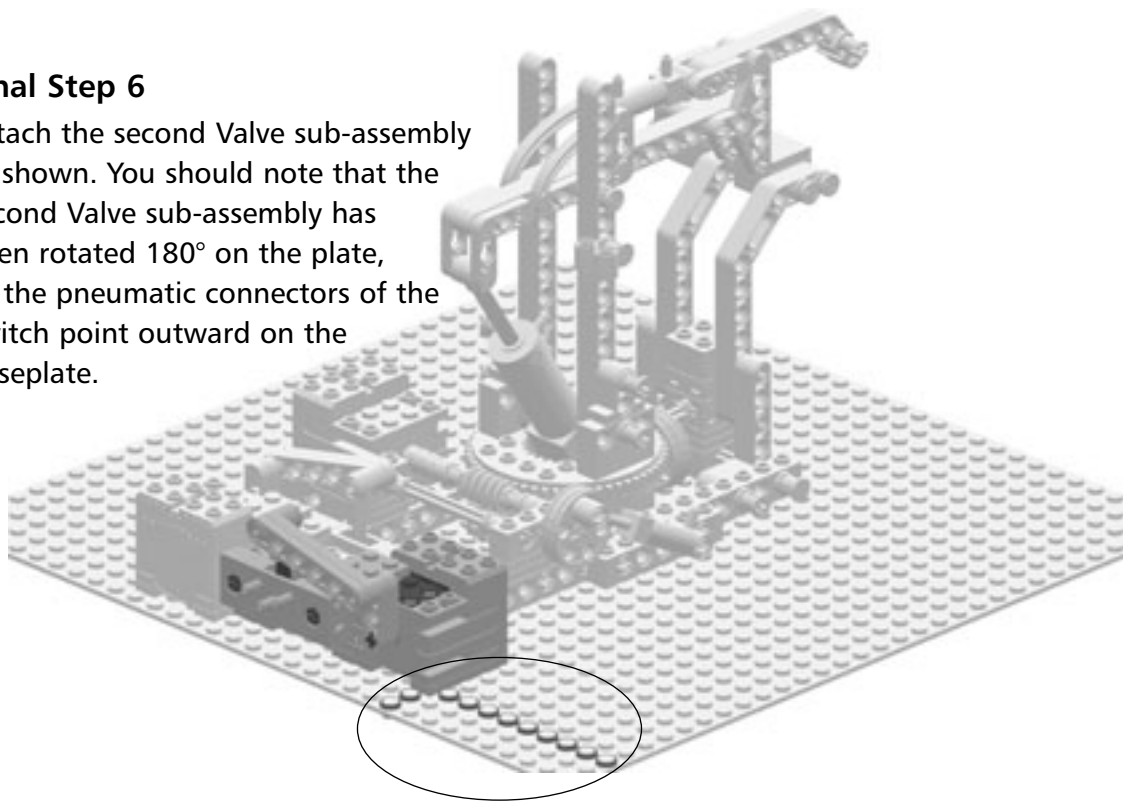
Final Step 5



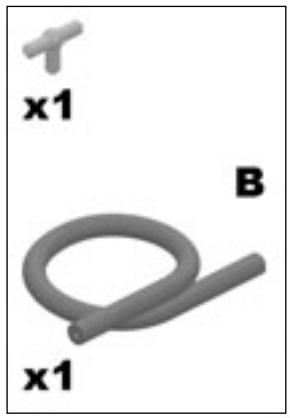
Attach one of the Valve sub-assemblies to the baseplate as shown.

Final Step 6

Attach the second Valve sub-assembly as shown. You should note that the second Valve sub-assembly has been rotated 180° on the plate, so the pneumatic connectors of the switch point outward on the baseplate.

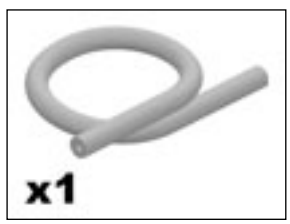


Final Step 7



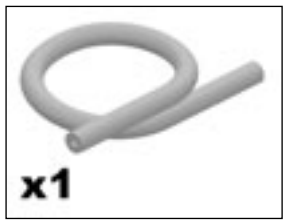
Using blue pneumatic tubing, attach the central connector of the pneumatic switch to a T-Connector as shown.

Final Step 8



Using a gray tube, connect the leftmost outlet of the Switchbox sub-assembly to the upper inlet of the large piston.

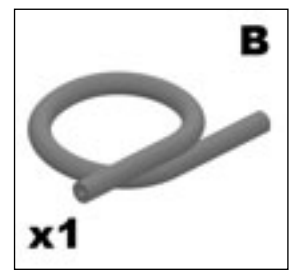
Final Step 9



Again, using a gray tube, connect the rightmost outlet of the Switchbox sub-assembly to the lowest inlet of the large piston.

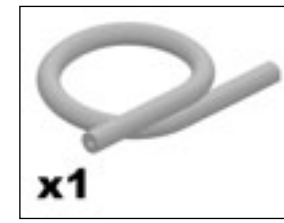
Rotate the model so that you are looking at the other side. Using a piece of blue pneumatic tubing, attach the central connector of the pneumatic switch to the T-Connector of **Final Step 7** as shown.

Final Step 10



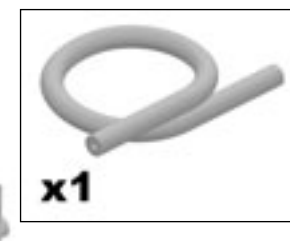
Connect the rightmost outlet of the Switchbox sub-assembly to the leftmost inlet of the small piston.

Final Step 11

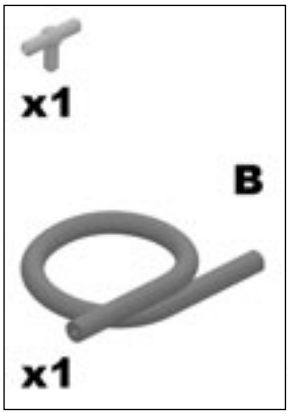


Connect the leftmost outlet of the Switchbox sub-assembly to the rightmost inlet of the small piston.

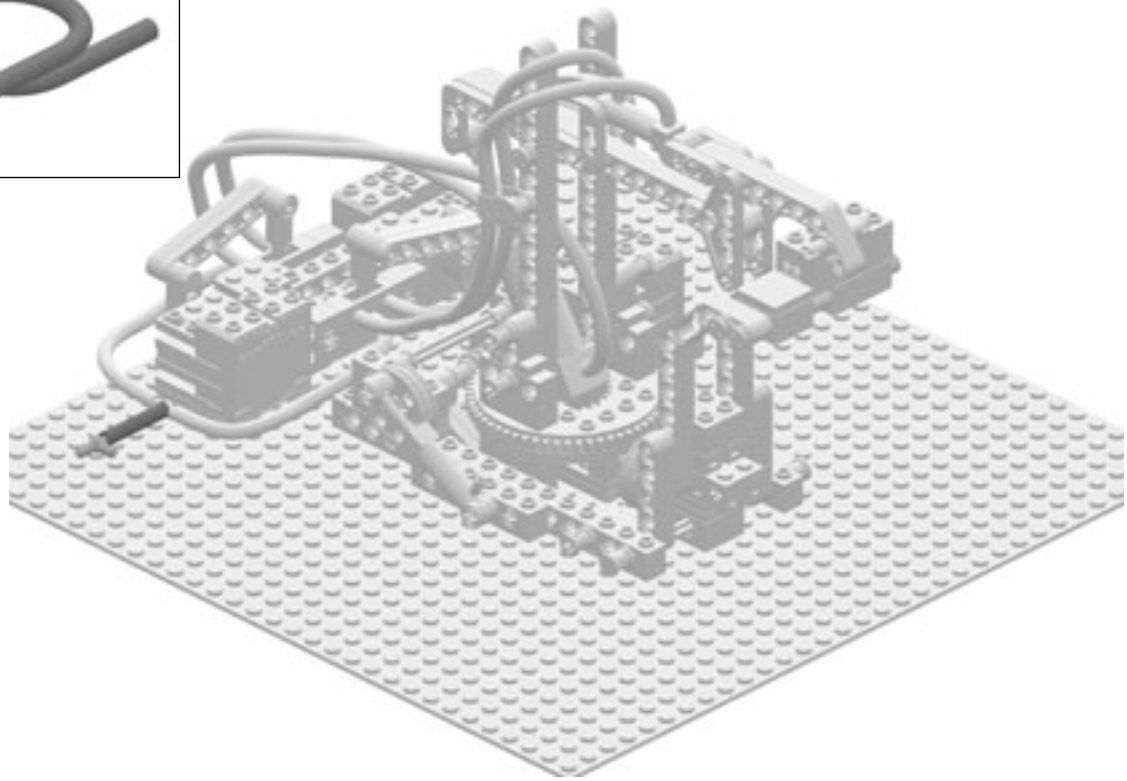
Final Step 12



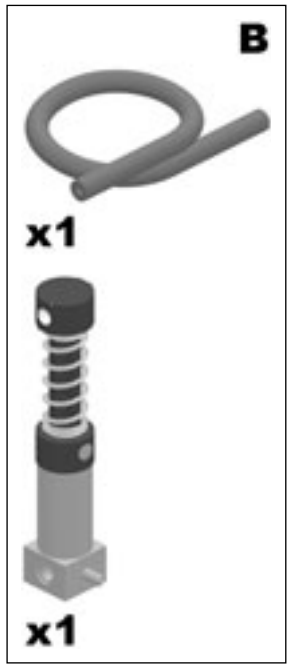
Final Step 13



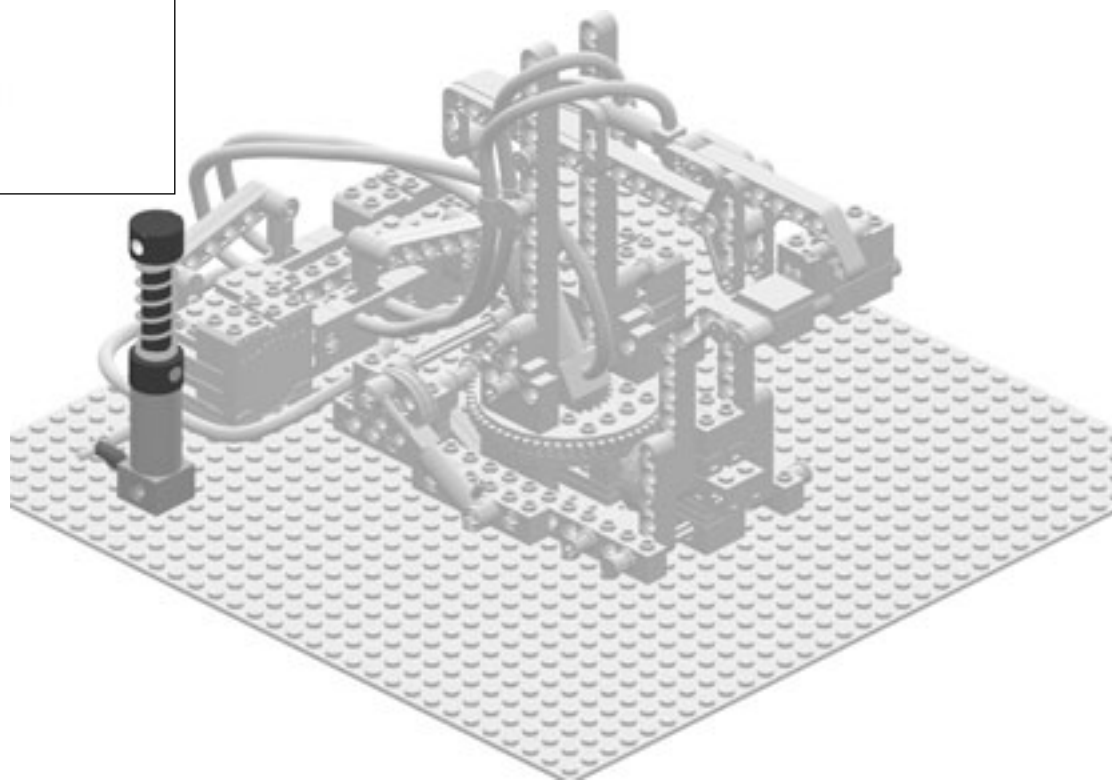
Using a short piece of blue pneumatic tubing, attach the T-Connector used in **Final Step 7** to a second T-Connector as shown.



Final Step 14

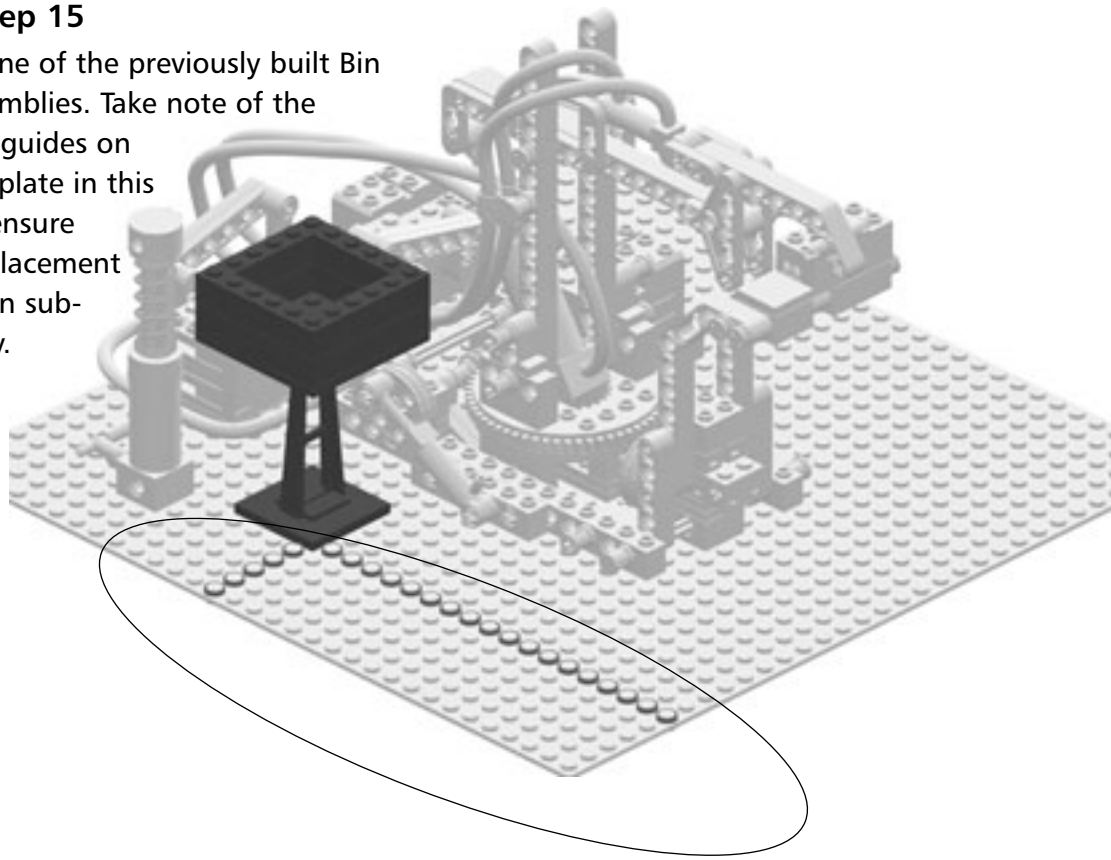


With another short piece of blue pneumatic tubing, attach the T-Connector from the previous step to the outlet of the manual pump.

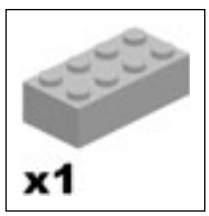


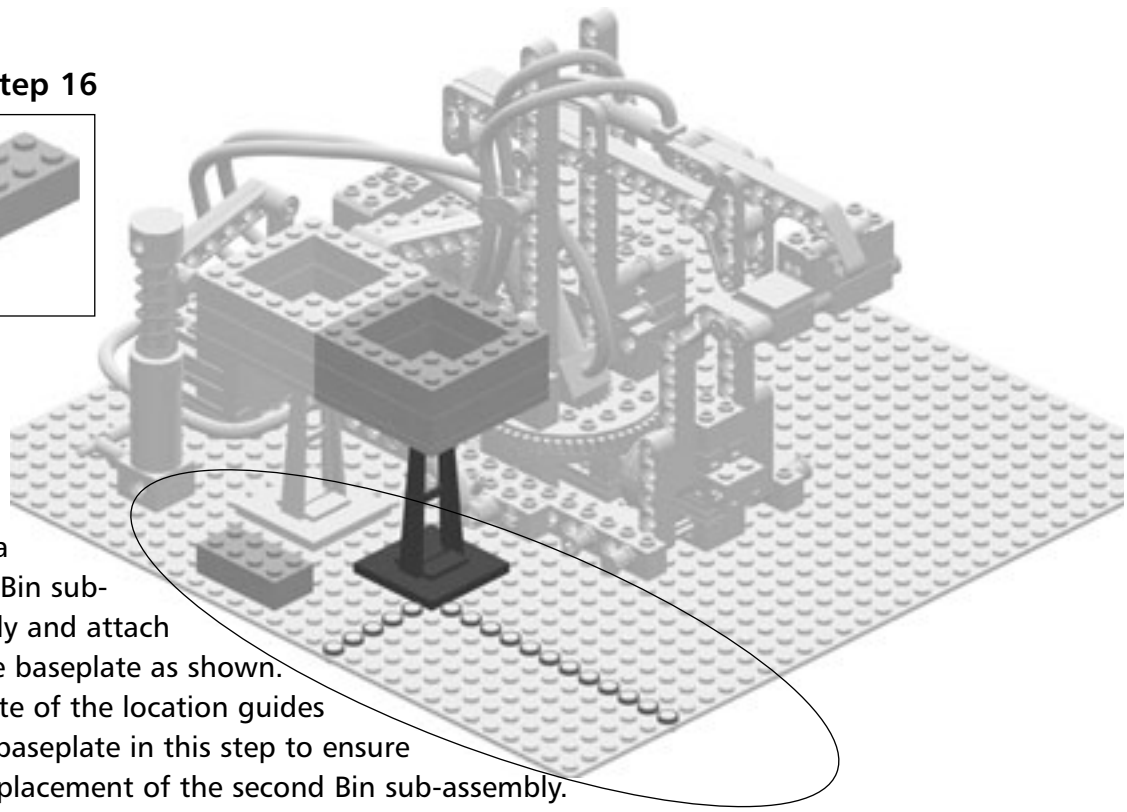
Final Step 15

Locate one of the previously built Bin sub-assemblies. Take note of the location guides on the baseplate in this step to ensure proper placement of the Bin sub-assembly.



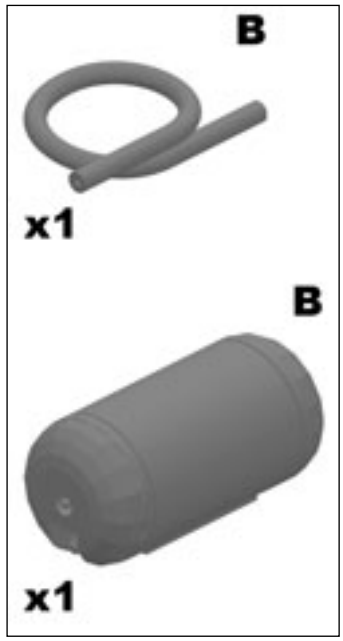
Final Step 16

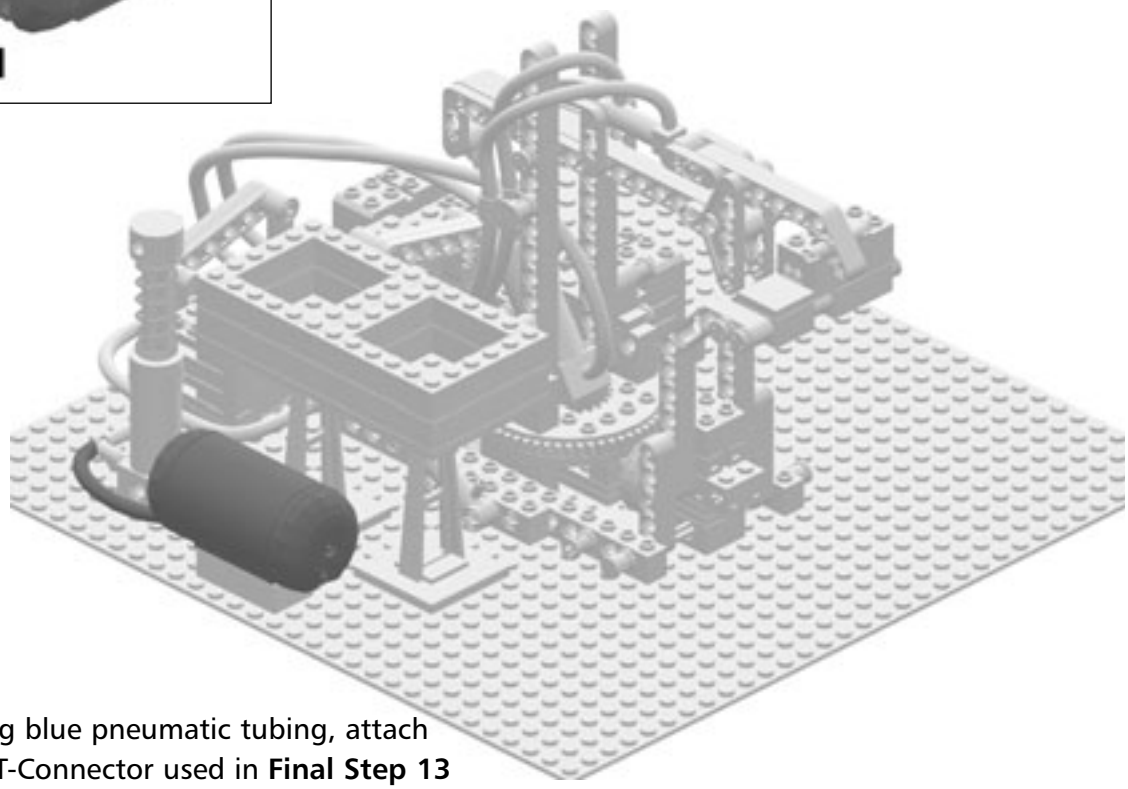




Locate a second Bin sub-assembly and attach it to the baseplate as shown. Take note of the location guides on the baseplate in this step to ensure proper placement of the second Bin sub-assembly.

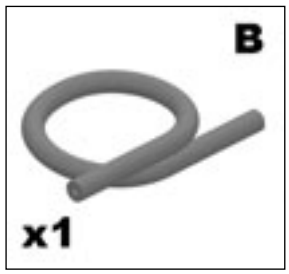
Final Step 17

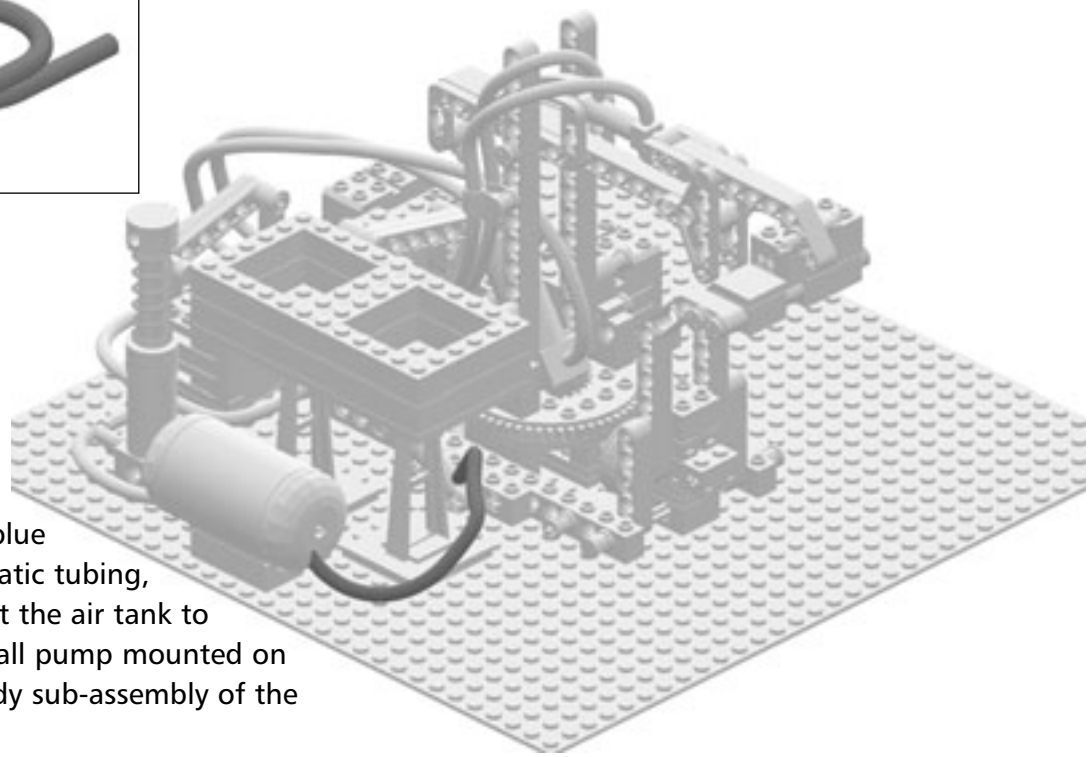




Using blue pneumatic tubing, attach the T-Connector used in **Final Step 13** to the air tank.

Final Step 18

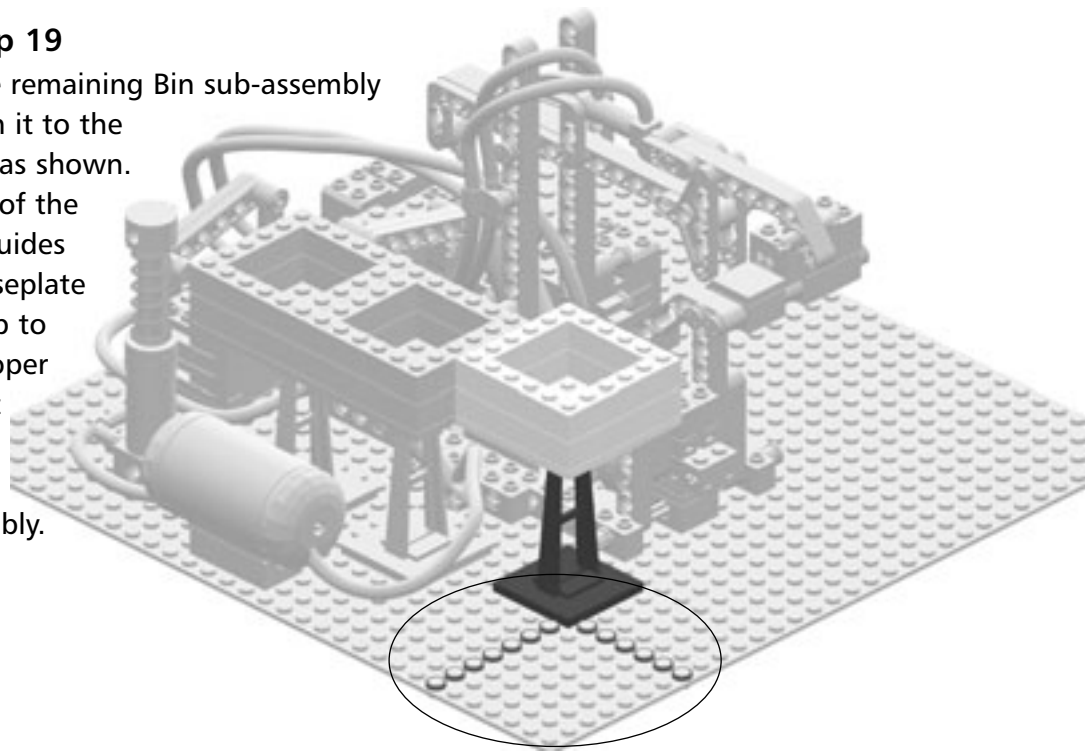




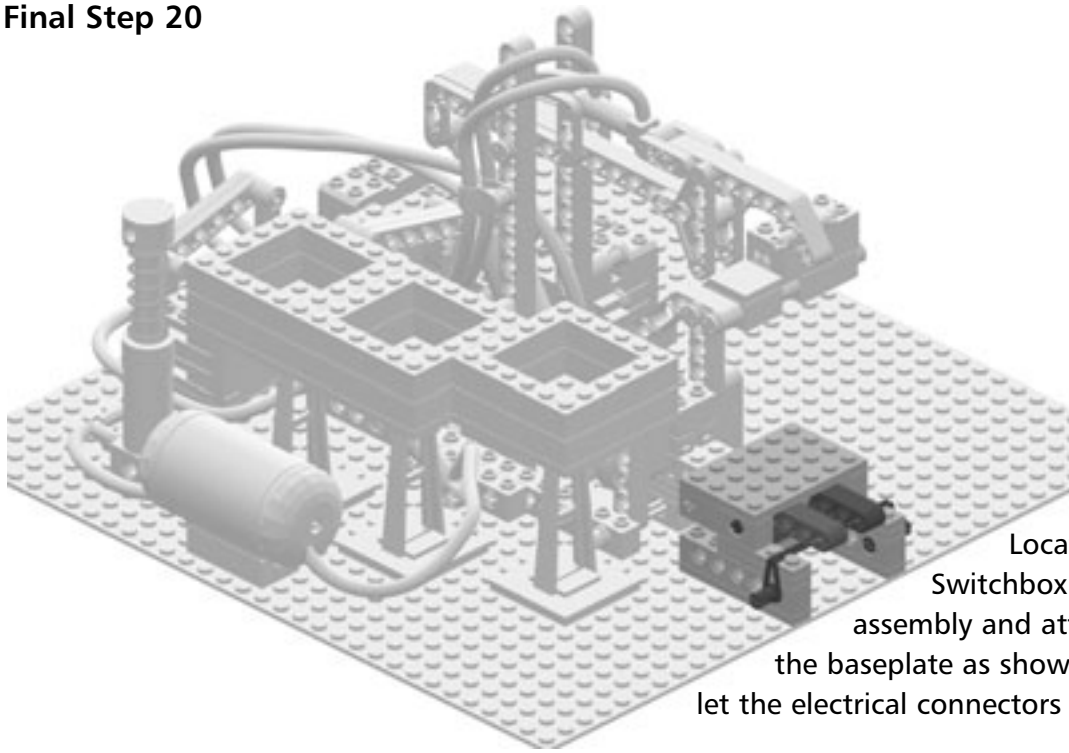
Using blue pneumatic tubing, connect the air tank to the small pump mounted on the Body sub-assembly of the robot.

Final Step 19

Locate the remaining Bin sub-assembly and attach it to the baseplate as shown. Take note of the location guides on the baseplate in this step to ensure proper placement of the final Bin sub-assembly.

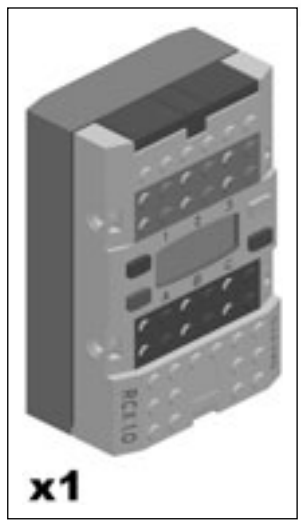


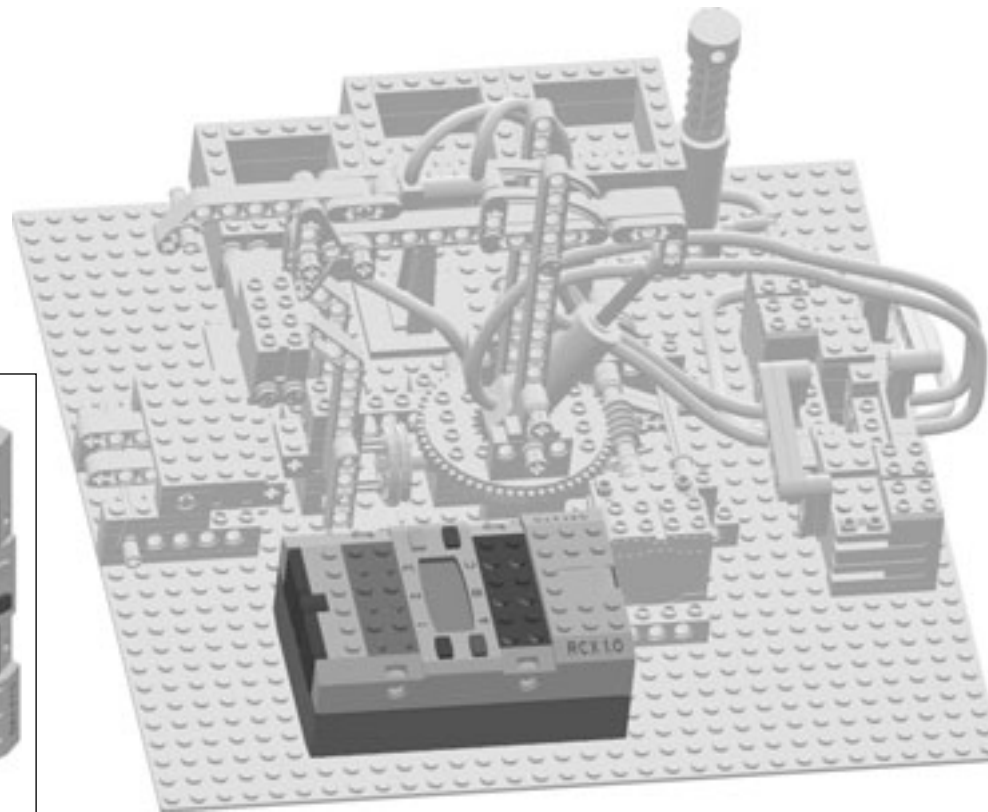
Final Step 20



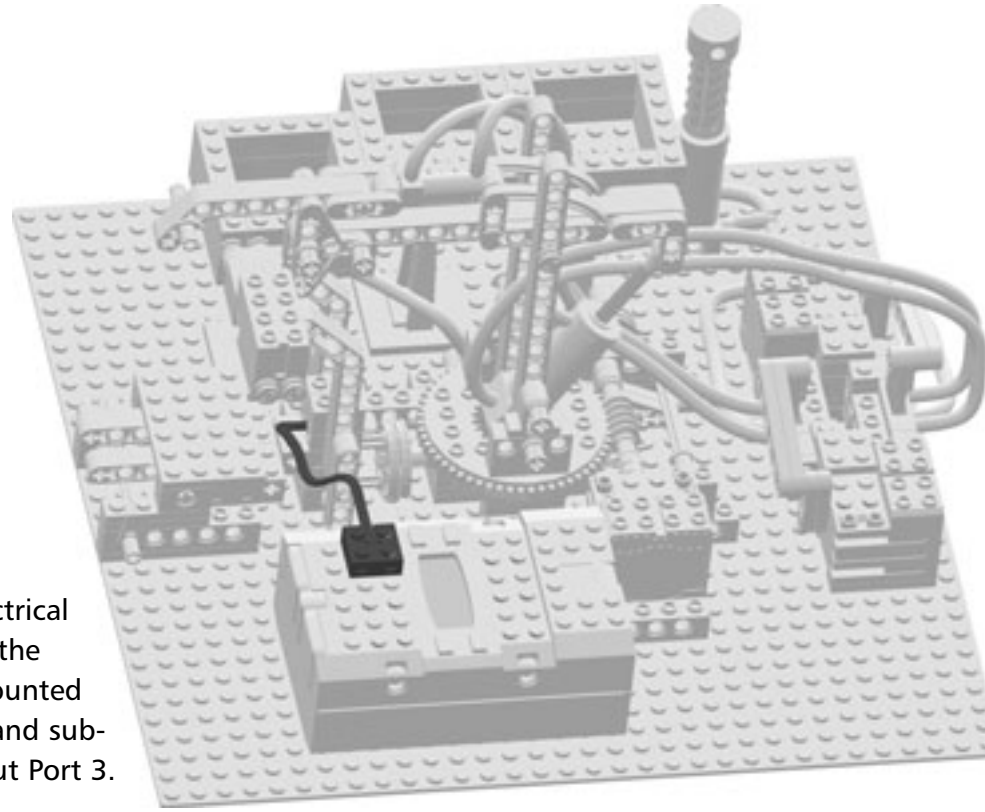
Locate the Switchbox sub-assembly and attach it to the baseplate as shown. For now, let the electrical connectors hang free.

Final Step 21



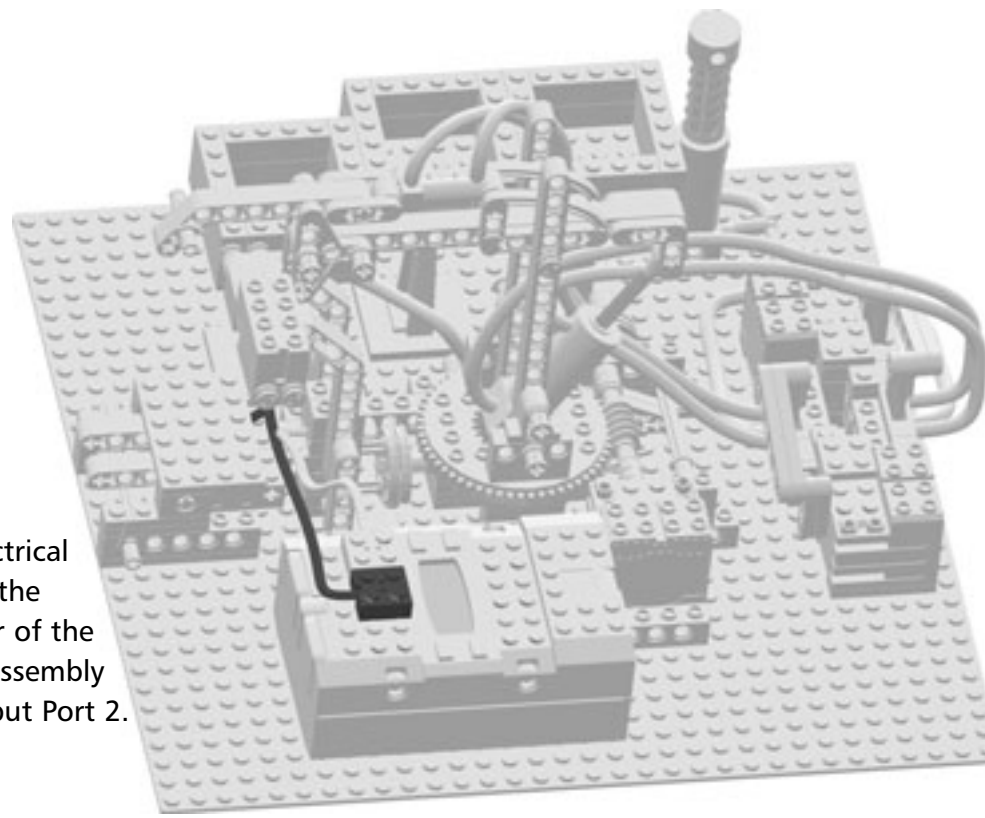


Mount the RCX on the baseplate as shown.



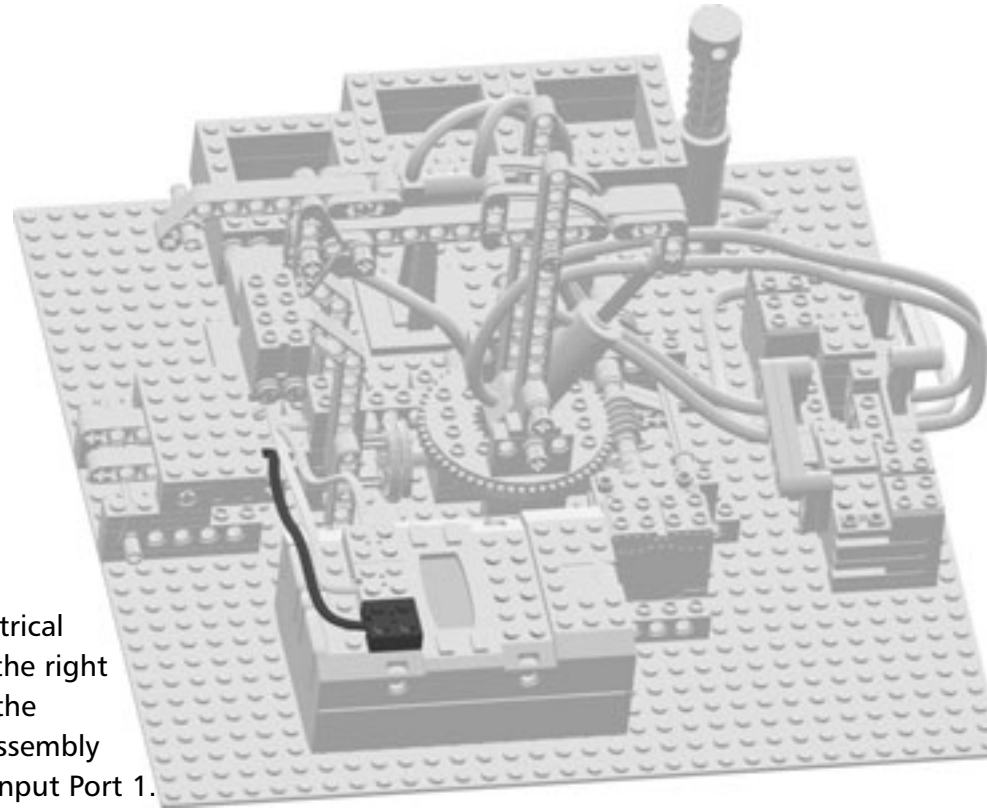
Final Step 22

Connect the electrical connector from the touch sensor mounted in the Pickup Stand sub-assembly to Input Port 3.



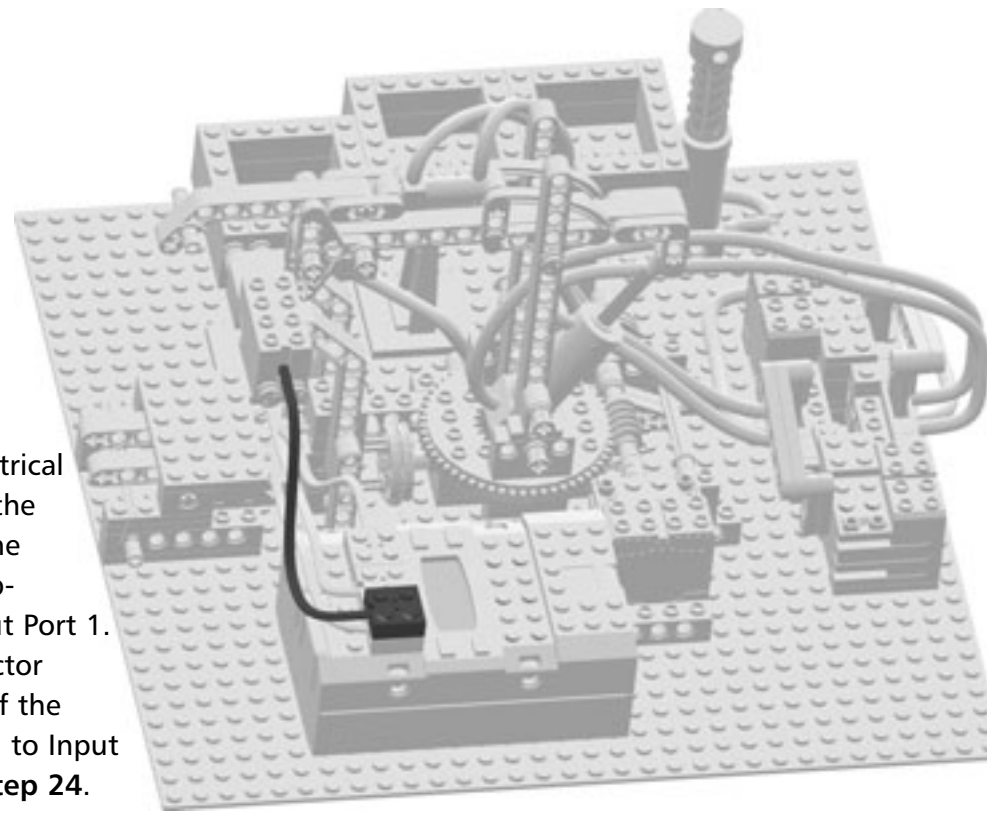
Final Step 23

Connect the electrical connector from the left touch sensor of the Switchbox sub-assembly (red lever) to Input Port 2.



Final Step 24

Connect the electrical connector from the right touch sensor of the Switchbox sub-assembly (green lever) to Input Port 1.



Final Step 25

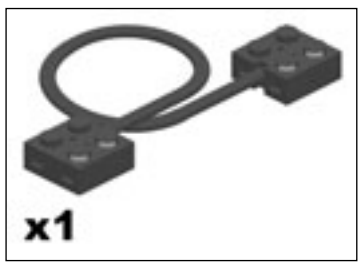
Connect the electrical connector from the light sensor of the Pickup Stand sub-assembly to Input Port 1. Place this connector directly on top of the connector added to Input Port 1 in **Final Step 24**.

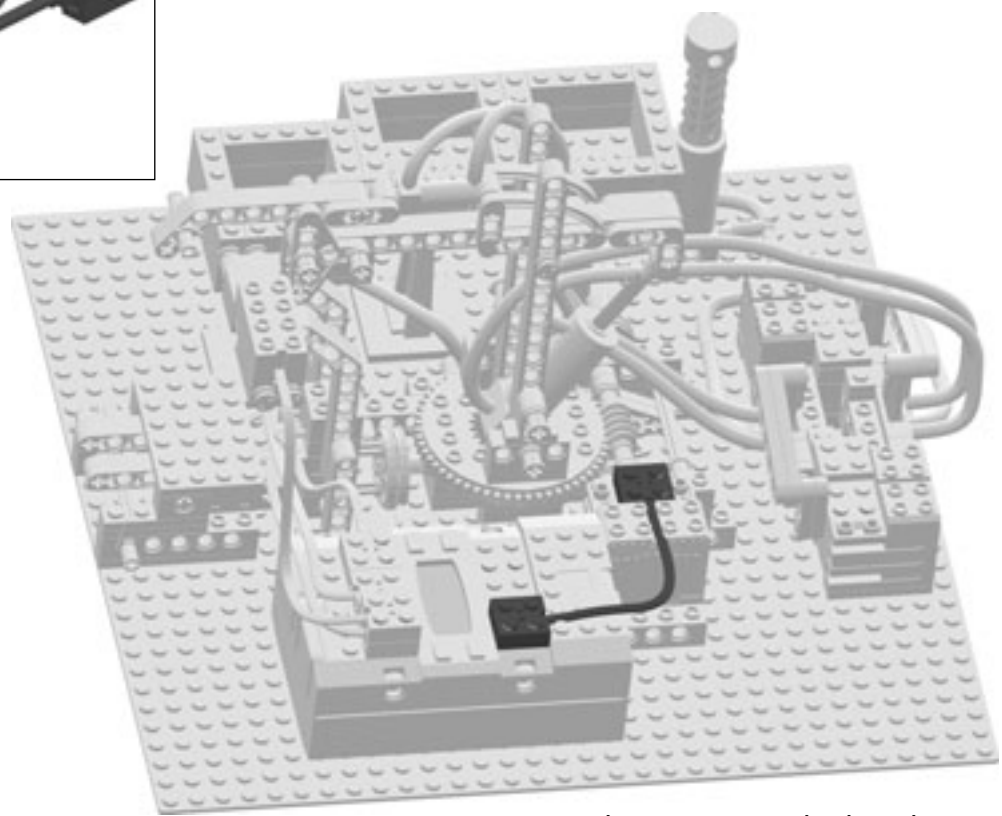
Bricks & Chips…

Multiple Connectors? Single Port?

How can the RCX distinguish between the readings coming from two sensors attached to the same port? This particular trick works because the light sensor returns readings that never reach the maximum value of its range. In fact, the light sensor returns a number representing a percentage which in normal conditions never read 100 percent. The touch sensor, configured as a light sensor, returns either 0 when released or 100 when pressed. Thus, in the program we can deduce that if the reading is 100 the touch sensor is pressed, otherwise the sensor is released and the reading is what returned by the light sensor. This trick is very useful to save input ports, provided that you don't need to read the light sensor when the touch sensor is pressed.

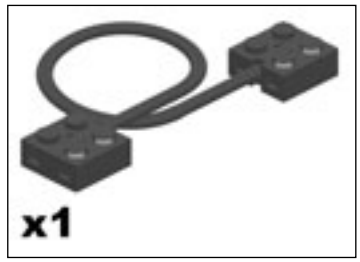
Final Step 26

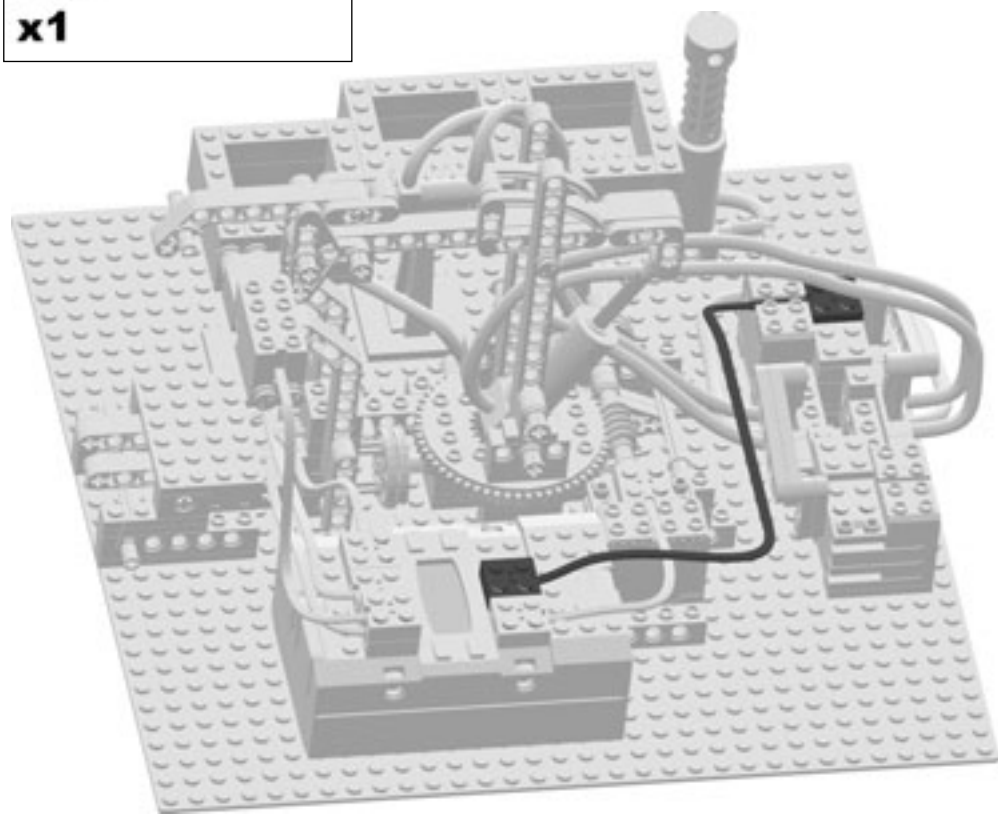




Connect the motor attached to the Body sub-assembly to Output Port A.

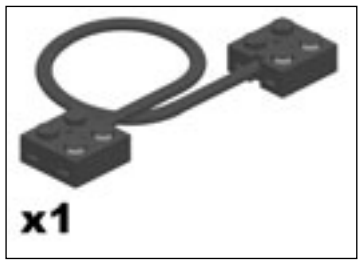
Final Step 27

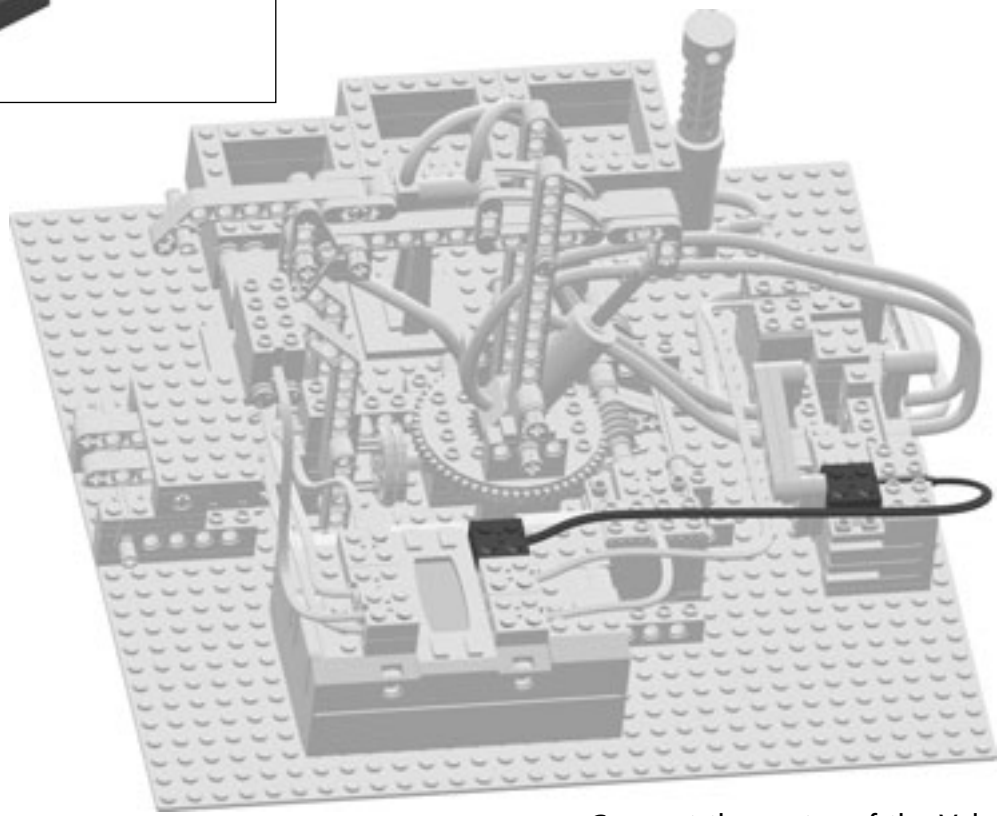




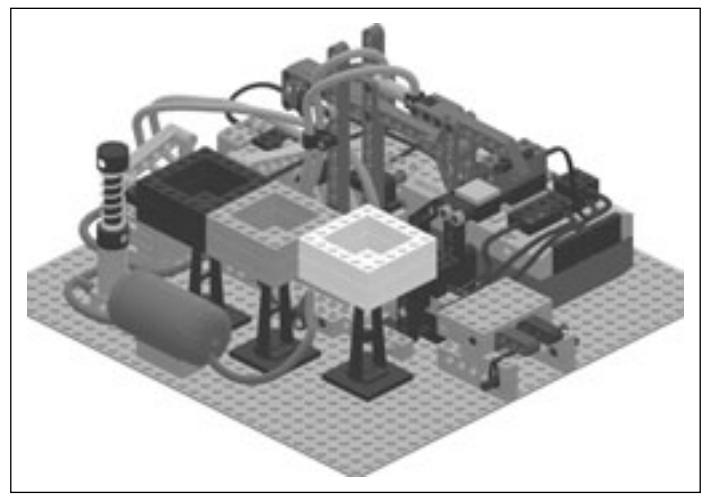
Connect the motor of the Valve sub-assembly that operates the large piston to Output Port B. If you don't have a connector of this length (it's not very common), either use a longer one or join two shorter connectors.

Final Step 28





Connect the motor of the Valve sub-assembly that operates the small piston to Output Port C.

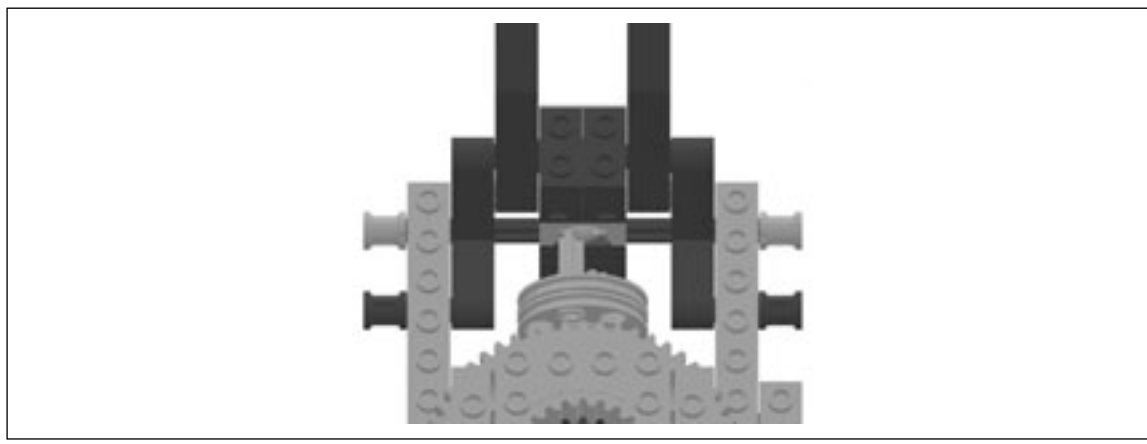


Tuning and Testing the Learning Brick Sorter

Now that the building phase is finished, you want to be sure that everything is connected properly and works as expected.

First, check to make sure the mechanical connections of the Brick Sorter are operational. Verify that the Arm sub-assembly is pointed toward the Pickup Stand sub-assembly. When the arm is lowered, the hand must align exactly with the yellow tile on the stand. If this is not the case, manually rotate the axle that operates the turntable (be sure that the RCX is switched off) until the arm is in the described position. At this point, the pin mounted on the pulley should close the touch sensor mounted on the Pickup Stand sub-assembly. This pin should be a bit off center of the sensor, as it appears in Figure 2.1.

Figure 2.1 Pin Alignment for the Activation of the Pickup Stand Touch Sensor



If your pulley is not in this position, you should locate the gear assembled in **Body Step 15**, pull the gear back a bit to disengage it from the worm gear (there's no need to remove the body from the baseplate), rotate the pulley until it reaches the position of Figure 2.1, and then replace the gear back against the worm gear.

Now we are going to test the connections of the motors and the pneumatic valves. Apply a small amount of pressure to the pneumatic circuit using the manual pump, and then switch the RCX on. Turning Motor A on in a reverse direction, the arm should rotate toward the bins. When the motor runs in a forward direction it should go back toward the Pickup Stand. If during the testing process you experience behavior opposite of what was just stated, check the cabling of the motor and correct it if necessary.

With a brief impulse to Motor B in the forward direction, the hand should open, while in reverse direction it should close. If this doesn't happen, check the cabling and the pneumatic tubes.

Similarly, a brief impulse to Motor C in the forward direction should lift the arm, while in reverse direction it should lower the arm. If this doesn't happen, check the cabling and the pneumatic tubes.

Bricks & Chips…

Using the Console Mode of the RCX

When you need to test your Learning Brick Sorter, as described in this section, you can take advantage of the LEGO remote control for the RCX. However, if you do not have a remote control, you can operate the motors using the *console mode* of the RCX.

1. Press the **View** button until the small arrow on the display points to the desired output port.
2. Keeping the **View** button pressed, press the **Run** button to power the motor forward or press the **Prgm** button to power the motor in reverse.

Programming the Learning Brick Sorter

Before entering the details of the program, I want to explain briefly the strategy we're going to use to make our Learning Brick Sorter able to learn. Let's start describing how the Learning Brick Sorter is expected to work:

1. The user places a 2x2 brick on the yellow tile of the Pickup Stand sub-assembly, and then presses any of the two levers of the switchbox to tell the Learning Brick Sorter to proceed. During this step, the Switchbox sub-assembly is not used to communicate any feedback, but only to inform the Learning Brick Sorter that there's a brick on the Pickup Stand sub-assembly ready to be sorted. It doesn't matter which lever the user presses.
2. The Learning Brick Sorter analyzes the color of the brick, looks up the information stored in its knowledge base, and decides which bin to drop it in.
3. The Learning Brick Sorter takes the brick and drops it in the chosen Bin sub-assembly, then waits for a feedback from the user.
4. The user sends his feedback to the Learning Brick Sorter using the Switchbox sub-assembly, pressing the green lever (if the bin was correct) or the red one (if the bin was wrong).
5. The Learning Brick Sorter updates its knowledge base with the new information, then moves the arm back to the pickup stand for a new cycle.

The key point is the design of the knowledge base obviously. We want our program to contain the minimal amount of predefined information, because our goal is to make

the Learning Brick Sorter adaptable to different situations. Thus, we won't store any associations between the colors and the bins, but we also won't store any values to define those colors. This way the Learning Brick Sorter will work with a set of any three colors, provided that their readings are well separated.

A "color" (actually the amount of light reflected by that color) can be stored into a single variable, but I find this approach a bit risky. If the user places the bricks at slightly different distances from the light sensor, the same color can result in different readings. For this reason, I chose to define colors using a range bounded by a lower and an upper limit. The Learning Brick Sorter has three bins, thus we need three ranges for three different colors:

```
#define RANGES 3
int min[RANGES];       // lower limits of the ranges
int max[RANGES];       // upper limits of the ranges
```

NOTE

You don't need to type any code, because you'll find the complete program included in the companion disk of this book.

To handle these ranges, we need an additional variable that tracks how many of them have been used:

```
int top_range;         // highest used range index
```

Initially *top_range* is set to −1, meaning that no range has been used yet. When a reading arrives, we scan the ranges from 0 to *top_range*, to see whether the value falls inside one of the defined ranges. If it does, this means the color belongs to that of the specified range, otherwise we increment *top_range* by one and define a new range.

```
light_value = SENSOR_1;

// search if the light value is contained in
// one of the already defined ranges
for(range=0;range<=top_range;range++) {
  if (light_value>=min[range] && light_value<=max[range]) break;
}

// if not, define a new range
if (range>top_range) {
  top_range++;
  // if all ranges have been used, beep for error and terminate program
  if (top_range>=RANGES) {
```

```
    PlaySound(SOUND_DOWN);
    StopAllTasks();
  }
  range=top_range;
  min[range]=light_value-RANGE_WIDTH;
  max[range]=light_value+RANGE_WIDTH;
}
```

There are a few things to notice about this portion of code: I used two new variables, *light_value* to store the reading of the light sensor, and *range* to point to the current range in the arrays. More important, I wrote a piece of code to handle the case where if the user attempts to use more than three different colors (the amount allowed by our program) the program plays the predefined *SOUND_DOWN* sound pattern and stops. If you use your Learning Brick Sorter properly, this portion of code will never execute, but foreseeing any possible problems or "unexpected" behavior of the user is a very good programming practice.

Finally, you might have noticed that I assigned the *min[range]* and *max[range]* variables with the value from the sensor plus and minus a constant called *RANGE_WIDTH*. This provides the tolerance for which we designed the ranges. Suppose *RANGE_WIDTH* is set to 1, when the Brick Sorter encounters a brick of a new color reading—for example—45, it will define a new range limited by the values 44 and 46. What happens if we set *RANGE_WIDTH* to higher values? We widen the bounds of the range, thus making the Brick Sorter more tolerant toward differences in the readings. For example, with *RANGE_WIDTH* set to 3, the brick of our previous case—whose reading was 45—will define a range limited by 42 and 48 instead of 44 and 46. The reason for want to increase the value of *RANGE_WIDTH* is to make the Learning Brick Sorter less sensitive to small fluctuations in the readings that might affect the behavior of the robot. The most typical cause for these fluctuations is the distance to which the brick is placed from the light sensor. In fact, the same brick placed at different distances results in different readings. On the other hand, if the values of *RANGE_WIDTH* are too high we limit the possibility of the Learning Brick Sorter being able to distinguish different colors up to the point that if ranges overlap, the Learning Brick Sorter can't work properly anymore. Low values of *RANGE_WIDTH* increase sensitivity, thus expanding the possibility for the Learning Brick Sorter to distinguish colors, at the price of making the placement of the brick more critical. I tested my Brick Sorter with the *RANGE_WIDTH* set to 2, but feel free to experiment with different values.

Now that we are able to define and distinguish colors, it's time to design the knowledge base. The knowledge base is the data structure that will contain the association between the colors and the bins. Let's think of the terms of the problem. Given a specific color and a specific bin, the Learning Brick Sorter may know it is the right combination, or that it is a wrong combination, or may know nothing about the two (when the program starts). This third case is the status of the knowledge base for any possible

combination of bins and colors. Thus, we need a logic based on three values, usually called a *trivalent logic*, and we define three constants with these meanings:

```
#define KB_YES 1
#define KB_NO -1
#define KB_UNKNOWN 0
```

Bricks & Chips…

Trivalent Logic

Jan Lukasiewicz, the Polish mathematician and philosopher, is considered the founder of the non-classical logical calculus. At the beginning of the 1900s, he demonstrated that trivalent logic has the same solid formal foundation that classical bivalent logic has. Quoting his work: "I maintain that there are propositions which are neither true nor false but indeterminate. All sentences about future facts which are not yet decided belong to this category." As many other great thinkers he was largely ahead of his time, and many years later multi-valued logic became a common tool of AI researchers.

For more information on Jan Lukasiewicz and his work, please refer to www.fmag.unict.it/PolPhil/Lukas/Lukas.html

Our knowledge base is simply a matrix that represents all the possible combinations of colors (ranges) and bins, where each cell contains what the program knows about that particular combination. Initially, all the cells are in *KB_UNKNOWN* status. The initial knowledge base can be seen in Table 2.1.

Table 2.1 Initial Knowledge Base

	Bin 1	Bin 2	Bin 3
Range 1	kb_unknown	kb_unknown	kb_unknown
Range 2	kb_unknown	kb_unknown	kb_unknown
Range 3	kb_unknown	kb_unknown	kb_unknown

Let's simulate what happens during a possible training session of the Brick Sorter. The user gives the Learning Brick Sorter a black brick. The Brick Sorter knows nothing about that color, but assigns the brick to the color "black" (actually its reading) to the first range, and places the brick in Bin 1. Bin 1 happens to be incorrect, so the user presses the red lever and passes a negative feedback to the Learning Brick Sorter. Now the program can update the knowledge base with this information. The updated knowledge base is shown in Table 2.2.

Table 2.2 Knowledge Base after the Black Brick (Range 1) is Placed in the Incorrect Bin

	Bin 1	Bin 2	Bin 3
Black	kb_no	kb_unknown	kb_unknown
Range 2	kb_unknown	kb_unknown	kb_unknown
Range 3	kb_unknown	kb_unknown	kb_unknown

If the user gives the Learning Brick Sorter another black brick, it would now try Bin 2, as it knows that the black brick doesn't go in the first one. But let's suppose the user chooses a white brick. The program defines a new range for it, and tries Bin 1 again. Now the bin is the right one, and the user returns a positive feedback. The updated knowledge base (Table 2.3) reflects the fact that the program knows where white goes, but knows also that Bin 1 cannot be used for the other colors:

Table 2.3 Updated Knowledge Base after Receiving "Correct" Feedback for Placing a White Brick (Range 2) in Bin 1

	Bin 1	Bin 2	Bin 3
Black	kb_no	kb_unknown	kb_unknown
White	kb_yes	kb_unknown	kb_unknown
Range 3	kb_no	kb_unknown	kb_unknown

Notice that the cells for Bin 2 and Bin 3 in the row assigned to white could have been assigned to *KB_NO* as well, but this has no effects of the algorithm, as every time a reading falls in the "white" range the *KB_YES* value forces the program to place the brick to Bin 1.

Now the user places a gray brick on the Pickup Standsub-assembly. The Learning Brick Sorter defines the third range, but it already knows it can't go to Bin 1, so it tries Bin 2, which happens to be the right one. This is how the knowledge base appears after the positive feedback:

Table 2.4 Updated Knowledge Base after Receiving "Correct" Feedback for Placing a Gray Brick (Range 3) in Bin 2

	Bin 1	Bin 2	Bin 3
Black	kb_no	kb_no	kb_unknown
White	kb_yes	kb_no	kb_unknown
Gray	kb_no	kb_yes	kb_unknown

At this point, after just three bricks, the robot is completely trained. In fact, even there's never been a black brick placed in Bin 3, but the program knows the black brick cannot go in Bins 1 and 2, thus it will make the right choice.

Having described the algorithm in plain words, it's time to write some code again. First, we have to solve a small problem: our knowledge base is organized as a bi-dimensional array, but NQC only supports arrays with a single dimension. The solution, however, is not overly complex. Imagine assigning each cell a progressive number, row-by-row, column-by-column, this way (notice that the bins and ranges in Table 2.5 is start from 0 instead of from 1. This trick would work with 1-based arrays as well, but it would require a bit more math.

Table 2.5 Converting the Knowledge Base into a Single Array Capable of Being Translated by NQC

	Bin 0	Bin 1	Bin 2
Range 0	Cell 0	Cell 1	Cell 2
Range 1	Cell 3	Cell 4	Cell 5
Range 2	Cell 6	Cell 7	Cell 8

We can now store the cells in a mono-dimensional array with nine cells, numbered from 0 to 8, and address them using a simple formula:

cell_address = range x 3 + bin

We already have a constant called *RANGES* that defines the number of rows of Table 2.5. Similarly, we can create a constant for the number of bins and make our program completely parametric. Now the declaration of the knowledge base becomes:

```
#define BINS 3
int kb[RANGES*BINS];  // the knowledge base
```

And we can address any cell with the formula:

cell_address = range x BINS + bin

Now we have all the elements to start coding the routine *Analyze(),* which reads the color of the brick, assigns it to a range, and decides which bin it must be placed into. This routine incorporates our previous work about defining ranges:

```
void Analyze()
{
  int light_value;
  // wait for coupled touch sensor released
  while (SENSOR_1>LITE_THRSHLD);
  light_value = SENSOR_1;
  // search if the light value is contained in
  // one of the already defined ranges
  for(range=0;range<=top_range;range++) {
    if (light_value>=min[range] && light_value<=max[range]) break;
```

```
  }

  // if not, define a new range
  if (range>top_range) {
    top_range++;
    // if all ranges have been used, beep for error
    // and terminate program
    if (top_range>=RANGES) {
      PlaySound(SOUND_DOWN);
      StopAllTasks();
    }
    range=top_range;
    min[range]=light_value-RANGE_WIDTH;
    max[range]=light_value+RANGE_WIDTH;
  }
  // look for a positive value in the knowledge base array
  for (bin=0;bin<BINS;bin++)
    if (kb[range*BINS+bin]==KB_YES) return;
  // otherwise, look for an "unknown" value in the knowledge base array
  for (bin=0;bin<BINS;bin++)
    if (kb[range*BINS+bin]==KB_UNKNOWN) return;
  // this point shouldn't ever be reached, unless the user provided some
  // inconsistent replies. If this happens, the program makes a random
  // choice.
  bin=Random(BINS-1);
}
```

You probably remember that one of the touch sensors of the switchbox shares the same Input Port as the light sensor. Because of this, we must be sure that the lever has been released before taking a reading from the light sensor. The purpose of the constant *LITE_THRSHLD* is to be able to distinguish when the touch sensor is closed.

```
#define LITE_THRSHLD 98
```

Under normal conditions, the light sensor never returns a value even close to 100. On the contrary, the touch sensor returns 100. This simple test against *LITE_THRSHLD* allows us to identify whether the touch sensor is pressed or not.

The routine continues with the code searching and assigning the ranges that define colors, and finally starts looking up the knowledge base. First, it scans the row corresponding to the given range for a positive value; that is. a cell assigned to the *KB_YES* value. If it finds one, the range has already an assigned bin and the routine can return the global variable *bin*, which contains the number of the bin to put the brick into.

If the program does not find a cell set to *KB_YES*, it scans the row again for the value *KB_UNKNOWN*. In fact, given that there are no positive values, and that negative values means, "don't use this bin," *KB_UNKNOWN* is our best second choice.

In theory, the routine should always find at least a *KB_UNKNOWN* value, unless the user has provided inconsistent replies. However, as I said before, good programming practice states we should try to anticipate any possible users' behavior. Thus, we have to handle this case too, so lacking any guiding rule, the routine simply makes a random choice. This makes our Learning Brick Sorter actually more adaptable. In fact, if the user changes his mind while using the Learning Brick Sorter—giving a negative feedback to a right choice—the user forces the Learning Brick Sorter to find a new solution to the problem: that is, a different bin for that particular color. After a few cycles, the Learning Brick Sorter will be trained to the new configuration.

Our next step is writing the routine that updates the knowledge base according to the feedback provided by the user.

```
void UpdateKnowledgeBase(int ok)
{
  int r;
  // the user said OK
  if (ok)
  {
    // this bin is the right one for the given range...
    kb[range*3+bin]=KB_YES;
    // ...and the wrong one for the other ranges
    for(r=0;r<RANGES;r++)
      if (r!=range)
        kb[r*BINS+bin]=KB_NO;
  }

  // the user said NO
  else
    // the bin is the wrong one
    kb[range*BINS+bin]=KB_NO;
}
```

This needs a few comments, as it reflects what I explained earlier when explaining how the knowledge base works. When the user confirms that the bin was the correct one, the program sets the corresponding cell of the knowledge base to *KB_YES* and the other cells related to the same bin to *KB_NO*, meaning that no other colors can go into that bin. When the user provides a negative feedback, the program simply sets the cell to *KB_NO*. Notice that the value *KB_UNKNOWN* is never assigned to a cell: Every feedback turns at least one unknown status into a yes or a no.

The remaining part of the program is rather trivial. Four simple routines provide control of the switch valves, operating the corresponding motors for a very short time defined by the constant *SWITCH_TIME*:

```
#define SWITCH_TIME 20
void OpenHand()
{
  OnFwd(OUT_C);
  Wait(SWITCH_TIME);
  Off(OUT_C);
}
void CloseHand()
{
  OnRev(OUT_C);
  Wait(SWITCH_TIME);
  Off(OUT_C);
}
void LiftArm()
{
  OnFwd(OUT_B);
  Wait(SWITCH_TIME);
  Off(OUT_B);
}
void LowerArm()
{
  OnRev(OUT_B);
  Wait(SWITCH_TIME);
  Off(OUT_B);
}
```

Controlling the rotation of the turntable requires a bit more code. The global variable *pos* maintain the information about which position the arm is currently located.

```
int pos;  // current arm position: -1=pick up station, 0=bin0, 1=bin1...
```

The routine *Move()* rotates the arm until the required destination is reached, at the same time updating the variable *pos* with the new position. Each step toward a new position is counted waiting for the touch sensor to be opened and then closed again:

```
void Move(int dest)
{
  int dir;

  // if arm is already at proper position, exit
```

```
  if (dest == pos) return;

  // set direction according to the difference between
  // the current position and the required destination
  if (dest > pos)
  {
    dir = 1;
    OnRev(OUT_A);
  }
  else
  {
    dir = -1;
    OnFwd(OUT_A);
  }

  // move until destination is reached
  while (dest != pos)
  {
    while (SENSOR_3 == 1);  // wait for sensor open
    while (SENSOR_3 == 0);  // wait for sensor closed again
    pos += dir;
  }
  Off(OUT_A);
}
```

Another simple routine that configures the sensors and initializes the global variables and the knowledge base:

```
void SystemSetup()
{
  int i;
  SetSensor(SENSOR_1, SENSOR_LIGHT);
  SetSensor(SENSOR_2, SENSOR_TOUCH);
  SetSensor(SENSOR_3, SENSOR_TOUCH);
  pos = -1;         // initial position is pick-up station
  top_range = -1;  // means no range is yet defined

  // clear the knowledge base
  for (i=0;i<BINS*RANGES;i++) kb[i]=KB_UNKNOWN;
}
```

Finally, we are ready to write the main task, which reflects the sequence of steps we defined at the beginning of this section. Together with all the constants and global variables we have used up to this point, summarized in a single place, here is that code:

```
#define BINS 3
#define RANGES 3
#define SWITCH_TIME 20
#define LITE_THRSHLD 98
#define OP_DELAY 75
#define RANGE_WIDTH 2
#define KB_YES 1
#define KB_NO -1
#define KB_UNKNOWN 0
int range;              // index of the selected range (0 based)
int top_range;          // highest used range index
int bin;                // index of the selected bin (0 based)
int pos;                // current arm position: -1=pick up station, 0=bin0,
1=bin1...
int min[RANGES];        // lower limits of the ranges
int max[RANGES];        // upper limits of the ranges
int kb[RANGES*BINS];    // the knowledge base
task main()
{
  SystemSetup();
  while (true)
  {
    // wait for button pressed
    while (SENSOR_1 < LITE_THRSHLD && SENSOR_2 == 0);
    PlaySound(SOUND_CLICK);

    // analyze knowledge base (set the range & bin variable)
    Analyze();

    // take the brick
    LowerArm();
    Wait(OP_DELAY);
    CloseHand();
    Wait(OP_DELAY);
    LiftArm();

    // go to bin and drop it
```

```
    Move(bin);
    OpenHand();

    // wait for user input
    while (SENSOR_1 <LITE_THRSHLD && SENSOR_2 == 0);

    // update knowledge base
    if (SENSOR_2 == 1)  // wrong
    {
      PlaySound(SOUND_LOW_BEEP);
      Wait(OP_DELAY);
      UpdateKnowledgeBase(false);
    }
    else  // right
    {
      PlaySound(SOUND_DOUBLE_BEEP);
      Wait(OP_DELAY);
      UpdateKnowledgeBase(true);
    };
    // go back to pick up station
    Move(-1);
  }
}
```

The code requires only a few explanations. The constant *OP_DELAY* is used as a short delay during operations: it's especially important to give pneumatic pistons the time necessary to complete their stroke before performing further operations. In reading the touch sensors in the switchbox, one has to consider that *SENSOR_1* is connected to the light sensor too. Thus, every test must compare the reading against *LITE_THRSHLD*, instead of simply checking a 1 or 0 as happens for *SENSOR_2*. Finally, when providing feedback through the levers in the switchbox, the user receives different sounds for the positive or negative response.

Training and Using the Learning Brick Sorter

Before switching the Learning Brick Sorter on, check that the hand is open and the arm is lifted and is positioned over the pickup stand.

Next operate the manual pump: the small pump connected to the motor that rotates the turntable will maintain the pressure while the Learning Brick Sorter is in action,. Initially you have to provide pressure in the pneumatic circuit. As the airflow provided by the motorized pump is slightly inferior to the average air consumption of the Learning

Brick Sorter, after several sorting cycles, you will notice the arm and hand moving slowly, and you will need to pump again to restore a good pressure level.

Take some 2x2 bricks in three different colors. White, gray and black are a perfect choice, but the Learning Brick Sorter will work with many other combinations of colors. You can test the readings of the light sensor for each color simply using the console mode of the RCX. Press the **View** button until the arrow in the display points toward Input Port 1; now the display shows the reading of the light sensor. Any choice of colors whose average reading are separated by at least the value of *RANGE_WIDTH* will work.

Switch the RCX on, place a brick on the yellow tile of the pickup stand, just next the light sensor, and press any lever on the switchbox. The Learning Brick Sorter will take the brick and will drop it in one of the bins. Now provide a Positive (green lever) or Negative (red lever) feedback using the switchbox, and the arm will come back in its rest position above the pickup stand, ready for a new cycle.

When you are finished and want to switch the RCX off, remember to do it only after you provided the last feedback. This ensures the arm is in its rest position and you don't have to reset it manually when you want to use the Learning Brick Sorter again.

Further Suggestions

My simple Learning Brick Sorter leaves the door open for many possible additions and improvements. On the hardware side, you can improve the management of the air supply providing an external compressor controlled by an automatic pressure switch. On the same theme, another option is replacing one of the valve switches with a version able to drive also the pump, as described later in this book in Masterpiece 4 "PneumADDic II" by Kevin Clague.

A nice addition is an automatic feeder for the bricks. Something as simple as an inclined plane can do the trick. If you need some further inspiration, I suggest you search the Internet: there's plenty of well-designed brick sortering robots, and many of them feature automatic feeding mechanisms of some kind.

You can try to add an additional bin to make the Learning Brick Sorter able to sort four different colors. You can move the valve switches to a different location on the baseplate to get some space for the fourth bin after the third one. The software needs very small changes: just set the *BINS* and *RANGES* constants to 4 instead of 2. Colors need to be chosen very carefully: white, light gray, dark gray and black are a good set. Reduce *RANGE_WIDTH* to 1 if necessary.

On the software side, there are many possible variations as well. For example, you can modify your Learning Brick Sorter able to recognize when the training phase is finished and the knowledge base is complete, so as not to ask for any feedback anymore. Or, in the opposite direction, you can make re-training more efficient: my program is already able to handle the situation where the Learning Brick Sorter is completely trained but the user wants to change the destination bin of each color. However, it's not particularly

efficient in this task. Can you improve the *UpdateKnowledgeBase()* routine to take maximum advantage from the feedback provided by the user?

NOTE

When updating the knowledge base with a new data that contradicts the information previously stored, some cells need to be assigned to *KB_UNKNOWN*.

Another possible extension to the software is making it able to place more than a single color in the same bin. For example, it could handle the case where you have five different colors and three bins, and two pairs of colors share a common bin. This requires that the knowledge base have more rows than columns, that is more ranges than bins. However, changing the constants *BINS* and *RANGES* is not enough: you have to modify the *UpdateKnowledgeBase()* routine, because the assumption that when a color is assigned to a bin the others cannot use that bin is no more valid.

If you are a teacher and use this project for some classroom work, you can assign some theoretical analysis of the algorithm. For example, you can ask them to discover what's the shortest sequence that makes the knowledge base complete in the luckiest case, and what's the longest sequence in the worst case.

Summary

As you discovered in the chapter, the apparently trivial brick sorting application offered many arguments for discussion and investigation.

A few tricks we used can prove useful for other robots and other situations. The position tracking system of my Learning Brick Sorter — built around a touch sensor and a pulley — not only avoids the missed-counts problem which affect the rotation sensors, but shows a general solution to emulate a rotation sensor when you don't have one. We have also seen a practical application of the well-known trick of connecting a light sensor and a touch sensor together on the same input port, thus expanding the possibilities of our RCX.

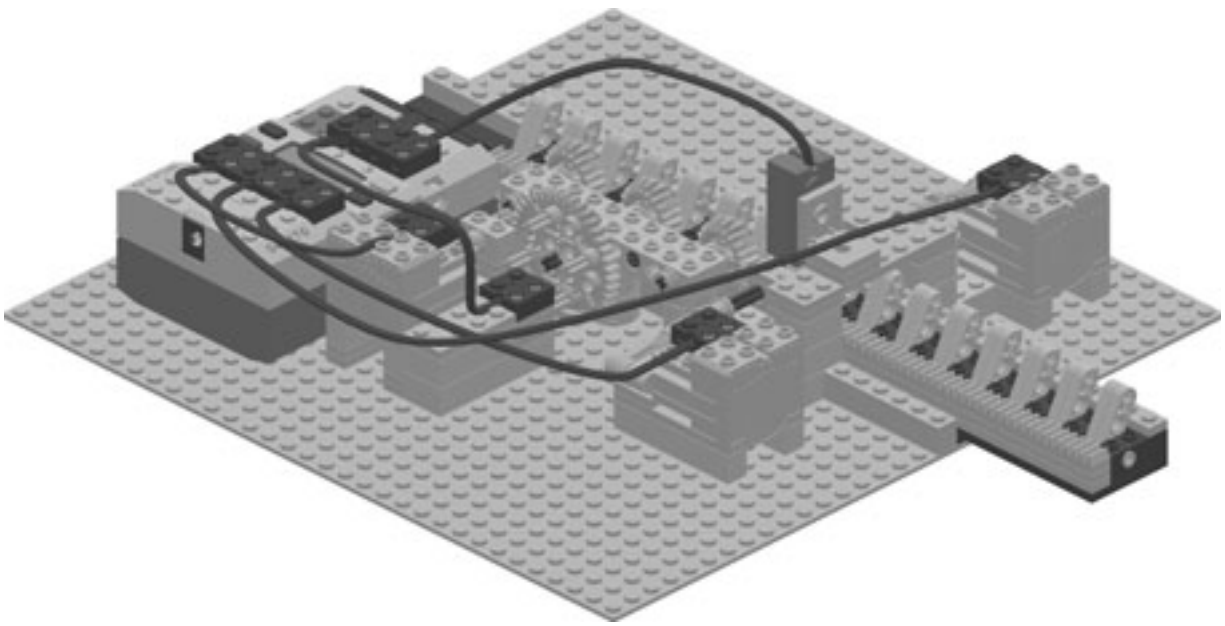
While developing the software, we have explored a few programming techniques. For example, how to emulate a bi-dimensional array over a mono-dimensional one, and more generally, how to write a solid program that can handle any possible situation and is easy to understand and maintain. We also had a quick look at trivalent logic, something different from the more common Boolean logic based on the true/false values, but a powerful tool when facing problems that contain some degree of indetermination.

More importantly, my Learning Brick Sorter lets you enter the world of machines that learn. Although the software of the Learning Brick Sorter is very simple compared to most actual AI programs, it nevertheless has merit and provides some insight that doesn't require the reader to have strong technical competence in this field already.

Masterpiece 3

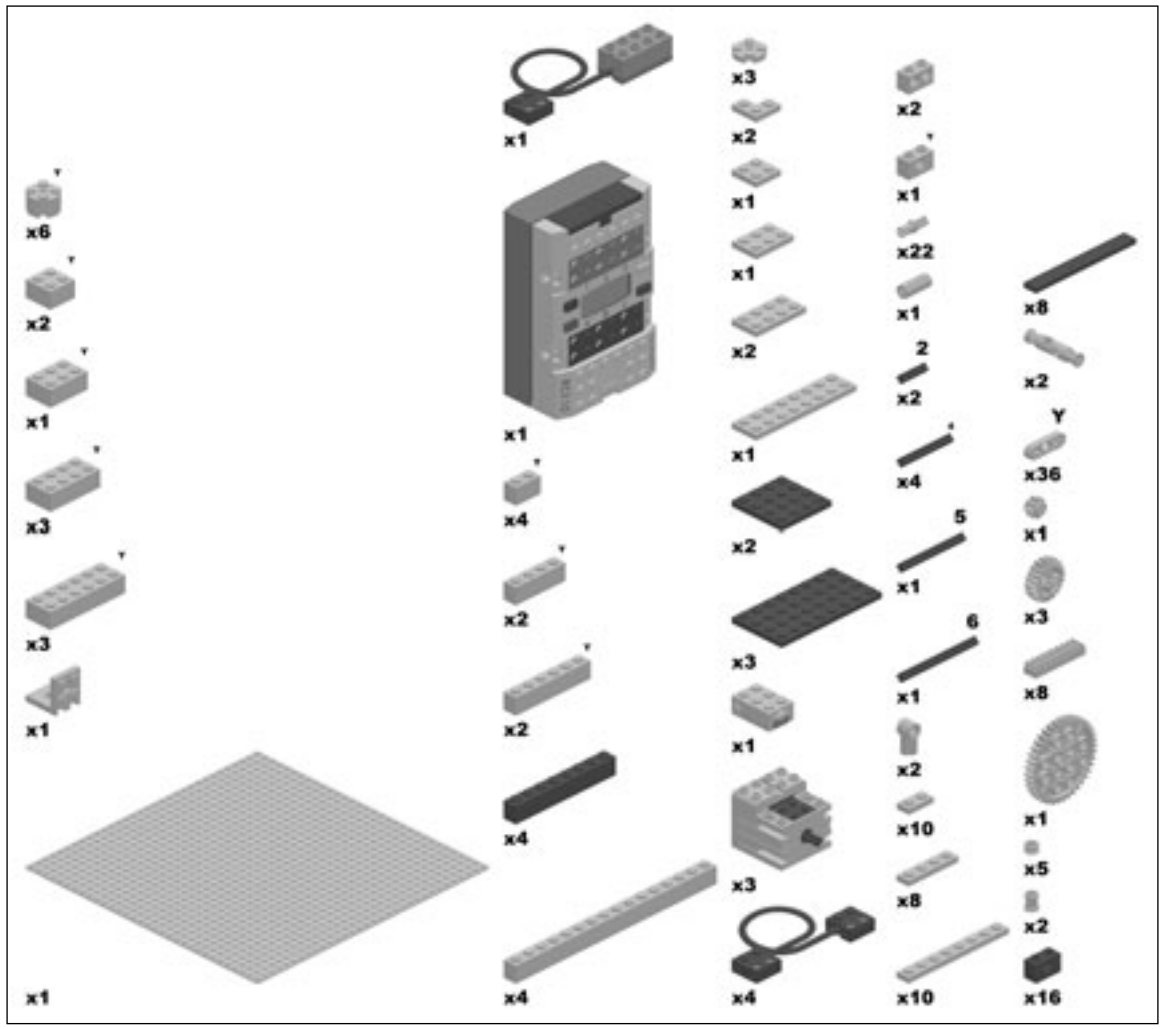
The LEGO Turing Machine

Giulio Ferrari



Bill of Materials

These are the parts necessary to complete of the LEGO Turing Machine as shown.



NOTE

If you're looking for information on a particular Lego piece or set, try www.peeron.com. This is a completely free, very complete, fan-created Web site. It contains listings and inventories, with illustrations and cross-references, plus shows which sets include any given part.

Bricks & Chips…

Additional Needs for the LEGO Turing Machine

This model doesn't need many extra parts. You will need a LEGO RIS 2.0 kit, which contains two motors, the light sensor, and the touch sensor, plus one additional motor. For the tape, though, you'll need some additional pieces: the 6632 TECHNIC 1x3 Liftarm and the 3743 TECHNIC 1x4 Gear Rack. Quantities depend on the length of the tape you want to build. In this project, we will build a tape with 16 cells that is 32 wide, so we need 32 liftarms (plus four for the switch mechanisms) and eight gear racks. For the base, we'll also need eight 4162 1x8 tiles. If you're missing some of the pieces, you can probably use something else or try a different solution: for example, you can substitute the eight tiles with a greater amount of shorter tiles, or leave a space between one tile and another.

In the LEGO Turing machine, we will use a combination of yellow bricks, gray plates, and black parts. You can change nearly every color however you prefer, but remember that you should use a light color for the liftarms and a dark color for the bottom of the tape to ensure that the light sensor reads the symbol on the cell correctly and that there is a significant difference in the light sensor's reading value.

Introduction

In Masterpiece 2, "The Learning Brick Sorter," we entered the world of Artificial Intelligence (AI), and saw how a robot can actually learn from its behavior. In this chapter, we will see another notable example of this concept, going back to its very beginning in the 1930s, with the birth of a very special device: the Turing Machine.

The Turing Machine was a very primitive sort of computer, the simplest possible machine that could perform computation, and an ancestor of any modern computer that we use everyday. Any digital computer or electronic device that is multi-purposed and uses the concept of algorithm is, in fact, a reincarnation of such a device. You can even consider the Turing Machine a grandparent of your RCX!

We will explain the concepts behind the invention, programming, and behavior of the LEGO Turing Machine. By following the instructions and directions in the chapter, you'll soon be able to build your personal Turing Machine purely out of LEGO pieces. We'll see a couple of examples of how to program the LEGO Turing Machine using NQC, and at the end of the chapter, we'll see hints and tips on how to expand your creation.

The History of the Turing Machine

Let's go back in time through some history of this fascinating device. The following historical sections are not essential for building the model, but are still extremely useful to explain the basic ideas that lead to the design and development of the Turing Machine,

and to understand the particular time period in which it was invented. Later, we will examine the machine's original architecture and the way its creator, the English mathematician Alan Turing, designed it to study computational problems.

In 1931, an Austrian mathematician named Kurt Godel demonstrated the revolutionary theory of the incompleteness of mathematics. This theory truly changed the science of mathematics. Mathematicians usually write theories (proposals for what might be) and then try to prove that the theories are correct using previously proven theories. The attempt to write theories down and prove that they are correct is considered a formal process. In extreme synthesis, Godel showed that with every formalization of arithmetics, there are truths that cannot be proved within the same formal system; that is, there is always something that can not be proved right or wrong within the rules of its system. But at that time, a big question remained unanswered: could there exist, at least in theory, a defined process or method to decide whether a mathematical proposition was provable? At the beginning of the century, the Prussian mathematician David Hilbert included this question in a list of 23 mathematical problems that were still undecided, and challenged mathematicians to try and solve them.

Alan Mathison Turing (1912-1954) is considered one of the most important scientists of the twentieth century, and a pioneer, if not the founder, of what we call AI today. Turing was a mathematician who spent a large part of his life studying a wide variety of disciplines including logic, probability, arithmetic, biology, and cryptography. Turing followed a very innovative approach to the problem on Hilbert's list, working not inside, but outside the formal system. Turing provided a new definition of "method" by analyzing what a person can do mechanically, and expressing this analysis on a theoretical machine that could perform basic operations. Turing's machine used a simple paper tape to store data and uses the concept of state changes, like that of a human being who changes their state of mind during an evolving mental process.

The Turing approach to the problem was really very innovative and revolutionary. He laid the foundation of the modern theory of computation and computability. He proved that determining the decidability of mathematical propositions is equivalent to asking, "What sorts of sequences of a finite number of symbols can be recognized by an abstract device?" Having defined the new concept of definite method, that which we now call algorithm, he could also try to answer Hilbert's question: does a defined process or method exist to determine whether a mathematical proposition is provable. A system that is able to perform the same operations as the Turing Machine is called *Turing complete*. It's interesting to see that there is no hierarchy of power in this computational system. If a system is Turing complete, in fact, it can do anything definable in terms of an algorithm. It's funny to think that a simple single-taped Turing Machine is as powerful as the expensive state-of-the-art computer that you just bought… and even more powerful if you think that, theoretically having an infinite tape, can handle infinite numbers!

NOTE

For more information regarding Alan Turing, the original inventor of the concept for our model please refer to www.turing.org.uk.

The Turing Test

In 1950, Turing published an article entitled "Computing Machinery and Intelligence," in which he asked the question "Can a machine think?" In the same article, he also posed another very interesting problem: if a machine can think, how could we tell? So he tried and solved the problem by inventing an imitation game, later called the "Turing Test for Intelligence," to demonstrate that computers in the future would be able to be programmed to emulate and rival human intelligence. The game consisted in a human being and a computer that were interrogated under specific conditions, so that the interrogator wouldn't be able to determine whom he was interacting with. The interrogation was performed using a specific text-only communication protocol. Turing held that if the questioner could not distinguish between the human and the machine within a reasonable time, then there is no reason why we could not consider the either the human or the machine as "intelligent."

This test soon became the "Holy Grail" of the entire AI community, and is still a fundamental reference point for many scientists and programmers worldwide. In 1990, Hugh Loebner created a contest based on the Turing Test, with a prize for the first computer whose results could be indistinguishable from a human test subject, while every year a medal is awarded to the "most human" computer. The contest is still open.

For more information regarding the Turing Test, please refer to www.loebner.net/Prizef/loebner-prize.html

The Original Turing Machine

Examining each and every part of the original setup that Turing invented will help us to understand thie device's precise role in the entire structure.

The first and main part of the original Turing Machine consists of a tape of infinite length, divided into cells. Each cell has just enough space for a single symbol to be written. A head reads and writes on the tape, and can move one cell at a time, moving to either the left or to the right. As you can see, the Turing Machine is a primitive machine that can handle very basic operations such as moving and writing symbols. So, what's the interesting part?

The device itself is always in one of a finite number of states. These states determine what action the machine should perform via a special programming feature called a state transition table (or action table). The state transition table tells the Turing Machine what to do when it is in a certain state and encounters a specific symbol, similar to how your state of mind tells you what to do in a certain situation to respond to specific external events.

The machine can perform four actions:

- Go left one cell
- Go right one cell
- Write a symbol
- Erase a symbol

In synthesis, you can think of the Turing Machine as a "black box" that is capable of reading, writing, and erasing symbols on a long tape divided into cells. Note that while the tape has to be infinite, the states and the symbol are finite. As you can easily see there are no places other than the tape to store data, so for the most complex operations (like multiplying two numbers) you need to put placeholders on the tape in addition to the numbers themselves. In fact, you have to hold information about what is happening, what is going to happen, and temporary data. Thus, for even very simple operations, the states quickly grow to a large number.

NOTE

For more information on the architecture of the Turing Machine and its variations, please refer to the following Web sites:

- www.unidex.com/turing/tm_intro.htm
- www.personal.kent.edu/~brosmait/computability/architecture.html
- www.webpearls.com/products/demos/pap/comap/chapter5/sec6/node2.html
- www.netaxs.com/people/nerp/automata/turing0.html

The State Transition Table

By convention, the machine starts in its first state at the first cell furthest to the left containing a symbol. Then it scans the tape by moving to the right or left, and handles the data by reading, writing, and erasing symbols. The table tells the machine the equivalent of: "if you are in a certain state and see a certain symbol, do a specific action, move one cell and change state." Let's see this process in a sequence.

1. The machine reads the contents of a cell, that is, the symbol written on the tape.
2. The machine uses the state transition table to map the current input (state+symbol) to an output (various actions).

3. The machine changes the symbol on the tape, erases it, or leaves it as is.
4. The machine moves the tape to either the left or to the right.
5. The machine may or may not change its internal state to a new state.
6. If in the "stop" state, the machine stops operating; otherwise, the process starts again from Step 1.

All of these decisions are based on its current state; that is, on its state transition table, which determines the machine's behavior. The first part of it indicates the situation of the device: its state and the symbol read on the current cell. The second part of the table contains the instructions for the machine's operations including what to write (if anything), where to go, and which state to shift to.

Here's a schema of the content of a single line of the table (Figure 3.1):

Figure 3.1 The Schema for a Single Line of the State Transition Table

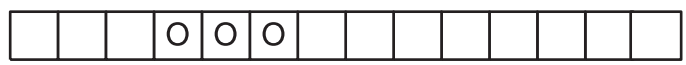
State	Read Symbol	Operation	Move Direction	Next State

As you can see now, programming a Turing Machine is quite difficult because there is little to no possibility of abstraction, which makes complex functions very hard to implement.

An Adding Machine

Let's suppose you want to add two integers. This can be done, depending on the restrictions you have on the number of symbols and the states you're allowed to use, in numerous different ways. One of the simplest methods is to represent an integer on the tape (which is empty) by the number of cells occupied by a symbol, in this case an "o." An empty cell means it is blank; that is, no symbols are written on it. For example, Figure 3.2 shows the representation of the number 3 on the tape:

Figure 3.2 Representing Three on a Tape

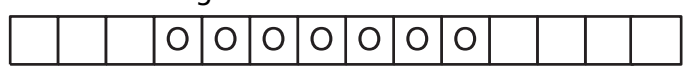


Let's write on the tape the two symbols that we want to add (for example, the numbers 3 and 4), separated by an empty cell (Figure 3.3).

Figure 3.3 Adding Three and Four

Our goal is, after as few as possible steps, the result of the addition (3+4=7), as can be seen in Figure 3.4.

Figure 3.4 The Result of Adding Three and Four



The machine should start at the first symbol on the left, add the two integers, and stop after showing the required answer. Table 3.1 shows how a state transition table could be done.

Table 3.1 A Transitions Table for Adding Two Integers

Input		Output		
State	**Read Symbol**	**Operation**	**Move Direction**	**Next State**
0	o		Right	
0	<Empty>	Write: o	Right	1
1	o		Right	
1	<Empty>		Left	2
2	o	Erase	Stop	

This way we have created a machine that is capable of adding any two integers, no matter what their size (provided that you have an infinite long tape). Input is determined by the combination of the current state (indicated as a number) and symbol. In this case, only two "symbols" are used: a cell can be empty or full. Note that when a cell of the table is empty it means that there is no action to take, meaning that a symbol is not re-written if already there. In the same way, if the next state is not specified, it means that there's no need to change it, and the machine simply stays in the current one. It is very interesting to see how the entire process works, by examining the tape in each step (see Table 3.2). I've indicated the current position of the machine's head with a gray cell.

Table 3.2 Steps for Adding Two Integers

State	Tape										
0		O	O	O		O	O	O	O		
0		O	O	O		O	O	O	O		
0		O	O	O		O	O	O	O		
0		O	O	O		O	O	O	O		
1		O	O	O	O	O	O	O	O		
1		O	O	O	O	O	O	O	O		
1		O	O	O	O	O	O	O	O		
1		O	O	O	O	O	O	O	O		
1		O	O	O	O	O	O	O	O		
2		O	O	O	O	O	O	O	O		
Stop		O	O	O	O	O	O	O			

It's very interesting to understand how the state transitions table (the program) works, and examine the machine's behavior step-by-step. Do you think that you'll be able to modify the table to let the machine return back to its original cell after adding the two numbers? I can assure you, it's not difficult. A quick tip: you actually have to change State 2 and add another state to let the machine go left until it finds a blank.

As you can see, now that we've gone a bit more in depth, this device is really more like a computer program (a sort of software) rather than a computer (hardware), its key part being the programming algorithm defined in the action table.

The UTM: Universal Turing Machine

In his works, Turing described a new, powerful concept applied to his machine. If an algorithm of a particular procedure can be coded with standard instruction (that can be read by a Turing Machine), then the work of interpreting the instructions itself is a mechanical process, and can be run by a particular Turing machine called the Universal Turing Machine (UTM). This is basically the concept of a modern computer: one machine that can be used for many different tasks, if programmed with the appropriate algorithm. The UTM, like modern computers, uses the idea of storing *instructions* in the same way in which *data* and numbers are stored. That was a revolutionary concept in 1936, many years before the first practical application of a digital computer.

The Differences between our LEGO Turing Machine and an Authentic Turing Machine

Building a real machine like the one that Alan Turing theoretically conceived is impossible, even without the limits of the Lego MINDSTORMS system. In March 2002, Aatish Bhatia published some pictures of his "LEGO Pseudo-Turing Machine" on the Web (www.generation5.org/aisolutions/ab_legoturing.shtml). This was the first attempt to build such a device with LEGO, although it remains an interesting experiment, it lacked a fundamental functionality that defines a Turing Machine: the possibility to write on tape. In that moment, I thought that it would be great if someone could push the experiment a bit further. I did some tests, and after many trials and errors, I triumphed! I built a limited but working device that has all the characteristics to be considered a true LEGO Turing Machine.

The first problem you encounter (and a large problem at that) is that the tape should be infinite. However, in actuality the tape does not need to be *infinite*, just *indefinite*, meaning that you build a finite device that can always be expanded for your needs. LEGO is the perfect tool to achieve this. It has the wonderful feature of being modular

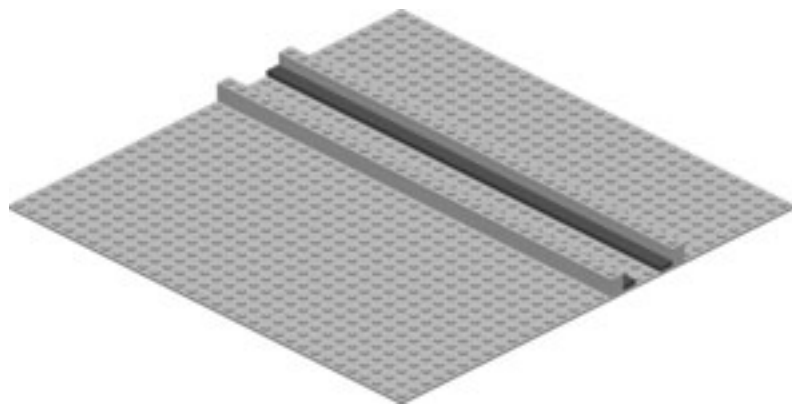
and easily modifiable, so you can always enlarge the tape for your needs. For most programs that you will be creating, a small tape is enough, so that you don't need to have thousands of pieces to build this model.

The second problem is that although the number of symbols a Turing Machine uses is finite, but can be high. This is hard to build with LEGO pieces, so you have to consider the constraints. We need to be able to create a mechanical device that can be set into discrete states, and the states need to be uniquely readable by the LEGO device. It's hard to build a tape that can handle more than two or three symbols, and in this chapter we describe a machine that has two possible symbols to keep things simple. This is not a real problem, as even if you had a tape whose cells could remember hundreds of states, you the human would be hard-pressed to write a program (state transition table) that would ever utilize all the states. So, the two symbols (empty and full) are enough to build and program many different tasks.

Building the LEGO Turing Machine

It's time to begin the construction of the LEGO Turing Machine. We'll build a device that is as similar as possible to the original concept of the Turing machine. The scheme will be a modular design: that is, splitting the project into sub-assemblies, each with a specific task. This approach is very useful as it allows you to separate the problem into smaller ones, giving you the possibility to concentrate on a single task at a time. We will start by creating a large support for the machine, and then we will build the tape, a motor-driven direction control, and the "head" of the machine. The head is the most important part, as it will read and write on the tape.

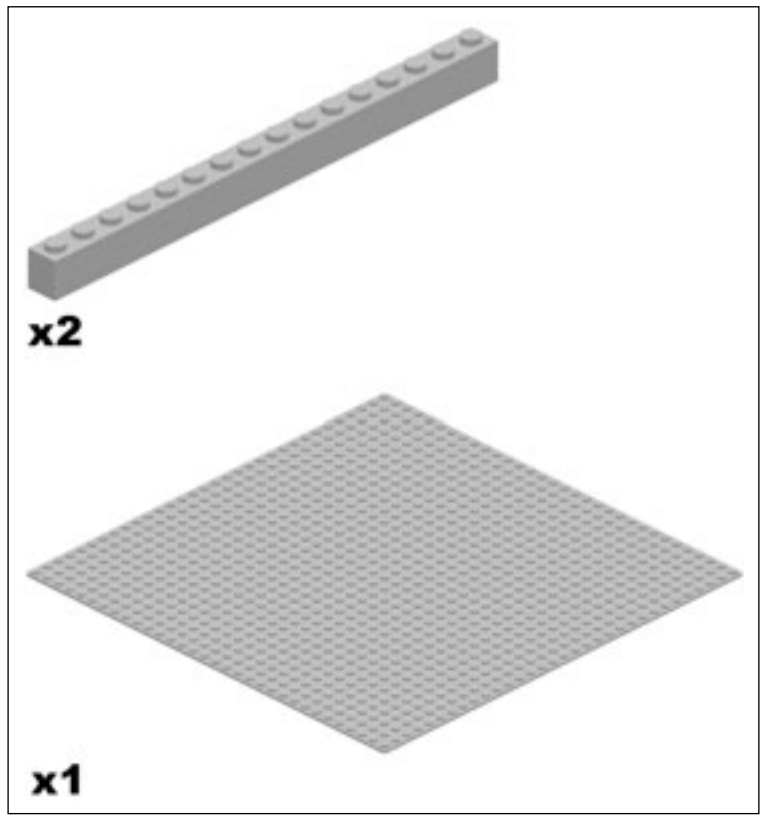
The Base

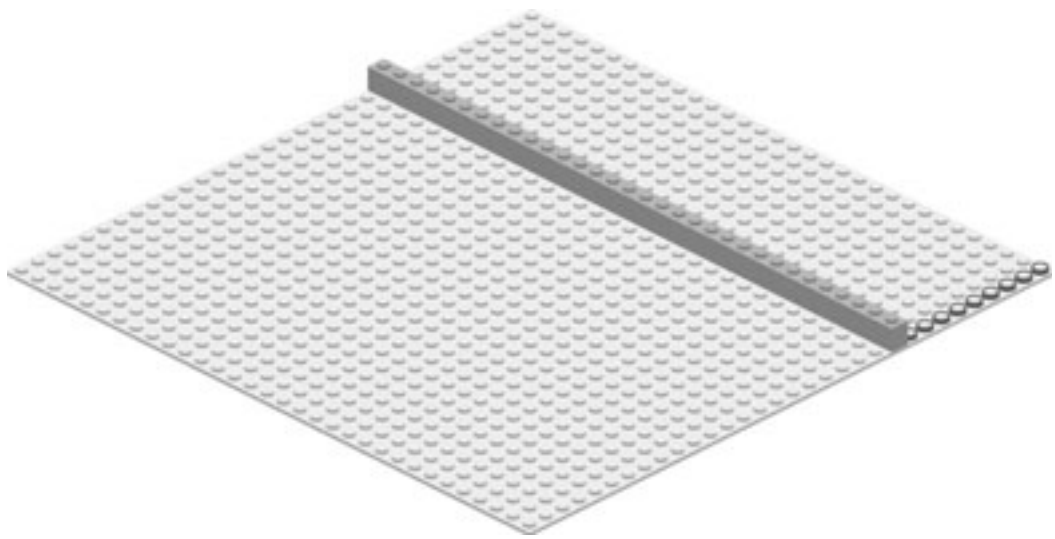


The Base sub-assembly serves as a container and connecting structure for the other parts of the model. On this base, we have to build a support for the Tape sub-assembly (built next) that should be free to move to the right or left without any constraints. The key point is that you have to build a solid mounting, especially if you think that the Tape

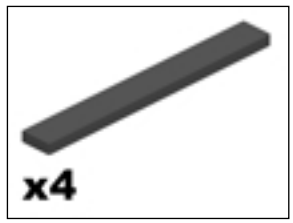
sub-assembly can be, at least in theory, very long. I chose to use the ground as a support, so this solves the problem of a long tape moving out of the model: it simply touches the floor and slides on it.

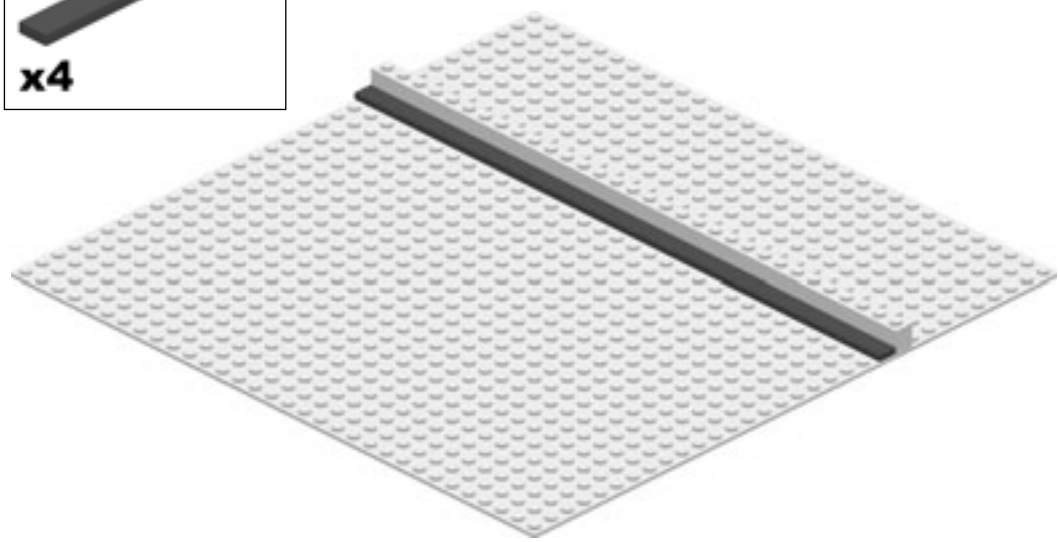
Base Step 0



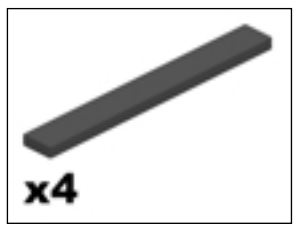


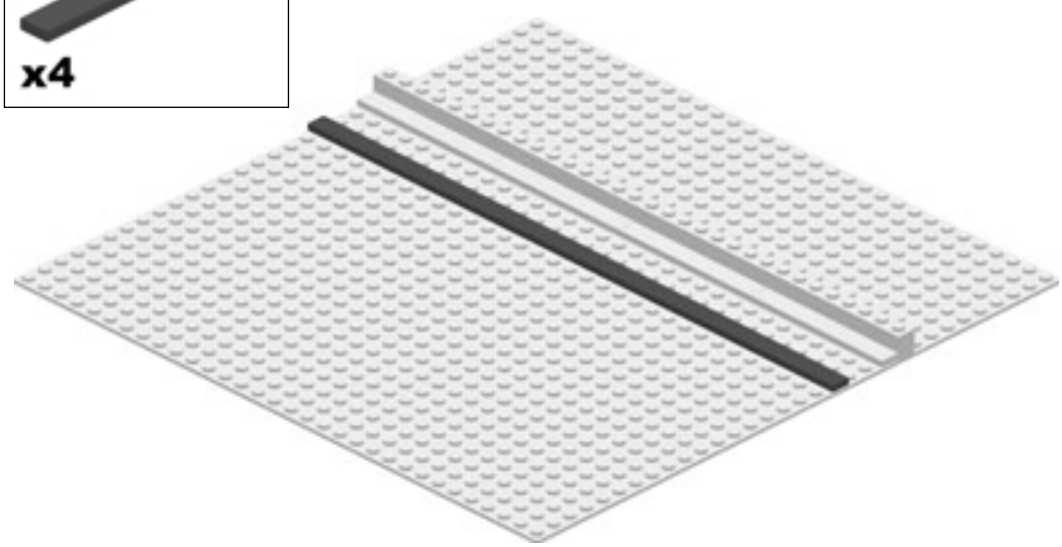
Base Step 1



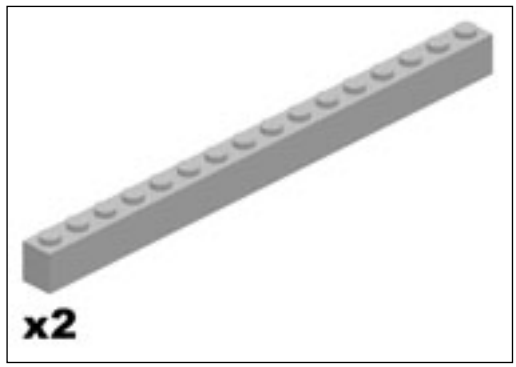


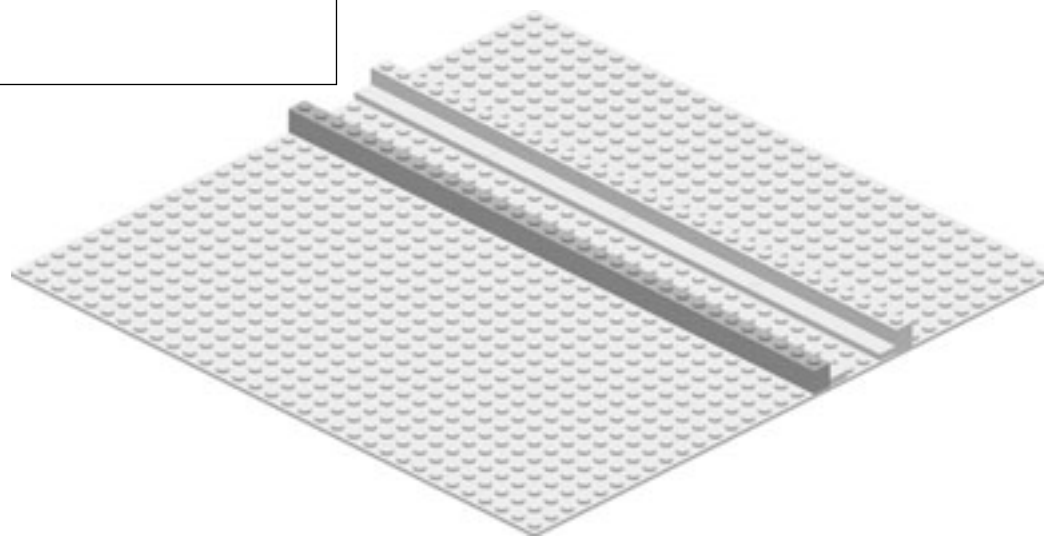
Base Step 2



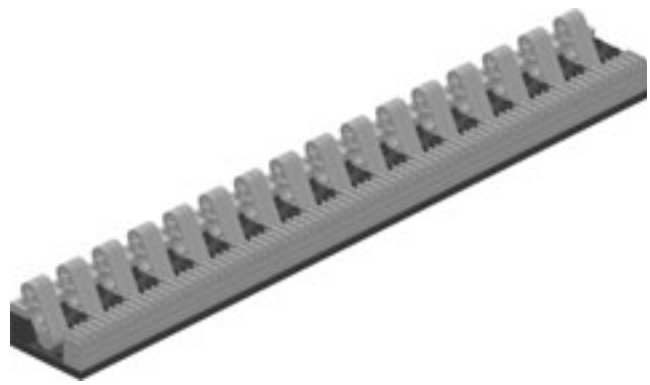


Base Step 3





The Tape

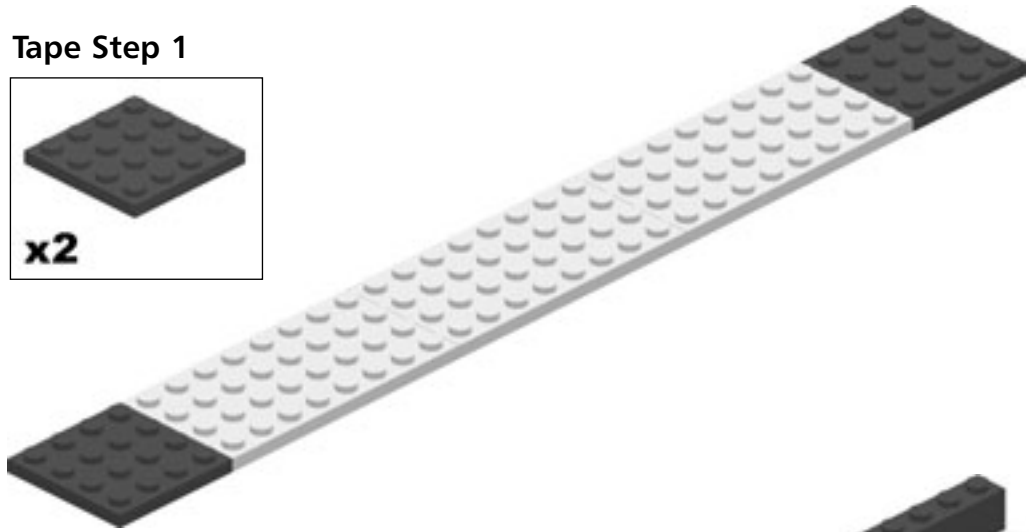


The Tape sub-assembly is where all of the data will be stored. We already mentioned that it has to be divided into separate cells, which can be empty or full. I decided to use 1x3 TECHNIC liftarms to build small switches that can be moved either up or down by the writing mechanisms. There is no fixed length for the Tape sub-assembly; therefore, you can construct the Tape sub-assembly to be as long as you want. The longer the length, the larger the numbers you will be able to handle. For my model, I built the Tape sub-assembly to be 32 studs in length. This equates to each cell occupying two studs, so the Tape sub-assembly can store 16 cells.

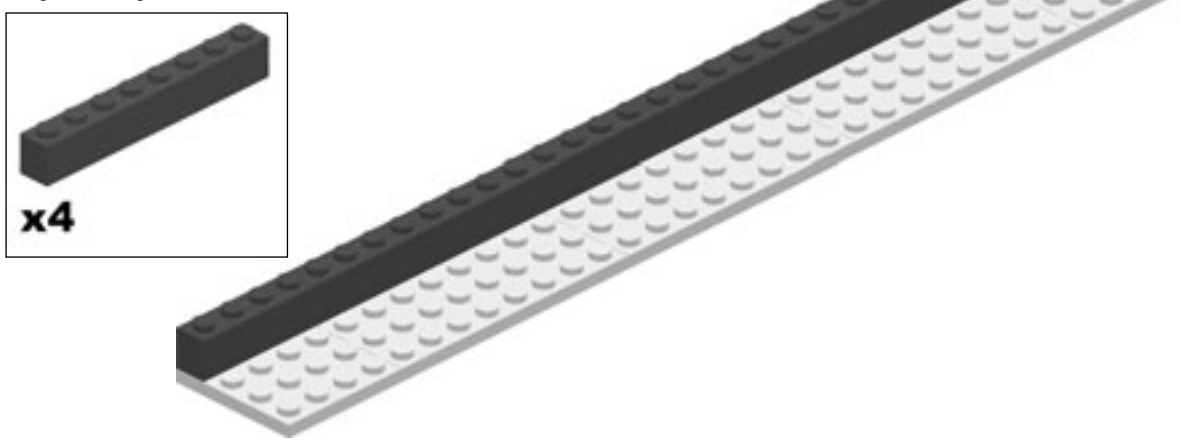
It's important to maintain a difference between the color of the liftarms, which should be light (in our case yellow) and the color of the rest of the tape, which should be dark (in our case black and light gray). This way you ensure that a correct reading from the light sensor is always returned.

Tape Step 0

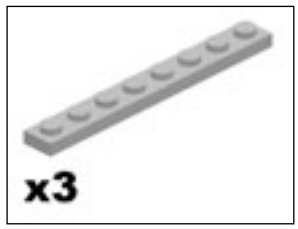
Tape Step 1



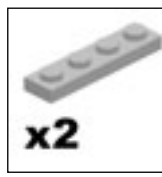
Tape Step 2



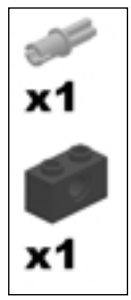
Tape Step 3

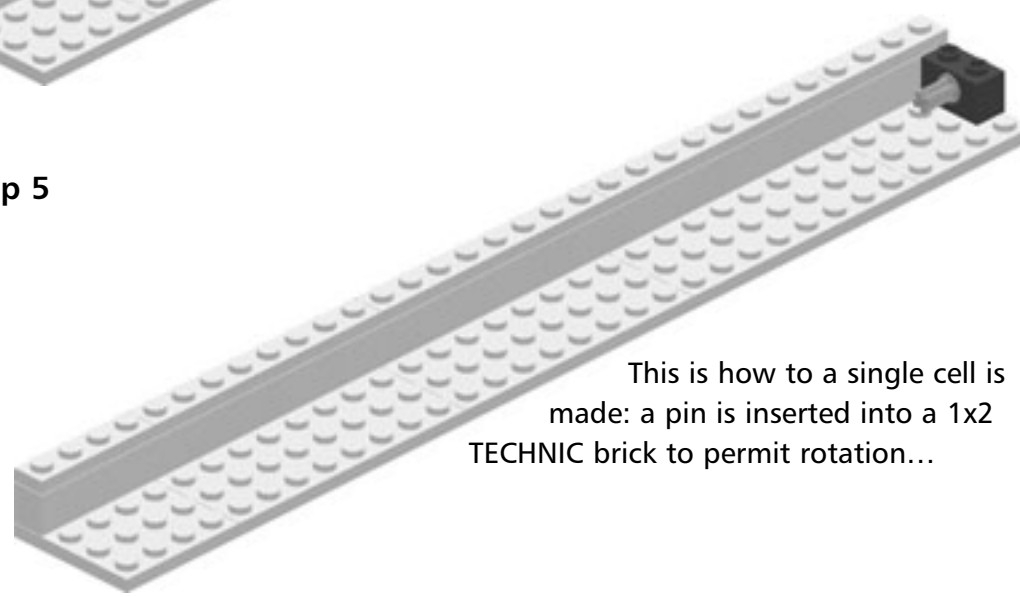


Tape Step 4



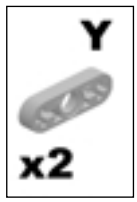
Tape Step 5





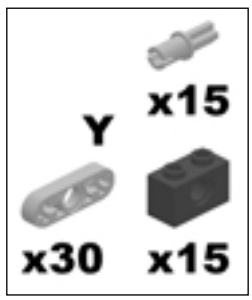
This is how to a single cell is made: a pin is inserted into a 1x2 TECHNIC brick to permit rotation…

Tape Step 6



…and two 1x3 liftarms will create the moving part of the cell.

Tape Step 7

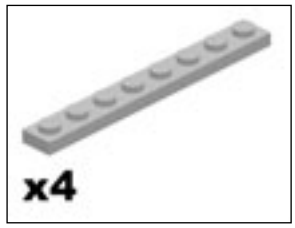


Repeat **Tape Step 6** and **Tape Step 7** until you have added all of the "cells" needed to fill the entire tape.

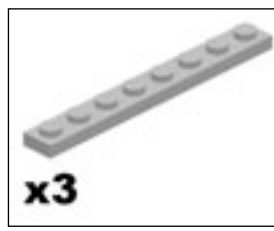
NOTE

The parts list for **Tape Step 7** reflects the number of pieces needed for a tape of this length. You may require more or less parts depending on the length of your tape.

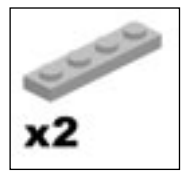
Tape Step 8



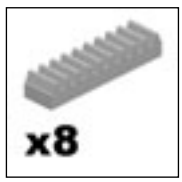
Tape Step 9



Tape Step 10



Tape Step 11



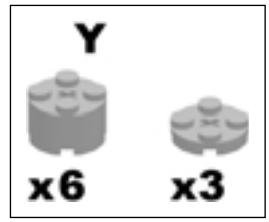
The gear racks are the pieces that allow movement control from the outside. If you were ever to build a longer Tape sub-assembly, the gears would have to be mounted along the entire length of the tape.

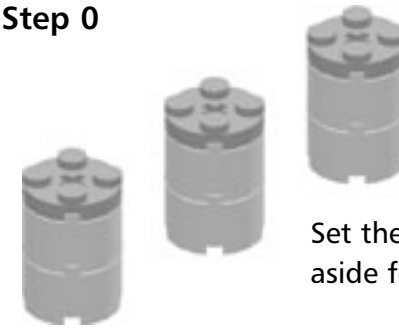
The Direction Control

The Direction Control sub-assembly controls the right and left movements of the tape. This should be a discrete step-by-step movement, meaning that the tape should move one cell at a time, no more, no less. This is achieved by using a touch sensor that can be closed by pegs connected to a 40T gear. The 40T gear is connected to a double-beveled 20T gear that works with the gear racks added in **Tape Step 11**. They act as if they are 10T gears. Every quarter of a turn of the 40T gear results in the Tape sub-assembly advancing by half gear rack o, a single cell.

Other systems to achieve the same result can be used: for instance, a rotation sensor, a light sensor that emulates a rotation, or a touch sensor working directly on tape. The one limitation to remember is that the movement must be smooth enough, and never use a timer. A timer on the motor would have to run and depend on the charge of the batteries of the RCX; thus accumulating small errors on errors, and you'd soon lose the correct position of the tape.

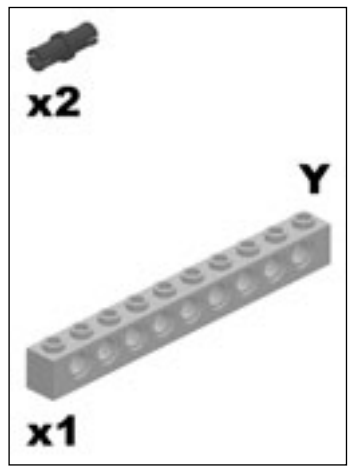
Direction Control Step 0



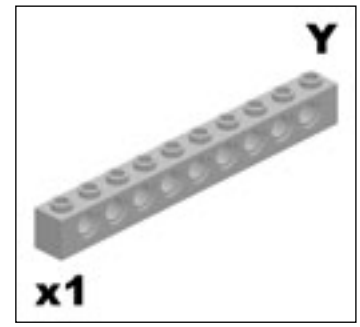


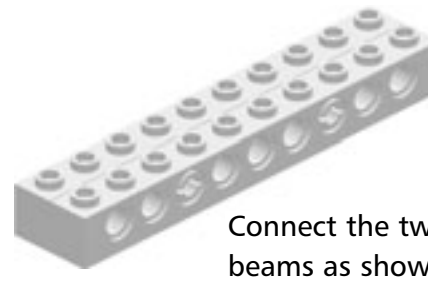
Set these columns aside for a moment.

Direction Control Step 1



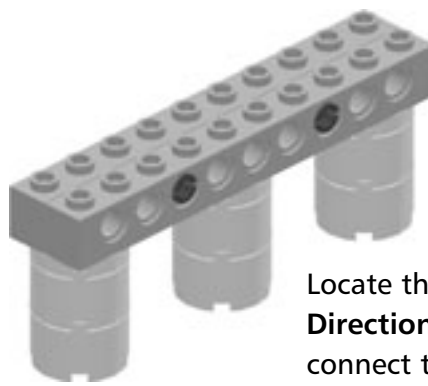
Direction Control Step 2





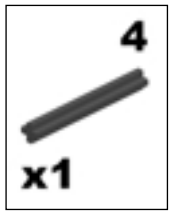
Connect the two yellow beams as shown.

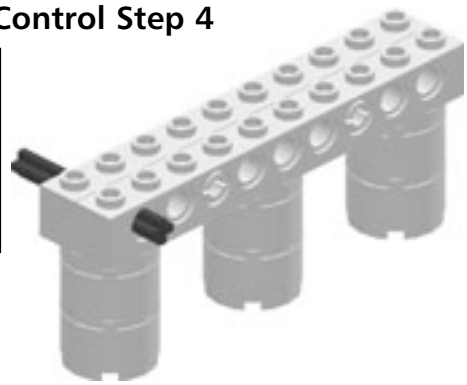
Direction Control Step 3



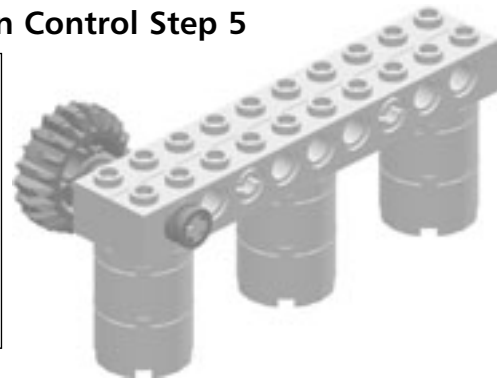
Locate the three columns built in **Direction Control Step 0** and connect the bricks as shown.

Direction Control Step 4



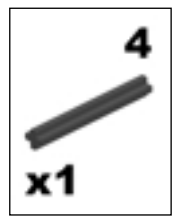


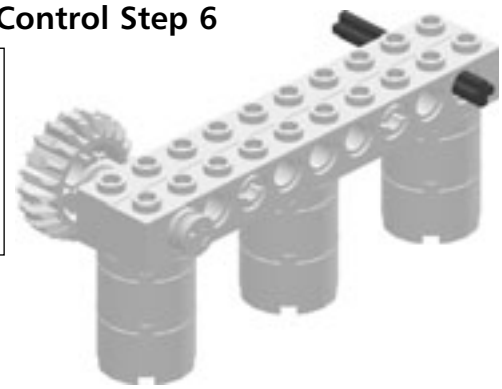
Direction Control Step 5



The double-beveled 20T gear is placed so that it touches the tape for only a half stud. This is so that the gear does not interfere with the liftarms of the Tape sub-assembly as it passes underneath the gear.

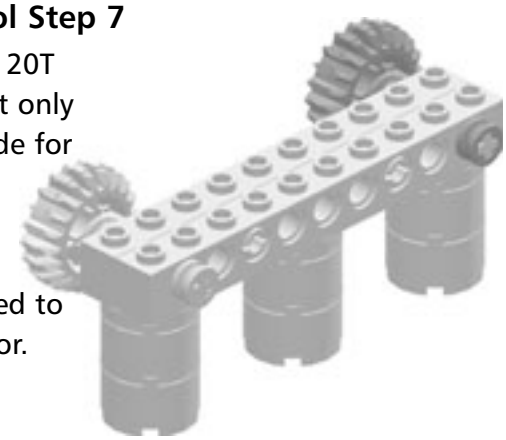
Direction Control Step 6



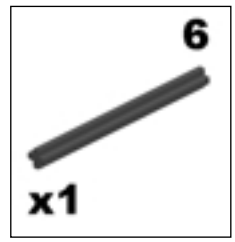


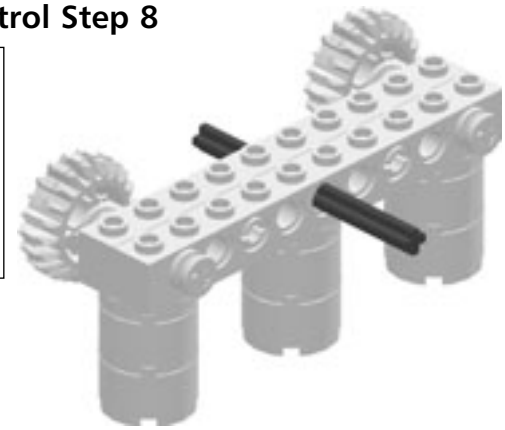
Direction Control Step 7

The two 20T gears act only as a guide for the tape and will not be connected to the motor.

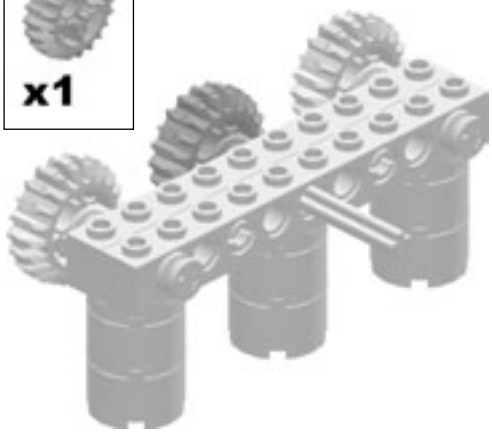


Direction Control Step 8

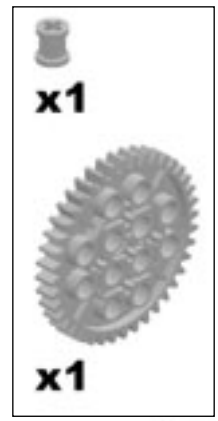




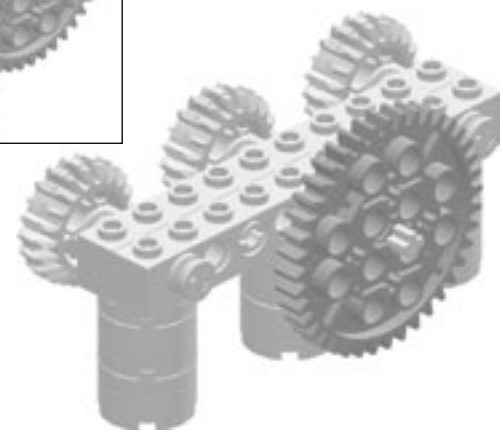
Direction Control Step 9



Direction Control Step 10



Slide the bushing onto the axle prior to adding the 40T gear.



Direction Control Step 11

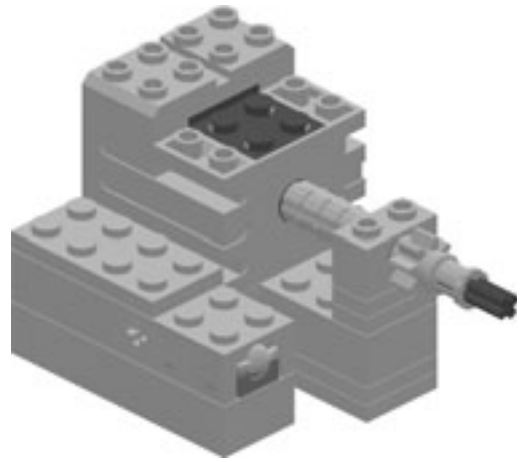
Place the pins in every other external hole on the 40T gear. This ensures that the touch sensor is closed every quarter of a turn.

Bricks & Chips….

Gray Pins or Black Pins?

For this step, we recommend using gray pins. When using a touch sensor, gray pins are preferred over black pins because the gray pin's ability to rotate. Their shape is also more precise when pushing the button on the touch sensor.

The Direction Control II



This is the second part of the Direction Control sub-assembly, where we connect the motor and the touch sensor. Later, we will connect the Direction Control II sub-assembly to the Direction Control sub-assembly.

In this sub-assembly, we use an 8T gear to slow the motor down to a reasonable speed, by gearing it down to 1:5.

Bricks & Chips….

Gears and Gear Ratios

The fundamental property of a gear is its number of teeth. It is so important that we even name the gears after it: for example, a 24-tooth gear becomes a 24T. By connecting two or more gears with a different number of teeth, you can control their speed. Think of connecting a 24T gear to an 8T, and slowly turning the 24T: You see, for every turn of the 24T the smaller gear makes three turns. This is a gain in *angular velocity*. In simple terms, angular velocity means a gain in speed. This conversion unfortunately has a cost: The *torque* of the system, that is the strength, is reduced exactly the same way (except for a small part of it that is lost due to *friction*) to one third.

Continued

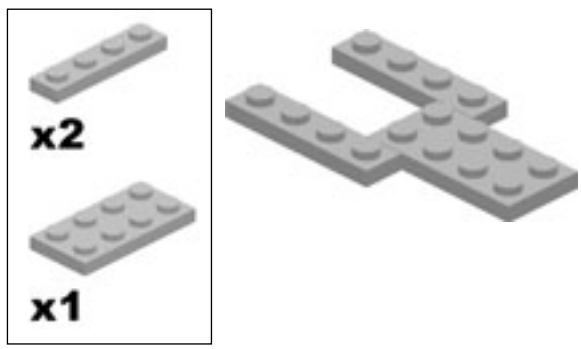
This is a symmetrical process; you can convert torque into velocity and vice versa:

- To *gear up* means that we make our system increase velocity and reduce torque, and ratio is usually written like 3:1.
- To *gear down* means that we make our system reduce velocity and increase torque, and ratio is usually written like 1:3.

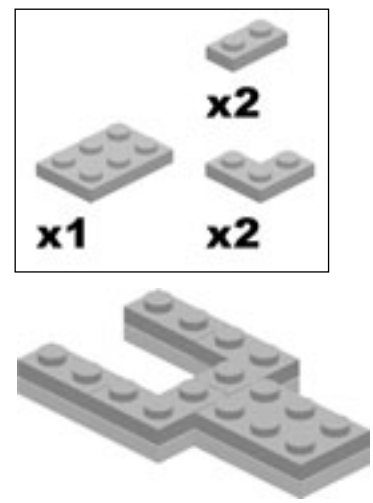
In the Direction Control process, we don't need a high velocity, but do require some torque. So, I have geared down the system by placing a 40T after an 8T, creating a ratio of 8:40 (simplified, that is 1:5).

For additional information on the topics of gears and gear ratios, we urge you to check out to the *Building Robots with LEGO MINDSTORMS* (Syngress Publishing, ISBN 1-928994-67-9) by Mario and Giulio Ferrari for detailed information about placing gears and other mechanical solutions.

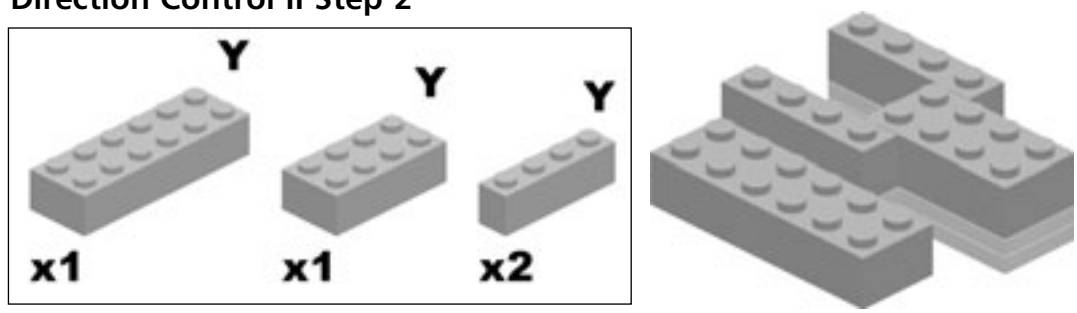
Direction Control II Step 0



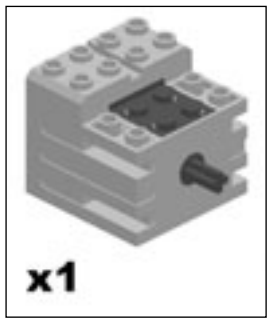
Direction Control II Step 1

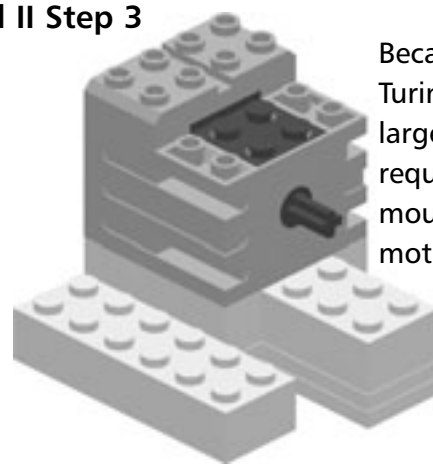


Direction Control II Step 2



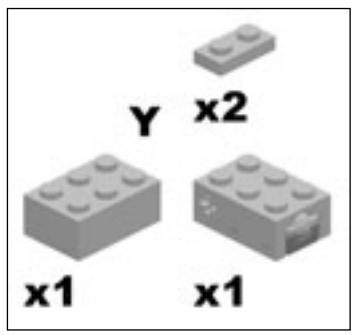
Direction Control II Step 3

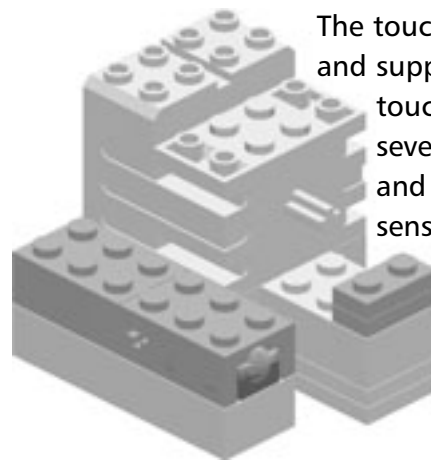




Because the motor in the Lego Turing Machine does not have a large load to bear, it does not require an extraordinarily strong mounting. You'll notice that this motor mount is lighter and simpler than the typical motor mount.

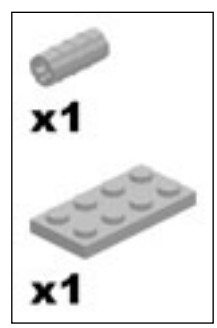
Direction Control II Step 4

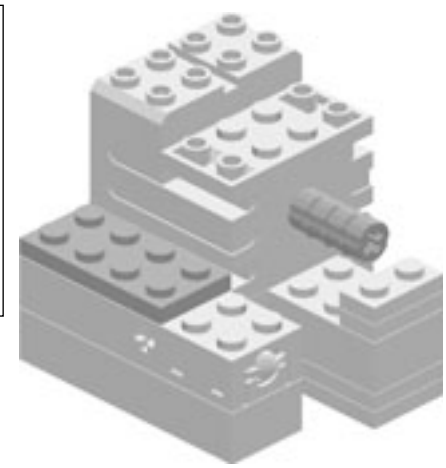




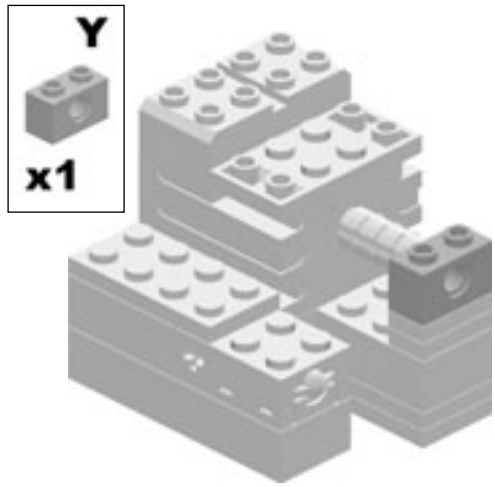
The touch sensor requires a base and support structure of bricks. The touch sensor will be pushed several times. The extra bracing and support will keep the touch sensor from moving out of position.

Direction Control II Step 5

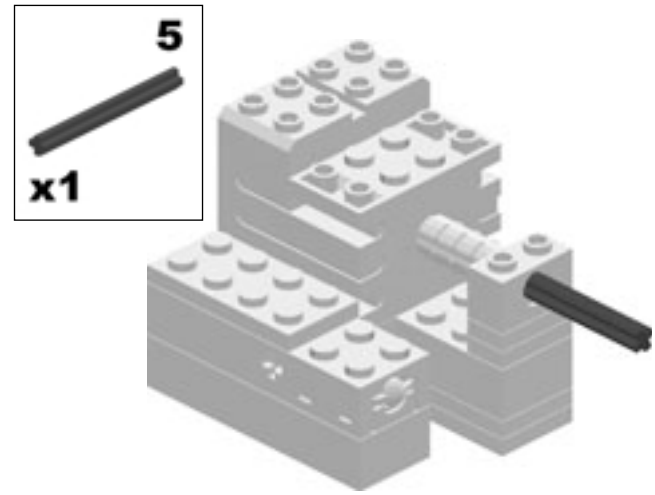




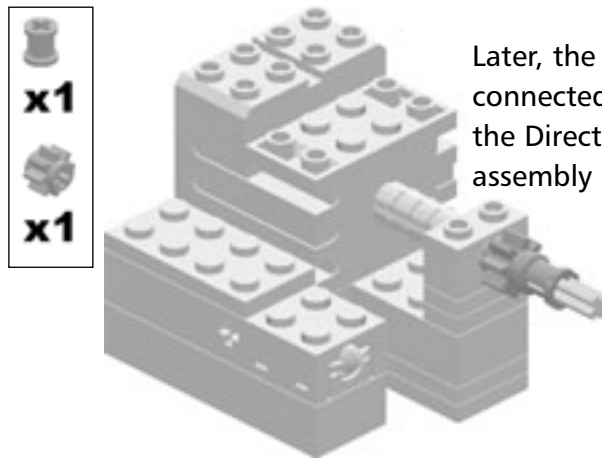
Direction Control II Step 6



Direction Control II Step 7

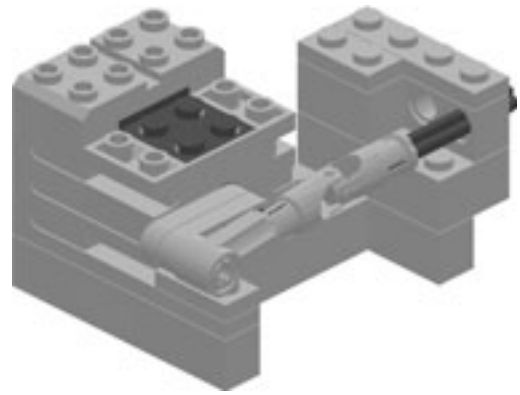


Direction Control II Step 8



Later, the 8T gear will be connected to the 40T gear of the Direction Control sub-assembly in **Final Step 0**.

The Erase Switch

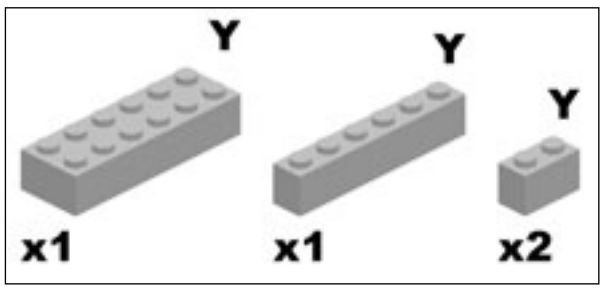


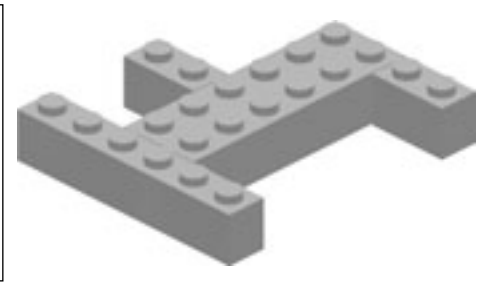
We have to build two mechanisms to perform the write and erase functions of the Lego Turing Machine's module. The head consists of:

- The Erase Switch sub-assembly (down switch)
- The Write Switch sub-assembly (up switch)

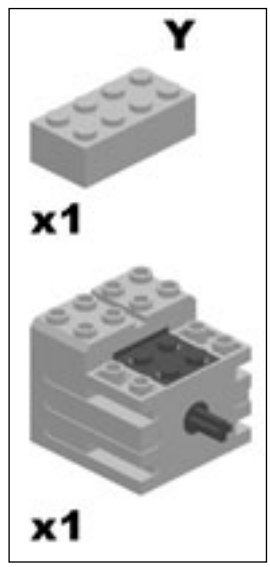
The Erase Switch sub-assembly pushes the liftarm of a given cell down to the erased (or empty) position. It converts the rotation of a motor into a plunger movement that pushes the liftarm of a memory cell on the tape into the erased position.

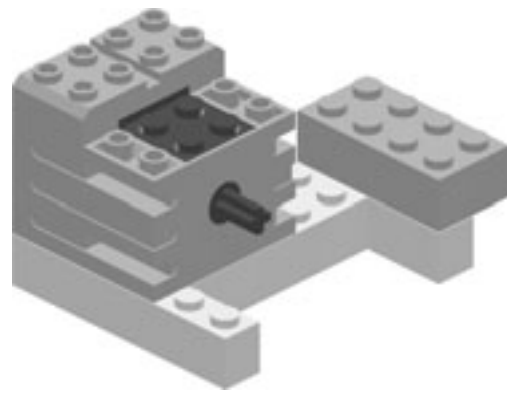
Erase Switch Step 0



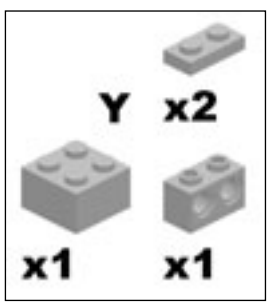


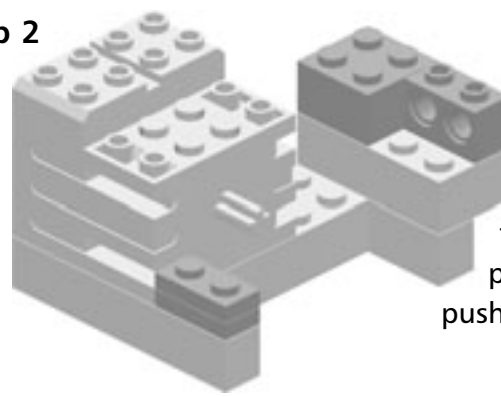
Erase Switch Step 1





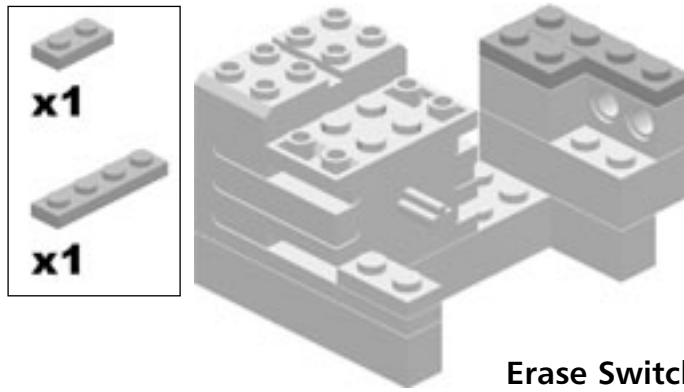
Erase Switch Step 2



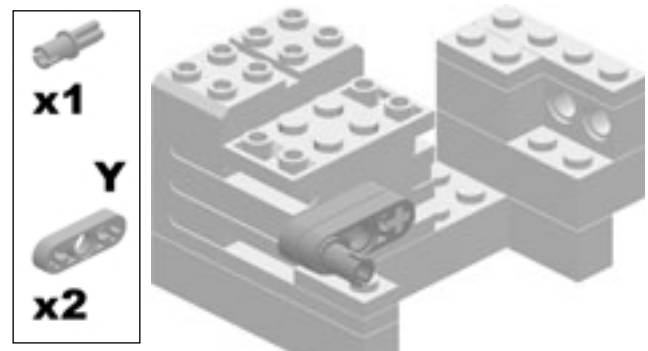


The use of a double-holed 1x2 TECHNIC brick is fundamental for the correct placement of the small pushing arm.

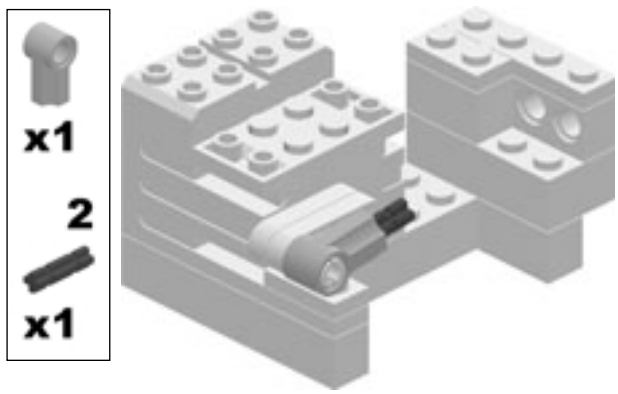
Erase Switch Step 3



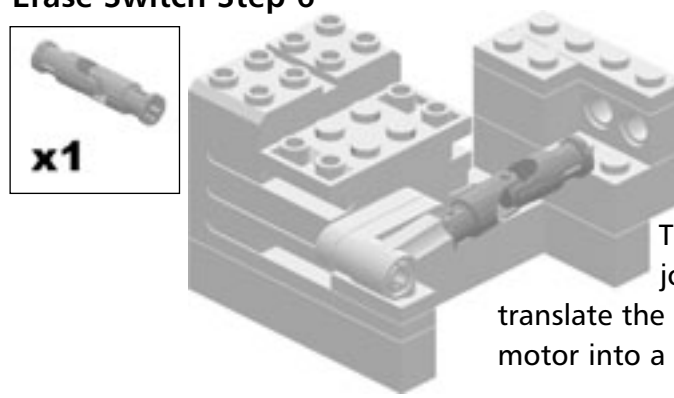
Erase Switch Step 4



Erase Switch Step 5

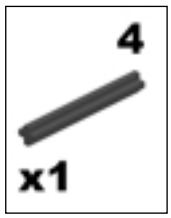


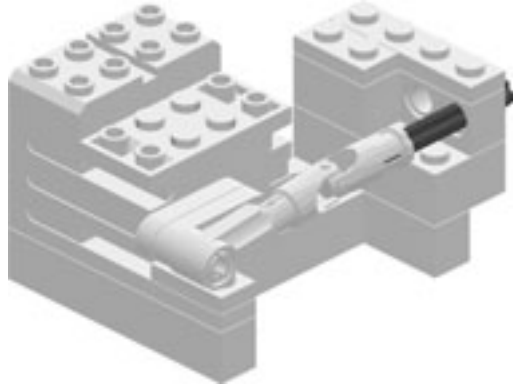
Erase Switch Step 6



The universal joint is used to translate the rotation of the motor into a pushing action.

Erase Switch Step 7



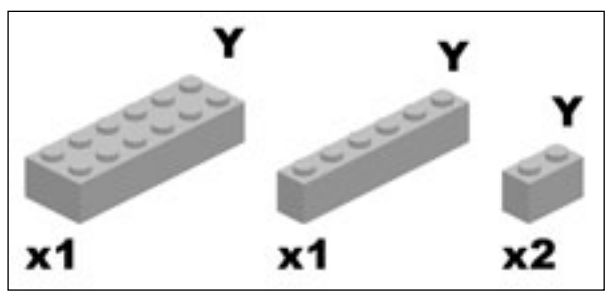


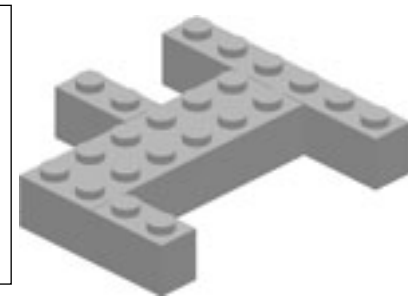
The #4 axle is the arm that will push the liftarms. Insert the arm from the right and connect it to the universal joint.

The Write Switch

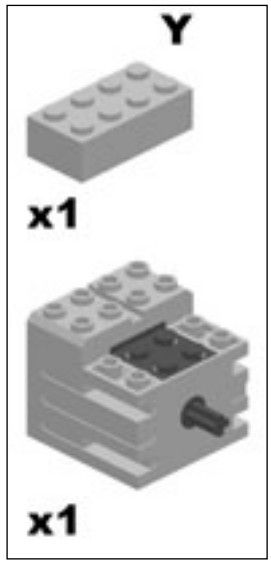
The Write sub-assembly is similar in construction (but obviously not in function) to the Erase Switch sub-assembly. The function of the Write sub-assembly is to push the liftarms on the Tape sub-assembly upwards. The upward movement of the liftarm equates to the process of writing a symbol in a cell on the Tape sub-assembly.

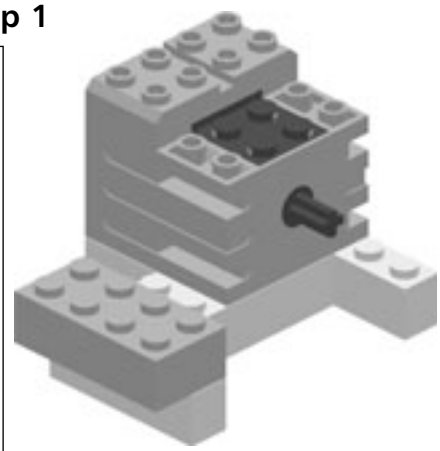
Write Switch Step 0



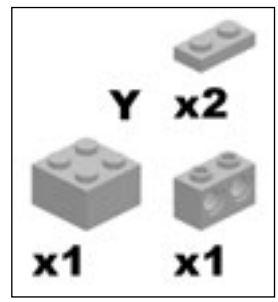


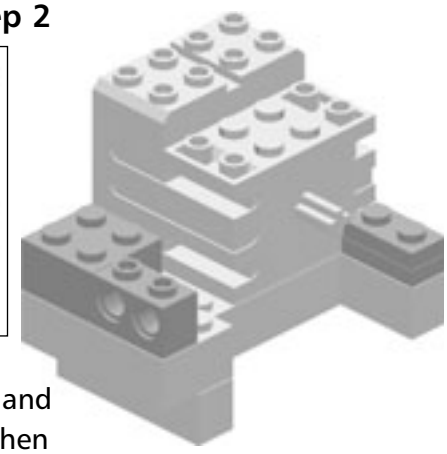
Write Switch Step 1





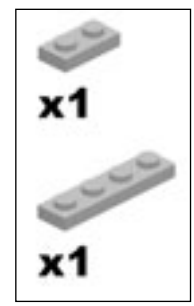
Write Switch Step 2

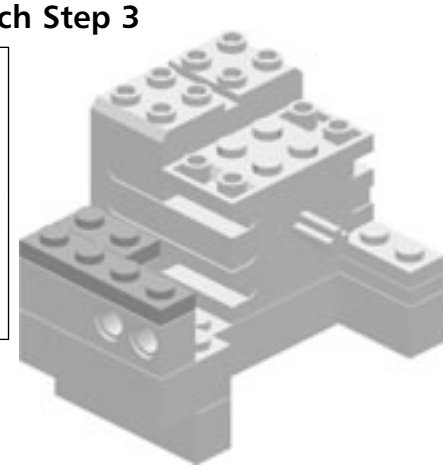




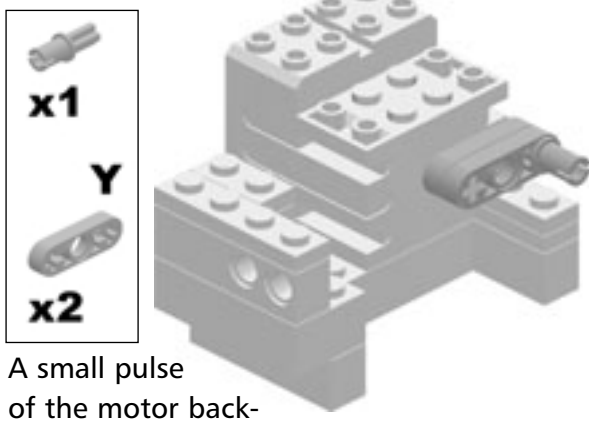
The plates on the right side will help and stop the liftarms when returning the axle to its starting position.

Write Switch Step 3



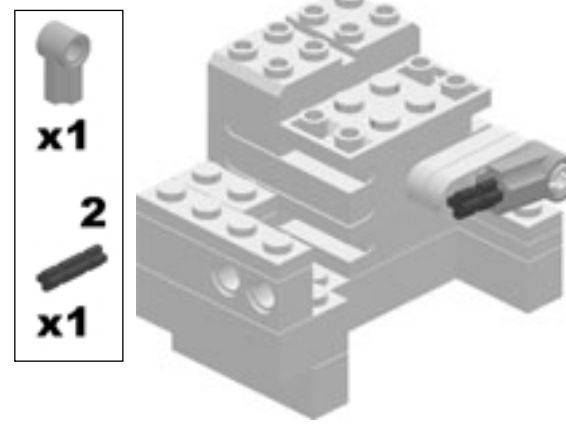


Write Switch Step 4

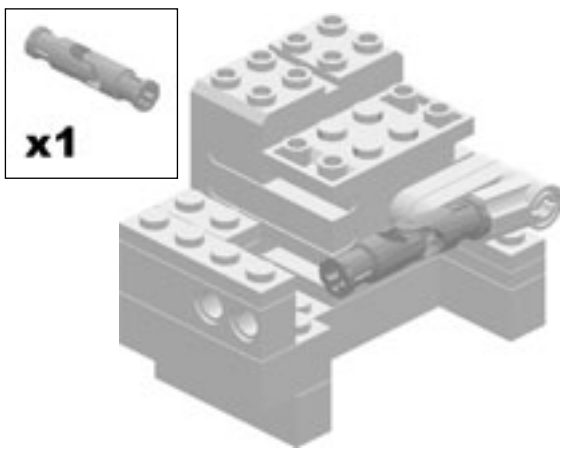


A small pulse of the motor backwards will be enough to reset the liftarm to its original position.

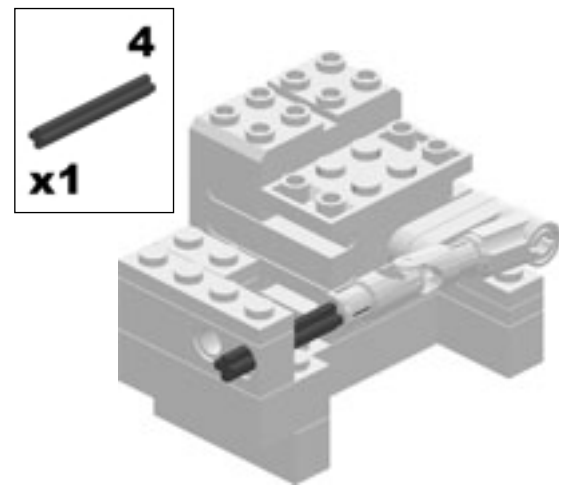
Write Switch Step 5



Write Switch Step 6



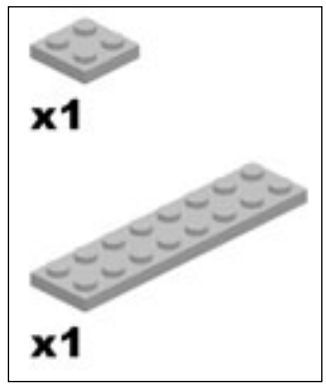
Write Switch Step 7



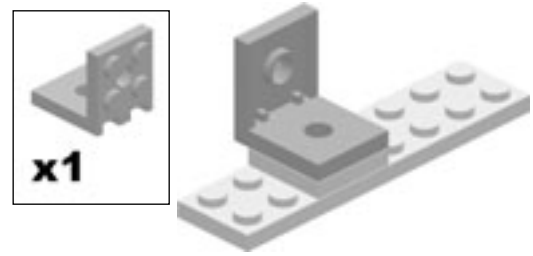
The Light Sensor

The light sensor for the Light Sensor sub-assembly is mounted on the head of the mechanism. This sensor enables the head to simultaneously read and write the same cell. It is very important that the sensor be situated as close as possible to the Tape sub-assembly for the reading of the sensor to be accurate—even if the external light conditions vary. However, in placing the Light Sensor sub-assembly close to the Tape sub-assembly you should be careful that it does not interfere with the movement of the liftarms moving on the Tape sub-assembly as it slides by the assembly just below.

Light Sensor Step 0

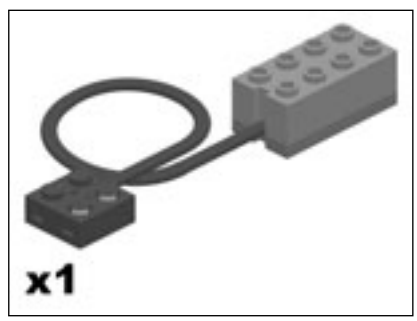


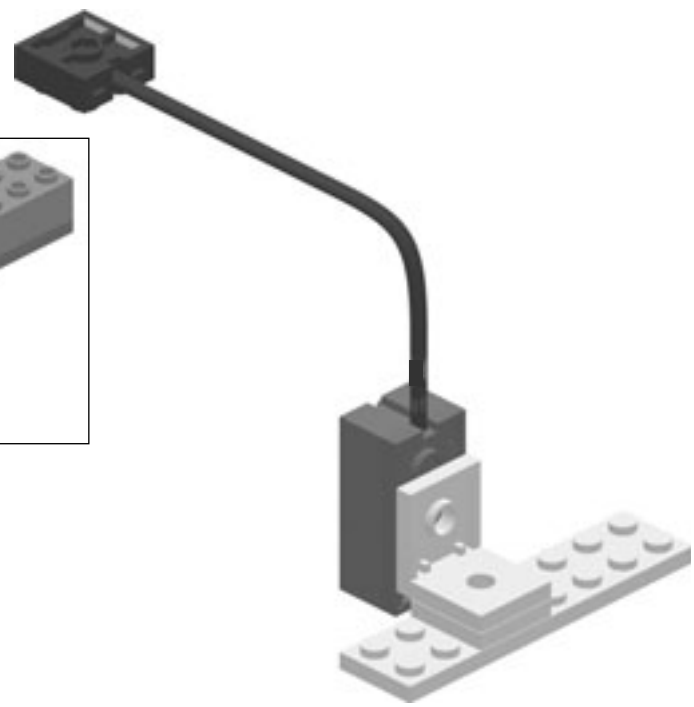
Light Sensor Step 1



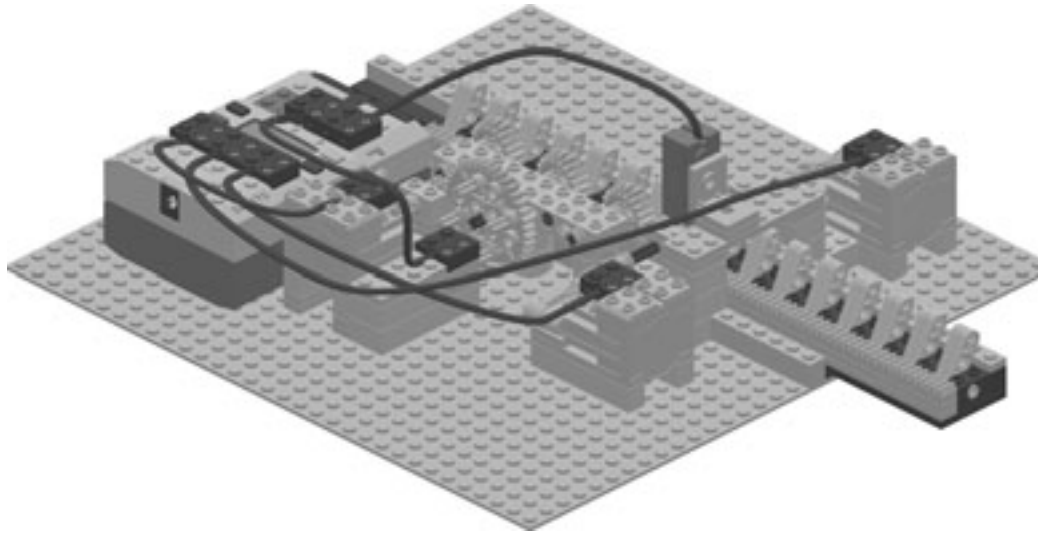
The light sensor is not placed in the center of the plate. Instead, it is shifted to one side to allow the movement of the liftarms.

Light Sensor Step 2



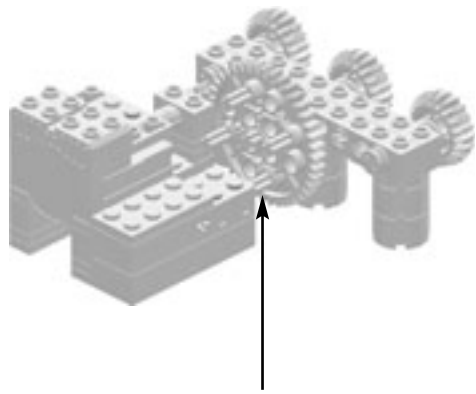


Putting It All Together



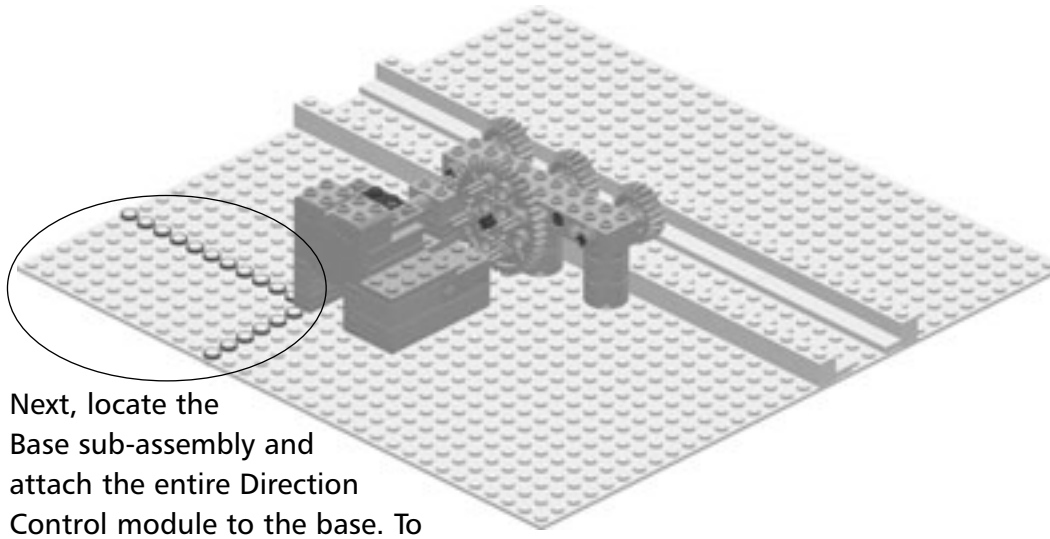
Let's connect all the sub-assemblies and finalize the machine by adding the RCX and some electric cables. Be careful to place the various parts correctly on the baseplate so that the required distance and nearness are maintained. Watch out for the electric cables interfering with the correct movements of the motors; the tape should be free to move either direction smoothly.

Final Step 0



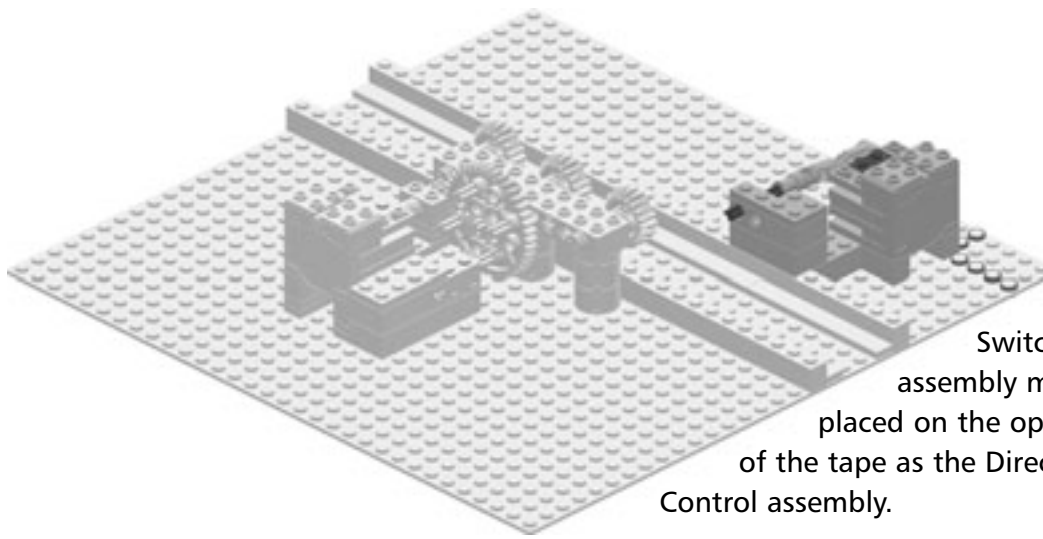
Locate the previously built Direction Control and Direction Control II sub-assemblies, and attach the axle with the 8T gear (not visible) to the 40T gear as shown.

Final Step 1



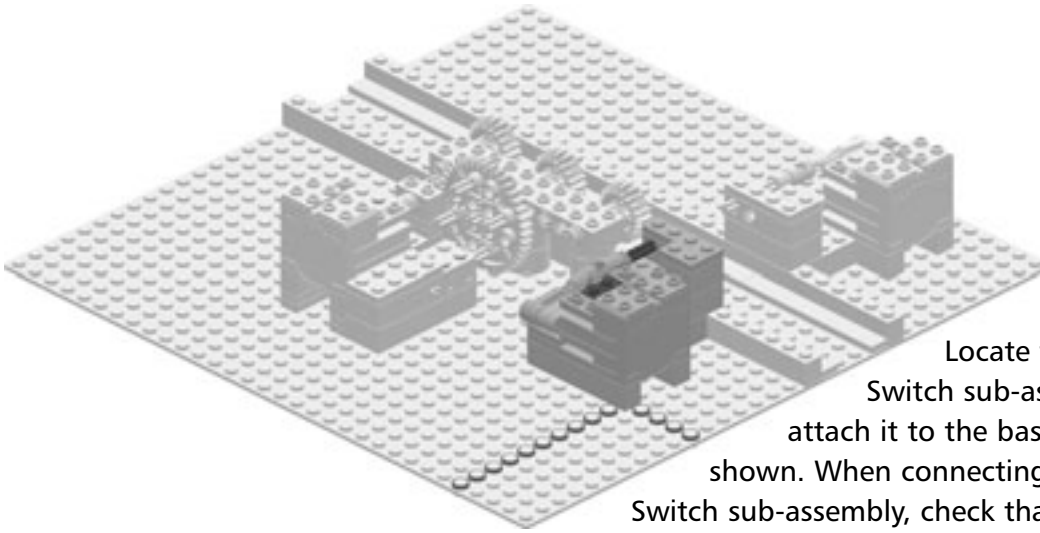
Next, locate the Base sub-assembly and attach the entire Direction Control module to the base. To assist you with the placement of the pieces in the step – refer to the circle provided on the baseplate.

Final Step 2



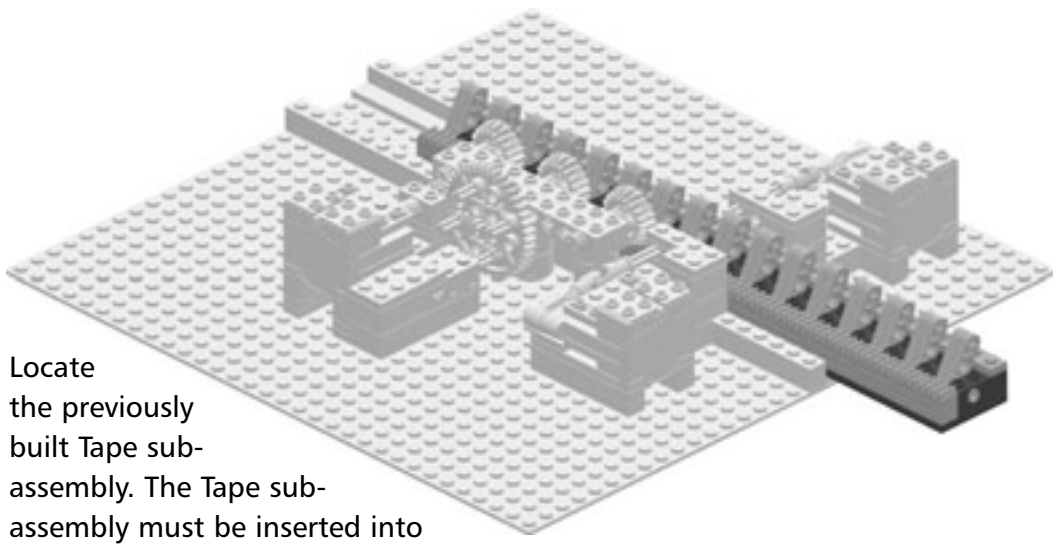
The Erase Switch sub-assembly must be placed on the opposite side of the tape as the Direction Control assembly.

Final Step 3



Locate the Write Switch sub-assembly and attach it to the baseplate as shown. When connecting the Write Switch sub-assembly, check that the two arms (the #4 axles) are on the same line, so they can work on the same cell.

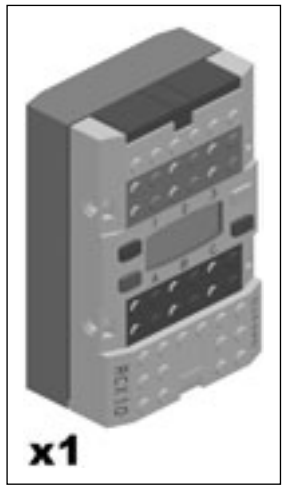
Final Step 4

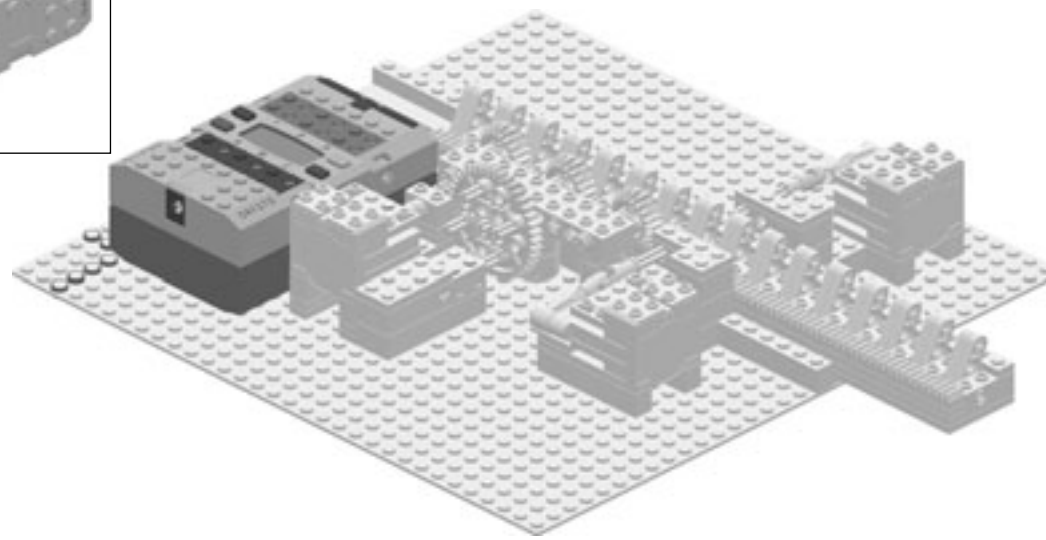


Locate the previously built Tape sub-assembly. The Tape sub-assembly must be inserted into the model very carefully. Make sure that a gray peg is closing the touch sensor at the same moment a yellow liftarm is aligned with the Write Switch sub-assembly and Erase Switch sub-assembly (the head).

Check also that the three 20T gears and the #4 axles of the Write and Erase Switch sub-assemblies are placed in line with exactly one stud on the Tape sub-assembly. This is so the switches don't interfere with movement of the Tape sub-assembly.

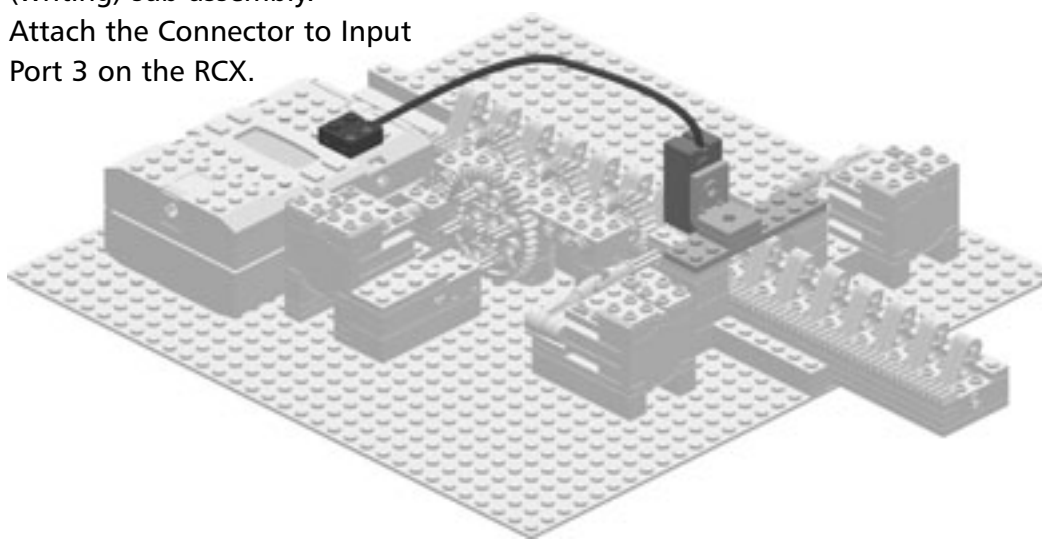
Final Step 5



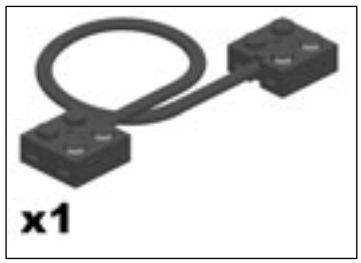


Final Step 6

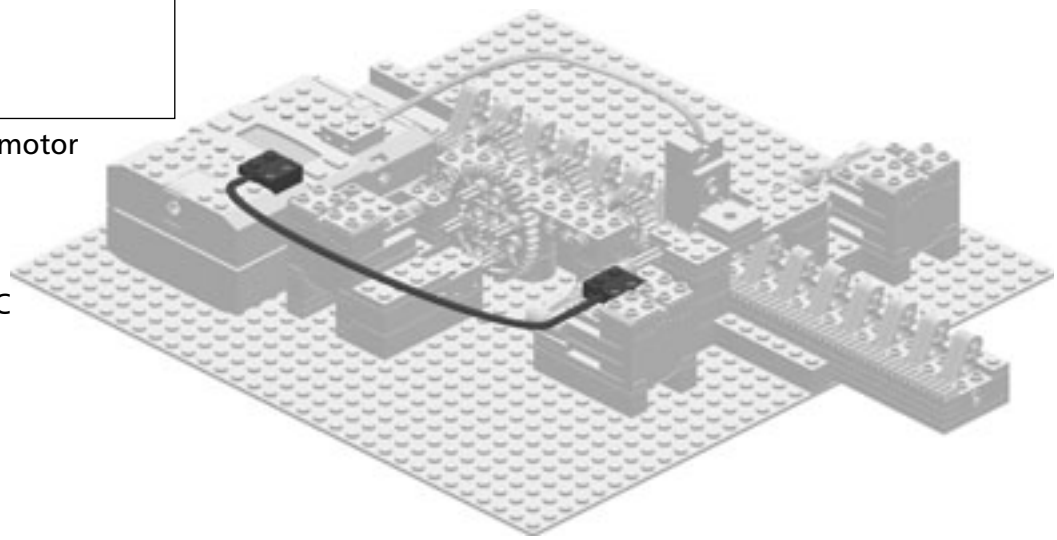
The Light Sensor sub-assembly is mounted over the Erase Switch sub-assembly (writing) sub-assembly.
Attach the Connector to Input Port 3 on the RCX.



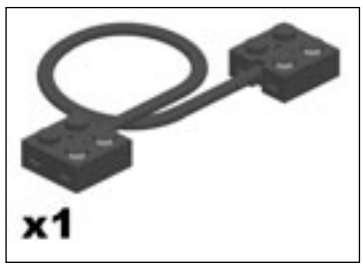
Final Step 7



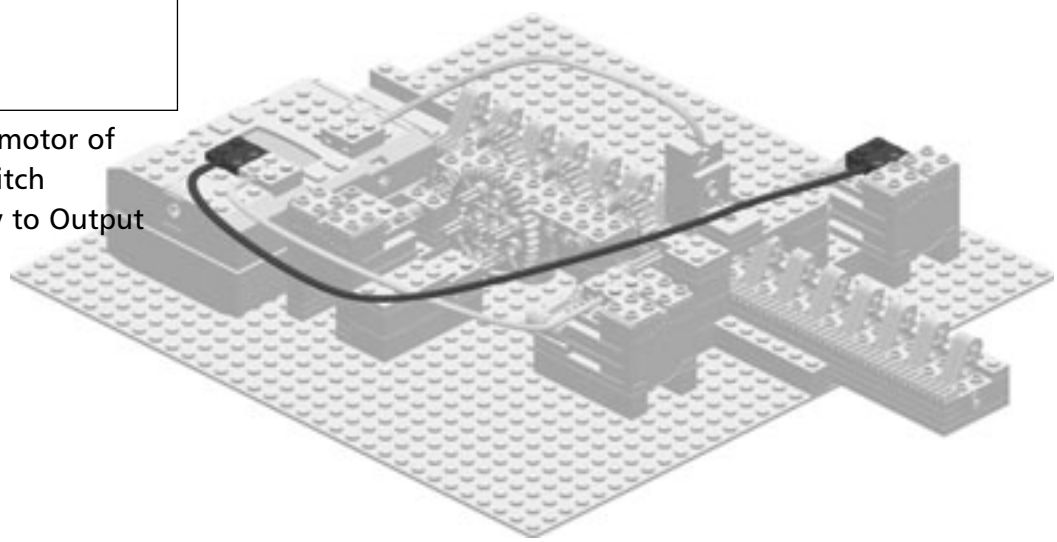
Connect the motor of the Write Switch sub-assembly to Output Port C of the RCX.



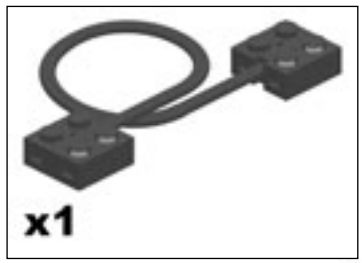
Final Step 8



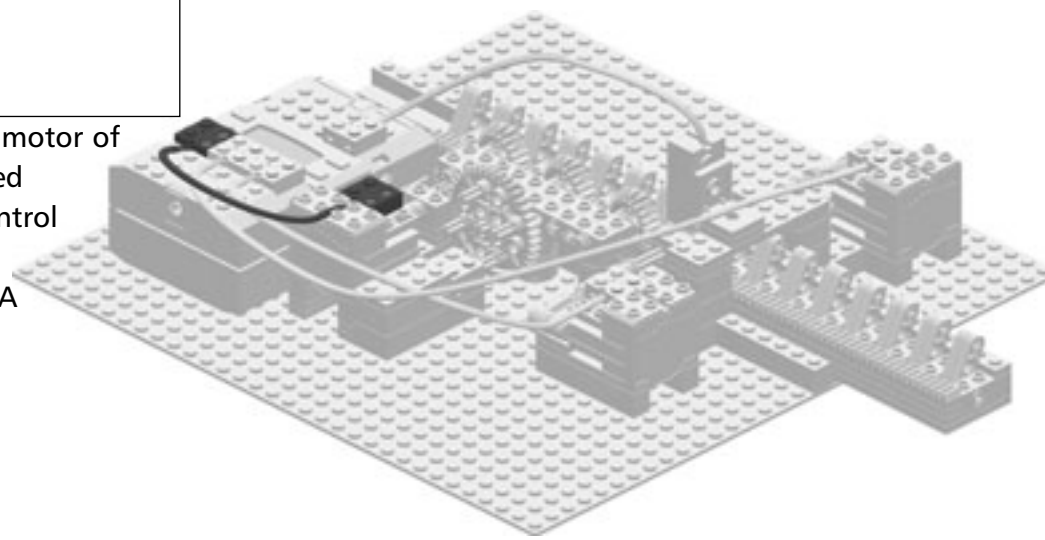
Connect the motor of the Erase Switch sub-assembly to Output Port B of the RCX.



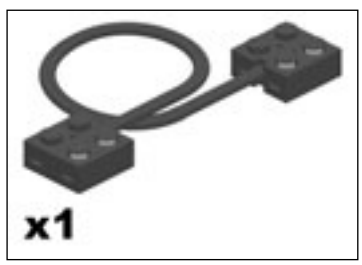
Final Step 9



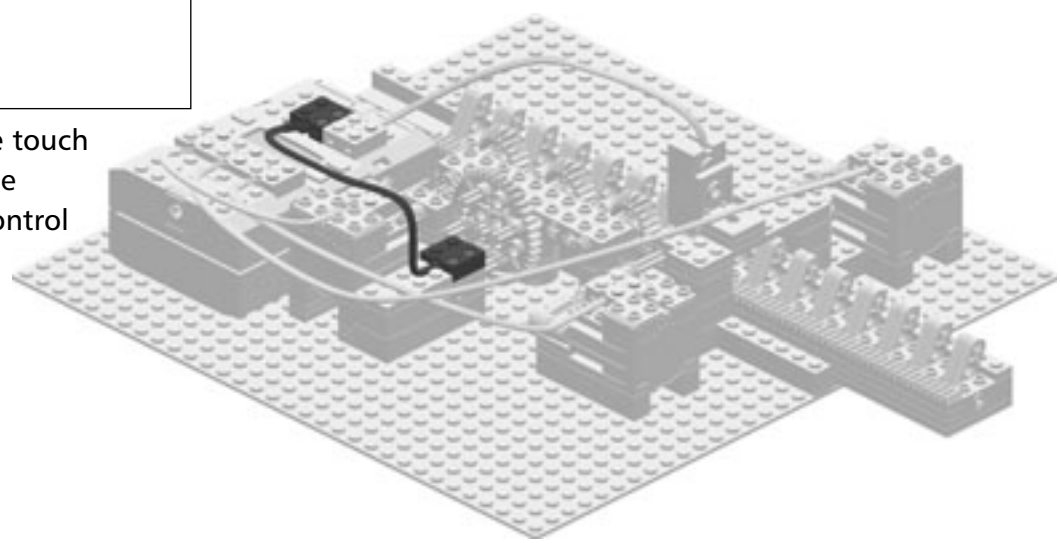
Connect the motor of the completed Direction Control Module to Output Port A of the RCX.

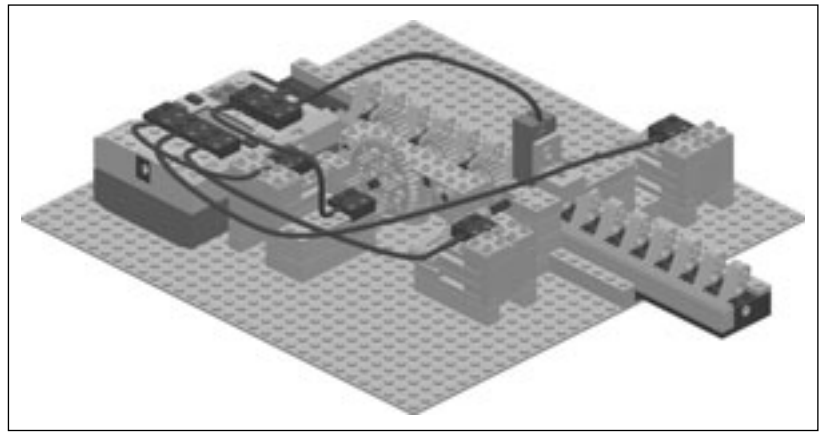


Final Step 10



Connect the touch sensor of the Direction Control Module to Input Port 2 of the RCX.





Programming the Machine

In this section, we will see how we can actually give "intelligence" to our Turing Machine. We will take the first example shown in this chapter. That is, making a machine that adds two integers and translates them into NQC code.

NOTE

The program for the operation of the LEGO Turing Machine can be found on the CD-ROM that accompanies this book.

We will follow good programming practice and split the programming into separate procedures. We will examine them one by one. There are multiple procedures, identified by the keyword *void*, meaning that they don't return any value, and a main program called *task main()*.

First, we should define two variables:

- One to store the current state the machine is in, indicated by a number
- The other used by the program to tell when it has finished its job and should stop (I choose to name it "done" because "stop" is a reserved keyword in NQC).

In NQC there are no Boolean variables to handle a True/False value, so the latter also should be declared as *int*. Both are initialized to zero.

```
int state = 0;
int done = 0;
```

Some setup must be done to tell the RCX that we will be using two sensors: a touch sensor for the step-by-step movements, and a light sensor to read the symbol inside the cell. Here we set motor's power to the maximum, and finally take advantage of a new feature of the RIS 2.0 firmware (version 0328) and set the RCX's LCD display to always

show the value of the *state* variable. This is very interesting since, when running the machine, we can observe and know in what state the machine is. This will also be very useful when debugging the program.

```
void setup()
{
  SetSensorType(SENSOR_2, SENSOR_TYPE_TOUCH);
  SetSensorMode(SENSOR_2, SENSOR_MODE_RAW);
  SetSensorType(SENSOR_3, SENSOR_TYPE_LIGHT);
  SetSensorMode(SENSOR_3, SENSOR_MODE_RAW);
  SetPower(OUT_A+OUT_B+OUT_C, OUT_FULL);
  SetUserDisplay(state,0);
}
```

Now we need to write a procedure to control the movement of the Tape sub-assembly one cell at a time. The motor of the Direction Control sub-assembly should be activated in either direction; then it has to run until the gray peg disengages the touch sensor and another gray peg engages it again. To do this, we've created two constants at the very beginning of the program:

```
#define TOUCH_LOW_THRESHOLD 500
#define TOUCH_HIGH_THRESHOLD 1000
```

The motor should stop only after reading a value higher than the *TOUCH_HIGH_THRESHOLD* and a value lower than the *TOUCH_LOW_THRESHOLD*. I chose to define these two values as constants, so they are easily modifiable and simple to determine the values that are acceptable for your sensor. (Remember that motors and sensors can be very different from one another, so you need some way to adjust the reading).

For this particular program, I chose to have the touch sensor use RAW values, instead of the usual Boolean mode. There are mainly two reasons for this choice. First, every sensor is different from the others, and the touch sensor makes no exception. In addition, a touch sensor doesn't have only two positions, on and off, but rather a series of intermediate values. To handle a movement that has to be very precise, like moving exactly one cell of the tape, you need to be able to control the reading in a very precise way.

Note that the values 500 and 1000 are just examples for two values that should be on the low part and high part of all the possible readings of the touch sensor (that usually range from around 250 to 1020). They should be just fine, but you may have to calibrate them to your system with the help of the View button on the RCX.

Here are the procedures to move one cell to the right or to the left:

```
void move_right()
{
  // run motor until the touch sensor engages a new peg
  OnFwd(OUT_A);
```

```
  while (SENSOR_2 < TOUCH_HIGH_THRESHOLD) {};
  while (SENSOR_2 > TOUCH_LOW_THRESHOLD) {};
  Off(OUT_A);
}
void move_left()
{
  // run motor until the touch sensor engages a new peg
  OnRev(OUT_A);
  while (SENSOR_2 < TOUCH_HIGH_THRESHOLD) {};
  while (SENSOR_2 > TOUCH_LOW_THRESHOLD) {};
  Off(OUT_A);
}
```

Note that when looking at the machine with the RCX close to you, which is supposed to be the front side, if you run the *move_right* procedure, the tape actually move leftward, and vice versa! This may sound a bit confusing, but in the original machine designed by Alan Turing, what is actually moving is the head, and not the tape. Therefore, we've kept the concept that "move right" means "move one cell to the right."

As you've seen, the Tape sub-assembly is built with multiple cells that contain two adjacent yellow liftarms as a switch. We choose to consider that if the liftarm is "down" (touching the worm gears), the cell is empty; if it is "up," the cell contains a symbol. Using the up and down switches, we can control the writing and erasing of the cells on the tape. A very short impulse to the motor is enough to let the liftarms pass from one side to the other. Another short impulse in the opposite direction resets the small arm (axle) to its original position.

```
void write()
{
  OnRev(OUT_C);
  Wait(5);
  OnFwd(OUT_C);
  Wait(5);
  Off(OUT_C);
}
void erase()
{
  OnFwd(OUT_B);
  Wait(5);
  OnRev(OUT_B);
  Wait(5);
  Off(OUT_B);
}
```

The main part of the program includes the task to coordinate the other functions and implement the state transition table that is the key to the operation of the Turing Machine. In NQC, there's no better way to turn the table into code than by using the *switch…case* and the *if* statements. The *switch* structure checks the state of the machine and the *if* statement checks if a symbol is read on the tape. A convenient way to handle sensor reading is to define a new constant: *#define LIGHT_THRESHOLD 750*

If the light sensor reading is higher than the *LIGHT_THRESHOLD* then the liftarm is in its upper position and the cell contains a symbol, otherwise the cell is empty. You will probably have to calibrate the threshold value for your configuration and your light conditions. Use the View button to discover the highest reading (with the liftarm below the sensor) and lowest reading (with liftarm in the other position) and find the mean between the two: this is your threshold. My sensor reads from 720 to about 780, so I calculated: (720+780)/2 =750.

```
task main()
{
  setup();
  do
  {
    switch(state)
    {
      case 0:
        if (SENSOR_3 < LIGHT_THRESHOLD)
        {
          write();
          state = 1;
        }
        move_right();
        break;
      case 1:
        if (SENSOR_3 < LIGHT_THRESHOLD)
        {
          move_left();
          state = 2;
        }
        else
          move_right();
        break;
      case 2:
        erase();
        done = 1;
```

```
            break;
        }
    Wait(100);
    }
    while (done != 1);
    Float(OUT_A);
    PlaySound(SOUND_UP);
}
```

After defining the machine's general setup, the program starts using the state transition table and the sensor reading to decide what to do (change state, move, write symbol), until it has finished its job and the *done* variable is set to 1 (true).

Now that the program is finished, you can test the Turing machine you just created! As you can see, the programming wasn't too difficult. Remember: You can adjust the robot's behavior by modifying the three constants we've defined. And if you want to change the state transition table, you just have to recode the *switch…case* statement of the main task, and leave the rest as is.

Off and Float Modes

Note that before emitting a happy sound to let you know that everything's over, Motor A is set to Float. This is very useful if you want to manually move the tape without having to switch the RCX off. If the motor is simply set to Off it will remain connected and braked and trying to move the tape would be hard to do, and would risk damaging some piece. If you put it in a Float state, it will be free to move and we can easily modify the values of the cells and prepare the machine for a new computation. Other methods include the use of a remote, or the so-called console mode. For additional information on console mode, refer to Chapter 2.

Operating the Turing Machine

Running the programmed Lego Turing Machine is very simple. You just have to put on the tape the combination of symbols that you want to calculate with your program. The cells are considered full (with a symbol) when the liftarm is in its upper position; that is, opposite to the direction control mechanism, and vice versa. According to Turing's definition, the machine starts operating with the leftmost cell of the tape placed in the read head (as described in "The State Transition Table" section in the beginning of the chapter.)

You may want to try the program we just described that adds two numbers.

1. Load the program into the RCX.
2. "Write" two numbers by hand on the tape, separated by an empty cell.
3. Place the head on the first symbol on the left.
4. Start the program by pressing the **Run** button. Remember not to move the tape if the motor is in the "Off" state.

See if everything's working or if it needs some debugging. The machine should go through the all steps shown in Table 3.2, and at the end, the tape should look like Figure 3.4, showing the result of the addition.

A More Complex Behavior

What we've seen so far is how to program our LEGO Turing machine to perform quite a simple task: adding two numbers. It's time now to see how to write a more complex program.

Let's consider the problem of making a Turing machine that, given an initial sequence of N symbols that represent an integer as already seen in the previous example, leaves the tape with the result of the operation "N div 2." Div is the operator that stands for an integer division (Table 3.3 shows some examples of inputs and outputs).

Table 3.3 Input and Outputs for the "N div 2" Example

Sequence	Result
oo	o
ooo	o
oooo	oo
ooooo	oo

There is more than one way to program a Turing machine, and you can always approach the problem from different points of view. In this case, I've chosen this strategy:

1. Start scanning the sequence, from left to right.
2. If the sequence contains a single symbol (o), the head erases it and stops.
3. If at least two symbols are present, both are erased and a new symbol is written on the right.
4. Return to the leftmost symbol and start again from Step 1.

Let's try and write a state transition table for the task just described. As you can see in Table 3.4, the programming is far more complex than the first example we discussed at the beginning of the chapter. Even simple problems on the Turing machine often require a solution that's not so simple and intuitive at all.

Table 3.4 A Transitions Table for Solving "N div 2"

Input		Output		
State	**Read Symbol**	**Operation**	**Move Direction**	**Next State**
0	o	Erase	Right	1
0	<Empty>		Stop	
1	o	Erase	Right	2
1	<Empty>		Stop	
2	o		Right	
2	<Empty>		Right	3
3	o		Right	
3	<Empty>	Write: o	Left	4
4	o		Left	
4	<Empty>		Left	5
5	o		Left	
5	<Empty>		Right	0

State 0 is used to erase the first symbol on the tape, while State 1 erases the second symbol, if present. If the sequence contains only one symbol, the machine stops in this state on the empty cell. State 2 moves the head right until it finds an empty cell, which marks the end of the input sequence. State 3 writes a new symbol on the right side, and State 4 returns to the left passing the output sequence. State 5 checks for any other left symbols remained, and sets the machine to start again from State 0, or stop.

In Table 3.5, we see an example of the steps required to solve the problem for a sequence of four symbols.

Expanding the System

Once you have performed a few experiments with your LEGO Turing Machine, you may want to go even further: expand its possibilities or modify the structure according to your preferences. Many improvements can be made to the model.

First, we've just built a tape that is capable of handling two symbols at a time: A cell can be empty or full. To allow the solution of more complex problems, like the very complex task of multiplying two numbers, the possibility of having more symbols can be very useful. In this case, a different solution must be found, like having pieces of different colors read by a light sensor (three colors are easily read by the sensor, but more often result in wrong recognitions). You can also stick to using a mechanical switch, with three or four possible positions, but using two light or touch sensors.

Table 3.5 Steps for Solving the "N div 2" Problem

State	Tape										
0		O	O	O	O						
1			O	O	O						
2				O	O						
2				O	O						
2				O	O						
3				O	O						
4				O	O		O				
5				O	O		O				
5				O	O		O				
5				O	O		O				
0				O	O		O				
1					O		O				
2							O				
3							O				
3							O				
4							O	O			
4							O	O			
5							O	O			
0							O	O			
Stop							O	O			

A more complex approach would consist of having the machine programmable from outside, and not from software that you have to recode every time. To do this, you should build a state transition table with some pieces like plates of different colors, and a reader that interprets and store the information. This is quite a difficult task to do, and you'll need a second RCX unit dedicated to this job, because at least two motors and one sensor are needed.

You can also try to build the same machine with fewer resources. For example, you could save one motor by using only one for the entire switch mechanism. This requires a complex structure in which running the motor in its two directions operate one switch or the other. The return of the pushing arm could be helped by using a small rubber band.

Another addition could be a mechanism to write and erase symbols on the tape (for its initial setup) without having to manually act on the tape. You should use buttons or

switches to control the movement and the write/erase functions, modifying the software accordingly, or using a specific program for this.

NQC is a wonderful language because it's easy to learn, powerful and very widespread. However, there are other languages—like leJOS (the LEGO version of the Java interpreter) or BrickOS—that would probably be much better than NQC to program a Turing machine. These alternate languages have more complex data structures and more flexible functions, so you can work on a different way to implement the state transition table, for example.

NOTE

For more information regarding Java Turing Machines and other software:

- www.nmia.com/~soki/turing
- www.nmt.edu/~prcm/turing
- www.turing.org.uk/turing/scrapbook/tmjava.html
- www.cheransoft.com/vturing

These are only a few ideas. Remember that LEGO is beautiful because everyone has a different way to approach and solve a problem, so take these tips only as simple suggestions that you can expand and revise as much as you want.

Summary

In this chapter, we took a journey into the very origins of AI, building a revolutionary device created to study computability: the Lego Turing Machine.

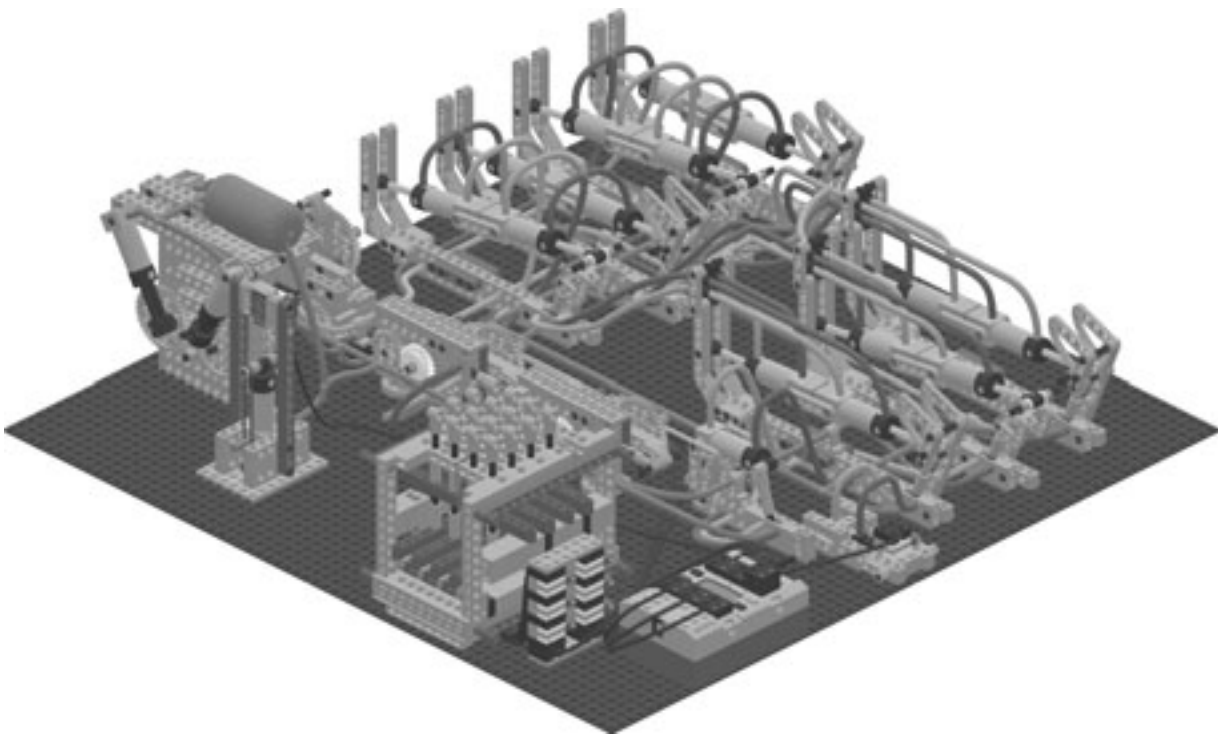
The purpose of the chapter was to see how you could create structures that contain data, like the tape and its modifiable cells, and handle reading and writing of symbols on it, using appropriate devices. We also learned how the state transition table, the "intelligent" part of the model, could be coded with a programming language and used to control the entire behavior of the machine.

Even though the LEGO Turing Machine may seem a fairly simple model, it covers many of the aspects that distinguish a true masterpiece: unconventional building solutions, a particular way of programming, expansion possibilities, and a model that is really beautiful to see in action.

Masterpiece 4

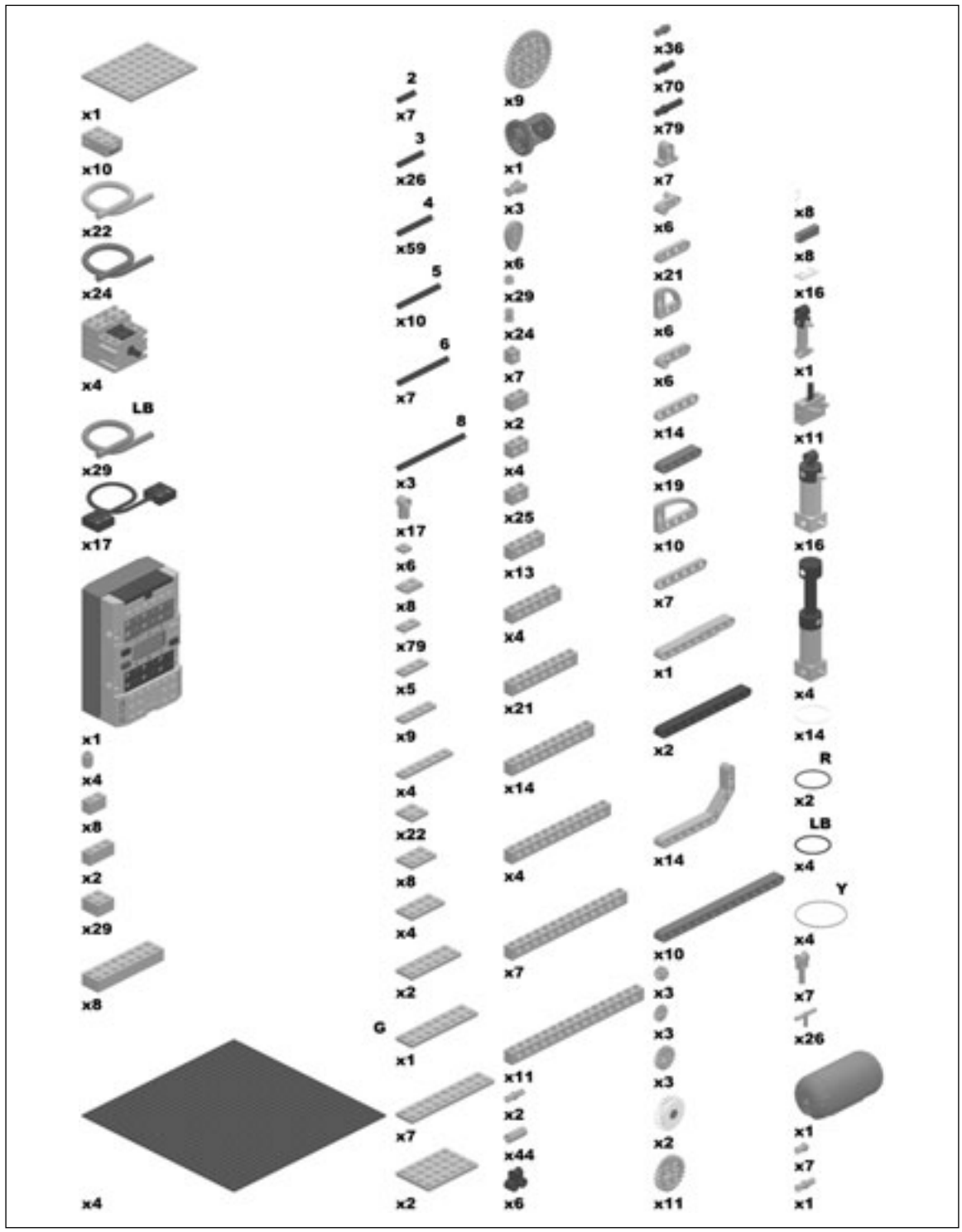
PneumADDic II

Kevin Clague



Bill Of Materials

These are the pieces that you will need to build PneumADDic II.



Introduction

This chapter describes PneumADDic II, my LEGO pneumatic adding/subtracting machine. PneumADDic II combines a MINDSTORMS Robotics Invention System RCX with motors, touch sensors, and LEGO TECHNIC pneumatic parts to make a device that mechanically adds and subtracts series of numbers by using pressurized air to manipulate a circuit. The fact that PneumADDic II performs these addition and subtraction functions mechanically, rather than digitally, is what really makes it unique. These days, these types of functions can be performed digitally, almost instantaneously, by the simplest of computers. The RCX itself can easily do so, as can any child's calculator. Why then, you might be asking yourself, would you build a machine that performed these functions mechanically, when the RCX brick is capable of performing the same calculations digitally? There are a couple of answers to that question.

When I first discovered LEGO pneumatics, I became very excited. My thought was that I would be able to build complex moving devices that operated on only compressed air. I've been a computer engineer for 20 years, and know quite a bit about designing computer circuits. I assumed that I could use many of my computer design skills to create complex never seen before LEGO pneumatic devices. I consulted JP Brown, a world-renowned LEGO builder that I had met on the LEGO MINDSTORMS Web site, about what I wanted to do. He told me that computer-like pneumatic circuits were not possible, due to the way the LEGO pneumatic switches work. I took this as a challenge and set to work to prove that it was indeed possible.

A second answer to the question of why the idea of a LEGO pneumatic adding machine appealed to me so much goes back even further into my past. When I was introduced to digital electronics as a teenager, the first circuit that captured my attention was a digital electronic adding circuit, similar to what you would find inside the RCX or a calculator. The heart of PneumADDic II mechanical adding machine is a pneumatic adding circuit that mimics the digital electronic circuit I first learned about 30 years ago. The fact that my knowledge has come full circle in such a way is extremely fulfilling, especially since I have been able to use LEGO parts to turn my theories and visions into reality.

LEGO TECHNIC pneumatic parts provide all the basics elements for building complete pneumatic circuits. These include the following elements:

- **Pneumatic Pumps** These pressurize air, which is the driving force behind all pneumatic devices and circuitry.
- **Pressure Tanks** These store the pressurized air for later use.
- **Pneumatic Switches** These switches steer where the pressurized air goes.
- **Pneumatic Pistons** These are used as actuators in LEGO designs.
- **Pneumatic Tubing and T-Connectors** These elements connect all of the other LEGO pneumatic components together.

NOTE

An actuator is a mechanism that can drive itself or other things to move.

PneumADDic II combines MINDSTORMS electric motors with pneumatic pumps to create a compressor that provides the compressed air which drives the pneumatic adding circuit. Another interesting facet of PneumADDic II is that it combines motors with pneumatic switches so it can control the values provided to the pneumatic adding circuit, and uses pneumatic pistons combined with MINDSTORMS touch sensors, so the RCX can tell when there is enough pressure to perform an addition operation, as well as determine the result of the addition process. If that was not enough, PneumADDic II also combines pneumatic pistons with pneumatic switches to create the computing elements of the pneumatic adding circuit. Finally, a calculator is useless without an interface for inserting numbers therefore PneumADDic II combines eight MINDSTORMS touch sensors into a 16-key keyboard.

As you can tell from the name, PneumADDic II is not my first attempt at a LEGO pneumatic adding machine. My first adder, aptly named PneumaADDic, was also a success. It could perform pneumatic addition, but was very slow, because the pneumatic adding circuit required a significant amount of pressure to operate correctly. It took a long time for the pneumatic pumps, powered by MINDSTORMS electric motors, to generate enough pressure to perform the functions I was looking for. This was compounded by the fact that the computing devices (called *gates*) in the adding circuit for PneumADDic were more mechanically complicated than they needed to be. This created a lot of friction, which in turn forced PneumADDic to run at an excessively high level of pressure. PneumADDic also needed a great number of parts to build only one pneumatic gate, and I only had parts for three. Despite its weaknesses, PneumADDic was the first of its kind and was nominated into the LEGO MINDSTORMS Hall of Fame contest.

PneumADDic II incorporates a simplified gate design, and utilizes a total of seven gates in the pneumatic adding circuit. But, before we dive into the actual building instructions for PneumADDic II, we should take a closer look at both pneumatics and digital computing, as these are the two disciplines that lie at the heart of this machine's abilities.

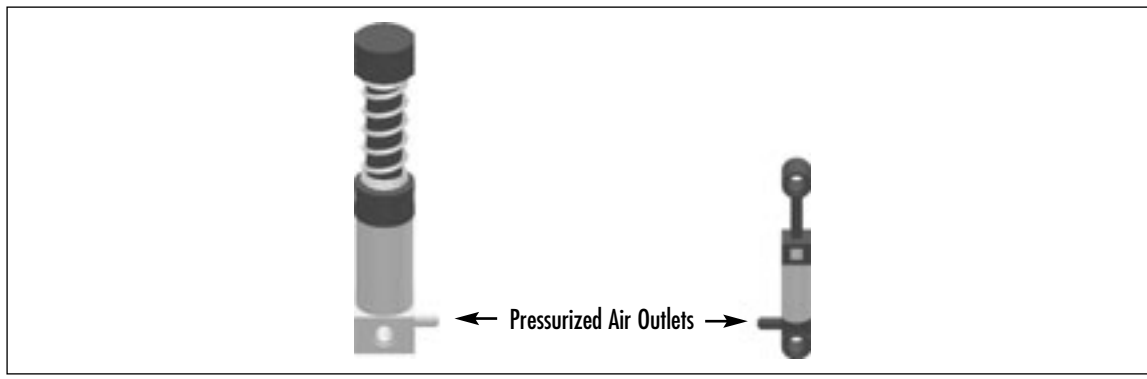
Pneumatics

As previously mentioned, pneumatic devices work by harnessing the power of compressed air. There are many examples of pneumatic devices in real life. Carpenters often use pneumatic nail guns when building or roofing houses, and pneumatic devices are commonly used in automated factories for bending and shaping metals. Hydraulics, a technology similar to pneumatics that uses pressurized fluid instead of compressed air, is

common in earth moving machines such as backhoes and bulldozers. However, probably the most common example of pneumatics in action is the tire on an automobile. Tires maintain their shape by storing pressurized air provided by an compressor.

Figure 1.1 shows the two types of pneumatic pumps available from LEGO: large pumps and small pumps.

Figure 1.1 LEGO Pneumatic Pumps



These pneumatic pumps have one pneumatic connector where air is forced out when the pump is compressed. This pressurized air is directed into a piston, which can be forced to expand (push the air out) and contract (suck the air in). The piston's movement, be it expansion or contraction, is used to deliver the power that drives a pneumatic device. Figure 1.2 shows the two kinds of LEGO pneumatic pistons available today.

Figure 1.2 LEGO Pneumatic Pistons

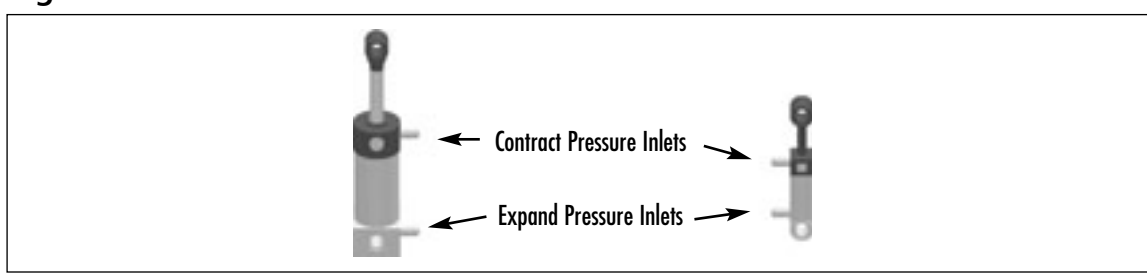


Figure 1.3 shows the function of a pneumatic switch that has a pressure inlet and two inlet/outlet ports that connect to a pneumatic piston. When the lever of the pneumatic switch is in the center, the pressure inlet is shut off, as are the two ports that connect to the piston. When the lever is flipped to the left, any pressure present on the left port is released to the atmosphere, and the pressure from the pressure inlet is forced out the right port into the piston. When the lever is flipped to the right, the pressure that was present at the right port is released to the atmosphere, and the pressure from the pressure inlet is forced out the left port into the piston.

Figure 1.3 Pneumatic Switch Positions

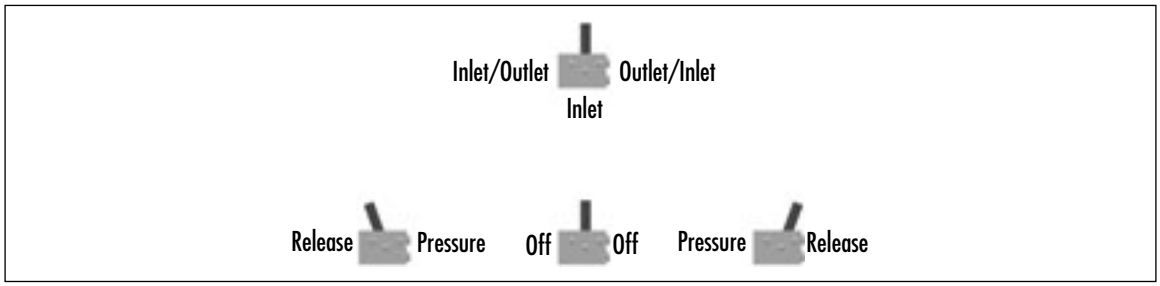
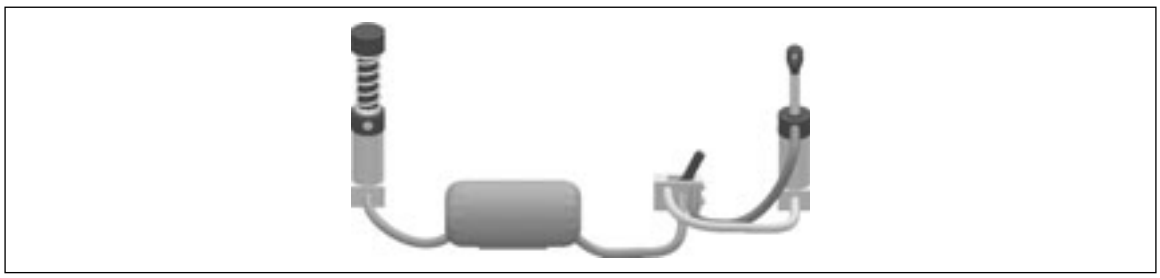


Figure 1.4 shows a complete pneumatic circuit starting with a pump that pressurizes the pressure tank. The pressure from the tank is fed to the pressure inlet on a switch. The inlet/outlet ports on the switch attach to a piston. LEGO makes three colors of pneumatic tubes:

- Blue
- Light gray
- Dark gray

Figure 1.4 A Pneumatic Circuit with an Expanded Piston



The most common method of attaching hoses is to use blue pneumatic tubes from the pumps to the tank(s), and from the tanks to the inlets on the switches. In Figure 1.4 I used light gray tubing to connect the left port of a switch to the base of the piston, and dark gray tubing to connect the right port of the switch to the top inlet of the piston.

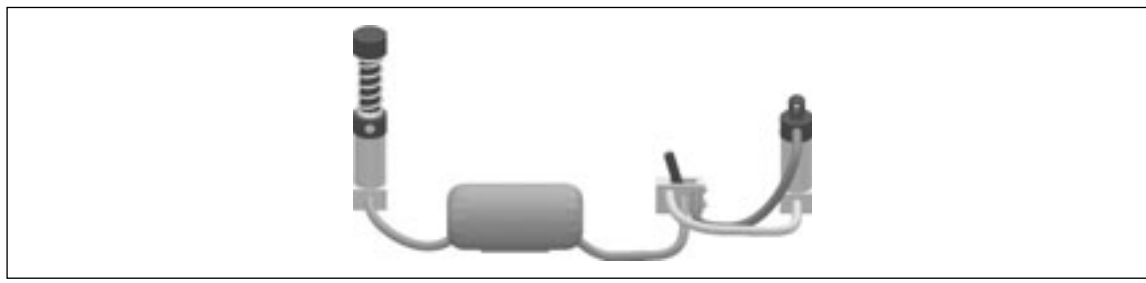
NOTE

When building PneumADDic II, use the hookup conventions described previously. The only exceptions are for special cases that are described in their corresponding steps.

In Figure 1.4 you can see that switch is flipped to the right, which means the pressure is forced from the left switch port and sent into the base of the piston, forcing the piston to expand.

In Figure 1.5 you can see that the switch is flipped to the left, which means the pressure is forced from right port of the switch and sent into the top inlet on the piston, forcing the piston to contract.

Figure 1.5 A Pneumatic Circuit with a Contracted Piston



Digital Computing

The computers and calculators that we use in our everyday lives compute and calculate using binary arithmetic. In the decimal arithmetic that we learned in school, a given decimal place can have one of ten possible values, 0 through 9. In binary arithmetic, a given binary place can have one of two values: 0 or 1.

In decimal mathematics, when we count up from 0, we go through the values, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, until we've used all the possible values, at which point we need to add 1 to the tens place and start over at zero in the ones place (the jump from 9 to 10). Once we've exhausted all values of the ones and tens places, we need to expand up into the hundreds place (the jump from 99 to 100).

In binary mathematics, instead of having ones, tens, and hundreds places, we have binary places, 1, 2, 4, 8, where each new place represents two times the value of the previous place. We can convert the decimal numbers we are familiar with to binary as shown in Table 1.1.

Table 1.1 Explanation of Decimal to Binary Conversion

Decimal	Binary	Explanation
0	0	
1	01	1 * 1 = 1
2	10	2 * 1 + 1 * 0 = 2
3	11	2 * 1 + 1 * 1 = 3
4	100	4 * 1 + 2 * 0 + 1 * 0 = 4
5	101	4 * 1 + 2 * 0 + 1 * 1 = 5

Binary addition is similar to decimal addition in that you start at the ones place, and work your way higher places carrying as needed, until you run out of numbers to add. For example, we want to add decimal 5 to decimal 7 as described in Figure 1.6

Figure 1.6 Binary Addition

```
Binary     Decimal
  101        (5)
 +111        (7)
----------
Binary     Decimal     Explanation
 111                   Carries
  101        (5)       Value 1
+ 111        (7)       Value 2
----------
  1100 = (8 * 1 + 4 * 1 + 2 * 0 + 1 * 0 = 12)
```

In each column, you need to be able to add three binary digits (often referred to as bits); a bit from Value 1, a bit from Value 2, and the Carry from the previous column. Table 1.2 shows all possible combinations of the three bits and the Sum and Carry. You can use Table 1.2 to perform the binary addition for each binary place column in Figure 1.4

Table 1.2 Computing the Sum and the Carry

Inputs Carry	Value 1	Value 2	Outputs Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Boolean Logic

A British mathematician named George Boole invented a branch of mathematics which became known as Boolean logic. Boolean logic deals with two valued number systems (binary), and defines a simple set of rules of Boolean computation. Boole defined three

operators: AND, OR, and NOT. The AND operator has two inputs and one output. With two binary values, you can have four possible value combinations: 00, 01, 10, 11. Table 1.3 shows the results of ANDing two values. In English, Table 1.3 states "Input 1 AND Input 2 must both be 1 to get 1 as a output, otherwise the output is 0."

Table 1.3 Truth Table for the AND Operator

Input 1	Input 2	AND
0	0	0
0	1	0
1	0	0
1	1	1

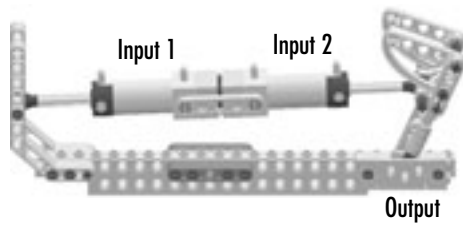
NOTE

In mathematical theory the AND, OR, and NOT operations are referred to as operators. When implemented in a mechanical design the device that performs these operations are referred to as *gates*.

The AND Gate

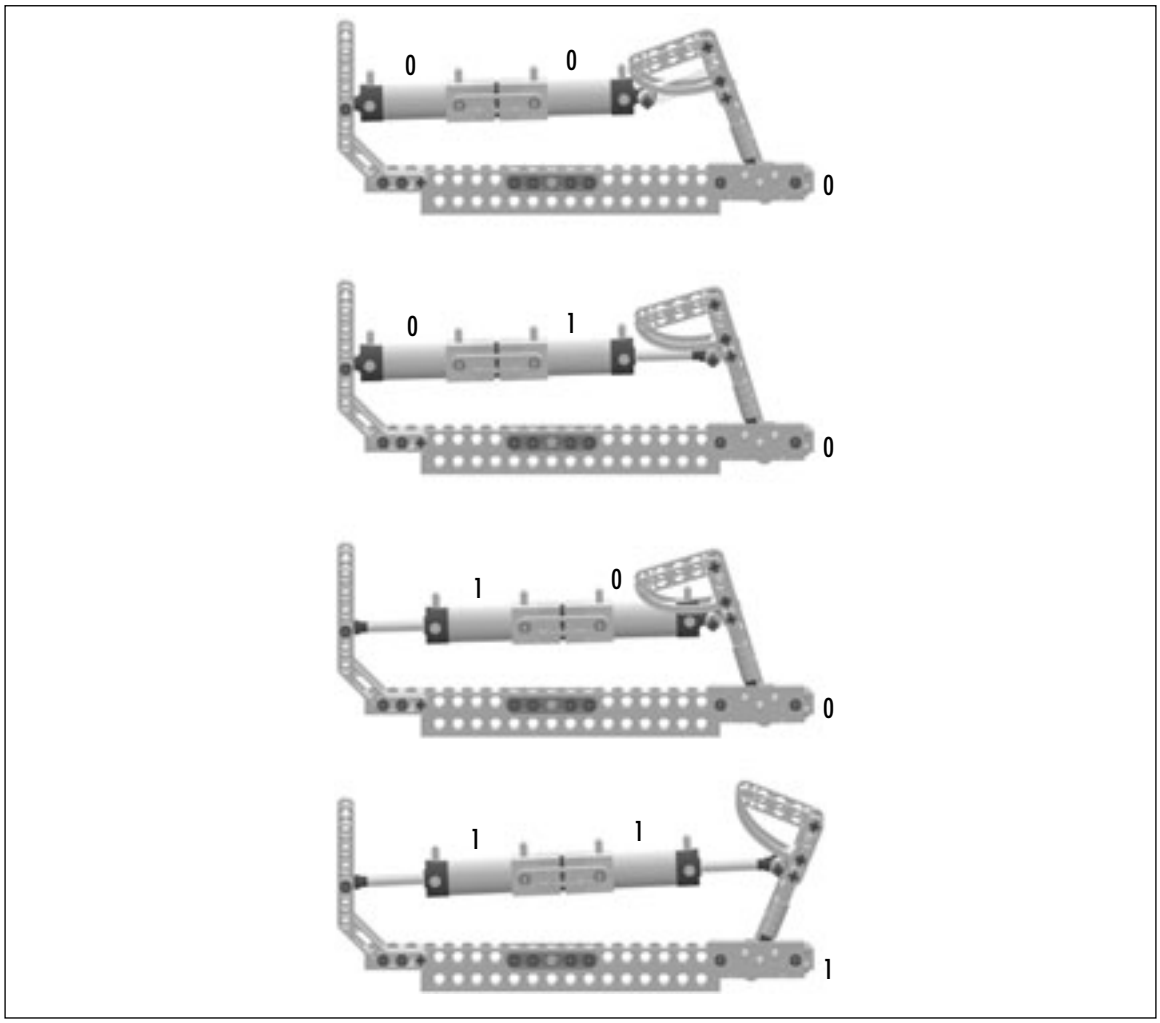
Let's take a moment and look at PneumADDic II's AND Gate sub-assembly, as can be seen in Figure 1.7. The AND Gate is composed of two pistons that act as inputs. The two pistons are connected to the pneumatic switch in the gate that plays the role of the output.

Figure 1.7 The Pneumatic AND Gate



In Figure 1.8, you can see four pneumatic AND Gate sub-assemblies showing each of the possible input states described in Table 1.3. Notice what happens to the handle of the pneumatic switch as the inputs change to different values. When both of the pistons are expanded, the switch is flipped to the right (what is considered the 1 position). In all other cases, the switched is flipped to the left (what is considered the 0 position).

Figure 1.8 Pneumatic AND Gate in All Possible States



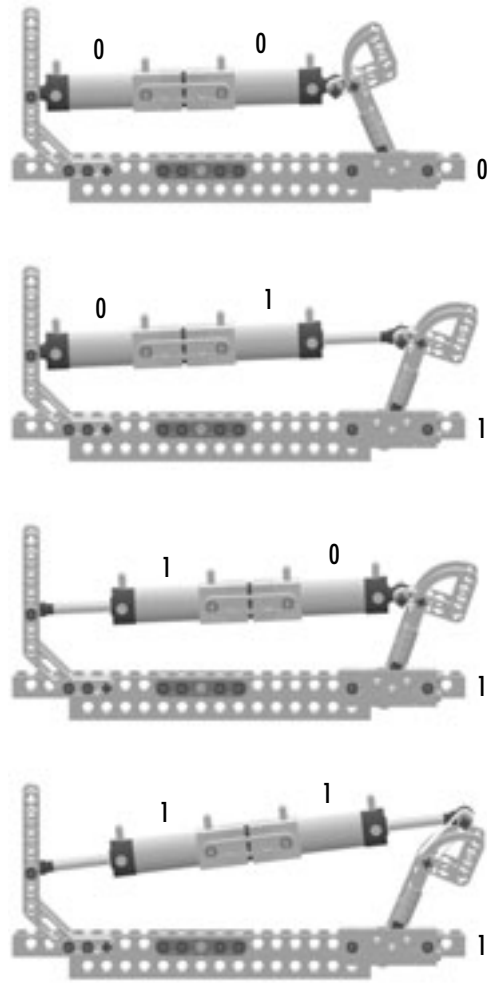
The OR Gate

The OR operator has two input values and one result. Table 1.4 shows all possible combinations of inputs to the OR operator, and the output result. In English, Table 1.4 states, "If Input 1 is 1 OR Input 2 is 1 (OR both), the output is 1. Otherwise the output is 0." Figure 1.9 shows the OR gate in all possible states. Like the AND gate, it uses two pistons as inputs, and a switch as the output. When either piston is expanded (in the 1 state), the pneumatic switch is flipped to the right (the 1 state), otherwise the switch is flipped to the left (the 0 state).

Table 1.4 Truth Table for the OR Operator

Input 1	Input 2	OR
0	0	0
0	1	1
1	0	1
1	1	1

Figure 1.9 Pneumatic OR Gate in All Possible States



The NOT Gate

The NOT operator has one input and one output. Table 1.5 shows the possible values for input to the NOT operator, and the resulting output. The NOT operator can be achieved

in pneumatics by hooking up the hoses from the output of one gate to the inputs of another gate reversed from the conventional way.

Table 1.5 Truth Table for the NOT Operator

Input	Not
0	1
1	0

In the gates of PneumADDic II when the pneumatic switch lever is flipped to the left, it means that the switch is outputting a 0, and when it is flipped to the right it means the switch is outputting a 1.

- When the lever is flipped to the right, the pressure is forced out the left port; in these situations, I always hook light gray tube to the left port of the gate's pneumatic switches and normally hook this to the base of the piston it is driving. This way the piston expands when the switch outputs a 1.
- When the lever is flipped to left, pressure comes out the right port; in these situations, I always hook the dark gray tube to the right port of a gate's switch and normally hook this to the top of the piston. When the switch outputs a 0, the piston contracts.

To NOT a value is to convert a 0 to 1, or 1 to 0. In the case of our pneumatics, if we want to NOT the input of a gate, we simply reverse the position of the previously described hoses to the piston. This is to say that we swap the light gray tube and dark gray tube connections from the standard previously described configuration. When the switch is outputting a 0, pressure is forced through the dark gray tube, and sent into the base of the piston causing it to expand. When the switch is outputting a 0, the pressure is forced through the light gray tube, and sent into the top of the piston, causing it to contract.

Before we start discussing how we combine these gates to perform binary arithmetic, let's take time to build the gates themselves so you can play with them and get a feel for how they work.

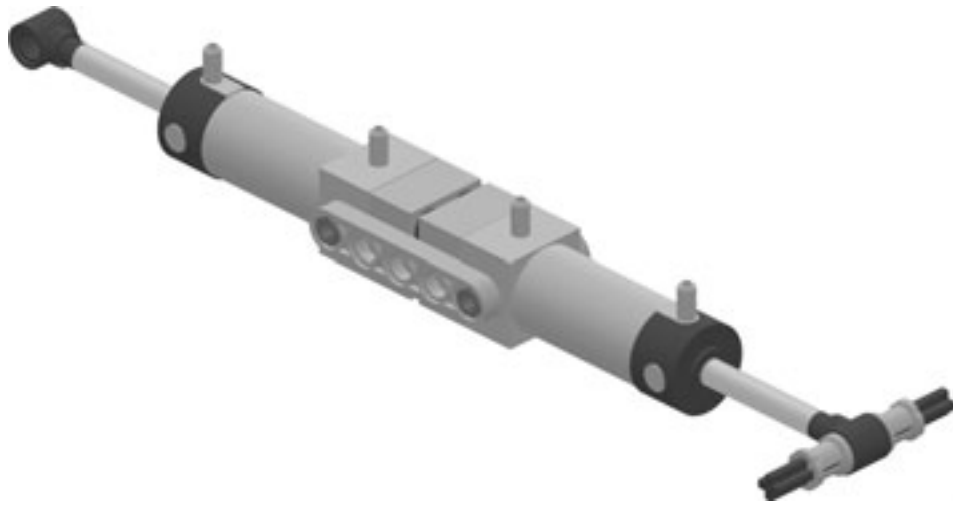
NOTE

To reiterate, in reversing the connections (pneumatic hoses) we can NOT the inputs of a gate. This is the reverse of the normal situation where light gray pneumatic tubing is used to make the piston expand, and dark gray pneumatic tubing is used to make the piston contract.

Building the Gates

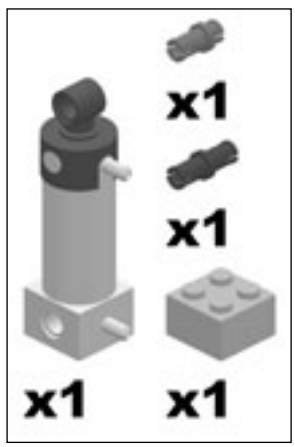
In this section, you will build both the AND Gate sub-assembly and the OR Gate sub-assembly. There are some similarities between the two kinds of gates, so we will start by building the common sub-assemblies used in both types of gates.

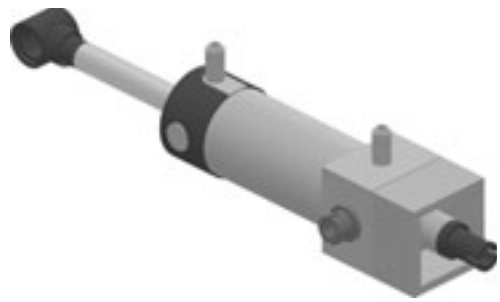
The Pistons



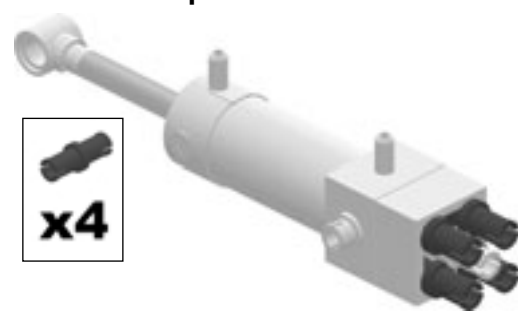
In the following steps, you will build a pair of joined pneumatic pistons that are the heart of the mechanical function of the pneumatic gates. You will need to build **seven** of these Piston sub-assemblies.

Pistons Step 0

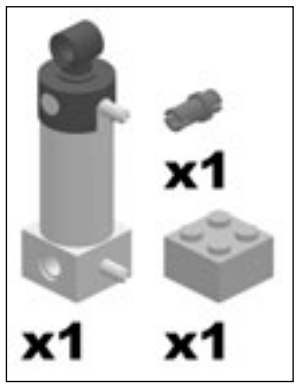


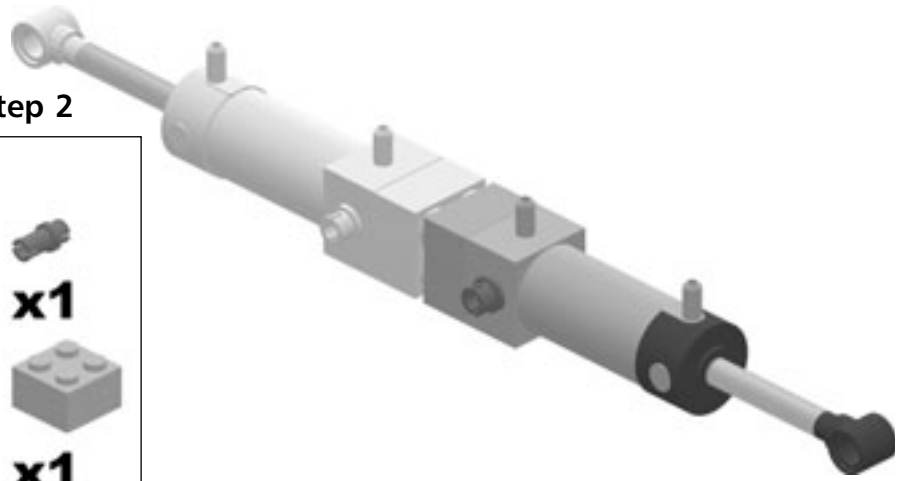


Pistons Step 1

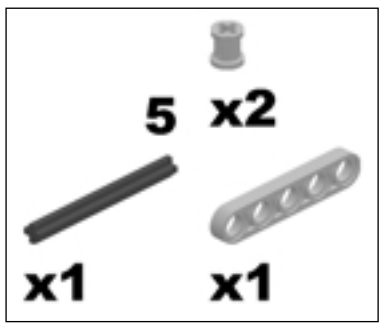


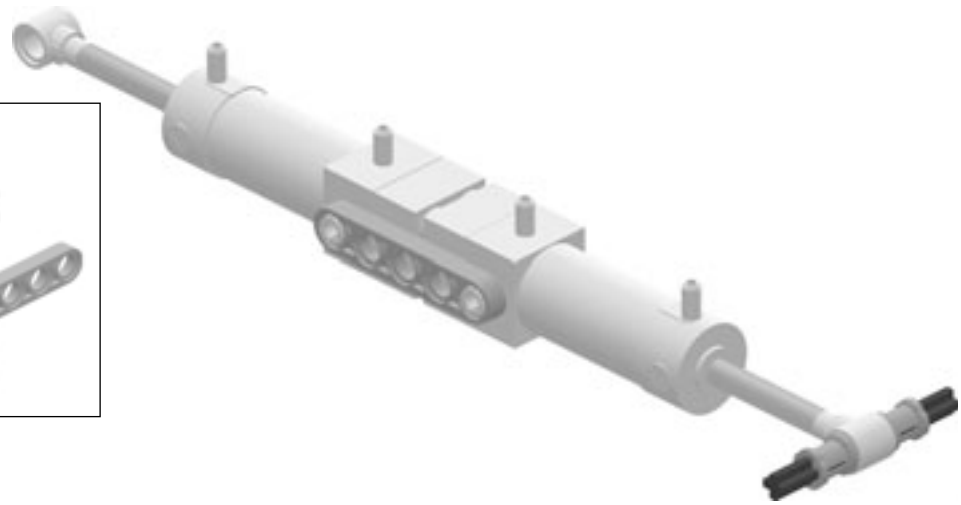
Pistons Step 2



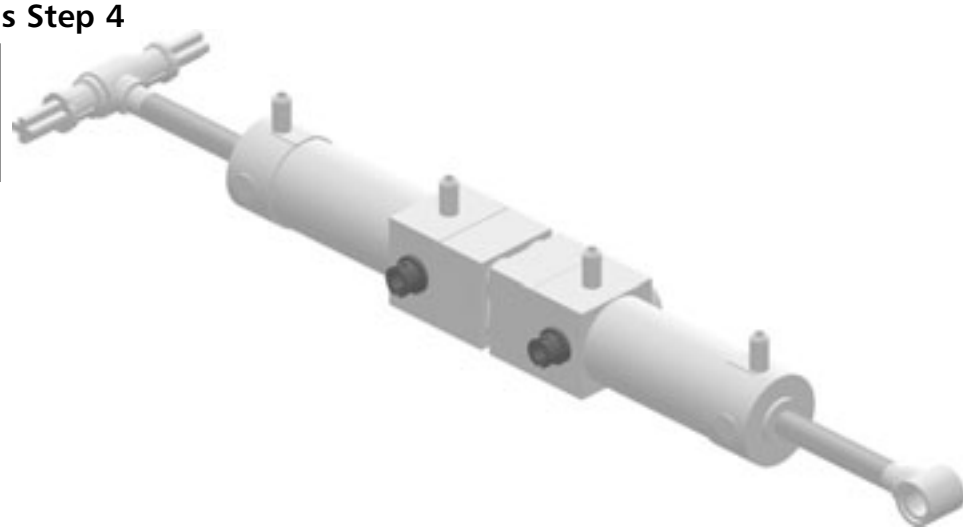


Pistons Step 3

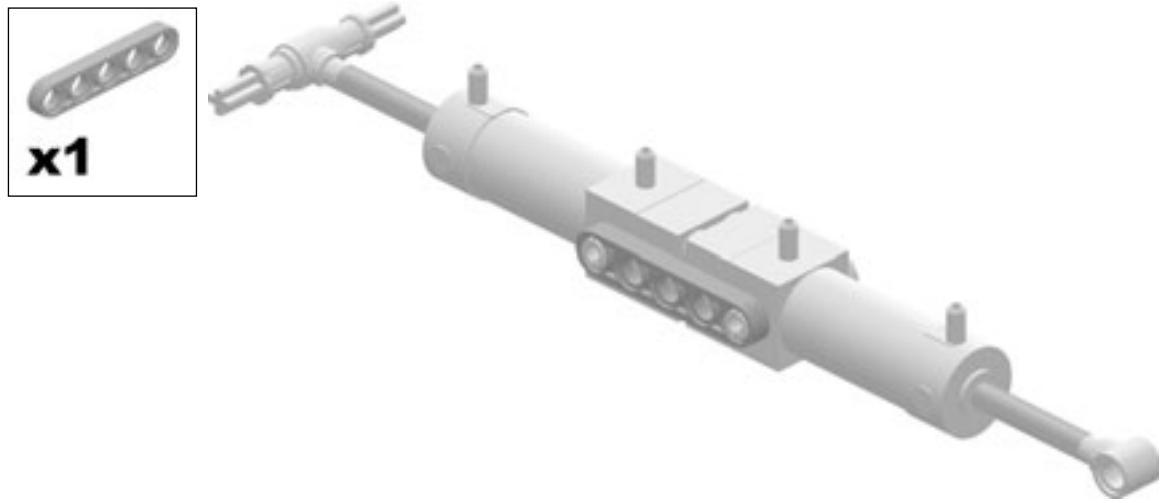




Pistons Step 4



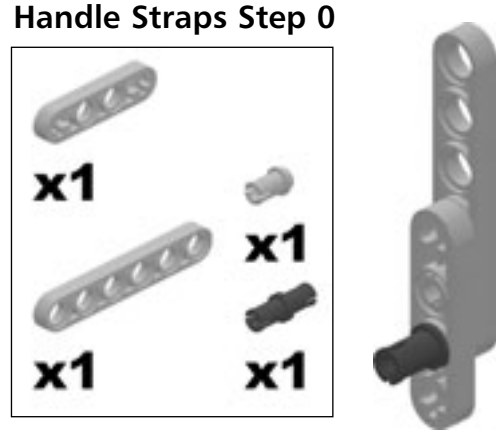
Pistons Step 5



The Handle Straps

The Handle Straps sub-assemblies prevent the pistons from pulling the handles off the pneumatic switches when the pistons on the OR Gate sub-assemblies are both expanded, or when the pistons on the AND Gate sub-assemblies are both contracted. You will need to build **seven** of these Handle Strap sub-assemblies.

Handle Straps Step 0

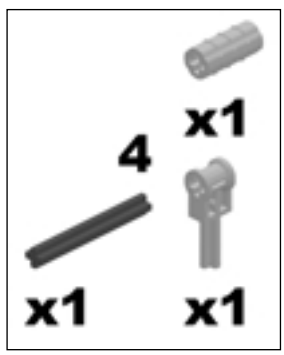


The AND Handles

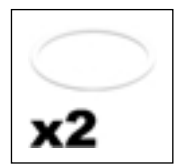
The AND Handle sub-assembly connects the Piston sub-assembly to the pneumatic switch for the AND Gate sub-assembly.

A key component of the gates' functionality is how the Piston sub-assemblies interact with the pneumatic switches. The AND Handle sub-assemblies you are about build, slip snugly over the lever of the pneumatic switches, and are hooked to the Piston sub-assemblies via a pair of rubber bands. The purpose of the rubber bands is to allow the pistons contract beyond the limits of the pneumatic switch without it. As the pistons contract beyond the limits of the pneumatic switch, the rubber bands stretch to compensate for the difference. In the case of the AND Gate sub-assemblies, you can see this when the inputs are 0, 0, as shown previously in Figure 1.9. You will need to build **four** of the AND Handle sub-assemblies.

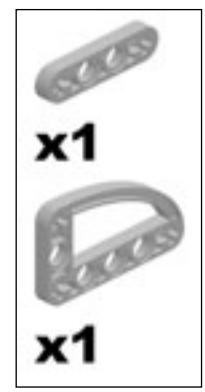
AND Handle Step 0



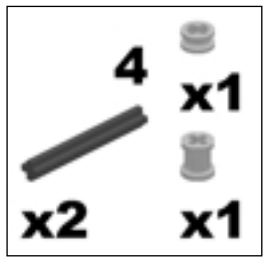
AND Handle Step 1



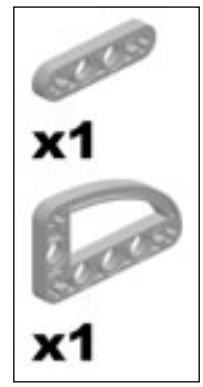
AND Handle Step 2



AND Handle Step 3



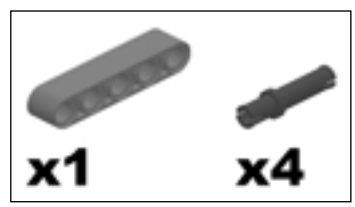
AND Handle Step 4



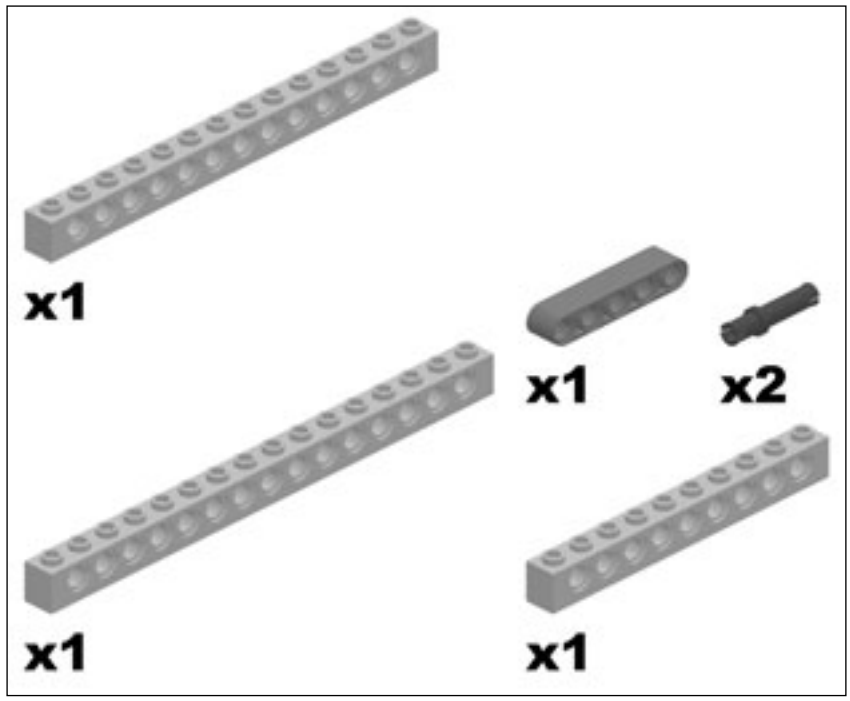
The AND Gates

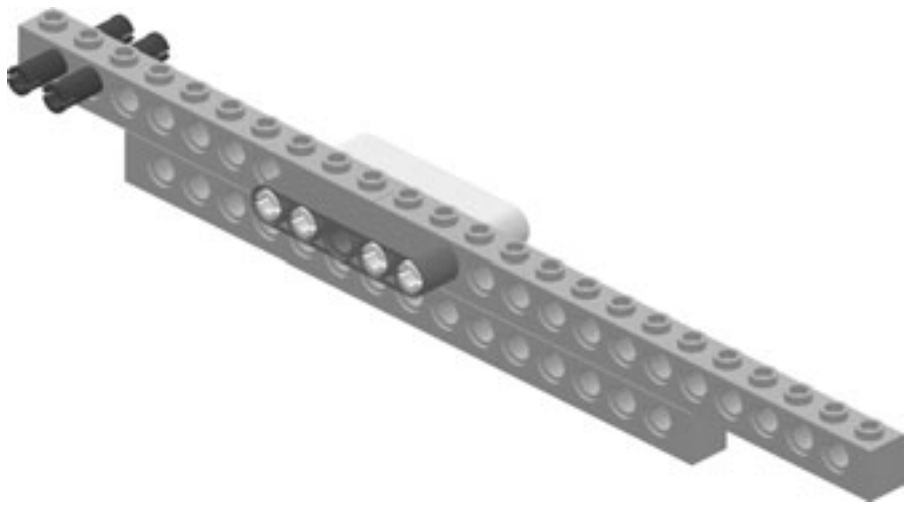
Now we have what we need to create the final AND Gate sub-assembly. When you finish the following series of building instructions, you should be able to manually expand and contract the pistons to see how they affect the position of the pneumatic switch. You will need to build **four** AND Gate sub-assemblies.

AND Gate Step 0

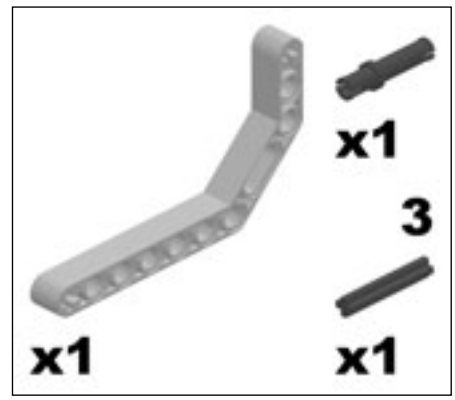


AND Gate Step 1



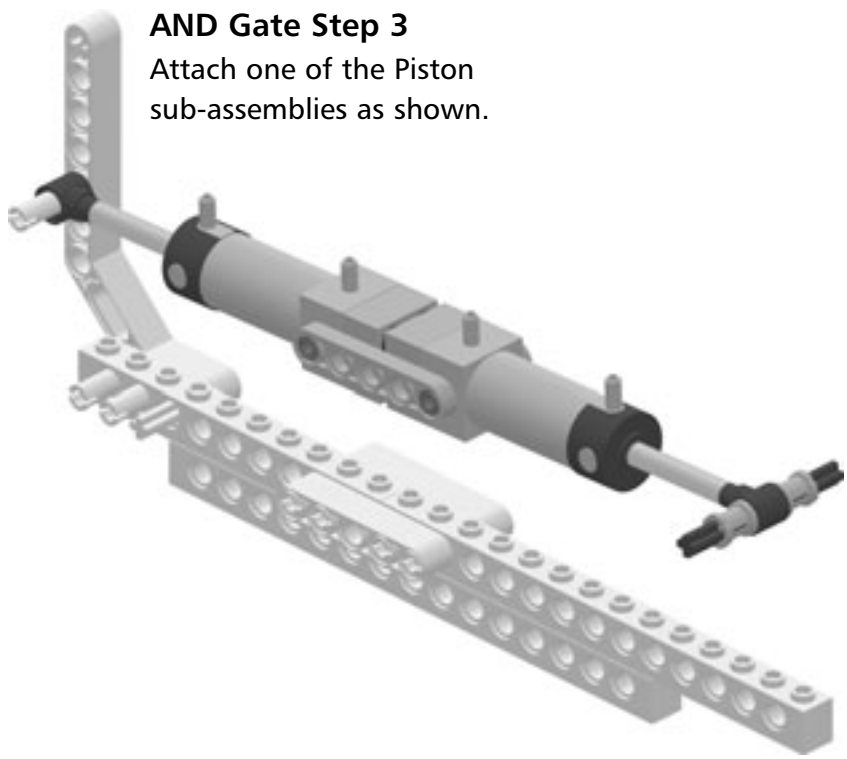


AND Gate Step 2

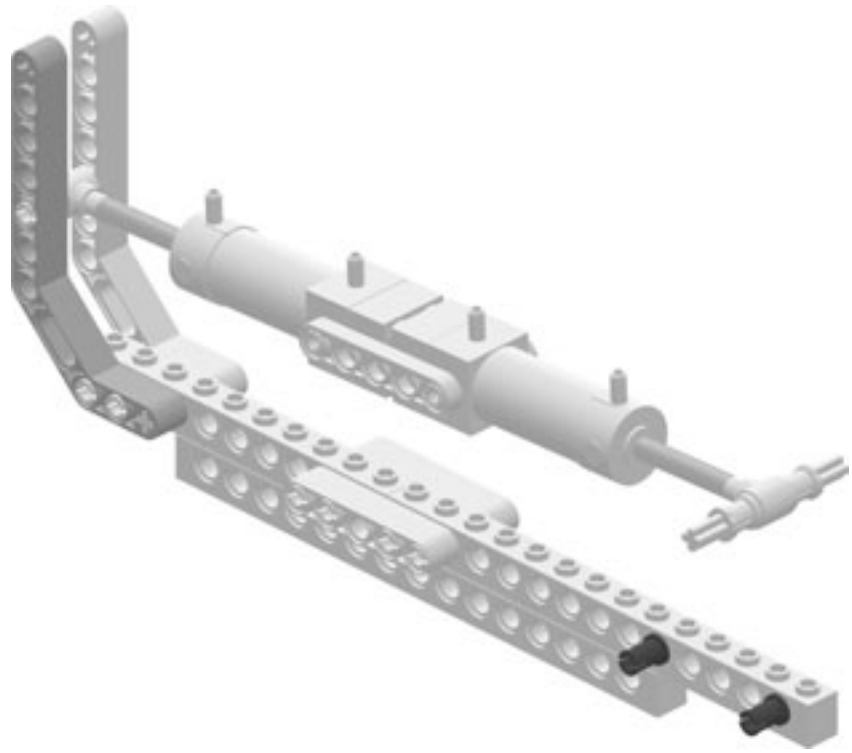


AND Gate Step 3

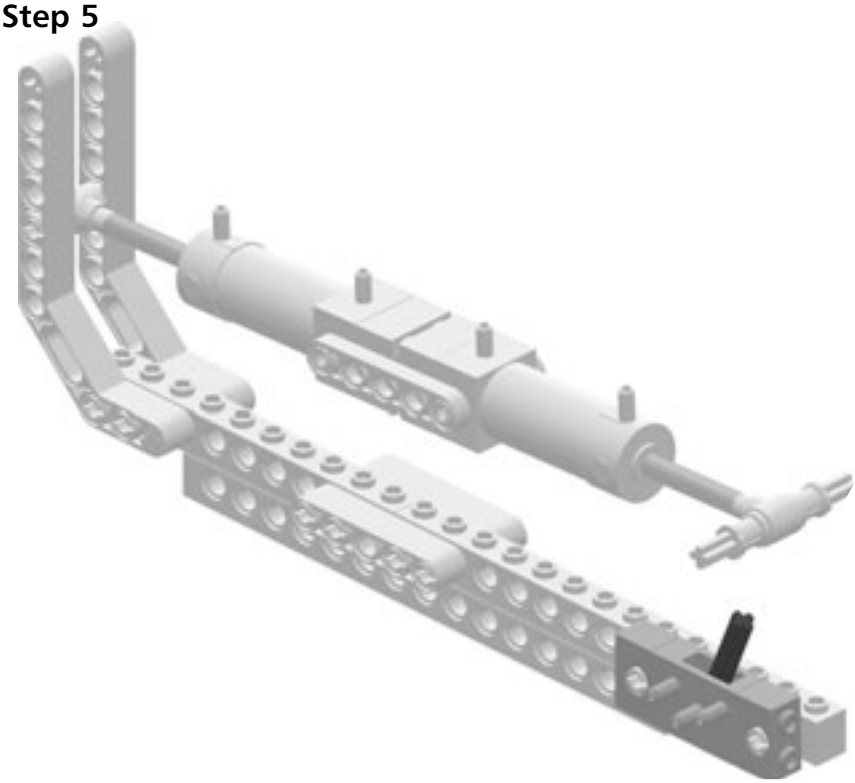
Attach one of the Piston sub-assemblies as shown.



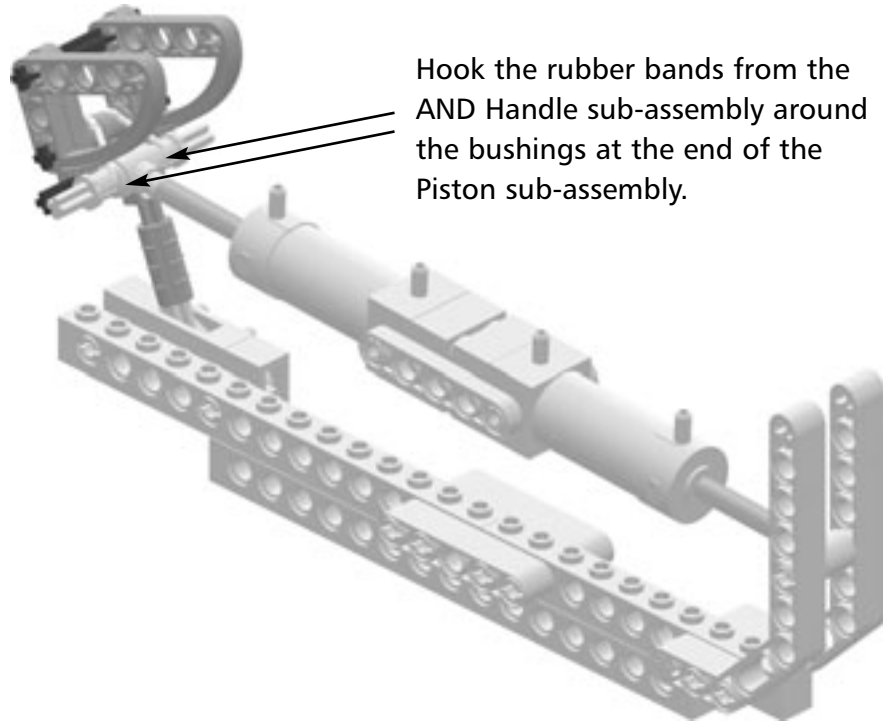
AND Gate Step 4



AND Gate Step 5

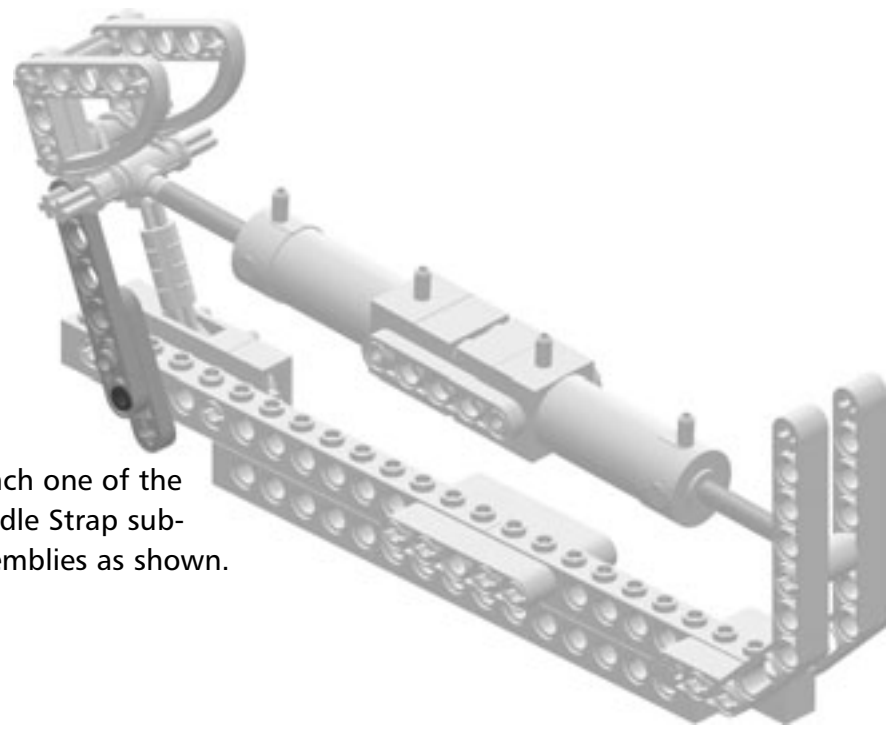


AND Gate Step 6



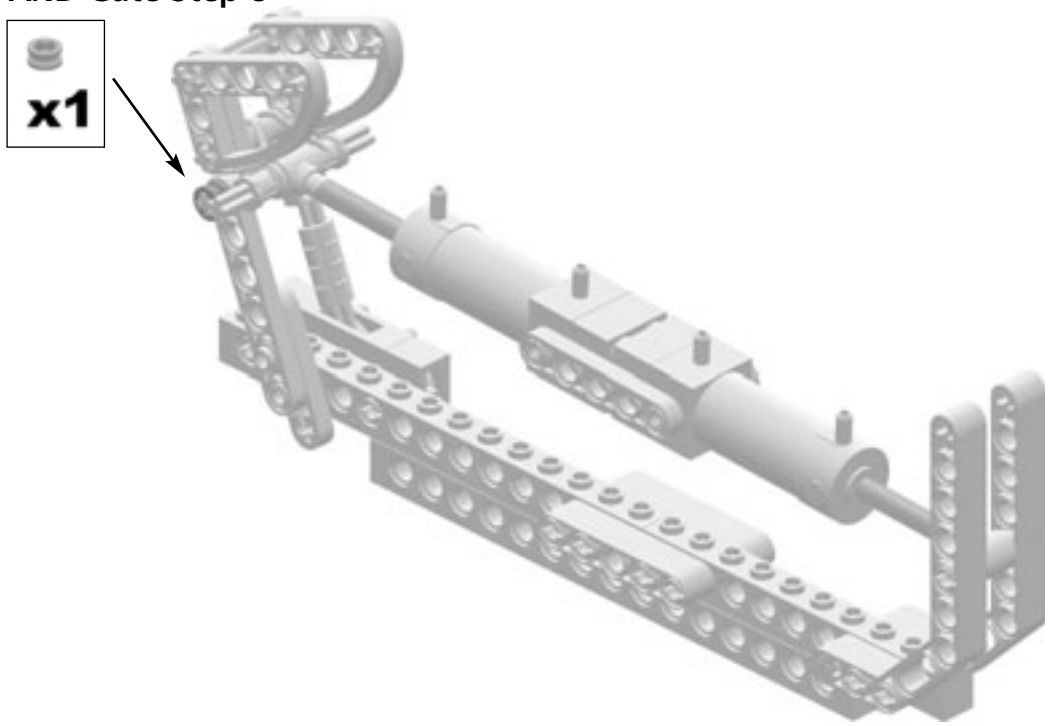
Hook the rubber bands from the AND Handle sub-assembly around the bushings at the end of the Piston sub-assembly.

AND Gate Step 7



Attach one of the Handle Strap sub-assemblies as shown.

AND Gate Step 8



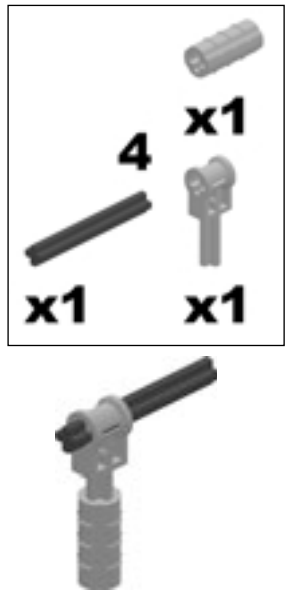
The OR Handles

There are two differences between the OR Gate and the AND Gate sub-assemblies.

- One difference between the OR Gate sub-assembly and the AND Gate sub-assembly is the distance between the pneumatic switch and the far end of the Piston sub-assembly.
- A second difference is the type of handle used to control the pneumatic switch.

In the OR Gate sub-assembly, the pistons expand beyond the pneumatic switch; this is accomplished by the OR Handle sub-assembly. The OR Handle sub-assembly also uses rubber bands and the rounded liftarm parts to guide the pistons back to the proper location when the pistons contract. PneumADDic II incorporates three OR gates, so you will need to build **three** OR Handle sub-assemblies.

OR Handle Step 0



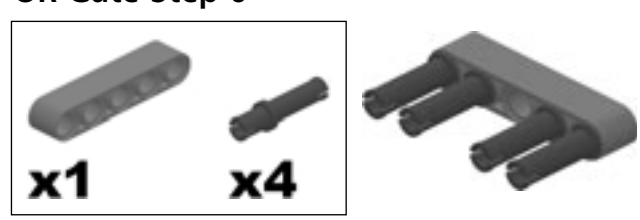
OR Handle Step 1

OR Handle Step 2

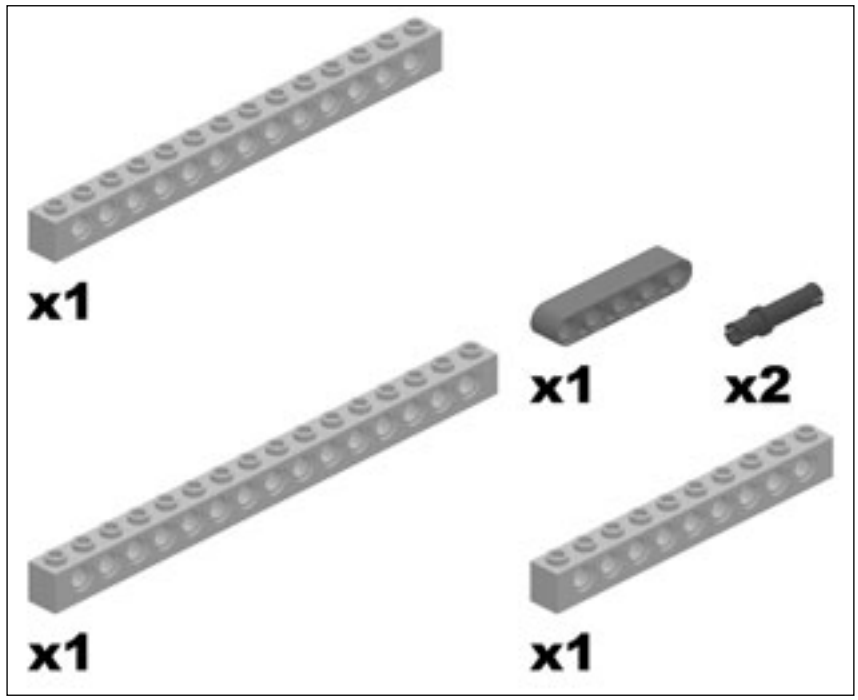
The OR Gates

Now we have what we need to create the final OR Gate sub-assembly. You will need to build **three** OR Gate sub-assemblies.

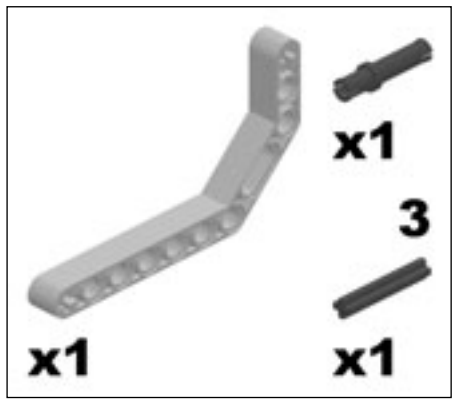
OR Gate Step 0

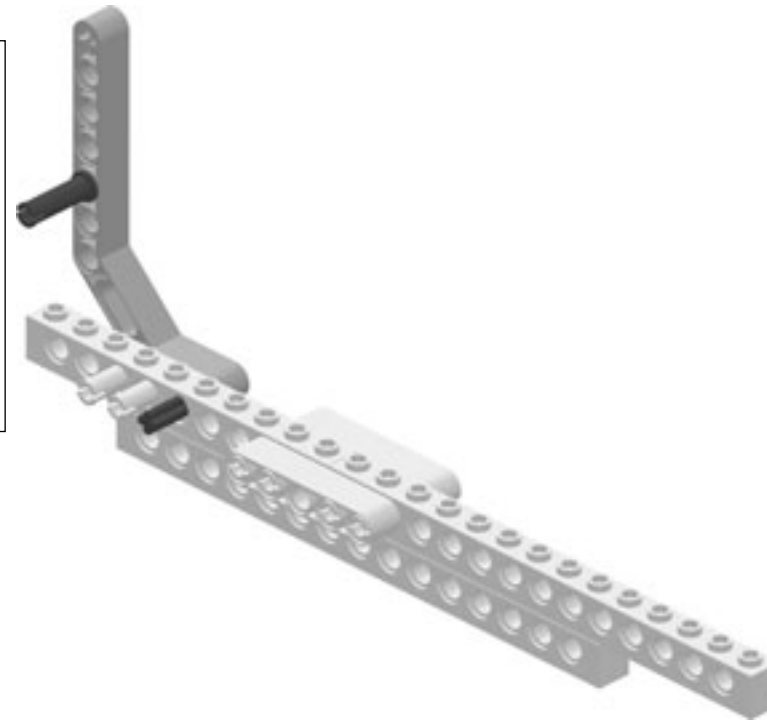


OR Gate Step 1



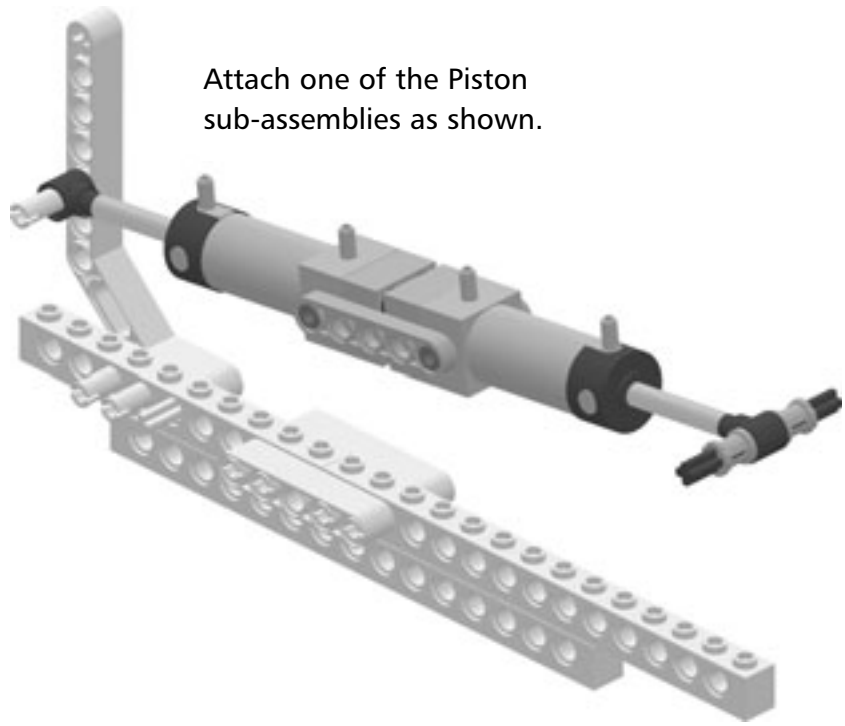
OR Gate Step 2



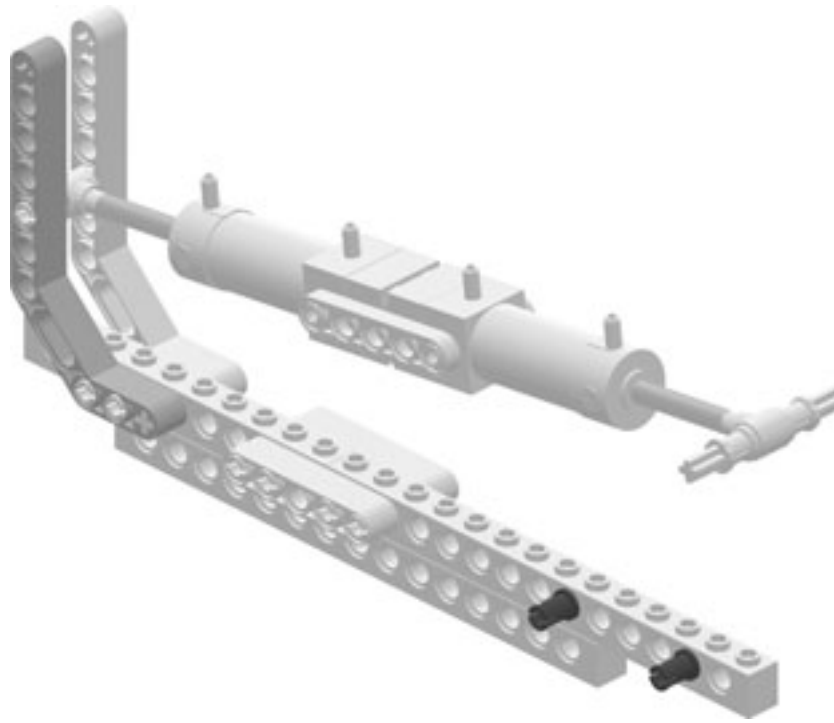


OR Gate Step 3

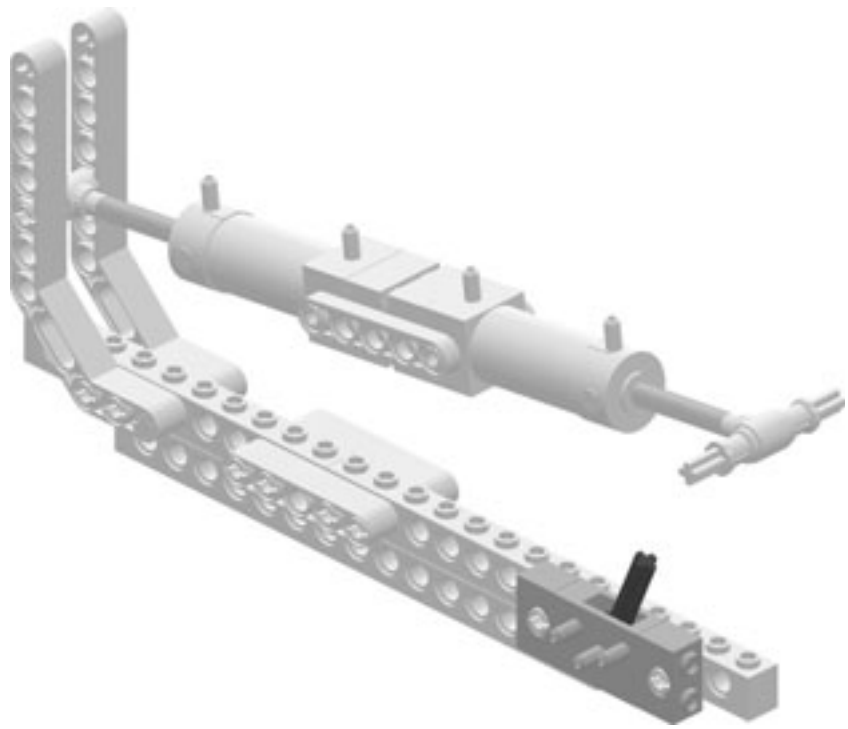
Attach one of the Piston sub-assemblies as shown.



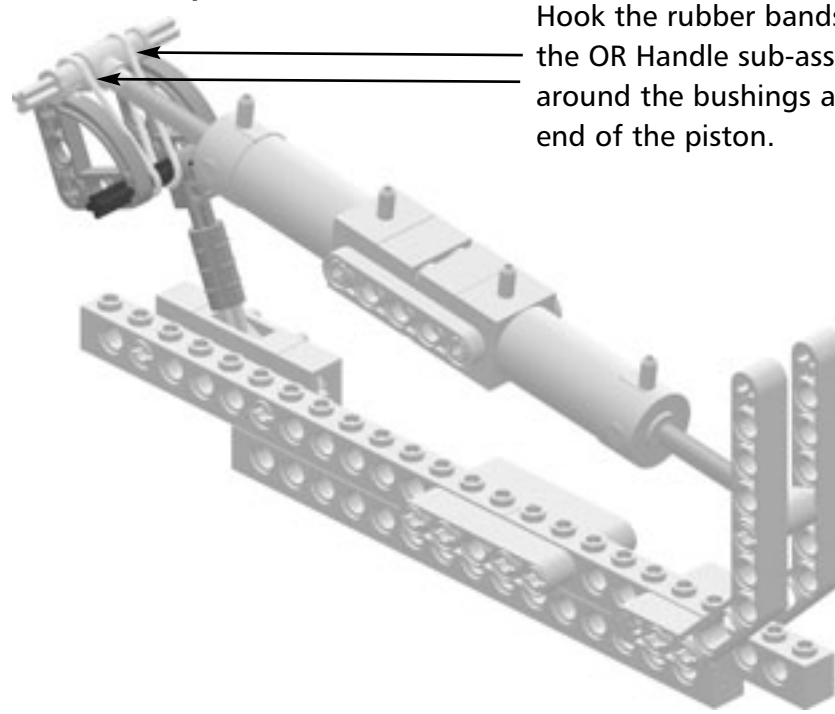
OR Gate Step 4



OR Gate Step 5

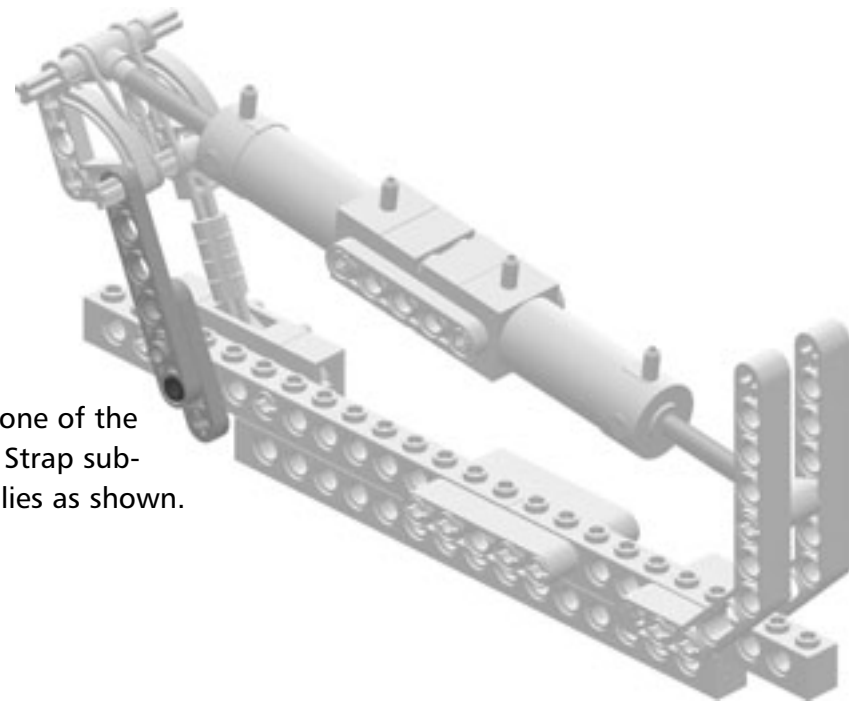


OR Gate Step 6



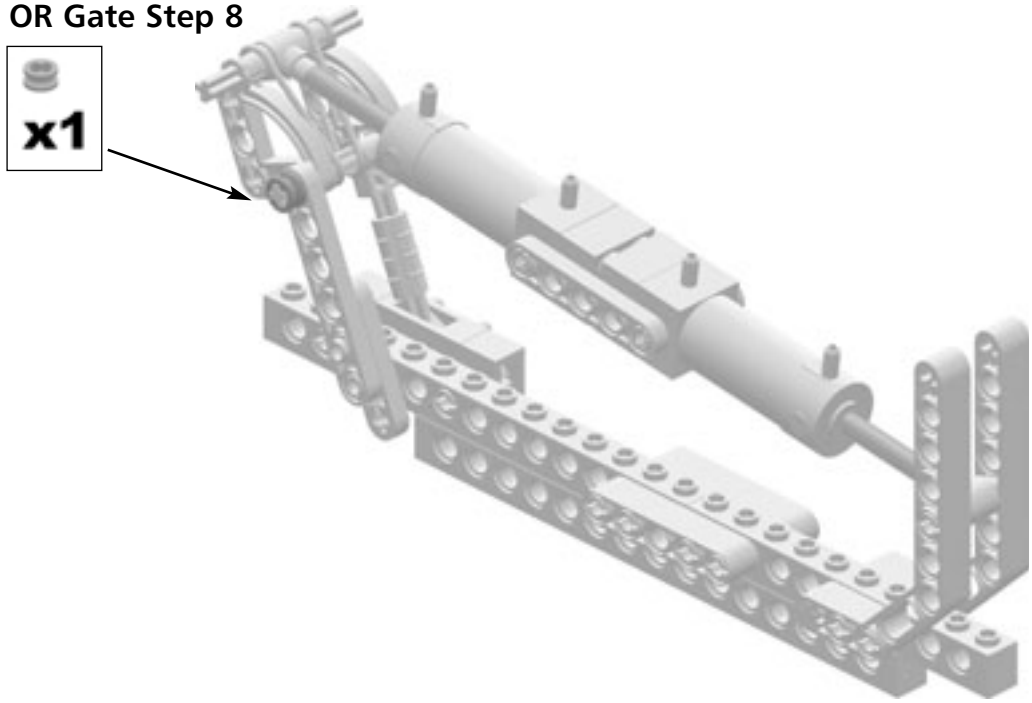
Hook the rubber bands from the OR Handle sub-assembly around the bushings at the end of the piston.

OR Gate Step 7



Attach one of the Handle Strap sub-assemblies as shown.

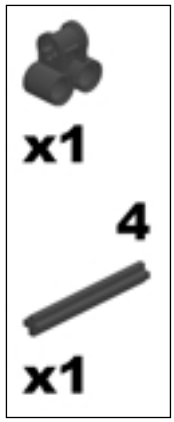
OR Gate Step 8

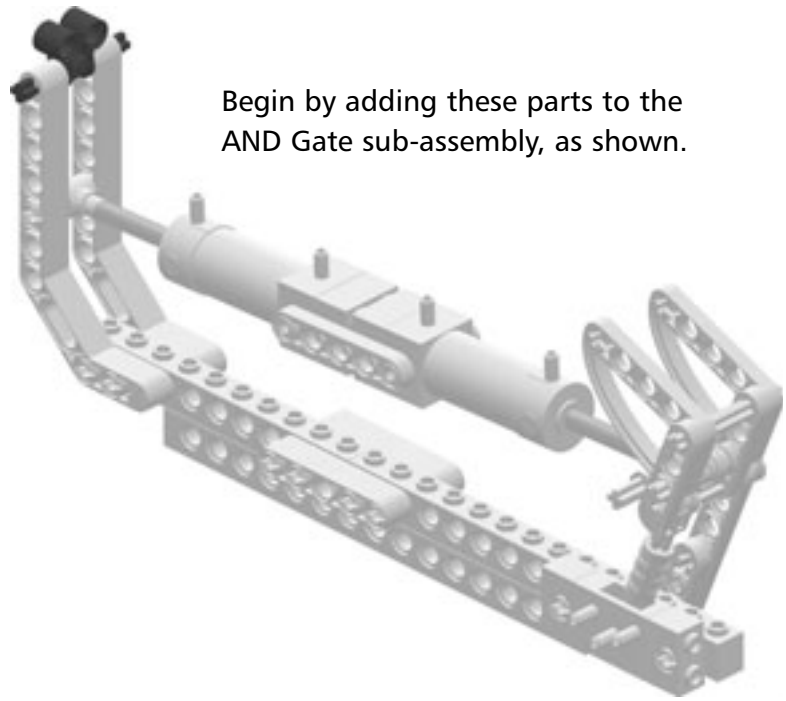


AND Gate Tube Guides

Some of the AND Gate sub-assemblies require tube guides to hold the pneumatic tubing in place. These tube guides prevent the Piston sub-assemblies from twisting. Adding the AND Gate Tube Guides sub-assembly is completed in just a few steps. You will only need to build **two** AND Gate Tube Guide sub-assemblies to add to the AND Gate sub-assemblies.

AND Gate Tube Guides Step 0



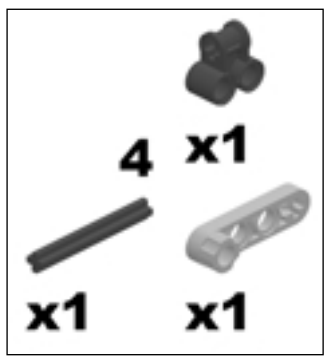


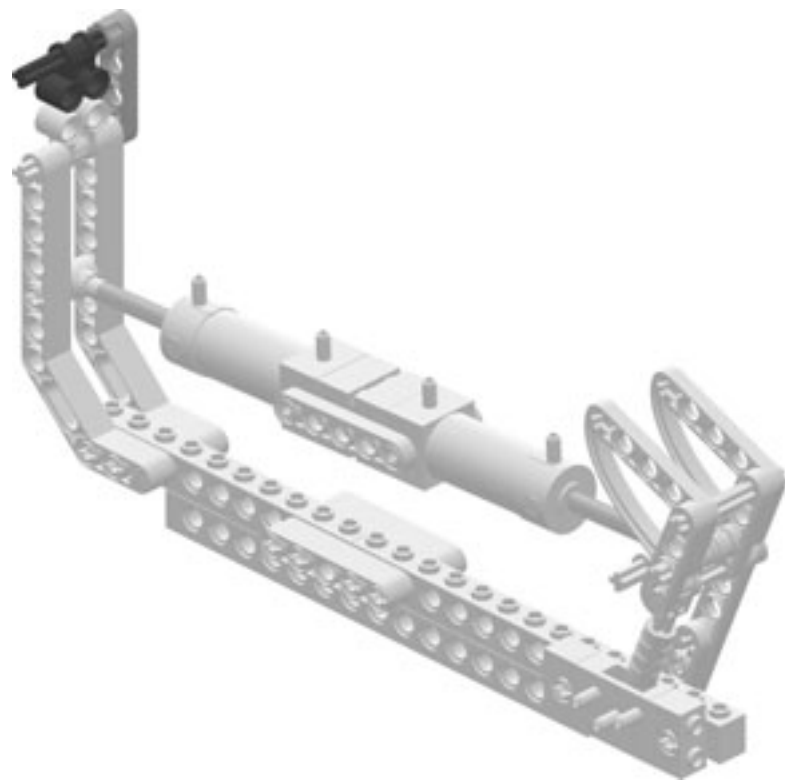
Begin by adding these parts to the AND Gate sub-assembly, as shown.

NOTE

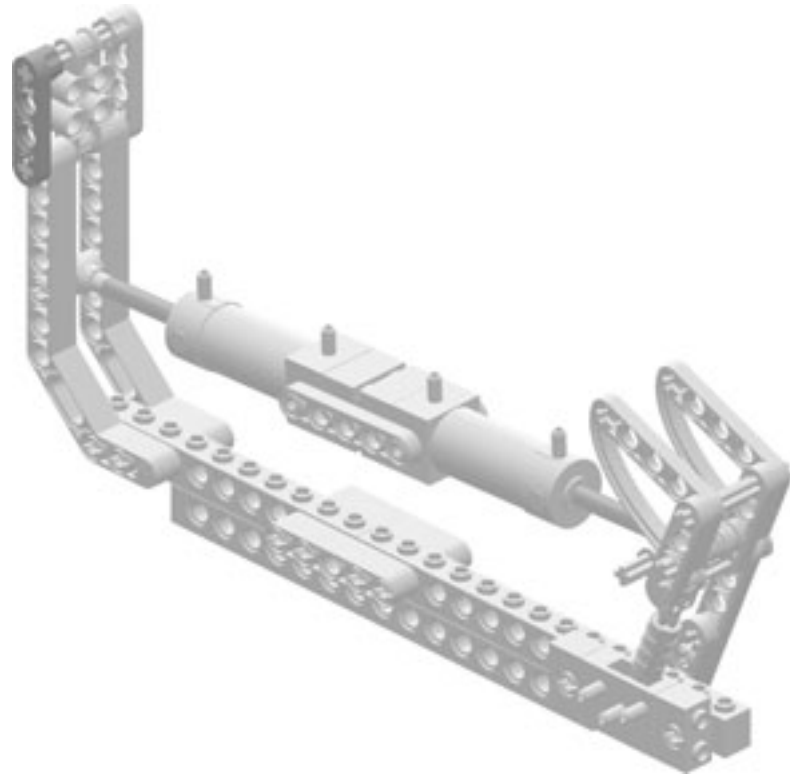
Remember to add these tube guides to just **two** of the AND Gate sub-assemblies.

AND Gate Tube Guides Step 1





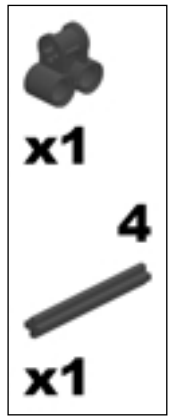
AND Gate Tube Guides Step 2

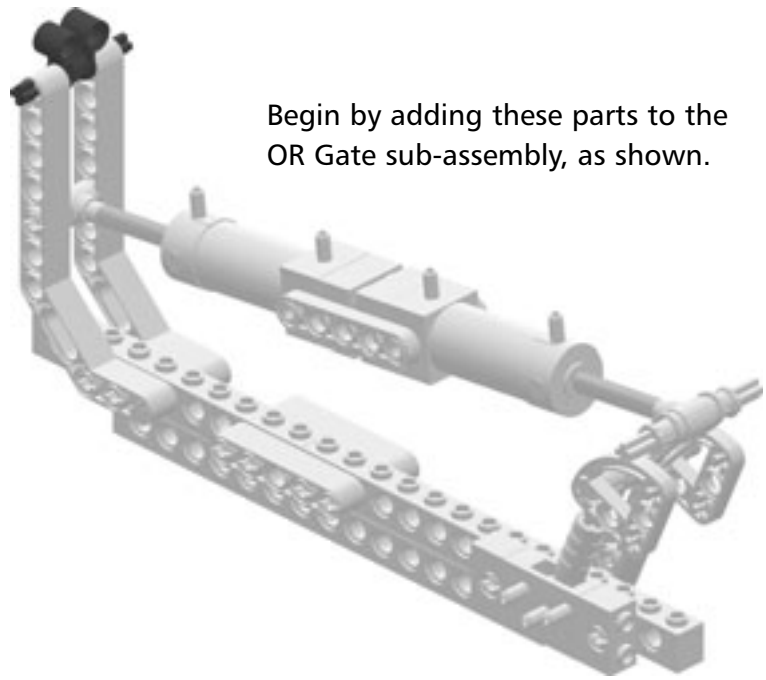


OR Gate Tube Guides

In this sub-assembly, you will build a single OR Gate Tube Guide sub-assembly that will be attached to **one** of the OR Gate sub-assemblies.

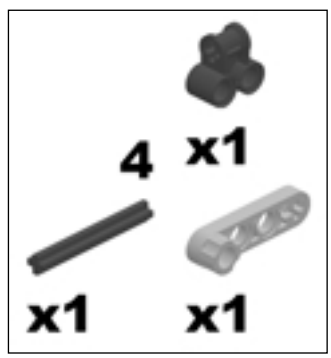
OR Gate Tube Guides Step 0

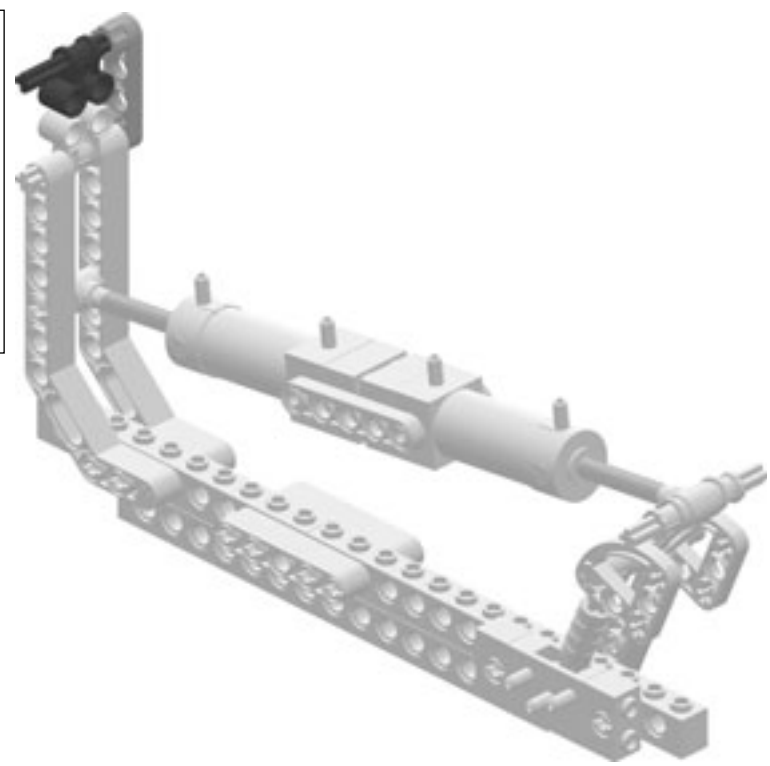




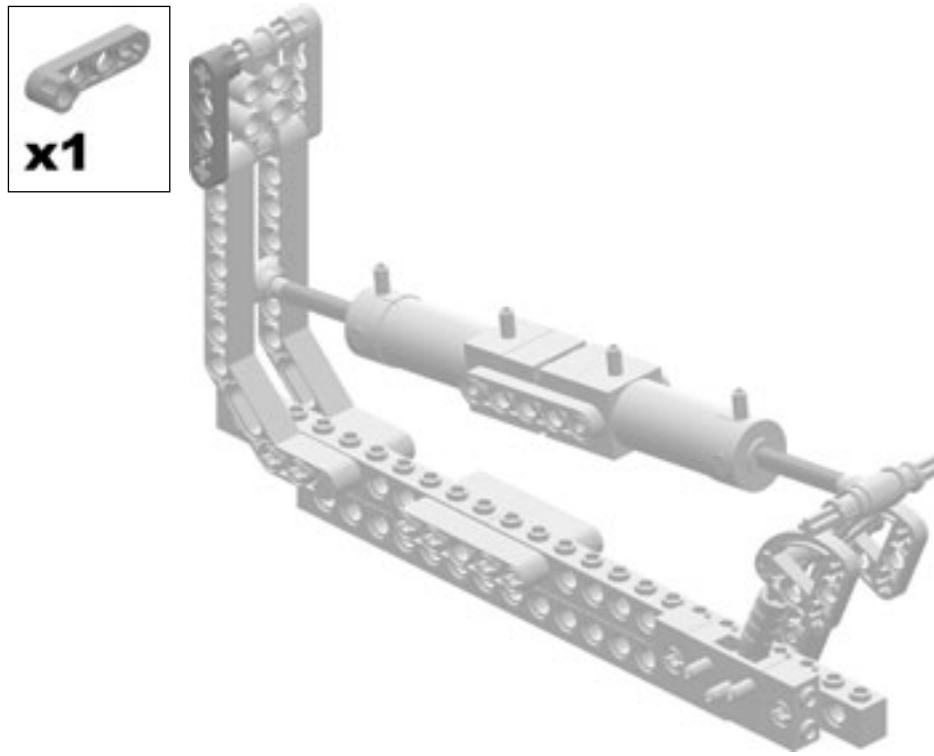
Begin by adding these parts to the OR Gate sub-assembly, as shown.

OR Gate Tube Guides Step 1





OR Gate Tube Guides Step 2



The Issue of Sensors

PneumADDic II uses a total of ten touch sensors, but only one RCX. The RCX has three sensor inputs. The question is, how do we get 10 touch sensors to connect to the RCX so that it can tell which touch sensors are being pressed? We do this by some electronic slight of hand.

The RCX contains a battery-operated digital computer. As we learned earlier, digital computing is limited to ones and zeroes. The sensors available for the RCX are analog, meaning that they can provide a wide range of readings. How is it possible for the sensors to communicate with the RCX, when the RCX is digital and the sensors are analog? The RCX contains a device called an *analog to digital converter*. The analog to digital converter computes a 10-bit binary number for any given analog value provided by a sensor.

Touch sensors are binary devices in that they only provide two values:

- Touched
- Not touched

The RCX reads values from the touch sensors using the analog to digital converter. It is possible to combine two or more touch sensors onto one RCX input port by assigning each touch sensor a different analog reading value to be interpreted by the RCX. This is accomplished by adding *electricity resistors* in between the touch sensors and the RCX analog input. Electricity resistors, or simply resistors, dispose of some of the electricity in

an electronic circuit by converting it to heat. PneumADDic II uses touch sensors and resistors in two different types of circuits:

- The series resistor circuit
- The parallel resistor circuit

Both of these circuits will be explained in more detail later in the chapter.

When I realized that I needed resistors to combine multiple touch sensors onto a single RCX sensor port, I was unsure of how much resistance was necessary, so I decided to use variable resistors called *potentiometers*.

You can adjust the amount of resistance in a potentiometer by turning a screw that protrudes from the device. The next challenge was to figure out how I could easily incorporate these potentiometers into LEGO models. I decided to create my own potentiometer bricks that could be snapped into place in a LEGO assembly, similar to the bricks that LEGO makes. When you create your own RCX sensors, they are traditionally called *home-brew* sensors. I call the resistor blocks I made *home-brew potentiometer bricks*.

NOTE

You will probably have to order your potentiometers from an electronics part supplier, as they are not a LEGO product. I got my potentiometers from Digi-Key Corporation (www.digikey.com), product number CT20P103-ND. These potentiometers are adjusted by turning a screw on the end of the case. It takes 10 complete turns of the adjustment screw to increase the potentiometer's minimum resistance of 0 Ohms, to its maximum resistance of 10,000 Ohms (or 10K Ohms).

The Potentiometer Brick

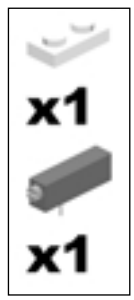
The next section of building instructions shows you how to create a home-brew potentiometer brick. The Potentiometer Brick sub-assembly requires the modification of some LEGO parts, soldering equipment and solder, and a strong adhesive. Some LEGO purists would find this appalling, but the potentiometer bricks work very well. You will need some special plates from LEGO, specifically electric plates. Electric plates look similar to normal plates, but contain metal to allow you to electrically connect cables together.

We need to cut away parts of the plates to allow access for electrical wiring. The Potentiometer Brick sub-assembly also requires soldering (similar to the effect hot glue, but for electrical connections), and some glue to fortify the sensors when they are done.

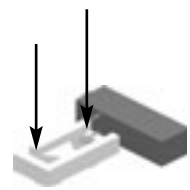
For the PnuemADDic II you will need to build a total of **eight** potentiometer bricks, therefore you will need the following supplies:

- 16 1x2 LEGO electric plates (you can get larger plates and cut them into 1x2 if you want)
- 16 1x2 standard LEGO plates
- 8 2x2 standard LEGO plates
- 8 10K Ohm potentiometers (10K means 10,000 Ohms, a measure of resistance)
- 1 foot of wire (standard copper telephone wire works well)
- Goggles
- Xacto knife
- A strong adhesive (Krazy Glue works well)
- Soldering pen
- Solder

Potentiometer Brick Step 0



Cut a 1x2 electric plate as shown to provide access to the underside electrical contacts. Solder one of the legs of a potentiometer to the electric plate as shown. Be careful not to melt the electric plate with the heat from the solder pen.

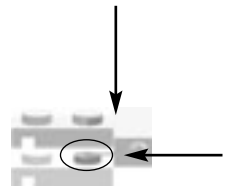


WARNING

It is hard to cut such a small piece of LEGO safely. It is recommended that you insert the 1x2 electric plate into the back of a larger plate. The larger plate will be easier for you to hold, and it will hold the electric plate. Be careful not to cut yourself.

Potentiometer Brick Step 1

Modify a second electrical plate, by removing the plastic on the electrical stud, and cutting access for a wire near the other stud as shown.



Solder the second leg of the potentiometer to the exposed electrical stud.

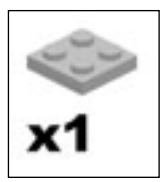
WARNING

A soldering pen is extremely hot. Please take care not to burn yourself.

Potentiometer Brick Step 2

Bend the two white electrical plates apart so you can solder one end of an electric wire to the underside of the stud on the bottom electric plate in this picture.

Potentiometer Brick Step 3

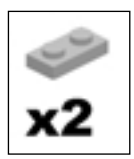


Insert the 2x2 plate in between the two electric plates. You may have to cut away part of the 2x2 plate to make the three plate combination fit together snugly.

When things fit snugly, secure the three plates together using the glue.

Solder the other end of the wire to the underside of the stud in the top electric plate in this picture.

Potentiometer Brick Step 4



Glue these last two 1x2 plates together to complete the 2x2 electric potentiometer brick.

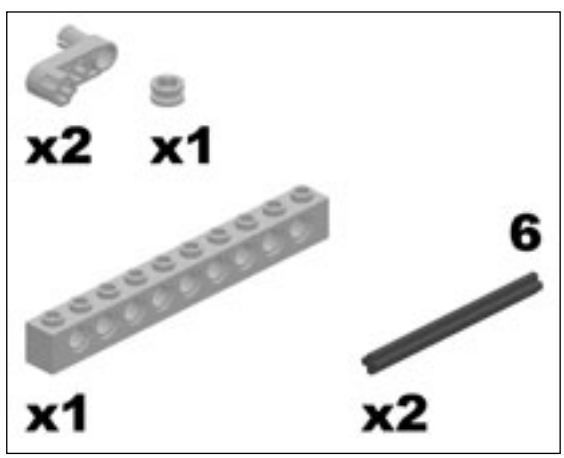
The Sum Sensor

This sum sensor combines a small pneumatic piston, a touch sensor and a Potentiometer Brick sub-assembly.

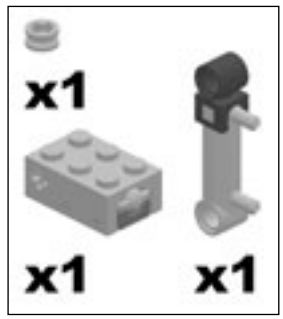
The RCX feeds the values you want added to the pneumatic adding circuit two bits at a time. The pneumatic adding circuit provides one output bit that is the result of adding (also called *summing*). The RCX needs to be able to read this sum bit and remember it, so it can display the final answer on its numeric display. The Sum Sensor sub-assembly you are about to build combines a small pneumatic piston with a touch sensor. The small piston is controlled by the AND Gate sub-assembly, which is also part of the pneumatic adding circuit. When the output of the adding circuit is zero, the Piston sub-assembly contracts, and does not press the touch sensor button. When the adding circuit outputs a one, the Piston sub-assembly expands, and presses the touch sensor button. The RCX reads the state of the touch sensor button using an input port.

Since the Sum Sensor sub-assembly shares an RCX input port with another sensor (the Pressure Sensor sub-assembly), the Sum Sensor sub-assembly requires a Potentiometer Brick sub-assembly. I will explain the potentiometer brick/touch sensor circuit in further detail when we connect the sensors to the RCX.

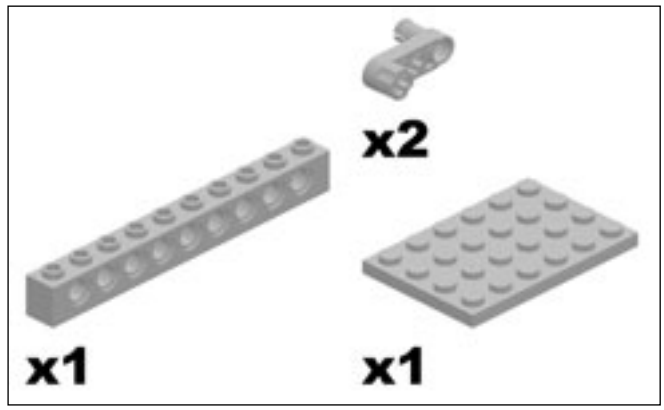
Sum Sensor Step 0



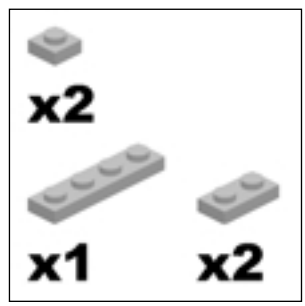
Sum Sensor Step 1



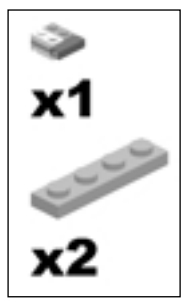
Sum Sensor Step 2

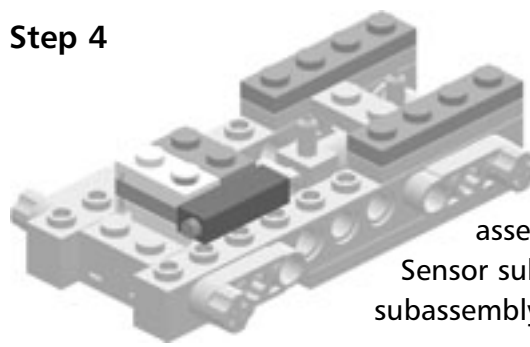


Sum Sensor Step 3



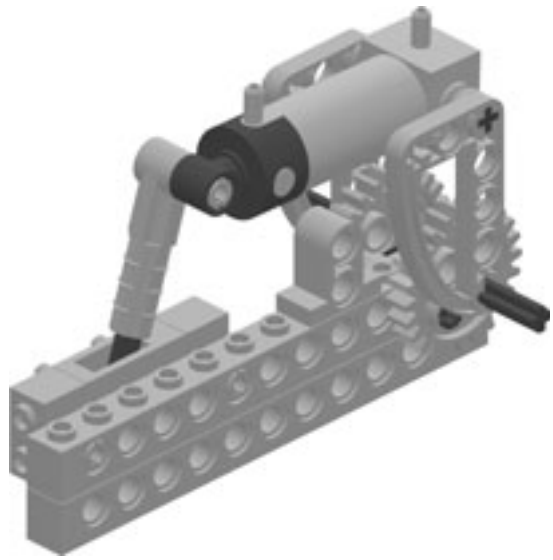
Sum Sensor Step 4





The Potentiometer Brick sub-assembly allows us to hook the Sum Sensor sub-assembly and the Pressure Sensor subassembly to the same RCX input port.

The Carry Memory



The original PneumADDic had only half the pneumatic adding circuitry that PneumADDic II does. To perform the addition for a given binary column, we need to add a bit from Value 1, a bit from Value 2 and the Carry. This is easily done in a two-stage addition, where we add a bit from Value 1 and a bit from Value 2 to get a *partial-sum*. We then add the partial-sum to the Carry to get the *final sum*.

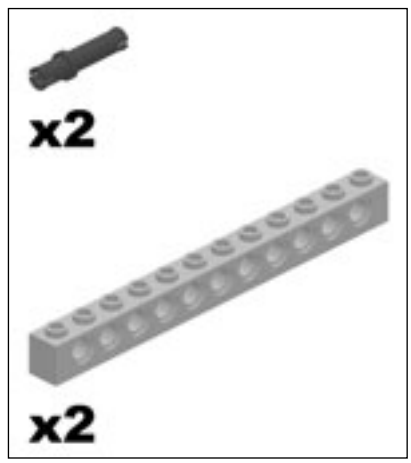
The original PneumADDic had one two-bit adder, so the complete addition for a given binary place had to be done as two consecutive pneumatic additions. The original PneumADDic also had to electronically remember the carry from binary place to binary place. PneumADDic II contains a complete two-stage pneumatic adding circuit, so a bit from Value 1 and a bit from Value 2 and the Carry can be collectively added together in a single calculation. PneumADDic II also pneumatically remembers the Carry from one binary place to the next. It uses a pneumatic piston that controls a pneumatic switch as a Carry Memory. In computers, a memory device has a value input, a value output, and an input that tells the memory device when to disregard the stored value, and when to store the current input. After the memory device is instructed to store a current input, the value input can change without the output value changing.

A medium-sized LEGO pneumatic piston creates a substantial amount of friction between the outside case (sometimes referred to as the *cylinder*), and the *piston head* (the component inside that causes the whole pneumatic piston to expand and contract). The piston head will only move if there is a strong force to push or pull it. The small LEGO pistons have less friction and the force of gravity is sometimes enough to move the small piston head (small pistons are also not strong enough to flip a pneumatic switch reliably). For a medium-sized piston, once the piston head is moved, friction makes it "remember" where it is, by not letting it move.

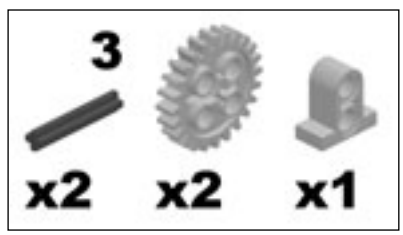
PneumADDic II uses air pressure to change the location of the piston head, and therefore remember a new value. The pressure inlets for the Piston sub-assemblies play the role of the value input for our Carry Memory. The RCX only allows pressure through to the Carry Memory inputs at certain times during the addition process. When pressure is applied, the Carry Memory remembers the new value. The pneumatic switch, which you remember has its lever position controlled by the pneumatic piston, acts as the memory output of Carry Memory. Pressure applied to the pneumatic switch is routed to the switch's ports based on the direction the lever is leaning.

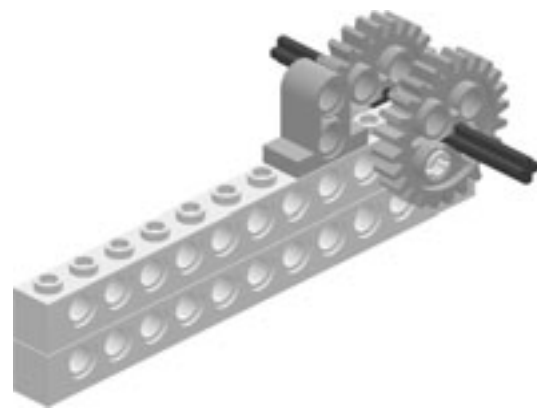
It takes time for the pneumatic adding circuit to calculate the answer. When new values are fed to the circuit via the RCX and motorized pneumatic switches, the changes affect the gates connected to the motorized switches. When the outputs of the first gates change, they affect the next gates in the circuit. When these outputs change, they affect the next gates and so on. We must let all those changes happen before we know that the Carry calculated by the pneumatic adding circuit is correct. Only then can we make the Carry Memory remember the Carry value, by applying pressure to the inlet of the switch that drives the Carry Memory.

Carry Memory Step 0

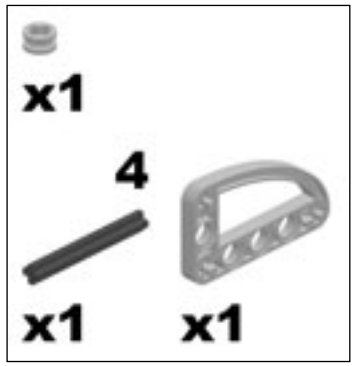


Carry Memory Step 1

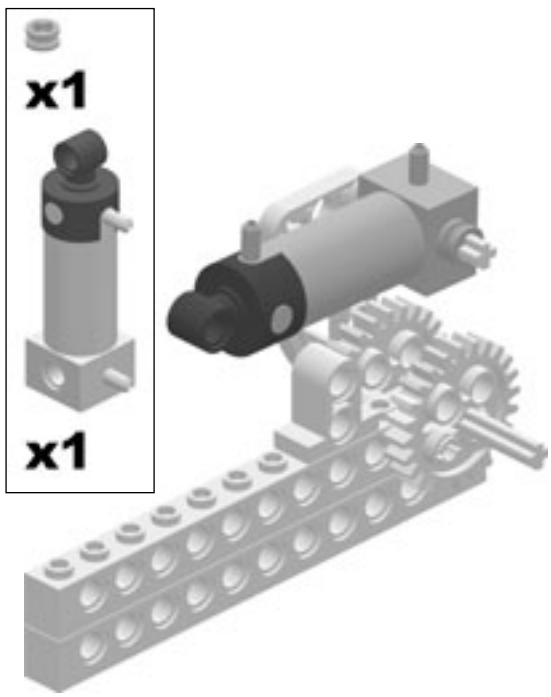




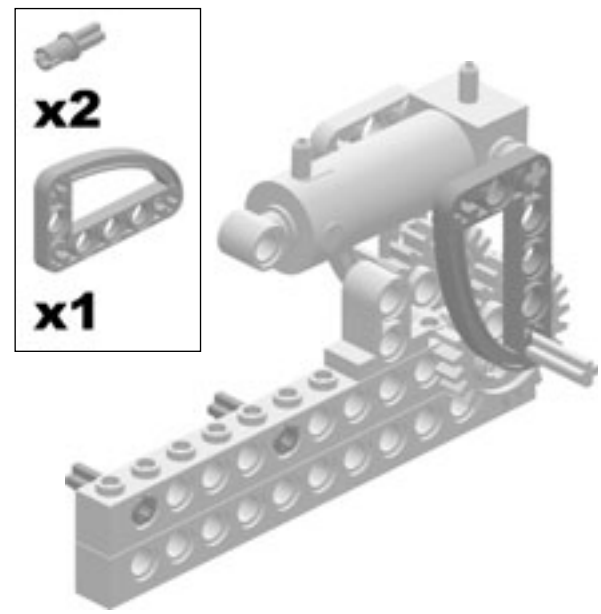
Carry Memory Step 2



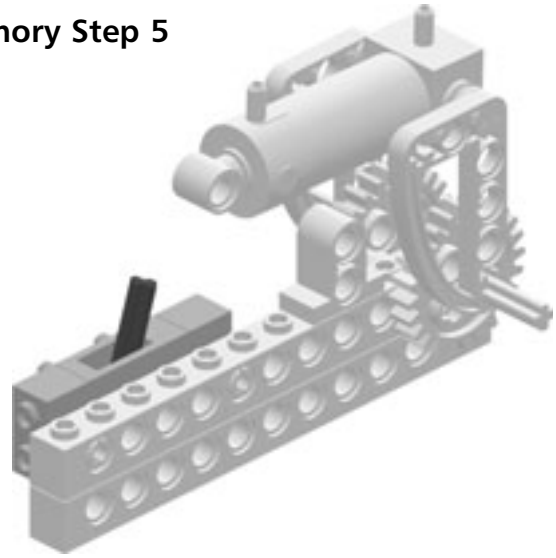
Carry Memory Step 3



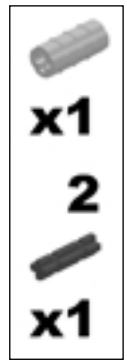
Carry Memory Step 4

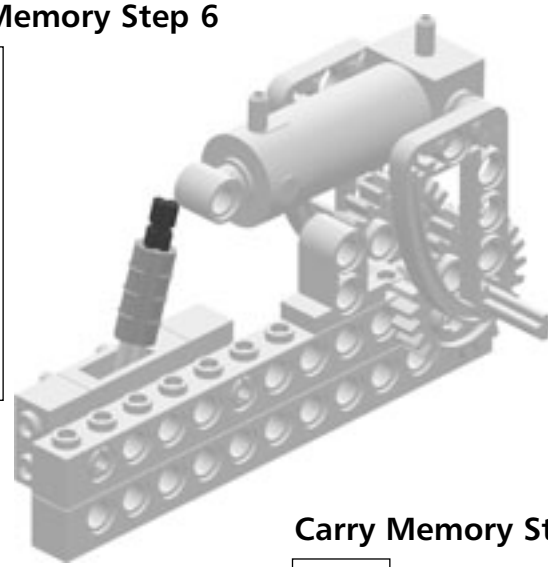


Carry Memory Step 5

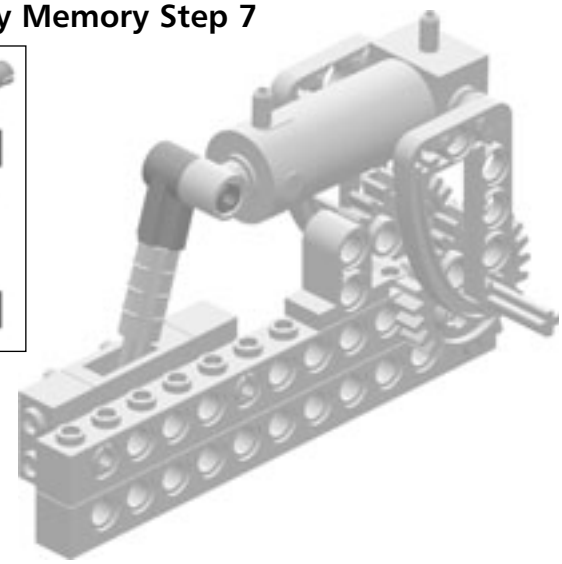


Carry Memory Step 6





Carry Memory Step 7



Powering PneumADDic II

One of the problems with the original PneumADDic was that it was painfully slow. It took about 10 minutes to add two three-digit numbers! This issue was attributed to the fact that the gates required high pressure, however a second issue was that the compressors (it had two) were not very strong.

PneumADDic II contains a single compressor comprised of two motors and four large pumps. When compressing a pump, the first half of the compression stroke does not require a lot of force. The closer the pump handle is to maximum stroke, the more force it takes to move the pump handle. PneumADDic II's compressor uses four pumps, but only one of the pumps is being driven to maximum compression at a time. On a clock, this is like making the first pump compress at 12:00, the second pump at 3:00, the third pump at 6:00, and the fourth pump at 9:00. This means that the motors only need to have enough force to compress one piston fully, and the next piston partially.

PneumADDic II's compressor uses two motors to drive the pumps giving the compressor twice the power to compress the pumps. By using four pumps and two motors, PneumADDic II's compressor is much more efficient than PneumADDic's two compressors.

PneumADDic II requires a total of three motorized switches. Two switches are for feeding values to the inputs of the pneumatic adding circuit. The third motorized switch controls when the Carry Memory remembers (or stores) the carry for a given binary place. The RCX only has three motor output ports. Each output port can drive at least two motors. The two motors in the PneumADDic II compressor can be driven by one RCX output port. But that only leaves us two motor ports and the need for three motorized switches. To solve this shortfall, I decided to combine one of the switches into the compressor.

The compressor uses a handy gearing device called a differential. A differential can be used many ways, and can be somewhat confusing to understand through reading, so I suggest you assemble one as shown in **Dual Motor Switch Step 9**. A differential has a total of four independent gears and two axles. PneumADDic II's compressor motors turn the case of the differential (one of the independent gears), which makes the two axles turn. The two axles can rotate independent of each other, but are related in how they do so.

If you rotate the differential casing (the dark gray partial cylinder in added in **Dual Motor Switch Step 9**), both axles turn at the same speed. If you prevent one axle from turning, all the rotation at the differential case is applied to the free axle. If you restrict the rotation of both axles using friction, the amount of power delivered to the differential is split between each axle based on how much resistance each axle is experiencing.

In the case of the Dual Motor Switch sub-assembly for the compressor, one of the axles is used to flip the switch handle from one position to the other. The other axle protruding from the differential is used to compress the pumps.

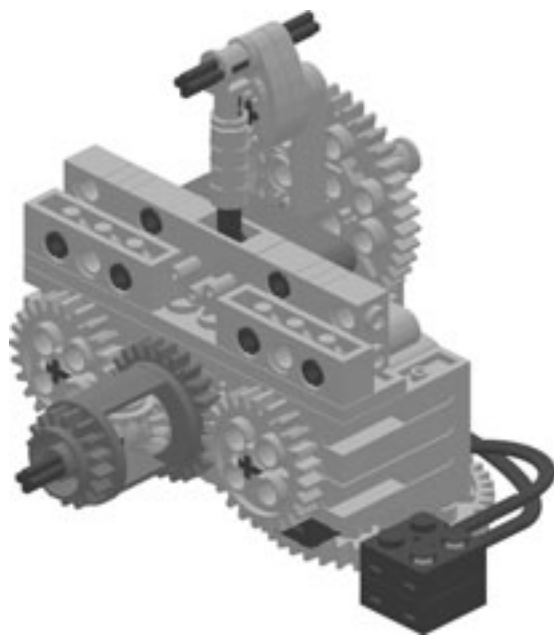
When the dual motors start running, the differential automatically determines which axle is requires less force to turn which in our case could be either the pumps in the

Pump sub-assembly or the pneumatic switch in the Dual Motor Switch sub-assembly. The pneumatic switch is easier to activate compared to the pumps, but once the switch has reached the switch lever limit, the switch lever will not move any more; therefore, at that point the pumps in the Pump sub-assembly are easier to turn. Once the pneumatic switch has been flipped all the power from the motors is applied to the pumps.

When the RCX wants to flip the switch, it applies power to the motors for a short time in the direction needed to flip the switch. When the RCX wants to run the compressor, without flipping the switch, it powers the motors in the same direction it used last time it flipped the switch. The pumps are not affected by the gear direction that compress the pumps. Only the switch is affected by the direction the motors are turning. The differential provides a wonderfully simple way to combine the switch and the pumps using the single set of motors.

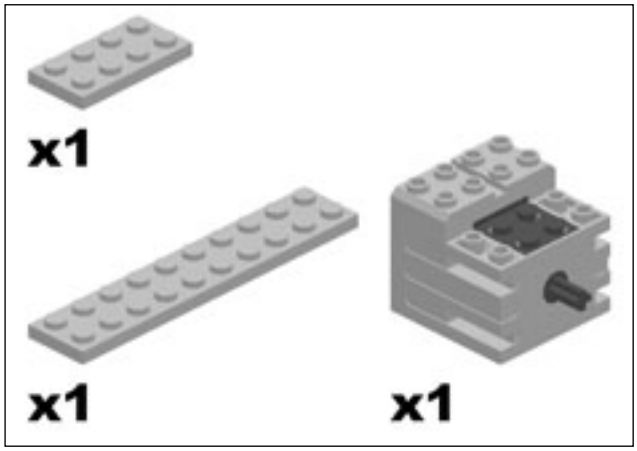
In the final assembly of PneumADDic II, we need to be able to see how to hook the ports of the Dual Motor Switch assembly to the rest of the pneumatic circuitry for PneumADDic II. If we add the entire compressor to the design at once, you cannot see the Dual Motor Switch sub-assembly, making it difficult to discern how to connect it to the rest of the circuit. To solve this building instruction problem, I've broken the compressor up into the Dual Motor Switch sub-assembly, and Pumps sub-assembly, which I add in two different steps in the final assembly. The following building instructions describe how to build the Dual Motor Switch sub-assembly.

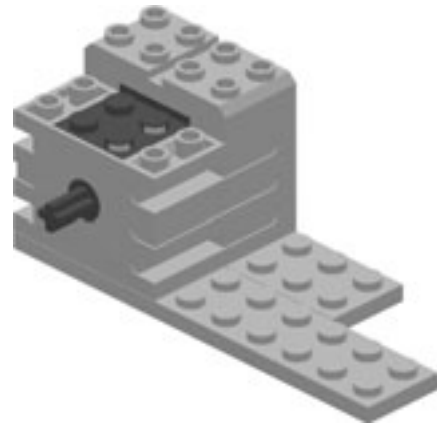
The Dual Motor Switch



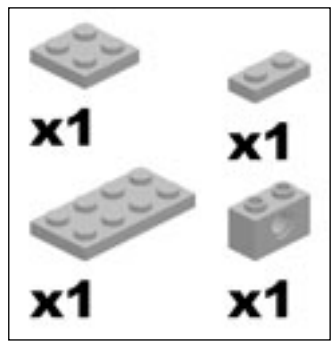
This Dual Motor Switch sub-assembly forms half of the compressor used to power the pneumatic adding circuitry.

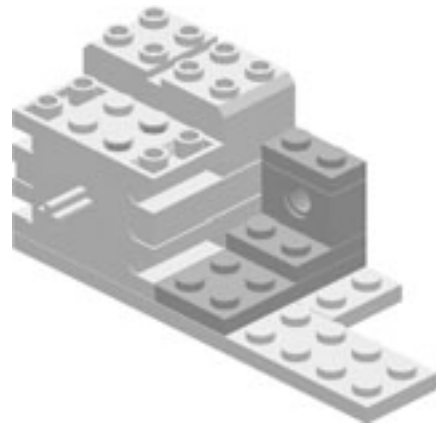
Dual Motor Switch Step 0



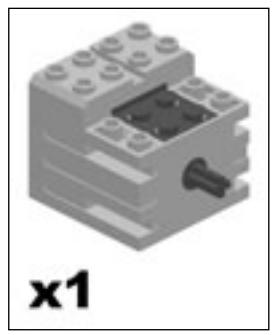


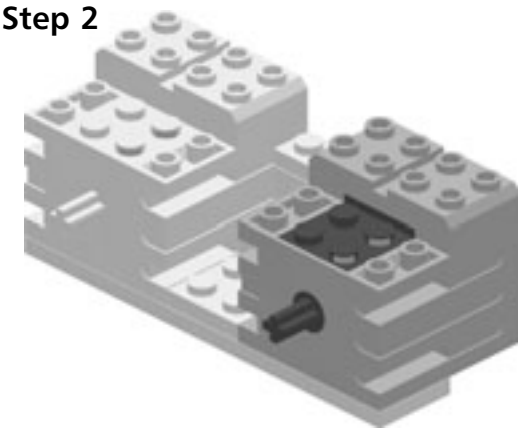
Dual Motor Switch Step 1



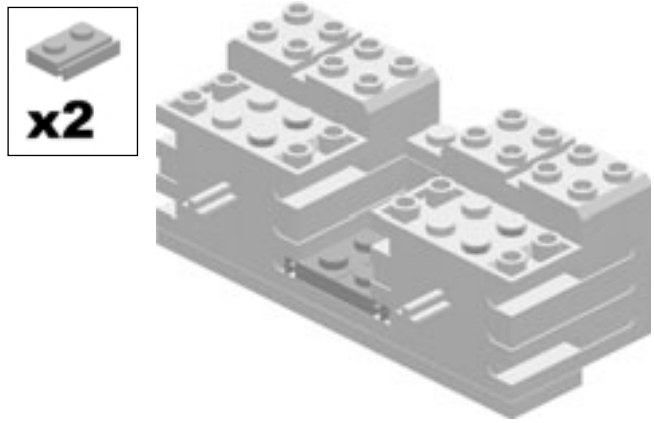


Dual Motor Switch Step 2

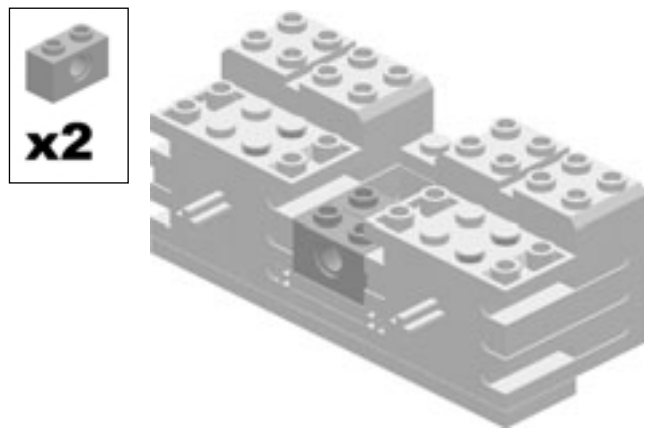




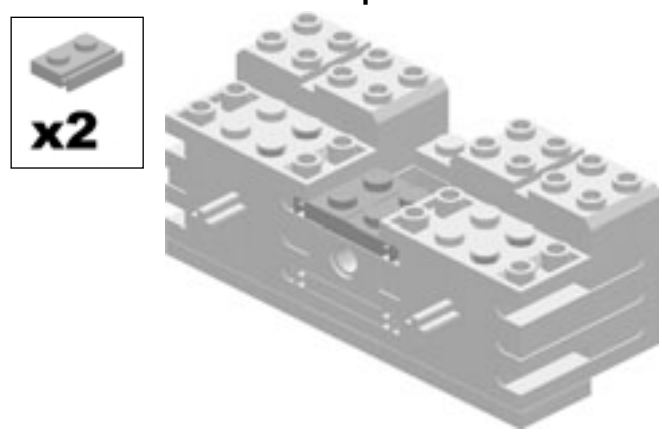
Dual Motor Switch Step 3



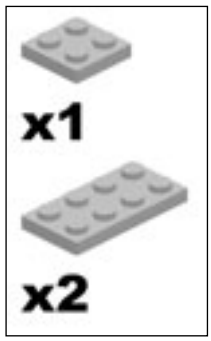
Dual Motor Switch Step 4

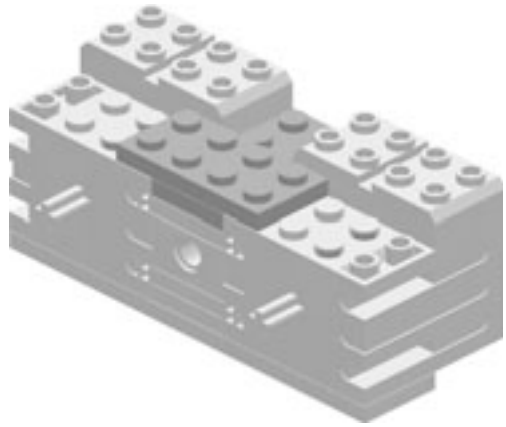


Dual Motor Switch Step 5

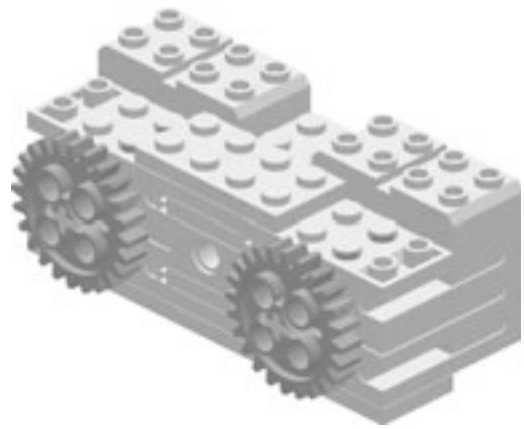


Dual Motor Switch Step 6

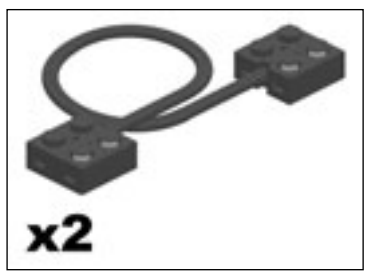


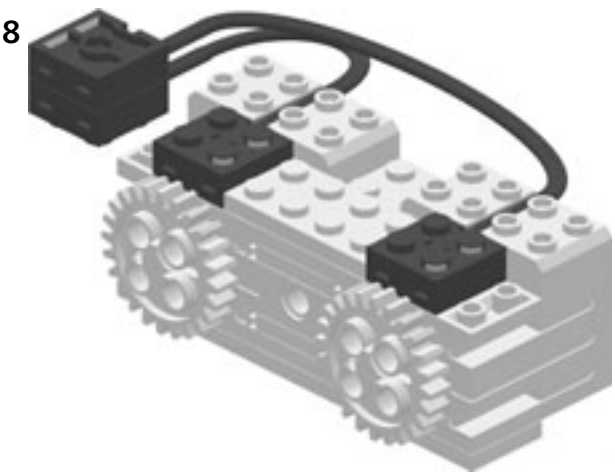


Dual Motor Switch Step 7

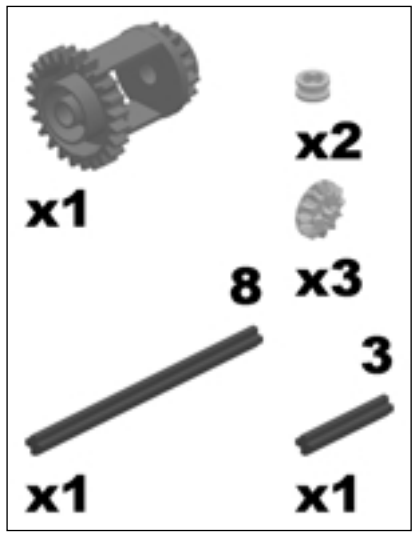


Dual Motor Switch Step 8

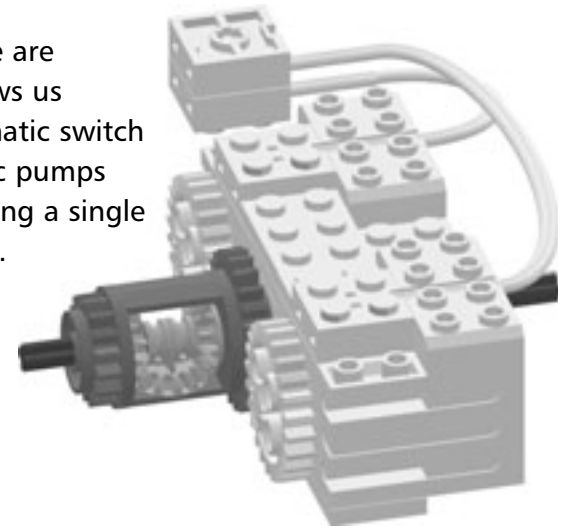




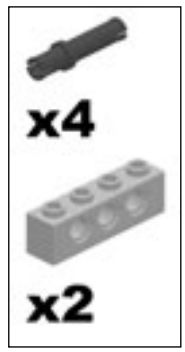
Dual Motor Switch Step 9

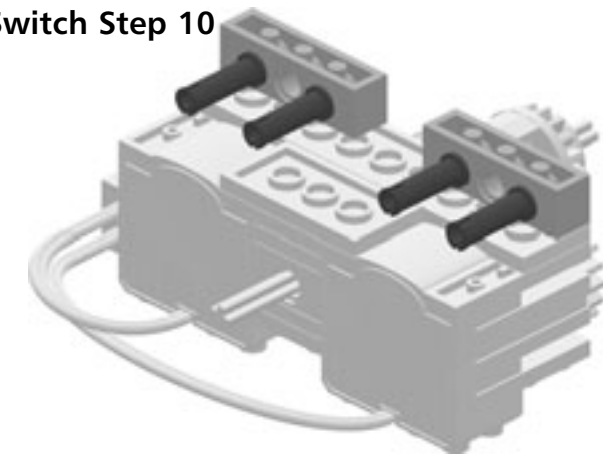


The differential we are building here allows us control the pneumatic switch and the pneumatic pumps while only occupying a single RCX motor output.

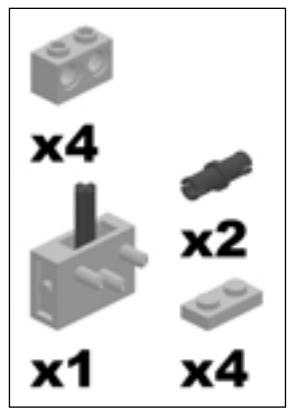


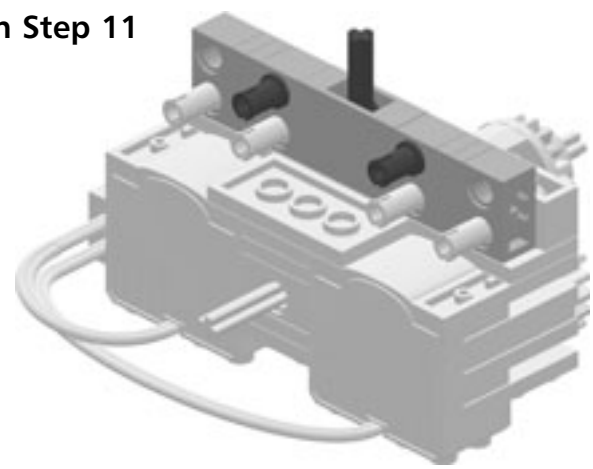
Dual Motor Switch Step 10



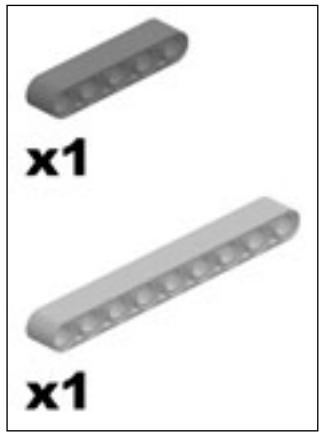


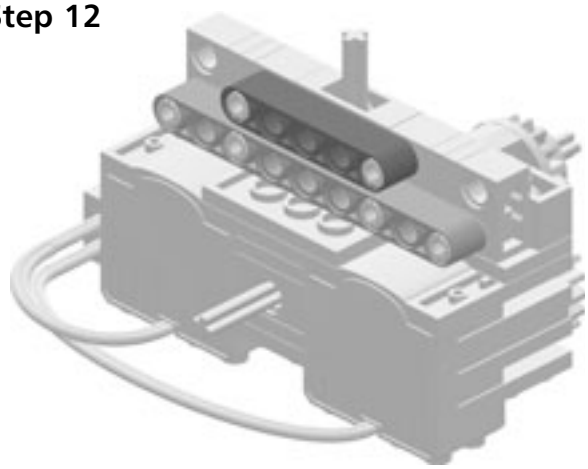
Dual Motor Switch Step 11



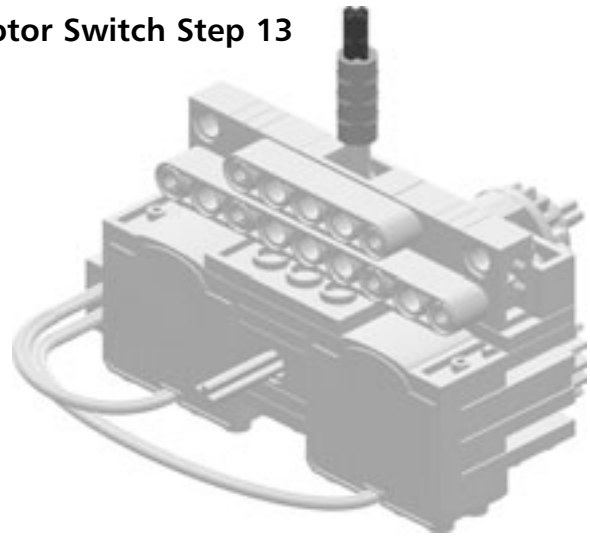


Dual Motor Switch Step 12

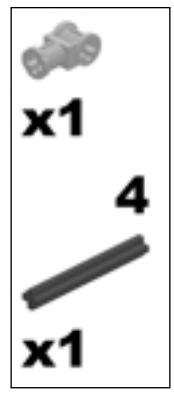


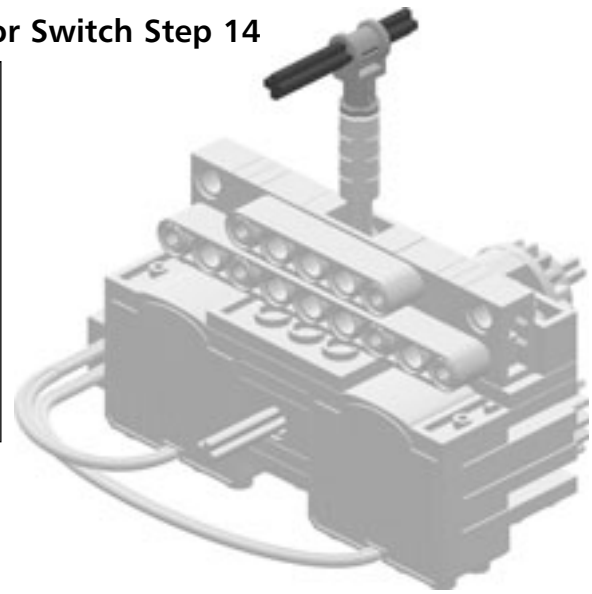


Dual Motor Switch Step 13

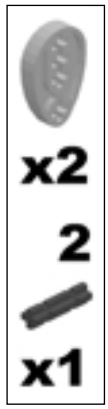


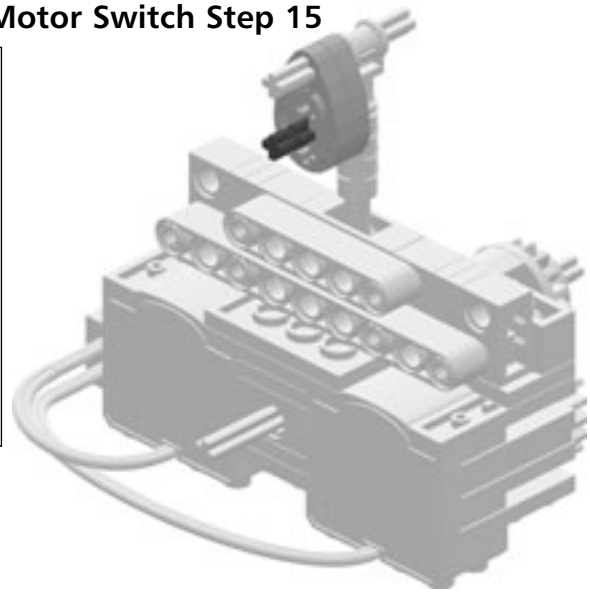
Dual Motor Switch Step 14



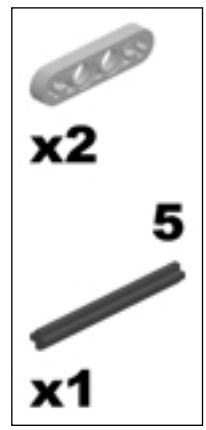


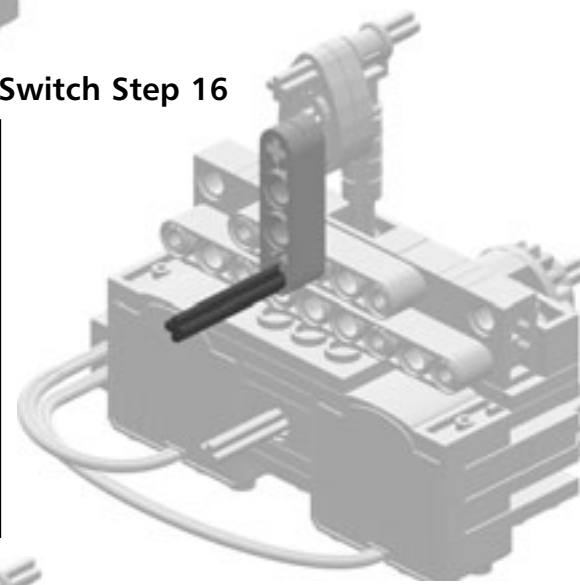
Dual Motor Switch Step 15



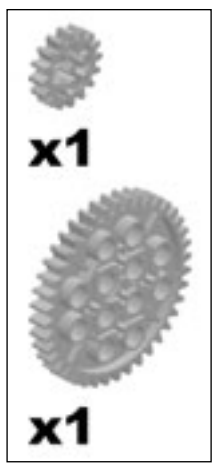


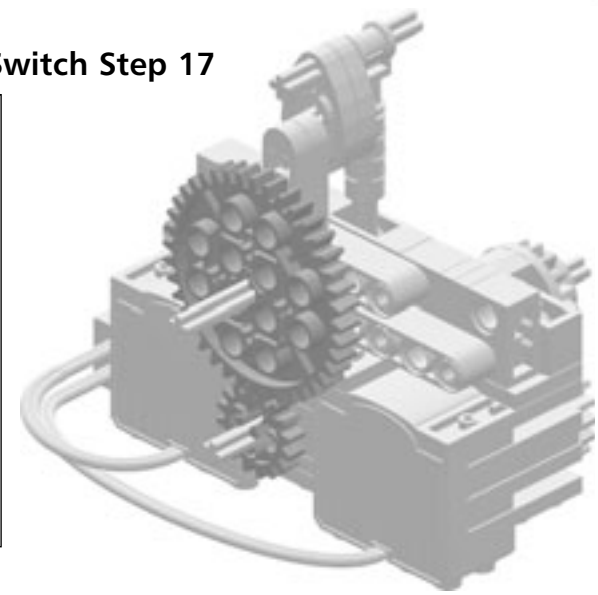
Dual Motor Switch Step 16



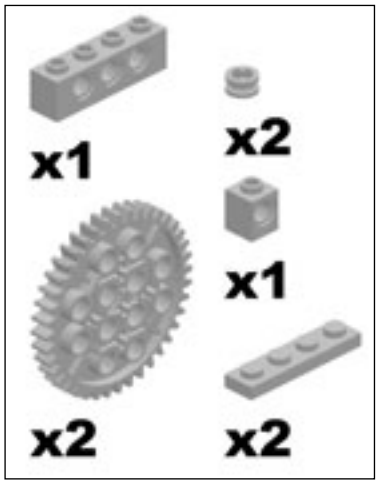


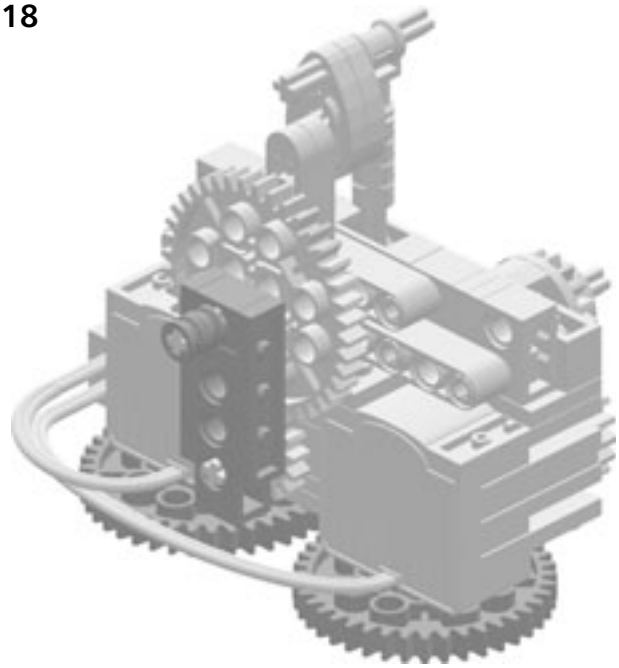
Dual Motor Switch Step 17



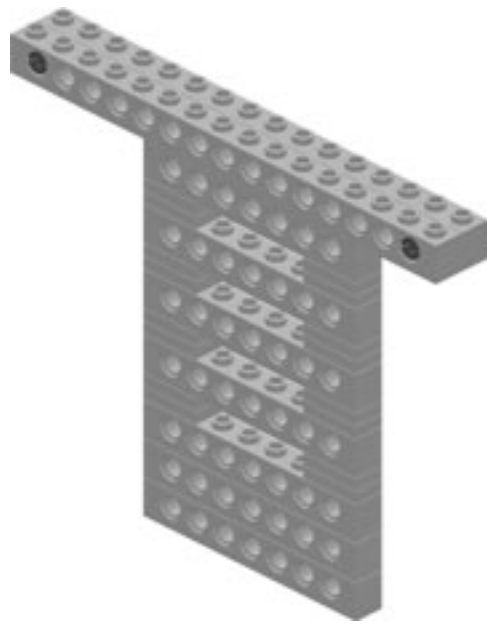


Dual Motor Switch Step 18



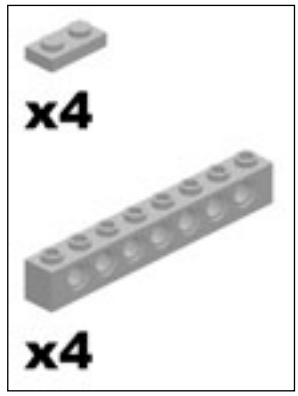


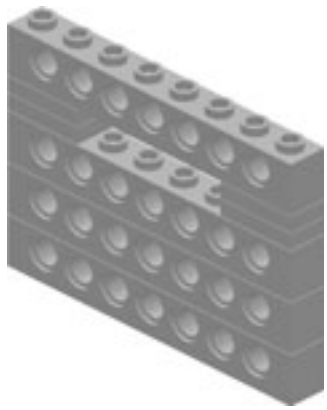
The Pump Walls



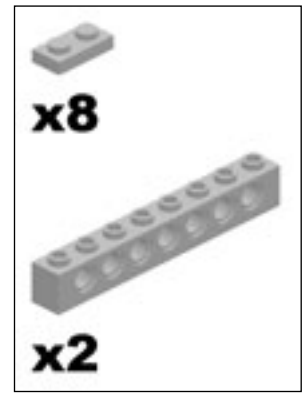
The compressor for PneumADDic II consists of two Pump Wall sub-assemblies, each of which contains two pumps. First, we will construct the walls of the pumping mechanism. Remember to build **two** of these.

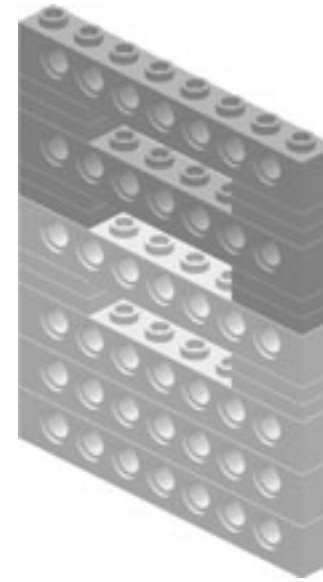
Pump Wall Step 0



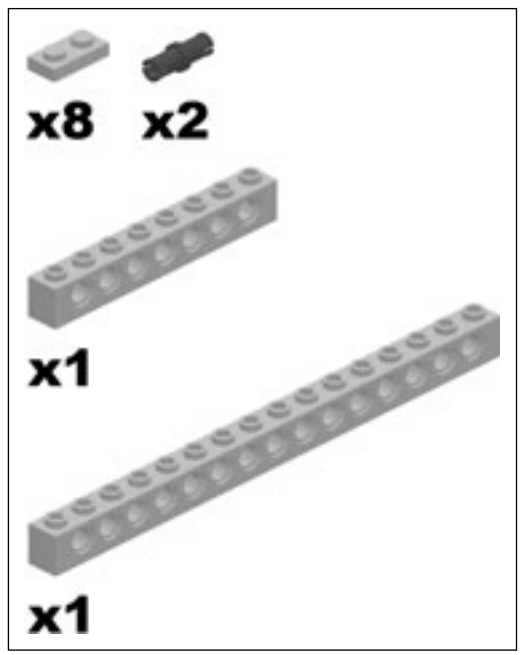


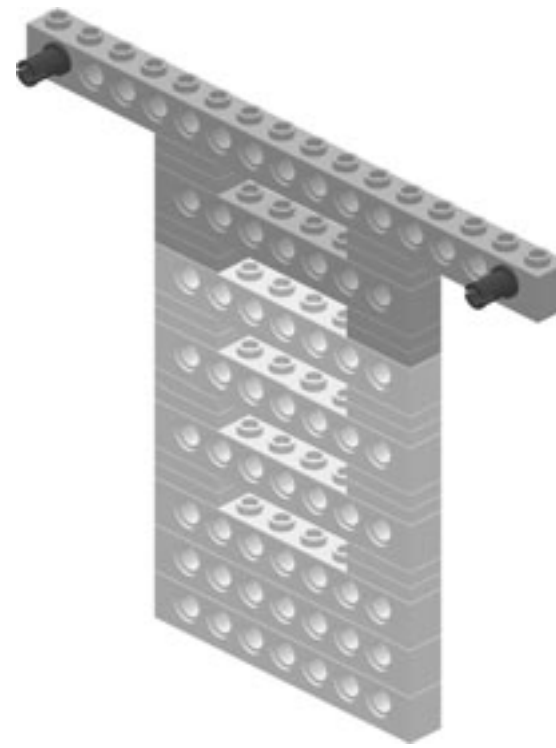
Pump Wall Step 1



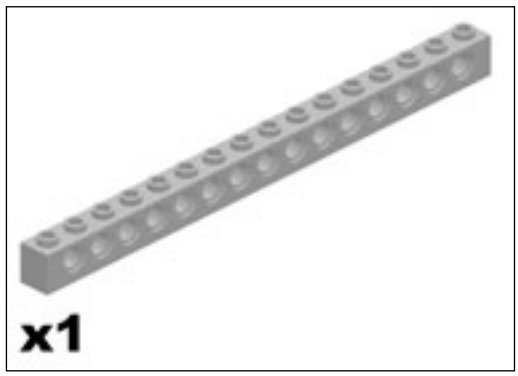


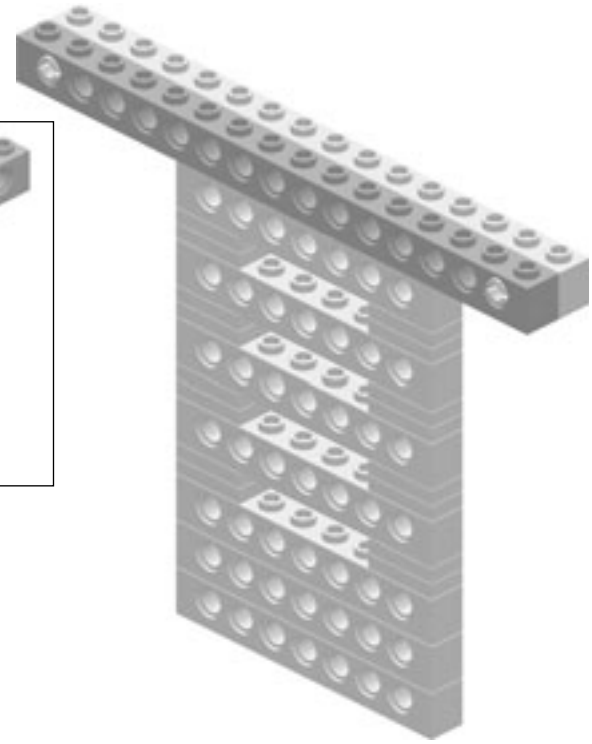
Pump Wall Step 2



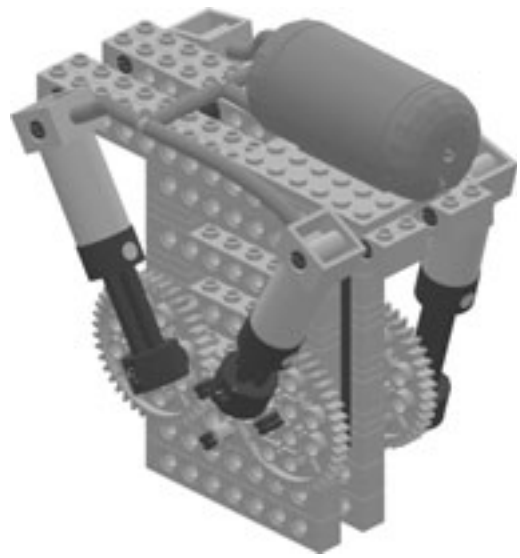


Pump Wall Step 3





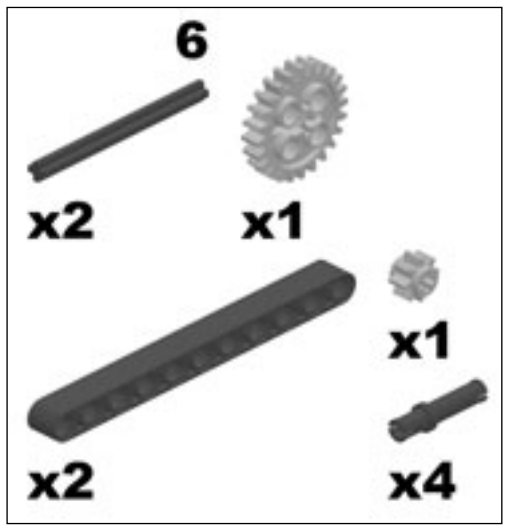
The Pump

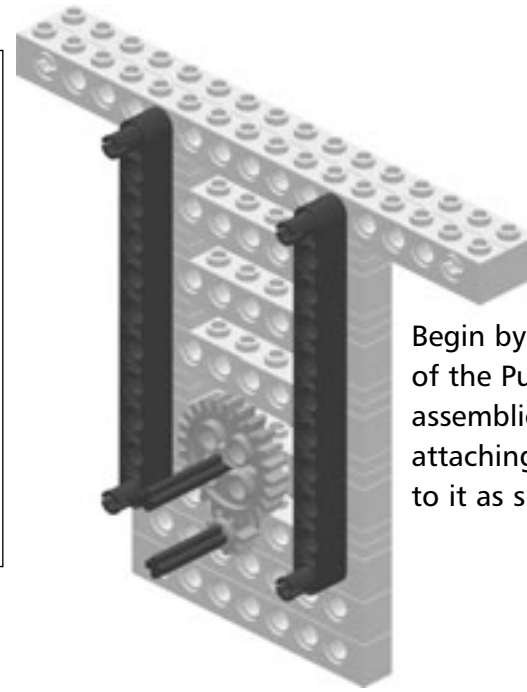


The Pump sub-assembly contains the pumps and an air pressure tank for the compressor. As described earlier, only one of the four pumps is driven to maximum compression at any one time, which decreases the stress placed upon the pump motors. When connecting the pumps to the gears, be careful to attach the pumps exactly as shown to ensure that the pumps only compress one at a time. For this sub-assembly, you will require four large

pumps. Large pumps are manufactured by LEGO with springs that return the pump handle back to its uncompressed state. Removing these springs makes the compression stroke of each pump less work for the motors; therefore, prior to starting this sub-assembly you should remove the springs from the pumps.

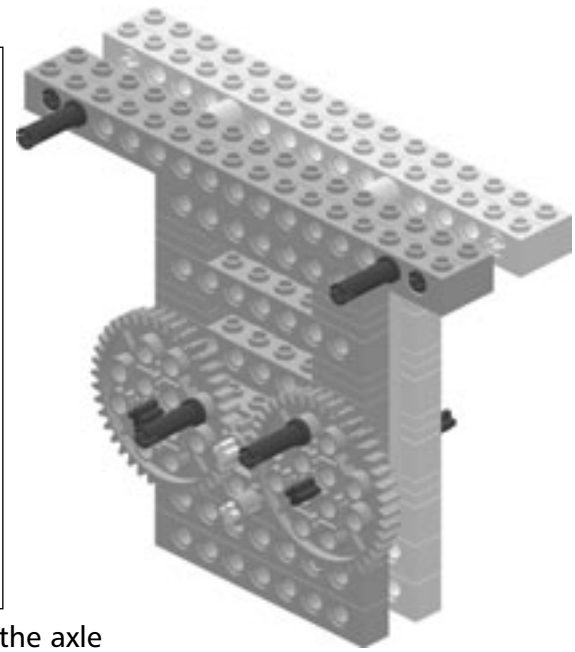
Pump Step 0





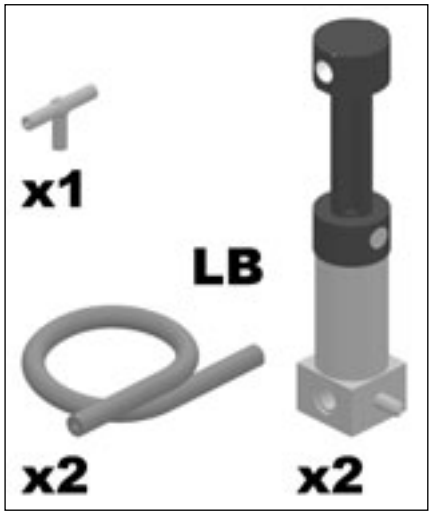
Begin by taking one of the Pump Wall sub-assemblies and attaching these parts to it as shown.

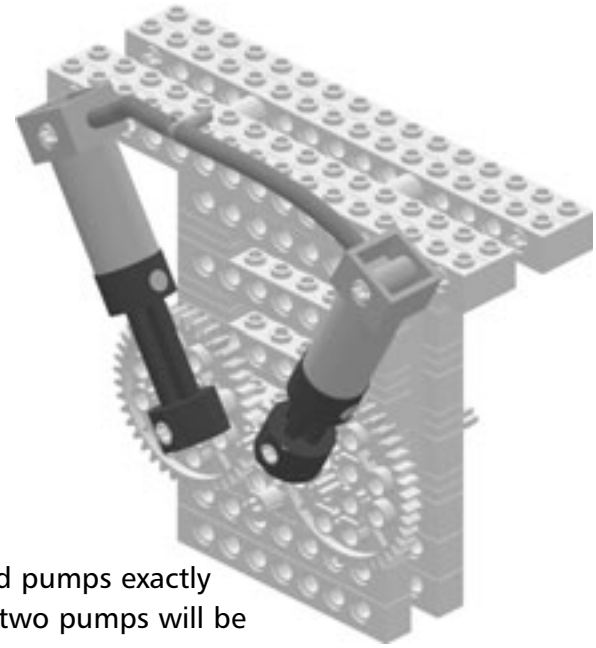
Pump Step 1



Note that you will need to slide the axle joiner onto the side of the lower #6 axle you added in **Pump Step 0** that is on the side of the wall that is not visible. For proper placement of the axle joiner please refer to **Pump Step 3**.

Pump Step 2



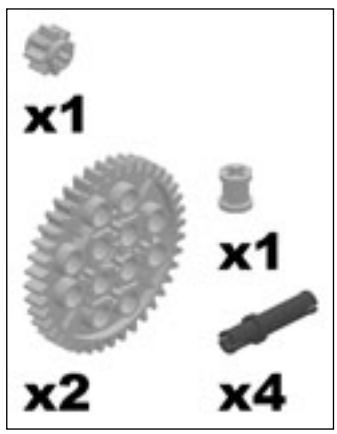


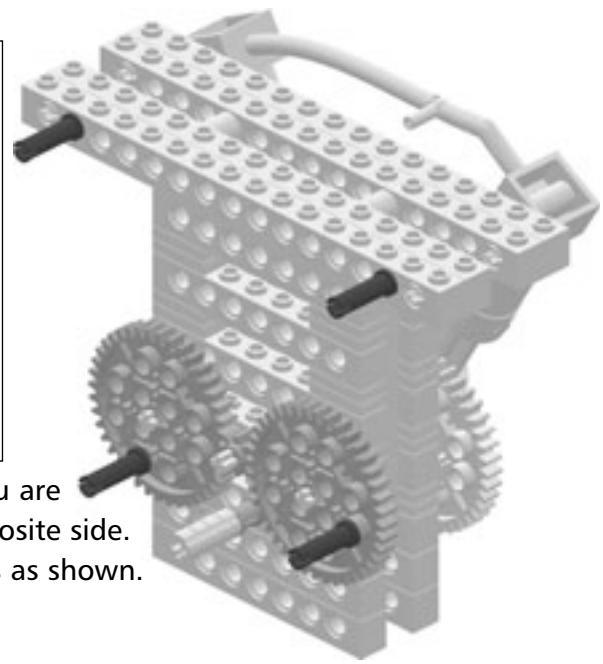
Be careful to align the gears and pumps exactly as shown. This ensures that no two pumps will be able to simultaneously compress.

NOTE

Notice that the springs have been removed from the large pumps. This reduces the effort needed by the motors to compress the pumps.

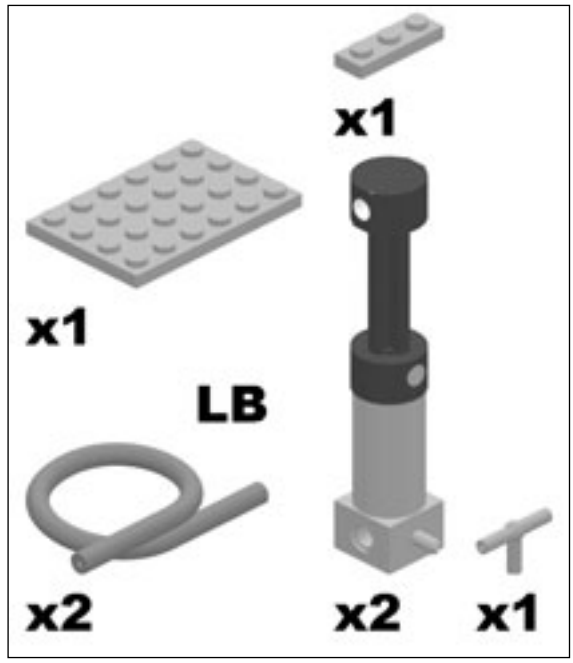
Pump Step 3

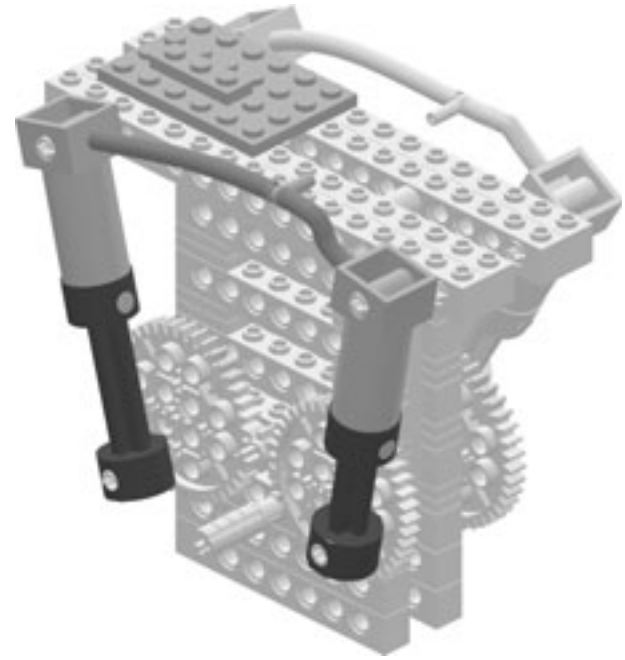




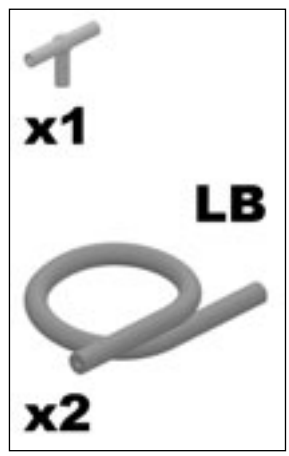
Rotate the model so you are now looking at the opposite side. Attach the various parts as shown.

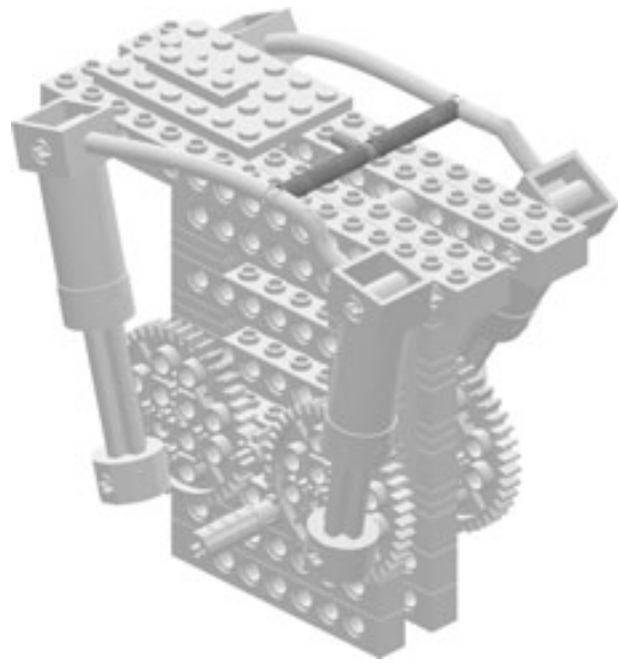
Pump Step 4



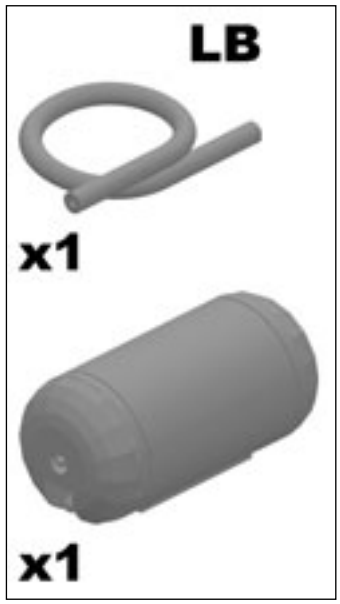


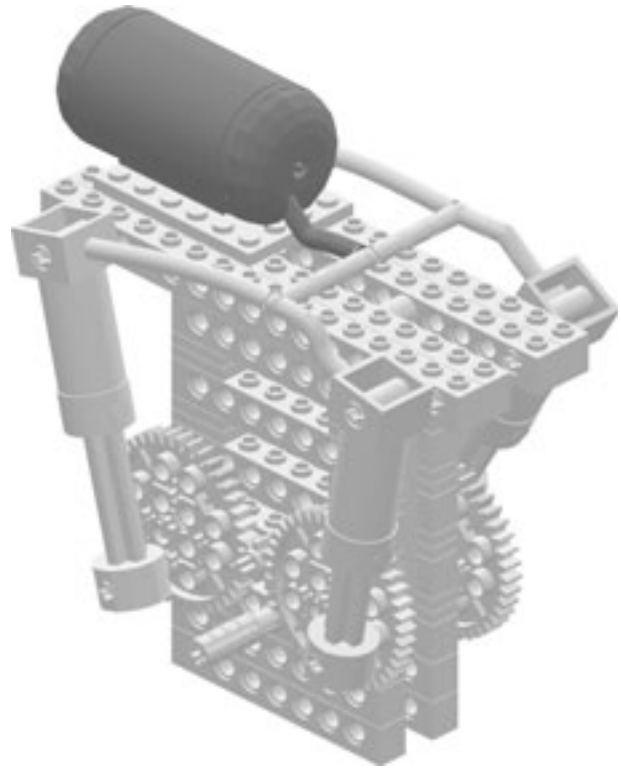
Pump Step 5



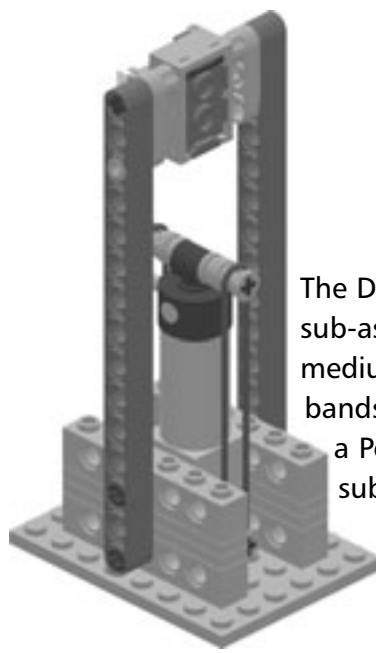


Pump Step 6





The Digital Pressure Sensor



The Digital Pressure Sensor sub-assembly consists of a medium piston, rubber bands, a touch sensor and a Potentiometer Brick sub-assembly.

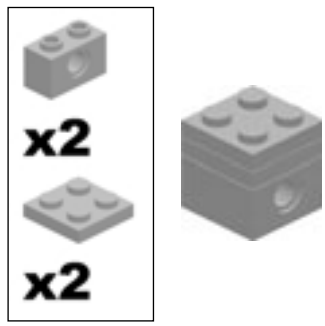
PneumADDic II needs to be able to recognize when to stop running the compressor, so that it does not over-pressurize the pneumatic adding circuit. Over pressurization will lead to self-destruction. If the compressor generates too much pressure, the friction that

holds the pneumatic tubes onto the pneumatic connectors are over-powered by the pressure and the weakest of the connections run the risk of disconnecting and depressurizing the entire circuit.

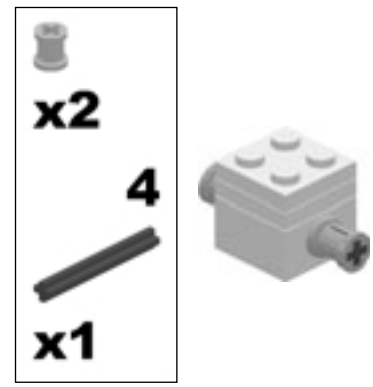
PneumADDic II also needs to be alerted to when the system is up to pressure, so it knows it is time to make the Carry Memory remember a new Carry value.

The Digital Pressure Sensor sub-assembly incorporates a medium piston, rubber bands, a touch sensor, and a Potentiometer Brick sub-assembly. When the system is up to pressure, the piston presses the touch sensor button. When the system is not up to pressure, the rubber bands compress the pneumatic piston, releasing the touch sensor button.

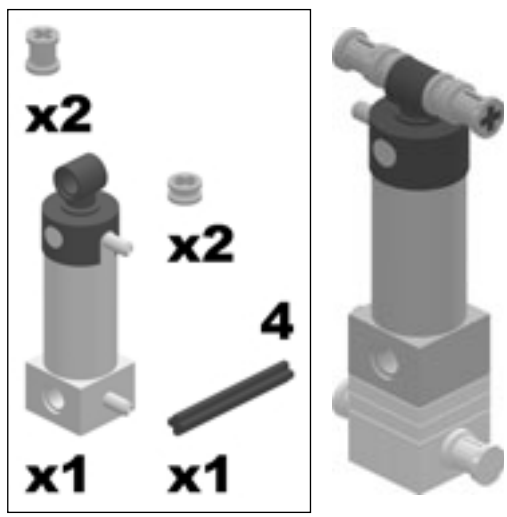
Digital Pressure Sensor Step 0



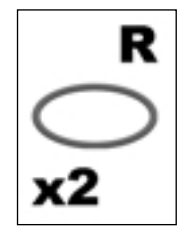
Digital Pressure Sensor Step 1



Digital Pressure Sensor Step 2



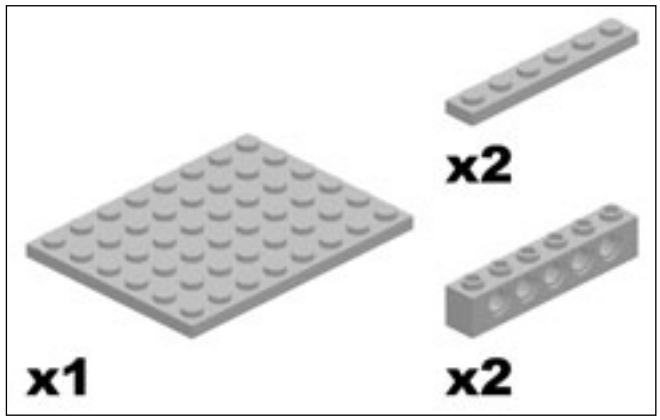
Digital Pressure Sensor Step 3

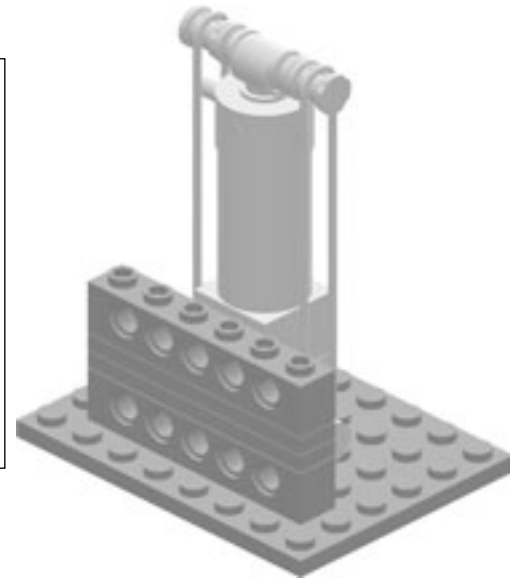


Wrap the rubber band around the top bushing three times (not shown in picture). Do this for both rubber bands.

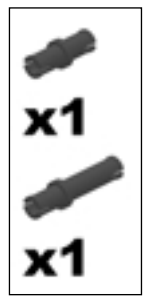
The red rubber bands compress the piston to the closed position, unless there is enough pressure from the pumps to expand the piston.

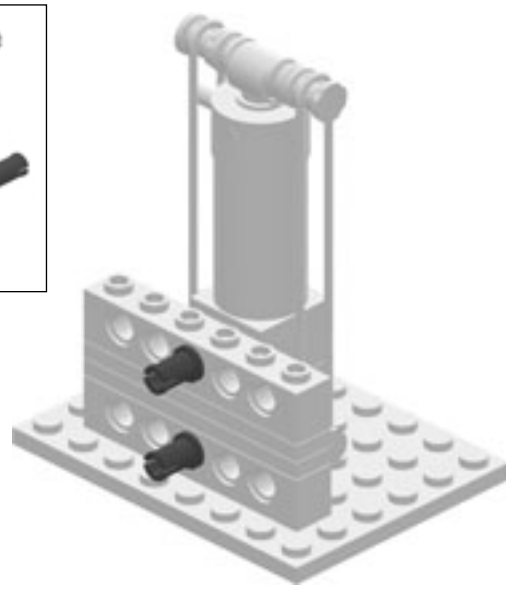
Digital Pressure Sensor Step 4



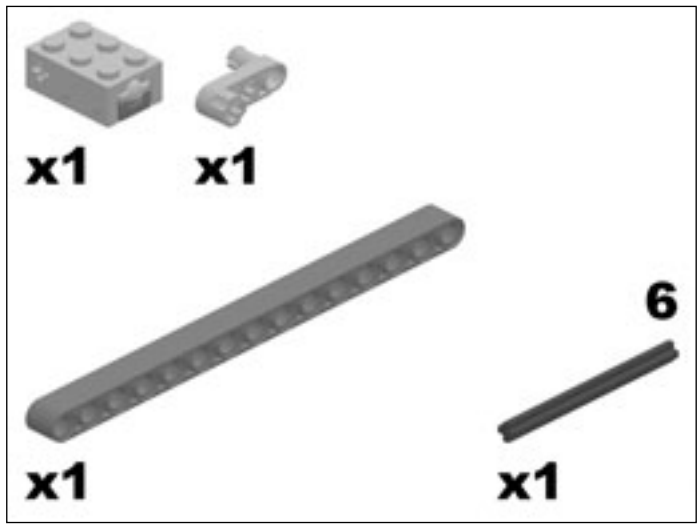


Digital Pressure Sensor Step 5

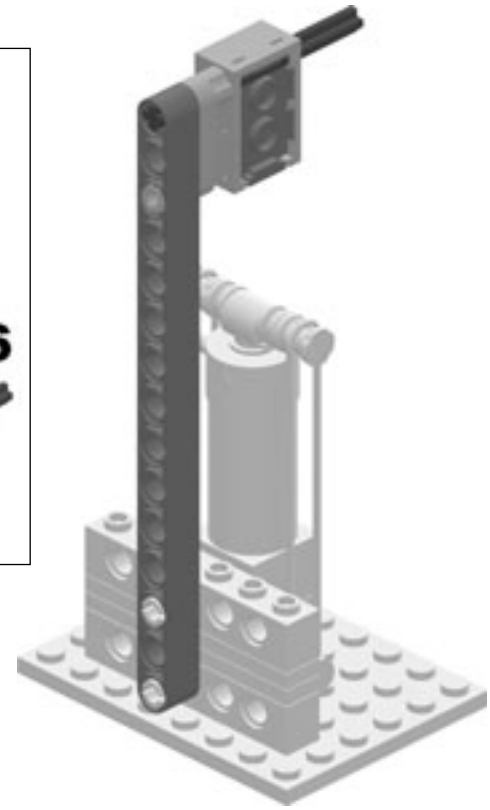




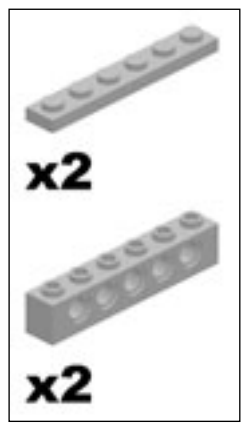
Digital Pressure Sensor Step 6

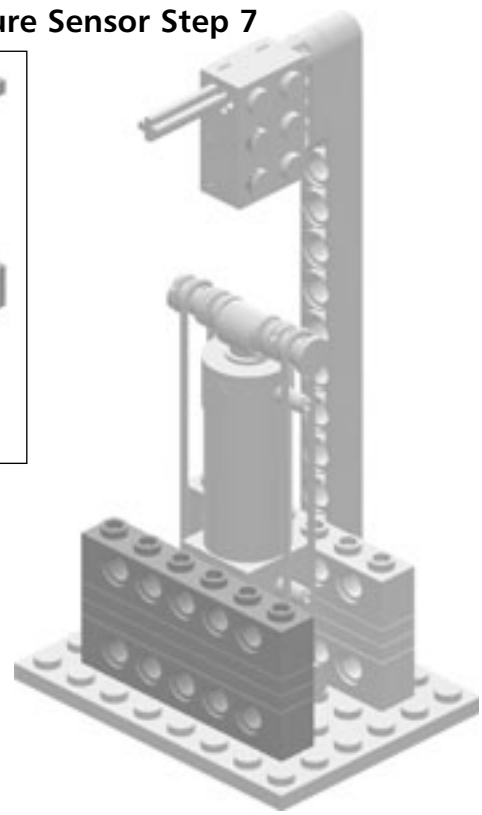


When the pressure is high enough to overcome the compression force of the rubber bands, the piston expands and depresses the touch sensor.

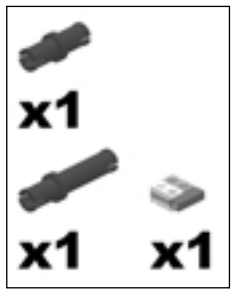


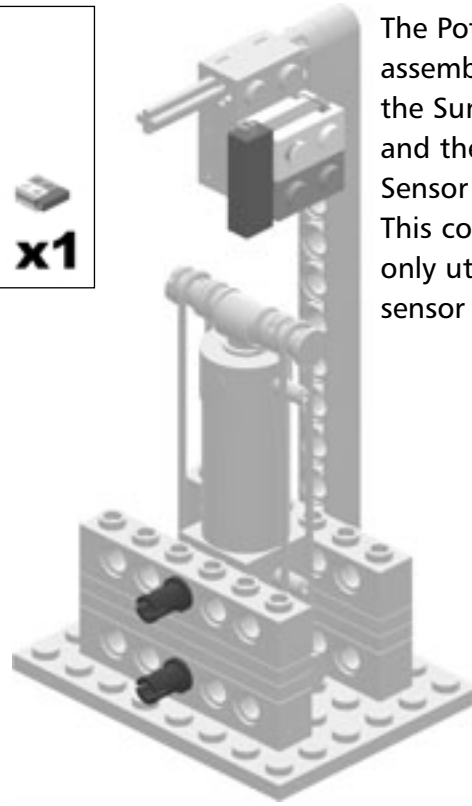
Digital Pressure Sensor Step 7





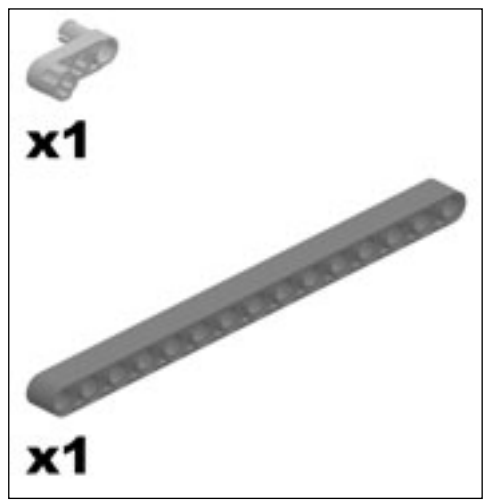
Digital Pressure Sensor Step 8

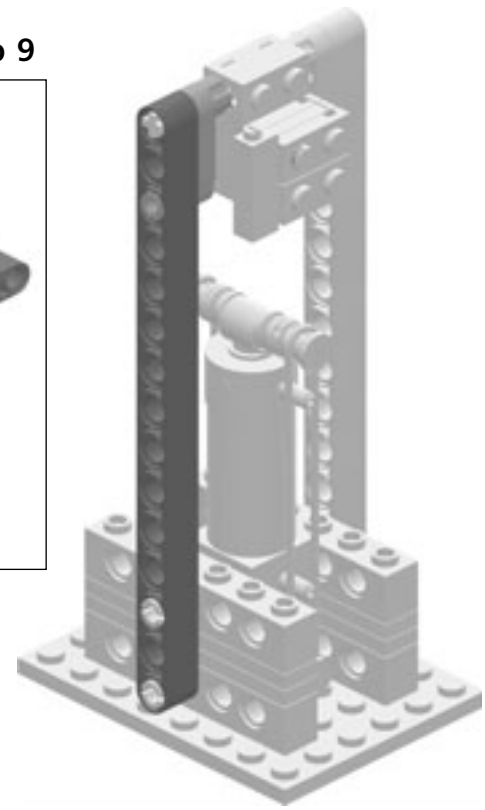




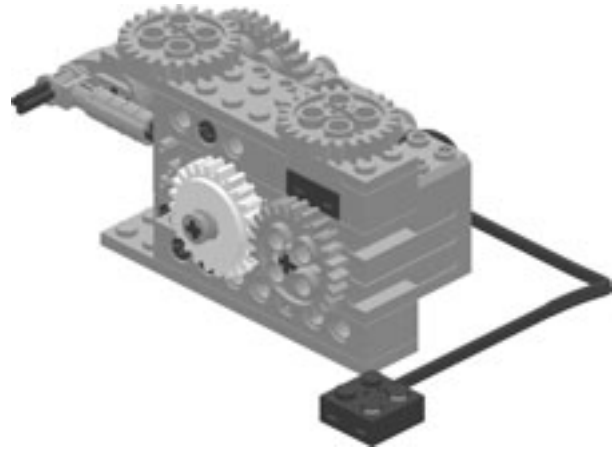
The Potentiometer Brick sub-assembly allows you to attach the Sum Sensor sub-assembly and the Digital Pressure Sensor sub-assembly together. This combination will then only utilize a single RCX sensor input.

Digital Pressure Sensor Step 9



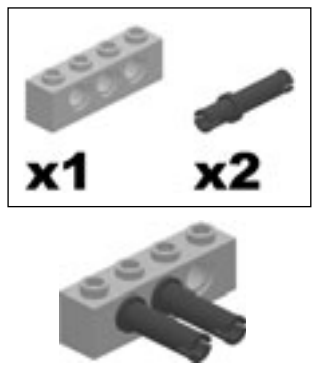


The Motorized Switches

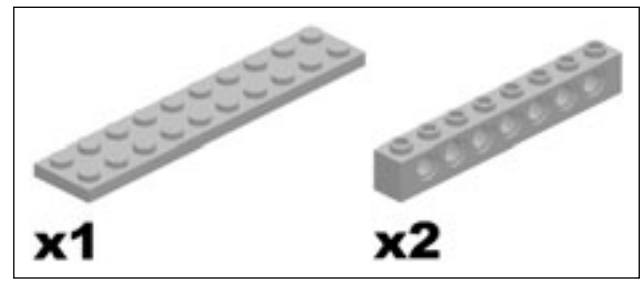


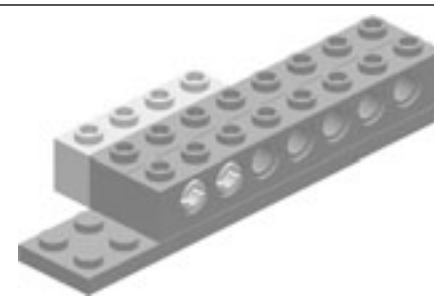
The Dual Motor Switch sub-assembly provides one motorized switch, but PneumADDic II needs two more. The following building instructions describe how to make PneumADDic II's Motorized Switch sub-assemblies. You will need to build **two** of these. The Motorized Switch sub-assemblies require a clutch gear (the white gear added in **Motorized Switch Step 10**), which prevents the motor from seizing, or causing undue stress to the switch or the RCX by allowing the axle of the motor to turn when the switch hits its limit.

Motorized Switch Step 0

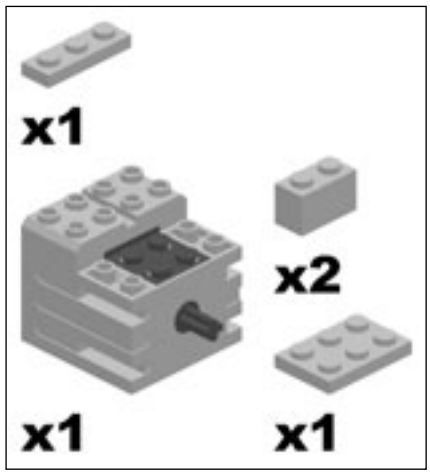


Motorized Switch Step 1

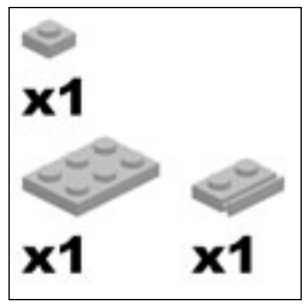


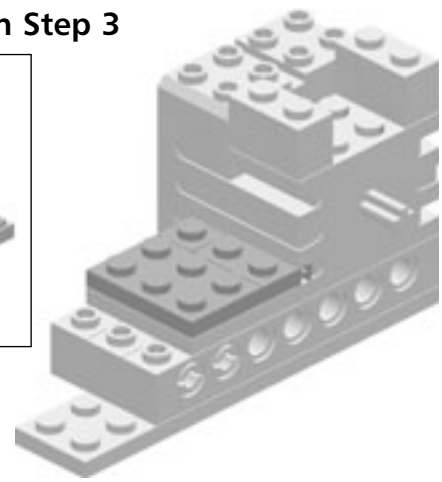


Motorized Switch Step 2

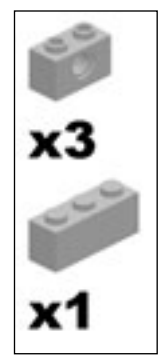


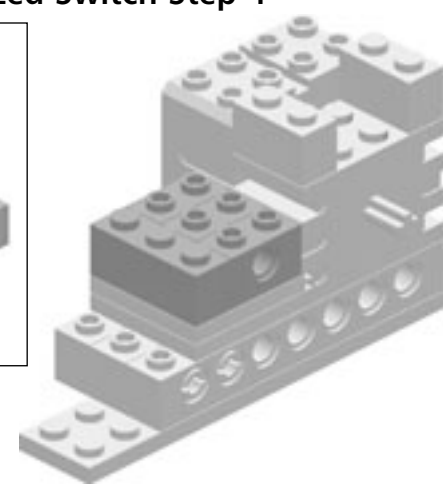
Motorized Switch Step 3



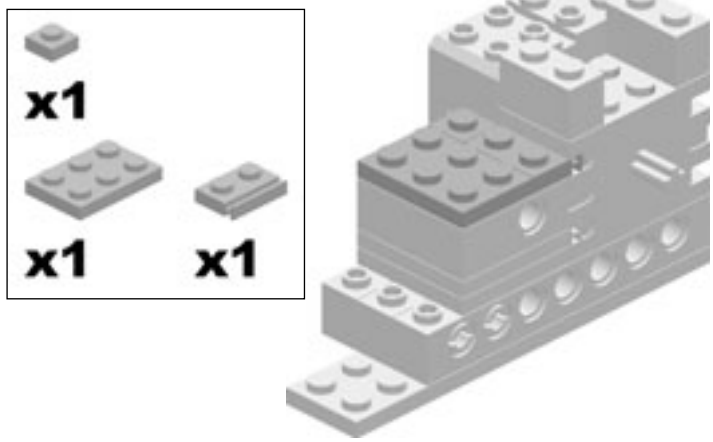


Motorized Switch Step 4

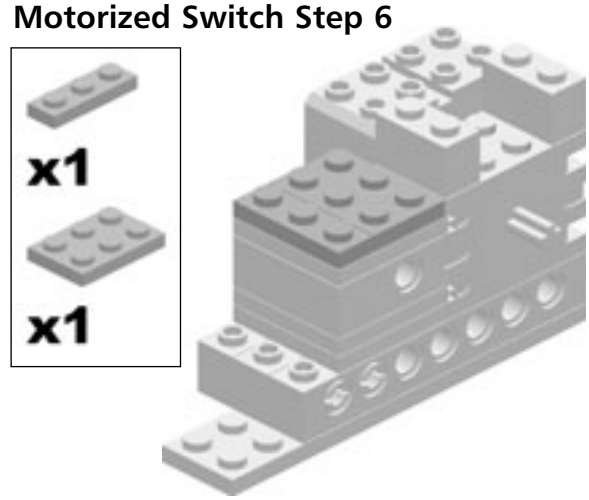




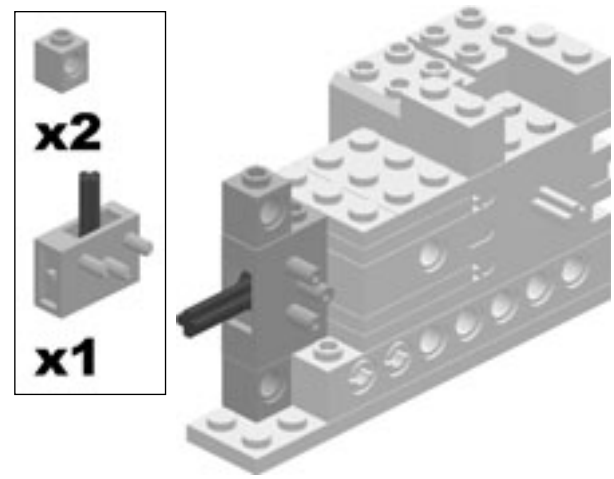
Motorized Switch Step 5



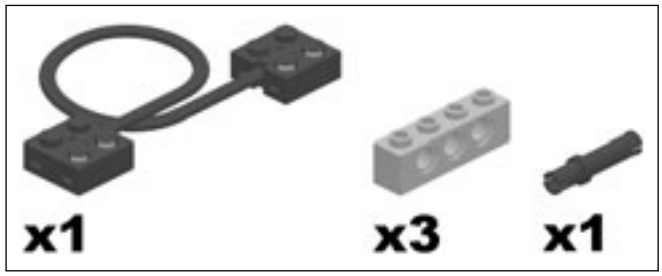
Motorized Switch Step 6

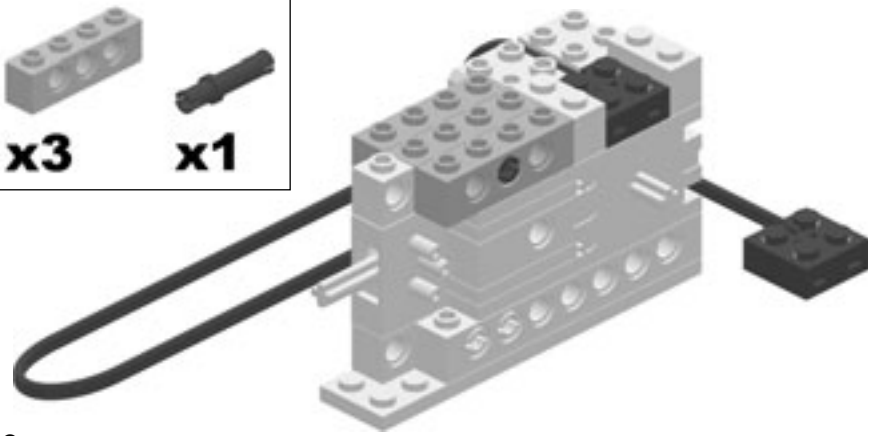


Motorized Switch Step 7

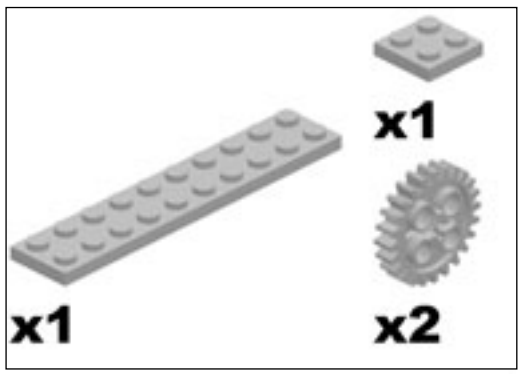


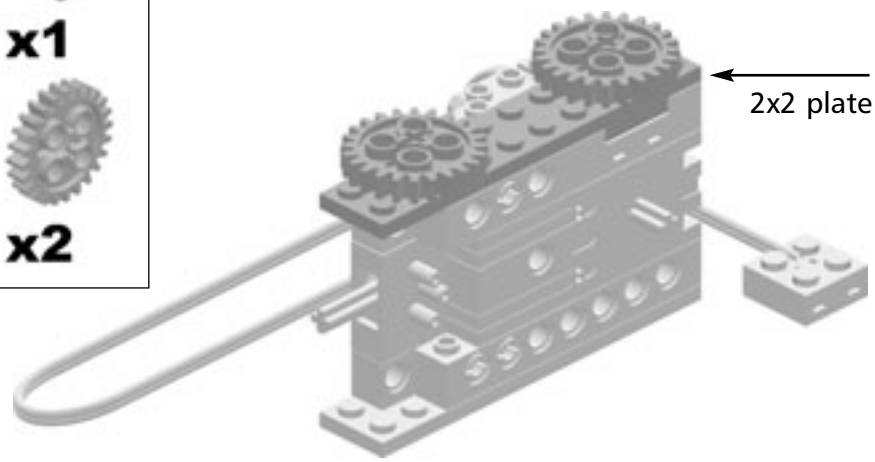
Motorized Switch Step 8



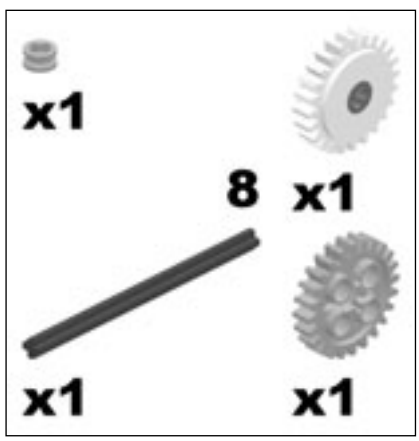


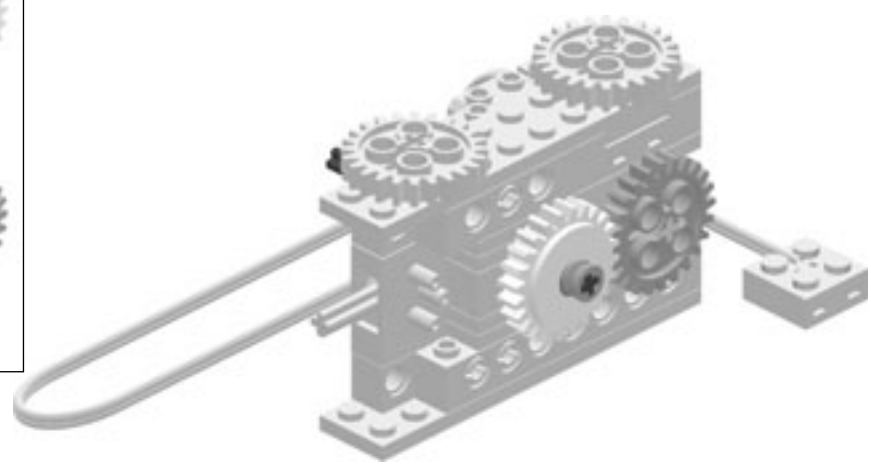
Motorized Switch Step 9





Motorized Switch Step 10

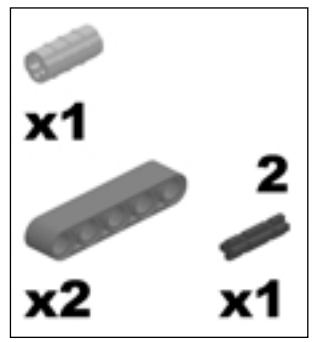


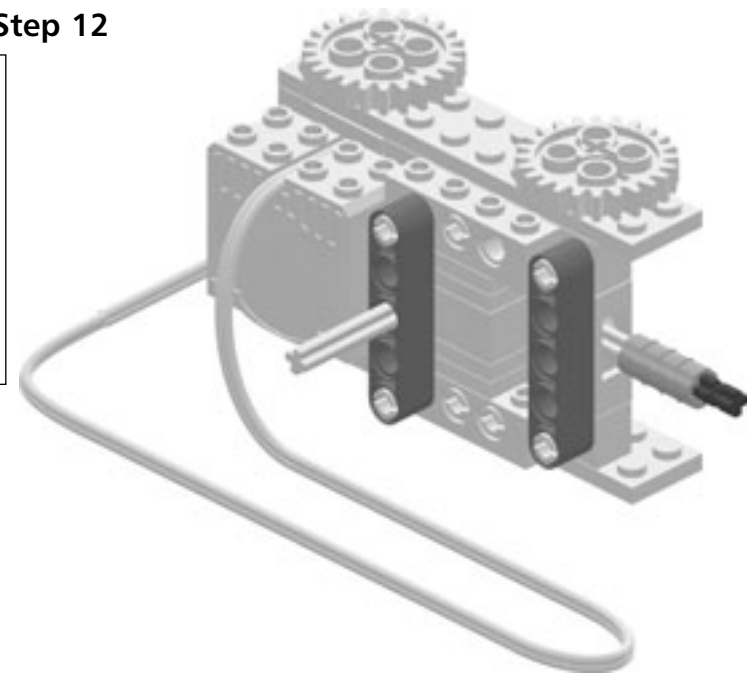


Motorized Switch Step 11

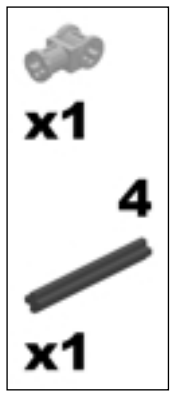
Rotate the model so you are now looking at the opposite side and insert the pins as shown.

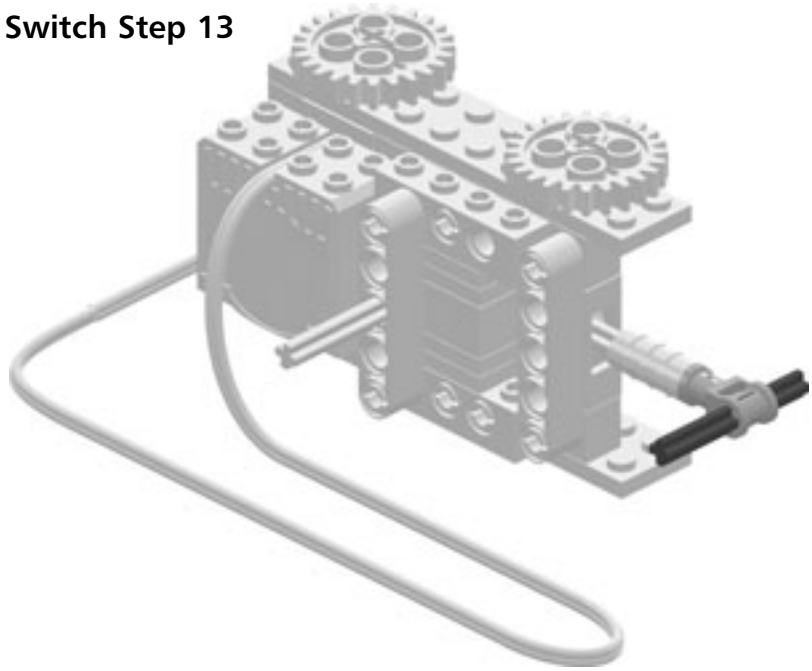
Motorized Switch Step 12



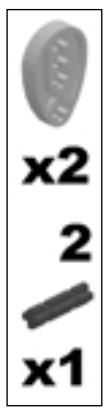


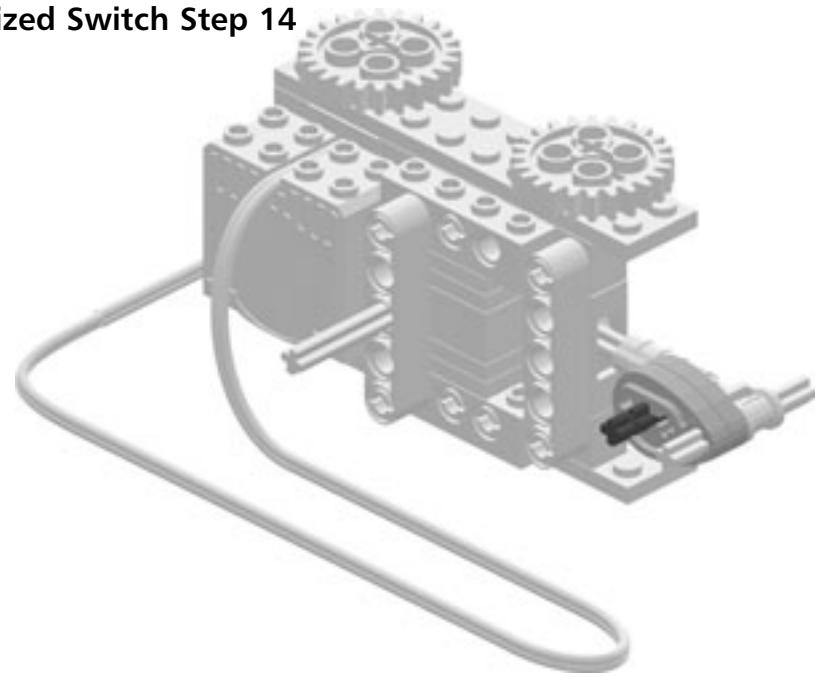
Motorized Switch Step 13



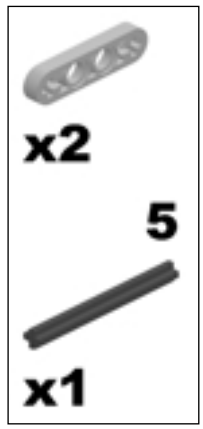


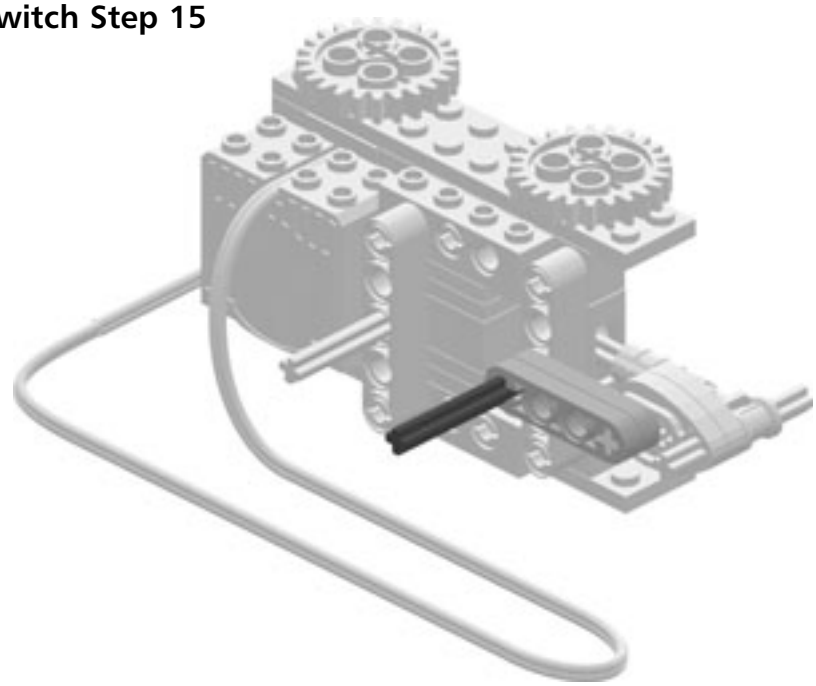
Motorized Switch Step 14



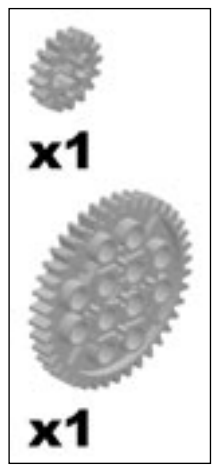


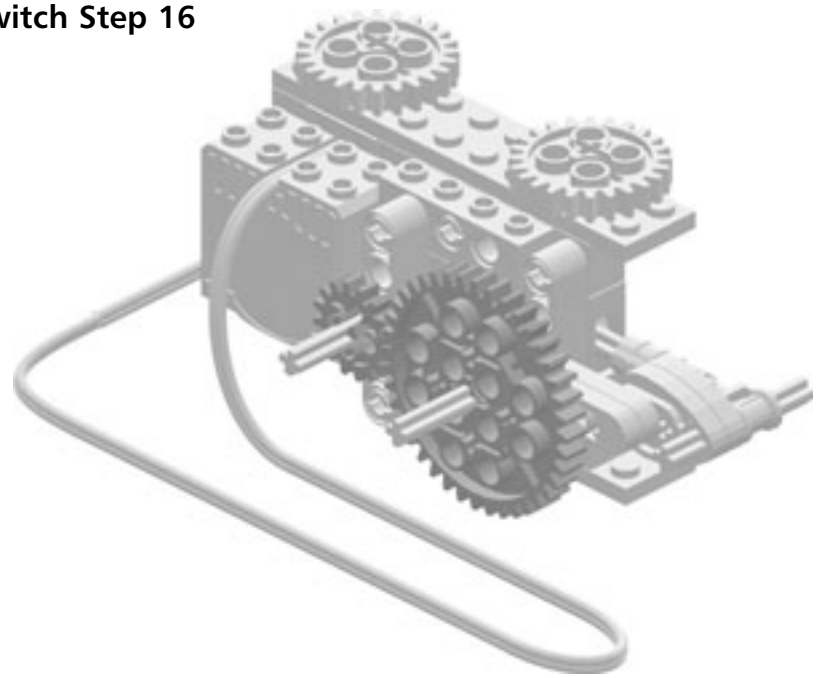
Motorized Switch Step 15



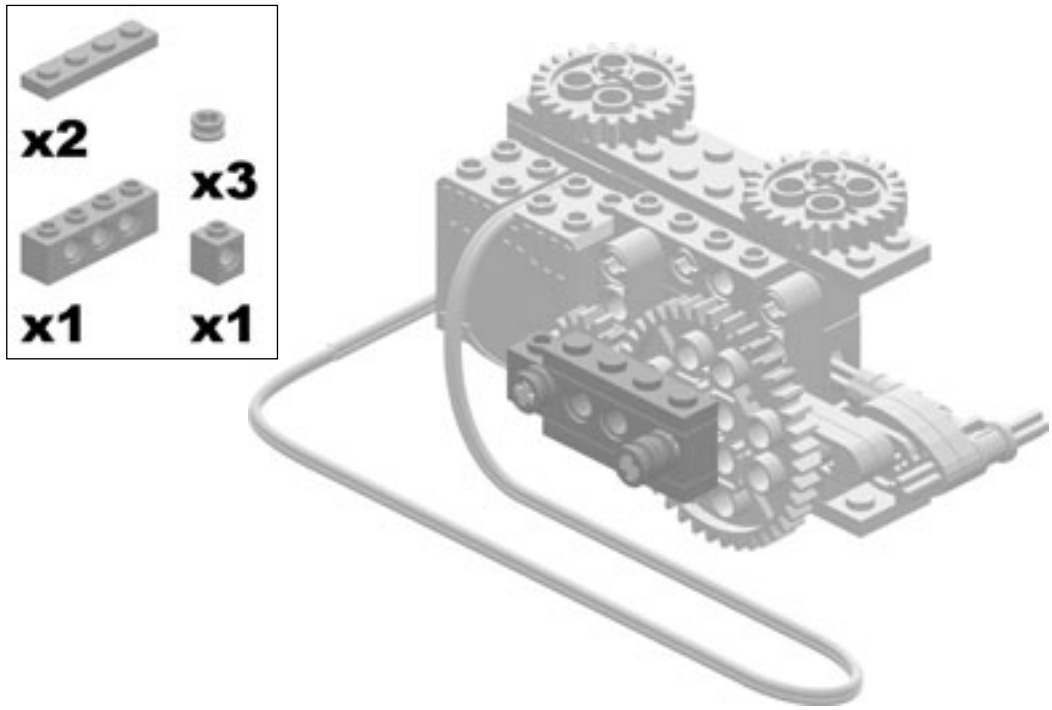


Motorized Switch Step 16





Motorized Switch Step 17



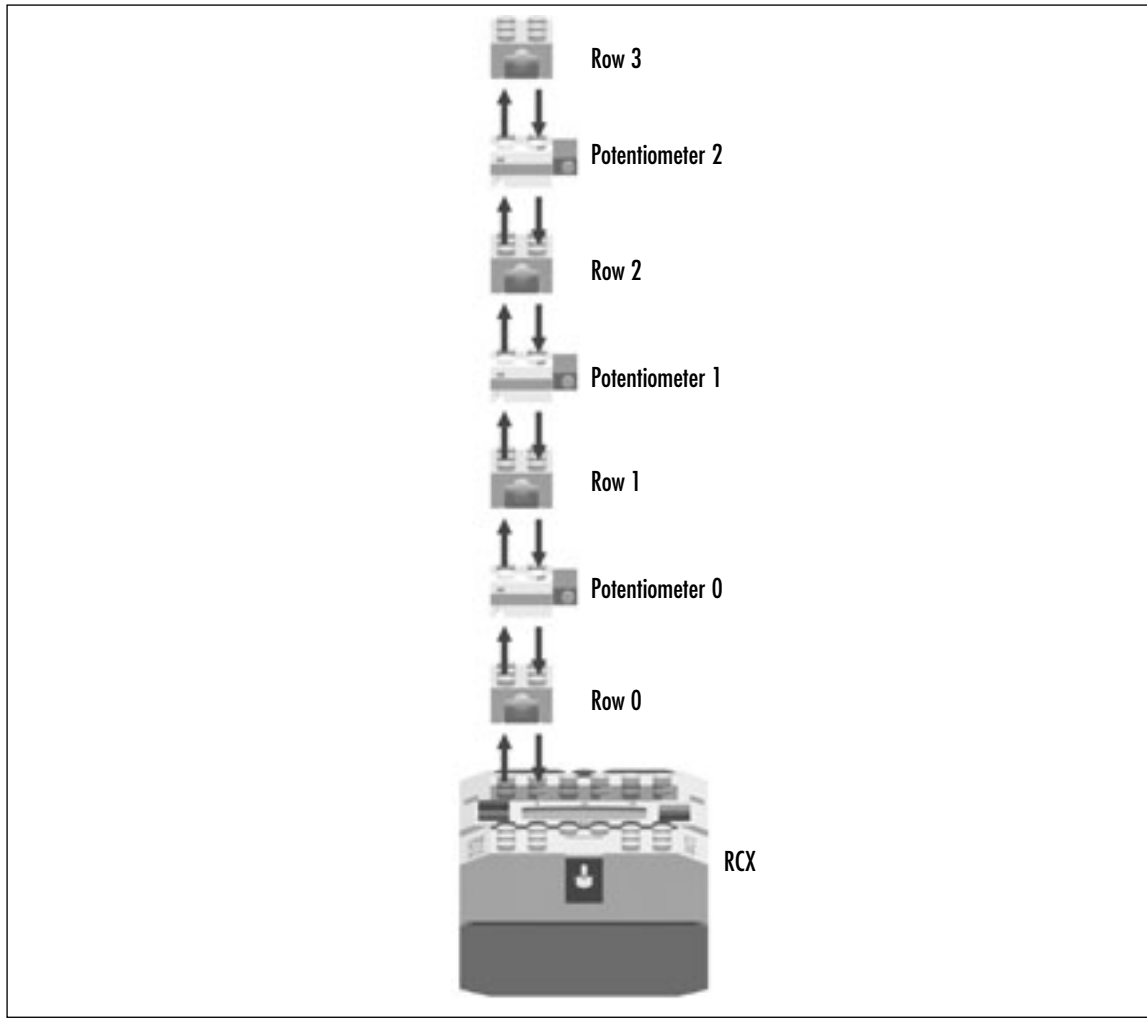
The Keyboard Module

PneumADDic II requires a keyboard in order for the user to input the numbers for the addition and subtraction calculations. PneumADDic II uses the same keyboard design previously used in the original PneumADDic. Why mess with perfection!

PneumADDic II's Keypad sub-assembly consists of 16 keys (four rows of four keys columns). The keyboard design incorporates eight touch sensors: Four touch sensors are used for the Keyboard Row sub-assemblies (one touch sensor for each of row), and four more touch sensors for each of the Keyboard Column sub-assemblies (one touch sensor for each of the columns). The Keyboard Row sub-assembly touch sensors connect to one sensor input of the RCX using a Potentiometer Brick sub-assembly. The Keyboard Column sub-assembly touch sensors connect to second RCX sensors input, again using a Potentiometer Brick sub-assembly.

Before attempting to build the Keyboard sub-assembly, you should understand how the Potentiometer Brick sub-assemblies allow you to connect more than one touch sensor to an RCX sensor port, and still tell which touch sensor is pressed. Figure 1.10 is a summary view of the keyboard row touch sensor/potentiometer brick circuit hooked to RCX Sensor Input 1. The potentiometer bricks have two paths for electricity to flow.

Figure 1.10 Combining Four Touch Sensors on One RCX Input



- One path (the left side of the Potentiometer Brick sub-assembly in Figure 1.10) is a wire where the electricity flows freely
- The other path the electricity must go through the potentiometer (the right side of the bricks in Figure 1.10)

Electricity flows from the RCX and wants to complete the circuit to the RCX, as shown by the arrows closest to the RCX. No electricity is able to flow from one point to the other unless one of the touch sensors is pressed. When a touch sensor is pressed, electricity flows *through* the touch sensor, and then tries to complete the circuit to return to the RCX. This is best seen by following the arrows shown in Figure 1.10.

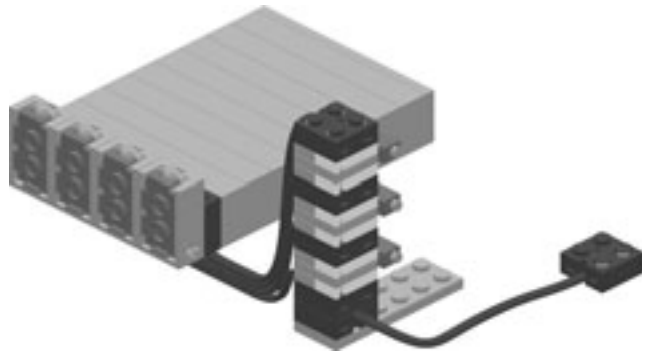
Four distinct actions occur during the process:

- When the touch sensor for Row 0 is pressed, the electricity flows through the touch sensor and directly back into the RCX. The RCX senses that all the electricity it has sent out has been returned.
- When the touch sensor for Row 1 is pressed, electricity flows through the touch sensor, but must travel back through Potentiometer 0, before it can get back to the RCX. Potentiometer 0 converts a portion of the electricity that flows through it into heat, therefore the RCX reads that less electricity is coming back, and receives a lower analog reading for Row 1 than it received for Row 0.
- When the touch sensor for Row 2 is pressed, electricity must travel through Potentiometer 0 and Potentiometer 1 before completing the circuit to the RCX. The reading that the RCX receives for Row 2 is lower than the readings for Row 1 and Row 0.
- Following this logic, the touch sensor for Row 3 receives an even lower reading than Rows 0 through 3, because the electricity in this circuit must pass through three potentiometer bricks.

As you can see, the potentiometer bricks provide a simple way of merging the four touch sensors onto a single RCX port. In electronics terms, the electricity must travel *in series* through a touch sensor, and then zero or more potentiometer bricks to return to the RCX. Series resistor networks are relatively simple to work with. We can use a series resistor networks for our Keyboard inputs because only one row and column touch sensor is pressed at a time (unlike the Sum Sensor sub-assembly and the Digital Pressure Sensor sub-assembly that can both be pressed at the same time).

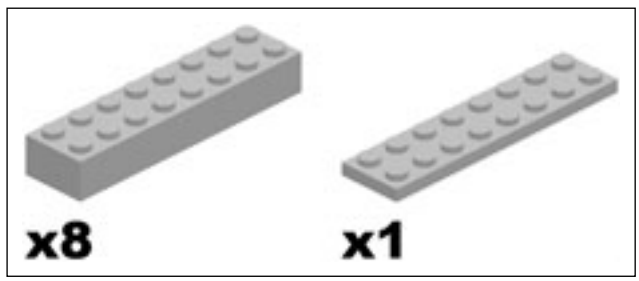
When we connect the Sum Sensor sub-assembly and the Digital Pressure Sensor sub-assembly together onto a single RCX input, things become more complicated. The Sum Sensor and Pressure Sensors' touch sensor buttons can both be pressed at the same time. However, we will address this issue, and how to prevent this problem, when we get to that step in the building instructions. First, let's build the keyboard.

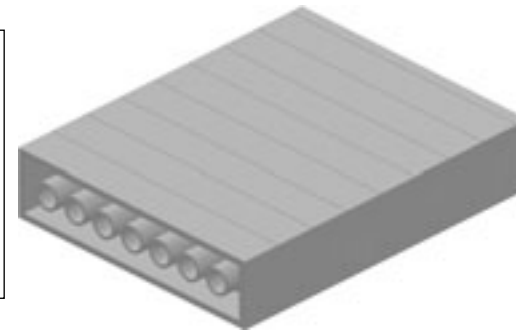
Keyboard Column Sensor



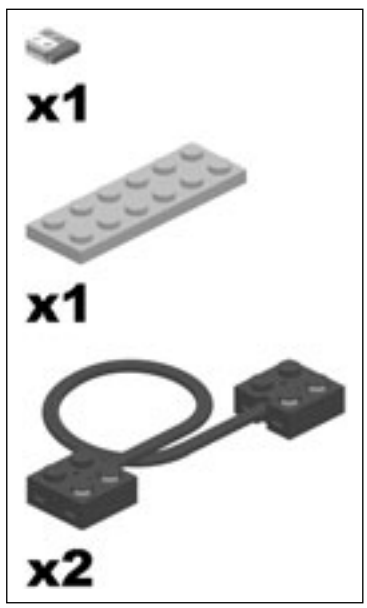
The Keyboard Column Sensor sub-assembly is an important part of PneumADDic II's keyboard. As you will see, there is one touch sensor for each of the columns on the actual keyboard.

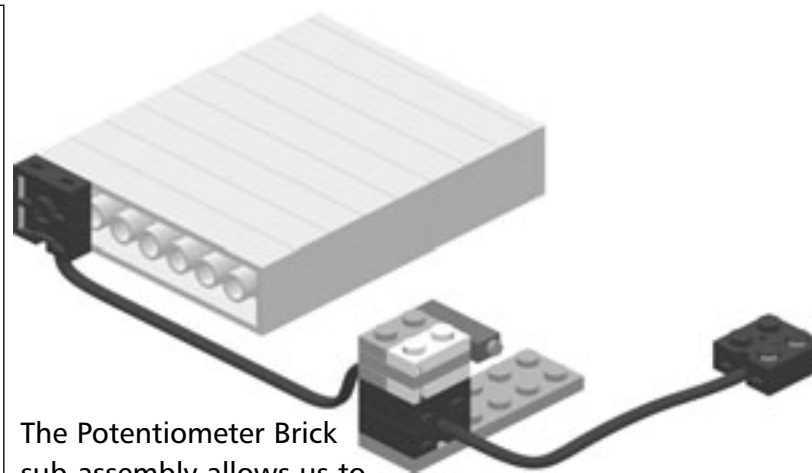
Keyboard Column Sensor Step 0





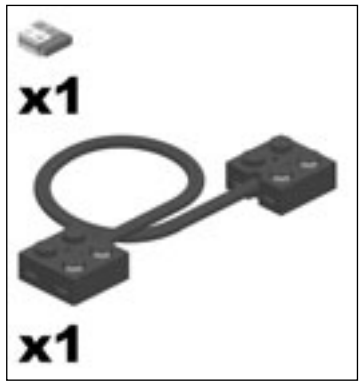
Keyboard Column Sensor Step 1

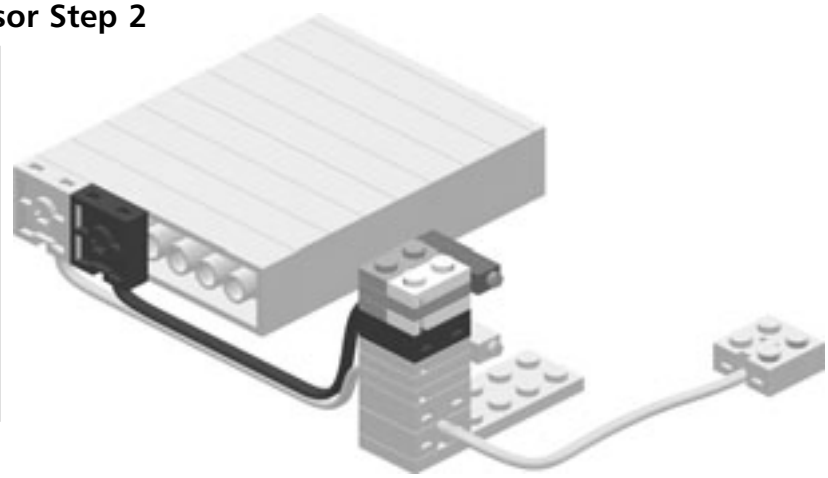




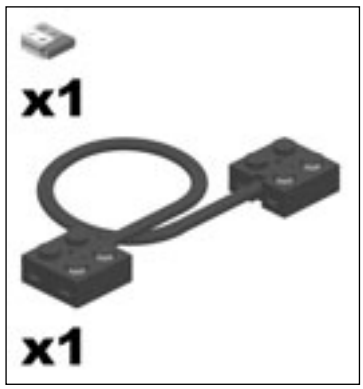
The Potentiometer Brick sub-assembly allows us to combine all four of the column touch sensors onto one RCX sensor input.

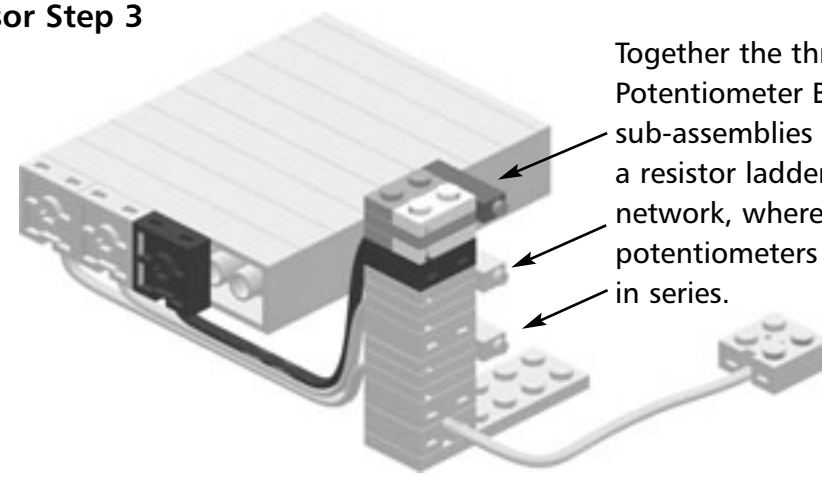
Keyboard Column Sensor Step 2





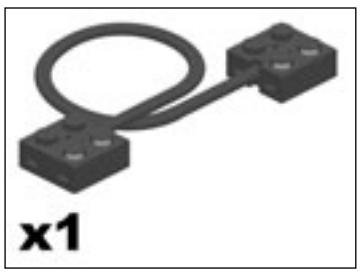
Keyboard Column Sensor Step 3

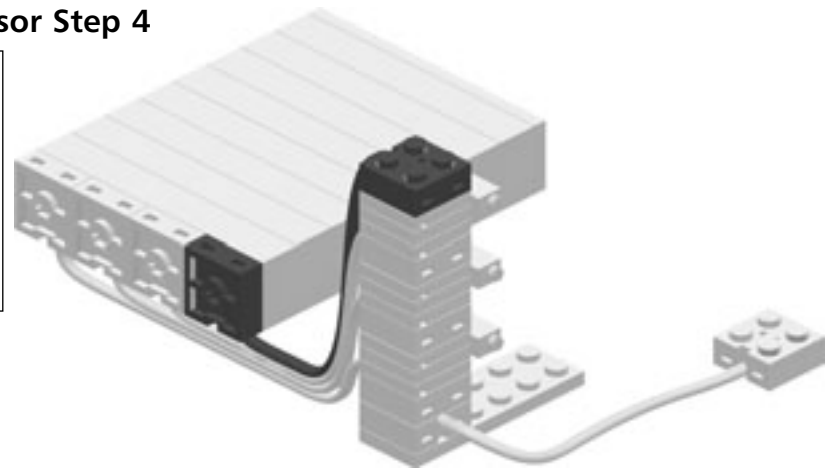




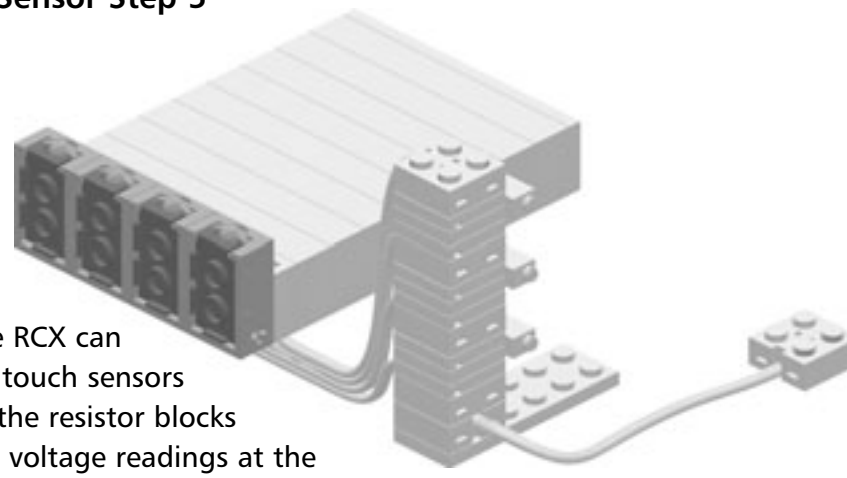
Together the three Potentiometer Brick sub-assemblies form a resistor ladder network, where the potentiometers are in series.

Keyboard Column Sensor Step 4





Keyboard Column Sensor Step 5

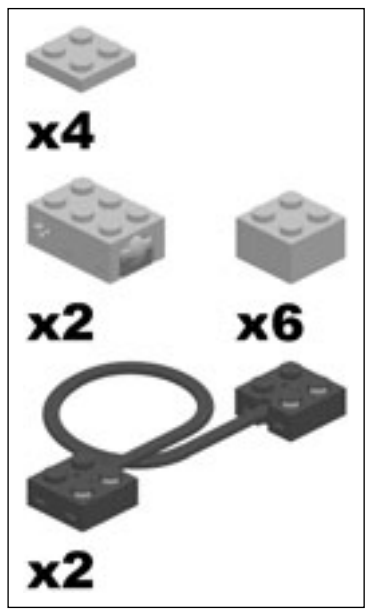


Only one of the four touch sensors is ever pressed at a time. The RCX can tell which of the four touch sensors was pressed because the resistor blocks force different analog voltage readings at the RCX sensor input for different touch sensors.

Left Keyboard Buttons

The Left Keyboard Buttons sub-assembly contains two touch sensors.

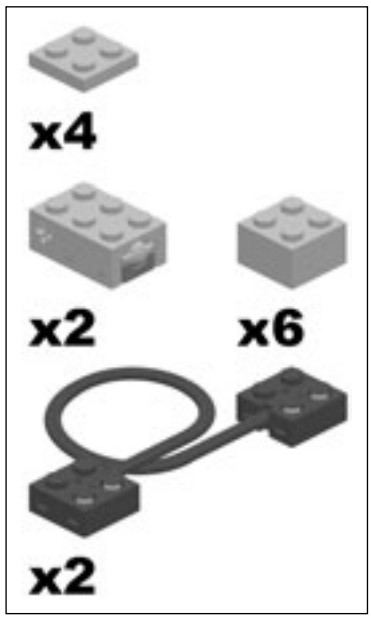
Left Keyboard Buttons Step 0



Right Keyboard Buttons

The Right Keyboard Buttons sub-assembly contains the other two touch sensors for the keyboard. The difference between the Right Keyboard sub-assembly and the Left Keyboard sub-assembly is the position of the touch sensors. The first touch sensor for the Right Keyboard sub-assembly is located on top of three 2x2 bricks and two 2x2 plates, whereas the first touch sensor for the Left Keyboard sub-assembly is located on top of two 2x2 bricks and no 2x2 plates. You will see why this difference is important when you add the Right and Left Keyboard Button sub-assemblies when you put the entire keyboard together.

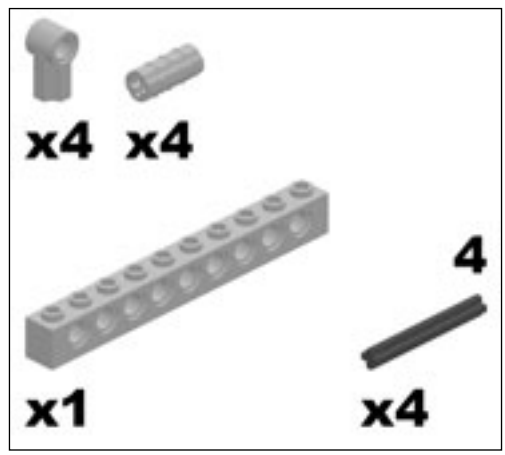
Right Keyboard Buttons Step 0



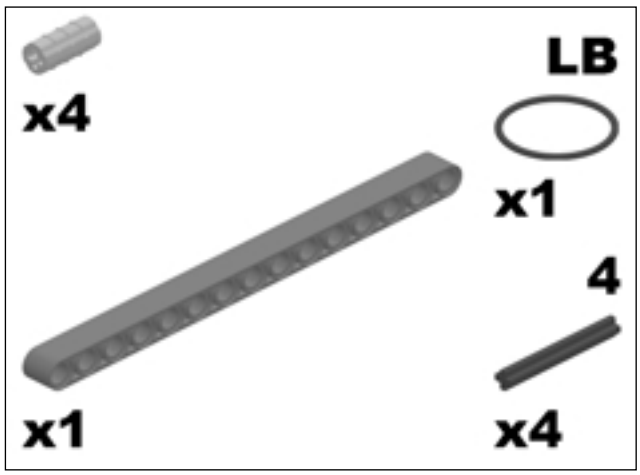
The Keyboard Rows

PneumADDic II's keyboard consists of four rows of four buttons per row. The following building instructions show you how to make one of the Keyboard Row sub-assemblies. You will need to build **four** of these sub-assemblies.

Keyboard Row Step 0



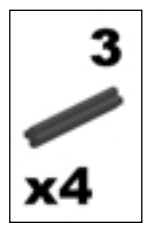
Keyboard Row Step 1

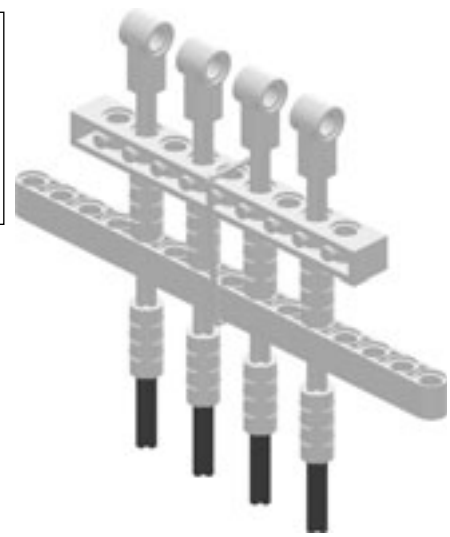


The long liftarms will be used to physically press each keyboard row's matching touch sensor when the keyboard is fully assembled.

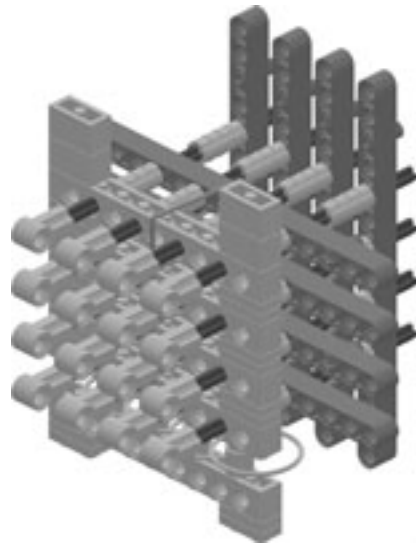
The rubber band pushes the individual keys back up so that when you release a key, it stops pressing on the row and column touch sensors.

Keyboard Row Step 2



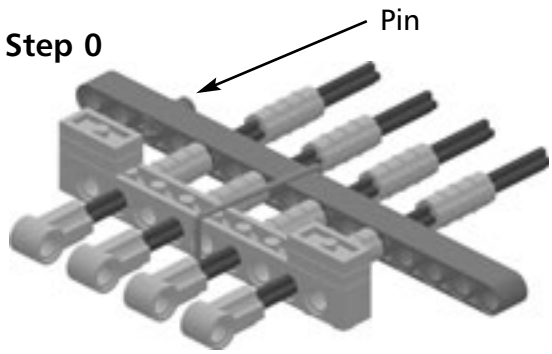


The Keypad



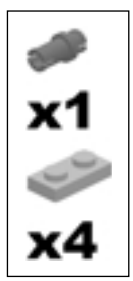
Now we're ready to assemble the entire keypad using the four Keyboard Row sub-assemblies you just built.

Keypad Step 0

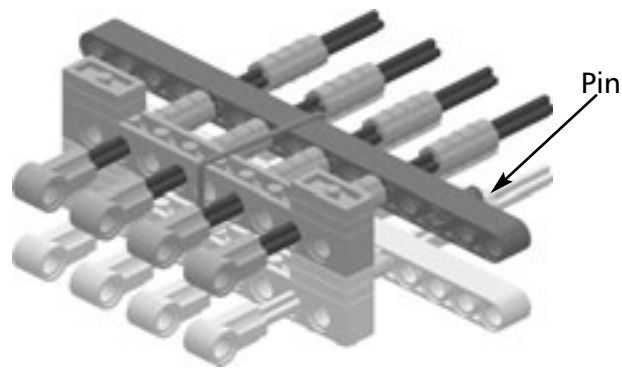


Locate one of the Keyboard Row sub-assemblies. Add the four 1x2 plates and pin as shown.

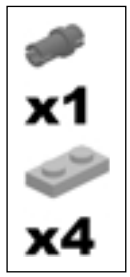
Keypad Step 1

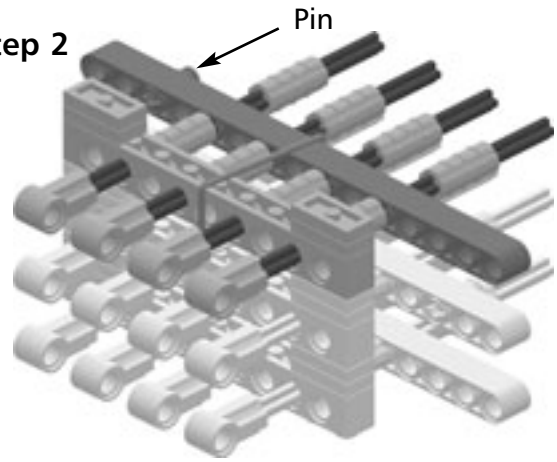


Add a second Keyboard Row sub-assembly, and attach the plates and pin as shown.



Keypad Step 2

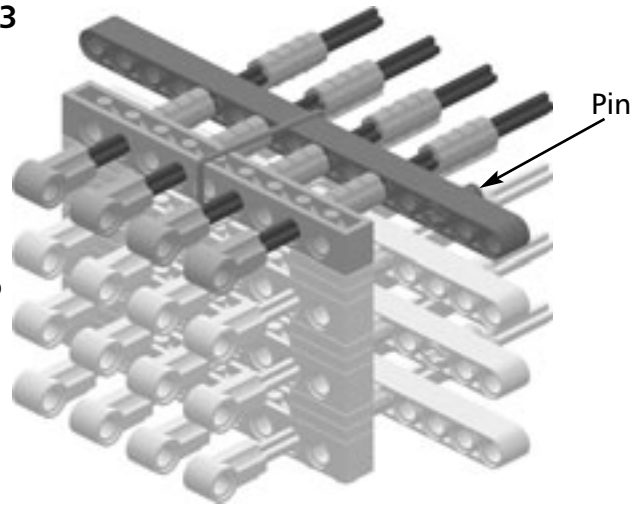




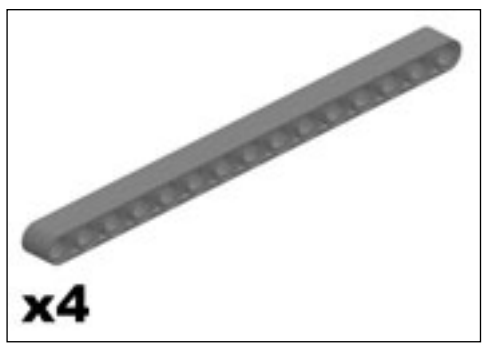
Add a third Keyboard Row sub-assembly, and attach the plates and pin as shown

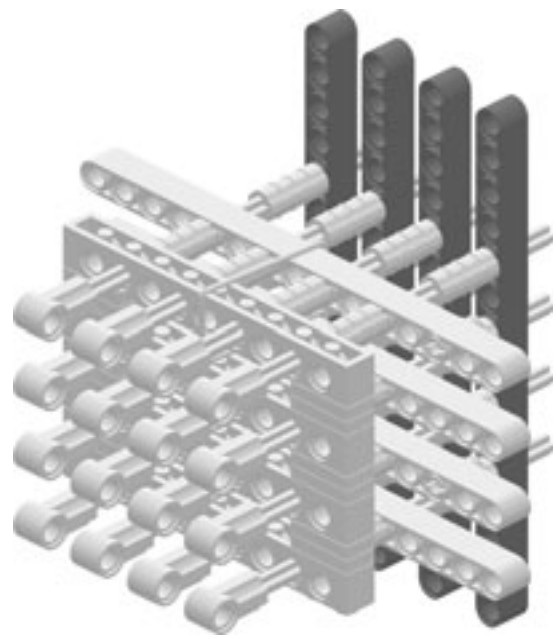
Keypad Step 3

Add the final Keyboard Row sub-assembly to the model, and attach the pin as shown.

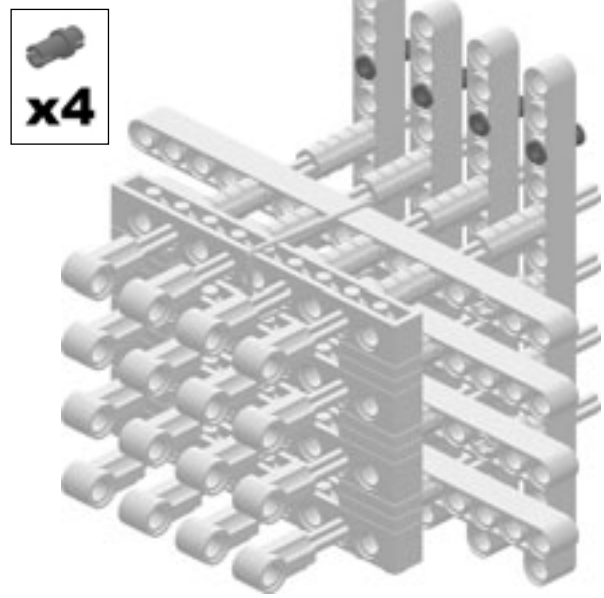


Keypad Step 4

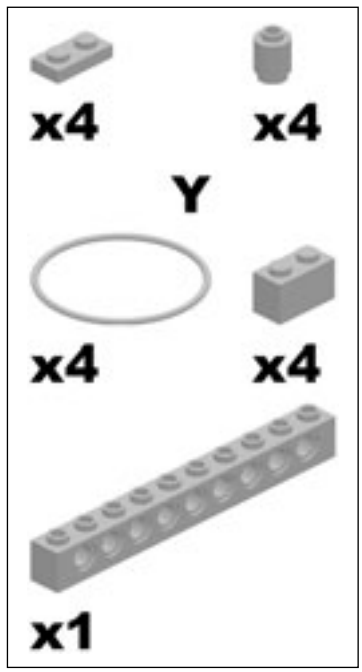


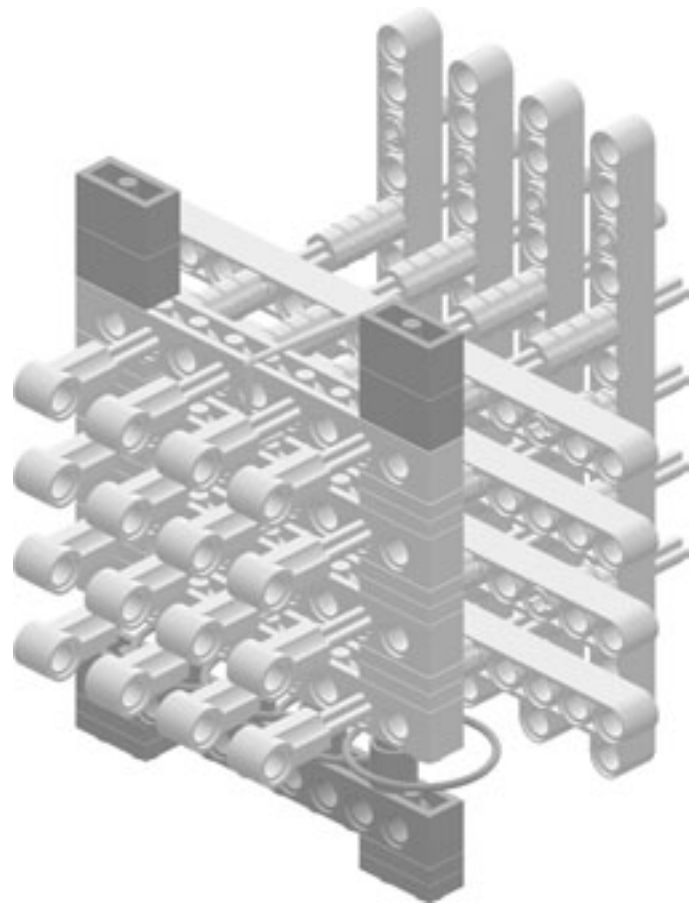


Keypad Step 5

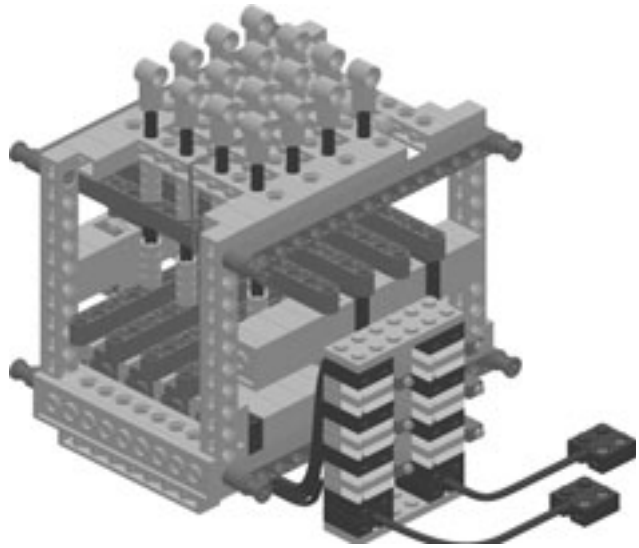


Keypad Step 6



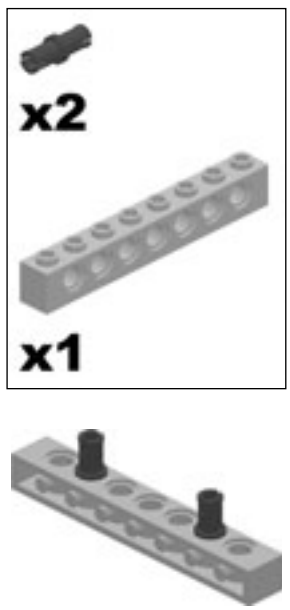


Completing the Keyboard Module

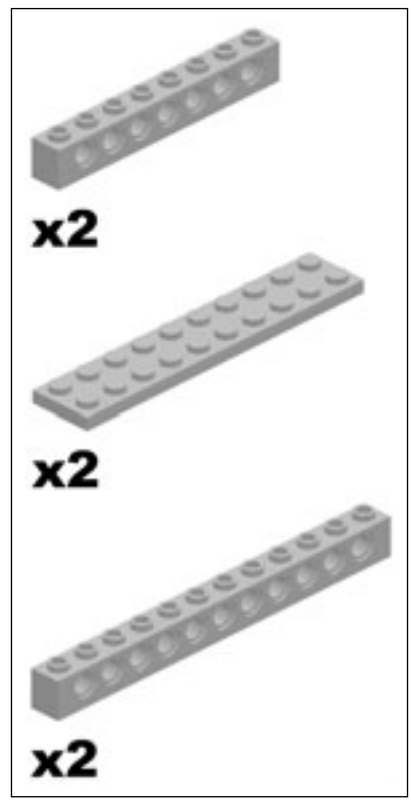


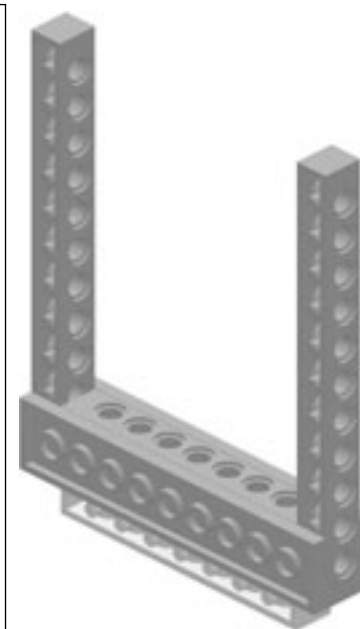
We now have all the pieces needed for final assembly of the keyboard. In the process of Keyboard Final sub-assembly, we will create another resistor network for the four Keyboard Row touch sensors.

Keyboard Final Step 0



Keyboard Final Step 1

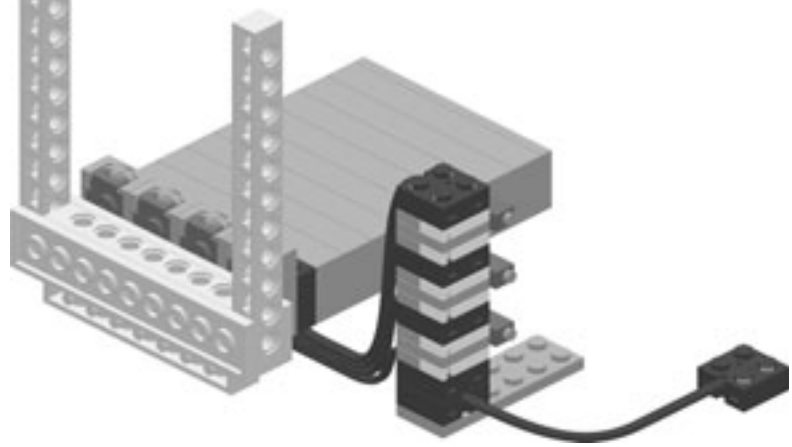




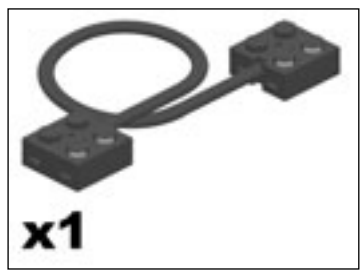
Start by building the base.

Keyboard Final Step 2

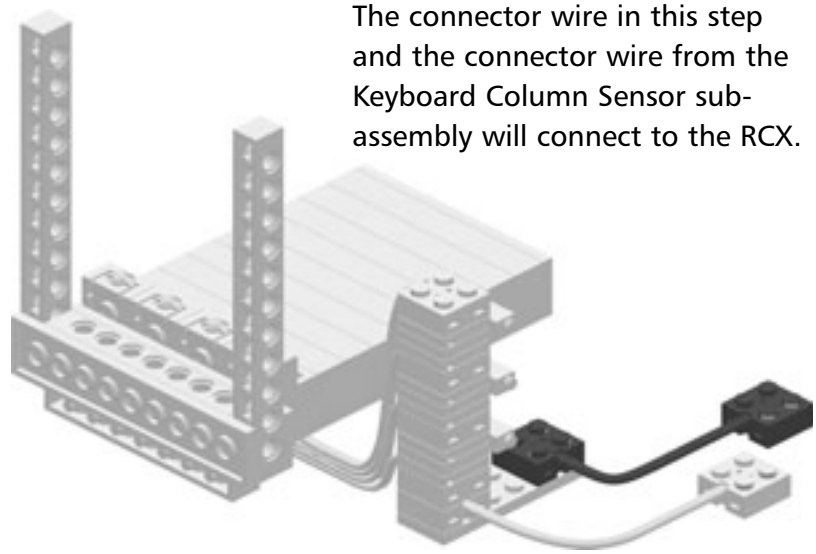
Add the Keyboard Column Sensor sub-assembly as shown.



Keyboard Final Step 3



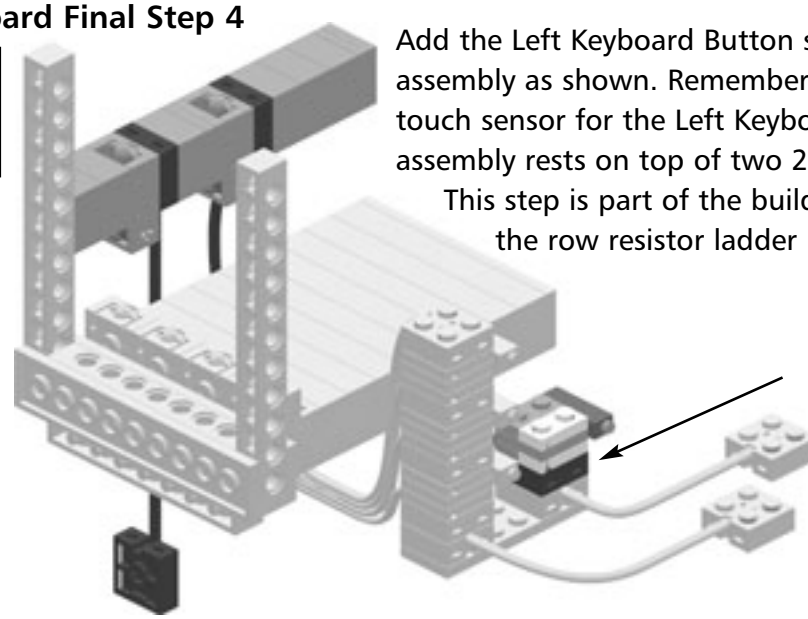
The connector wire in this step and the connector wire from the Keyboard Column Sensor sub-assembly will connect to the RCX.



Keyboard Final Step 4

Add the Left Keyboard Button sub-assembly as shown. Remember that the touch sensor for the Left Keyboard sub-assembly rests on top of two 2x2 bricks.

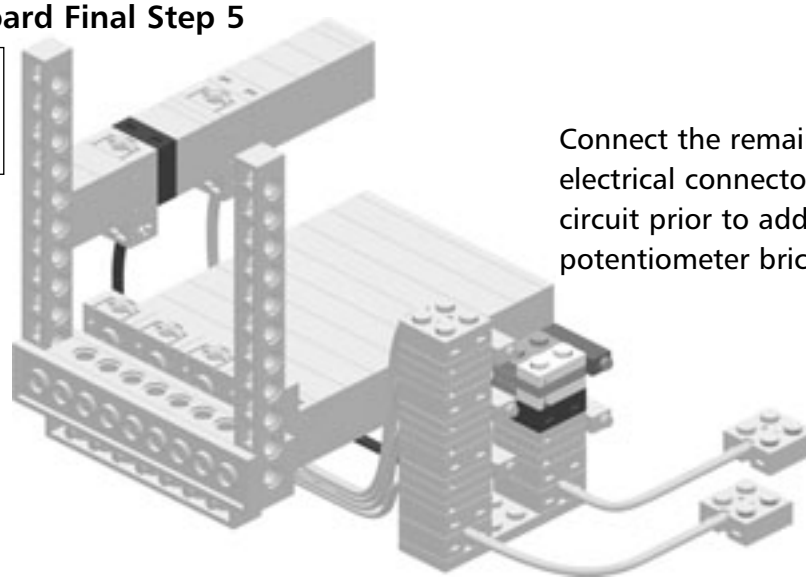
This step is part of the building up of the row resistor ladder network.



Notice that one of the free hanging electric connectors has been attached here.

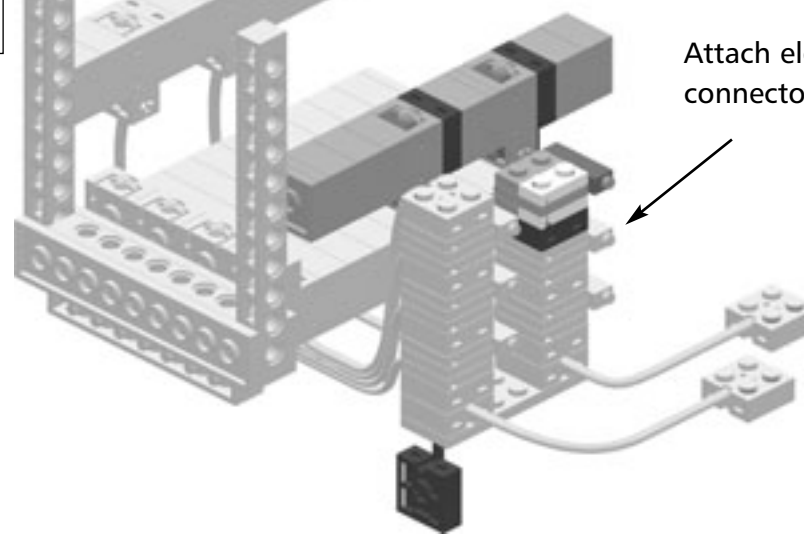
Keyboard Final Step 5

Connect the remaining electrical connector to the circuit prior to adding the potentiometer brick.



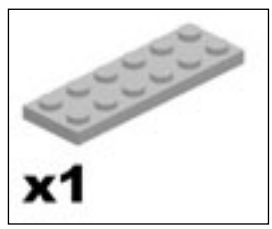
Keyboard Final Step 6

Add the Right Keyboard Button sub-assembly. This will complete the resistor ladder network.

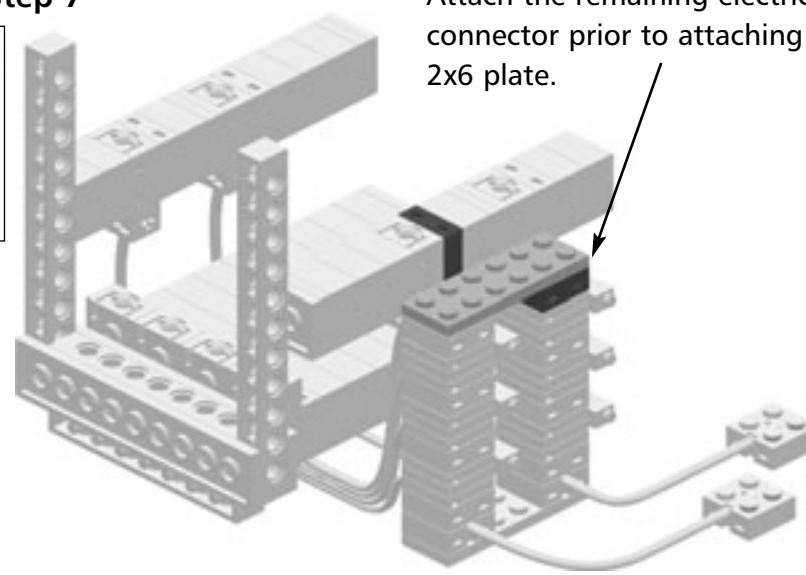


Attach electrical connector as shown.

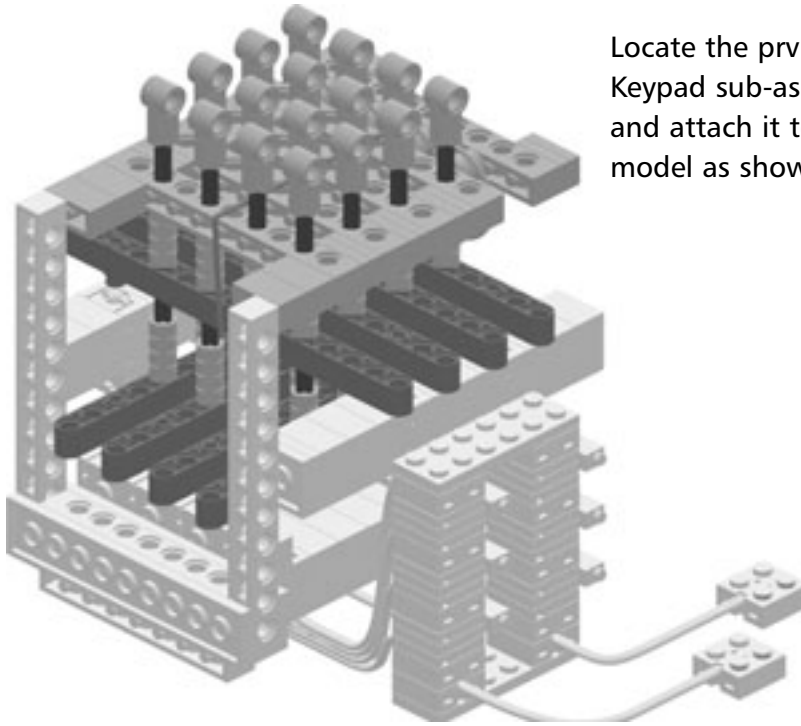
Keyboard Final Step 7



Attach the remaining electrical connector prior to attaching the 2x6 plate.

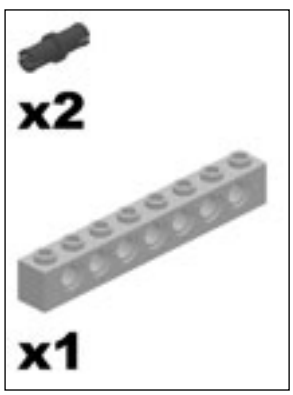


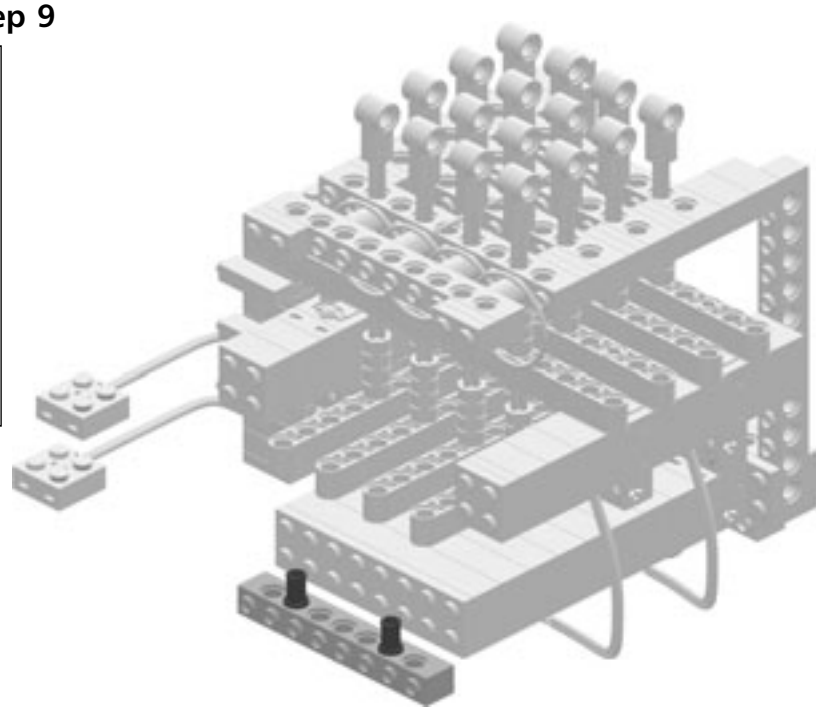
Keyboard Final Step 8



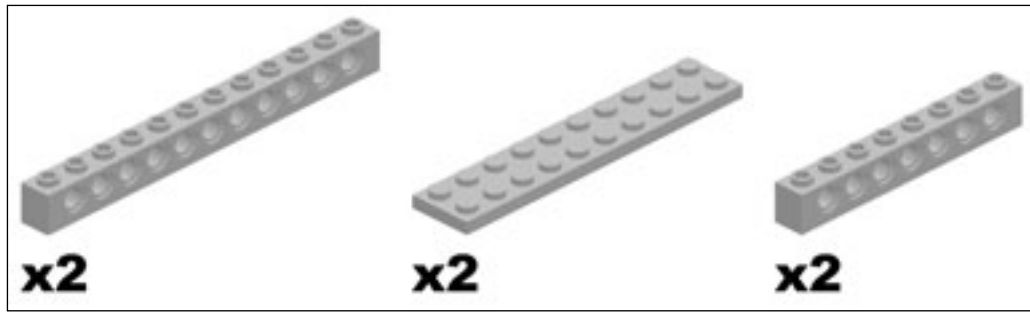
Locate the prviously built Keypad sub-assembly, and attach it to the model as shown.

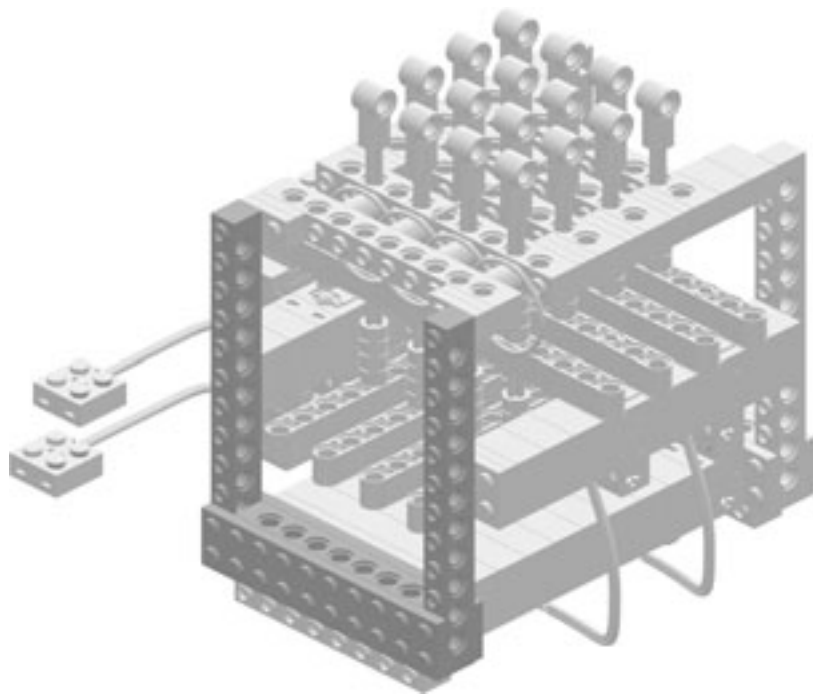
Keyboard Final Step 9



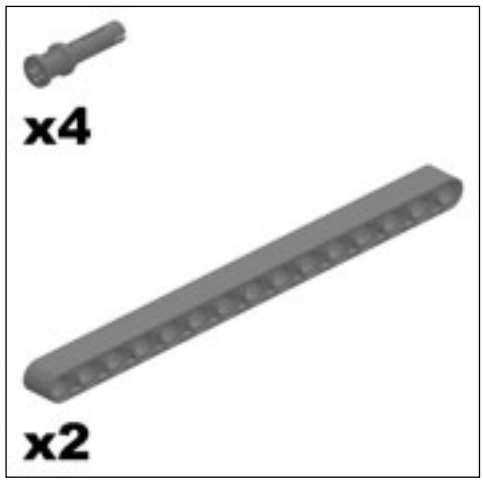


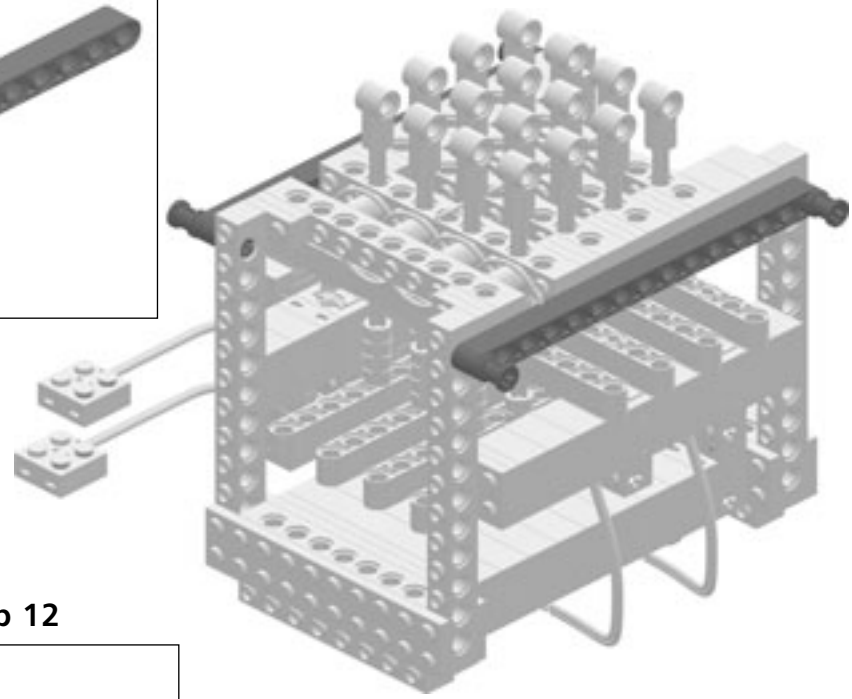
Keyboard Final Step 10



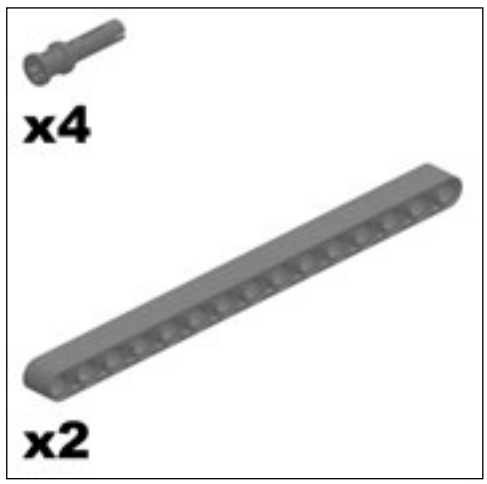


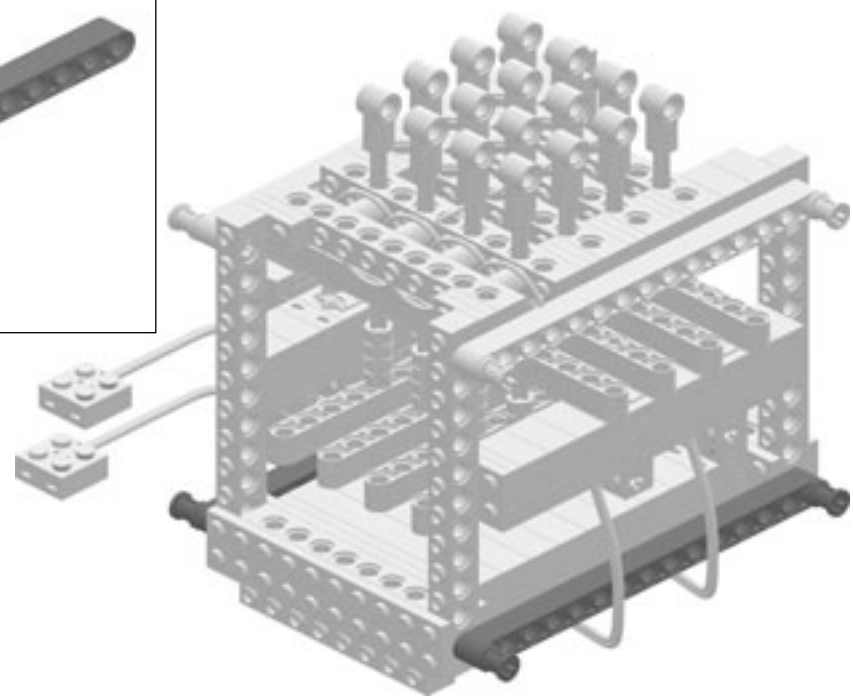
Keyboard Final Step 11



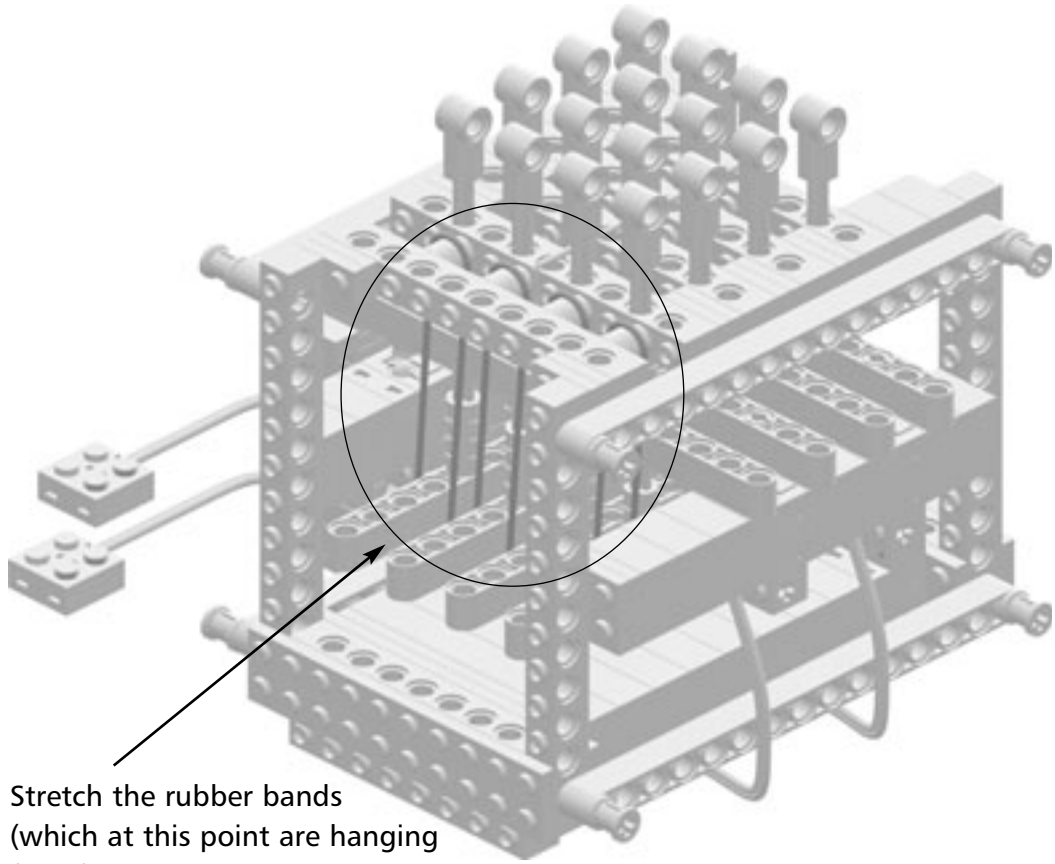


Keyboard Final Step 12





Keyboard Final Step 13



Stretch the rubber bands (which at this point are hanging freely) down and over their respective column's liftarm.

Adding Two Bits

Before we get into the final assembly of PneumADDic II we need to take a look at how exactly the pneumatic adding circuit works. As I mentioned earlier, the original PneumADDic used three gates to add two bits. PneumADDic II has two of these two bit "adders". When adding the bits in a given binary place together, we need to add three bits together: a bit from Value 1, a bit from Value 2, and the Carry from the previous column. PneumADDic II uses the first of its two-bit adders to add the bit from Value 1 to the bit from Value 2, and uses the second of its two-bit adders to add the Sum from the first two-bit adder to the Carry.

The two bit adders are made out of the AND and OR Gate sub-assemblies that we created earlier. To explain how to make a two-bit adder, I'll repeat the truth tables for the AND operator (Table 1.6), OR Gate (Table 1.7) and NOT (Table 1.8) gates here.

Table 1.6 Truth Table for the AND Operator

Input 1	Input 2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.7 Truth Table for the OR Operator

Input 1	Input 2	OR
0	0	0
0	1	1
1	0	1
1	1	1

Table 1.8 Truth Table for the NOT Operator

Input	NOT
0	1
1	0

A two-bit adder has two inputs and two outputs. In our first two-bit adder, the inputs consist of a bit from value 1 and a bit from value 2. The outputs of the two-bit adder are *Sum* and *Carry*. Table 1.9 shows all possible input combinations to a two-bit adder, and the Sum and Carry outputs. When translated to decimal number system, it says 0+0=0, 0+1=1, 1+0=1, and 1+1=2. In binary, there is no 2, only 0 and 1, so decimal 2 ends up as a Carry in binary.

Table 1.9 Input/Output Combinations for the Two-Bit Adder

Inputs		Outputs	
Value 1	Value 2	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

If we compare the Carry column in Table 1.9 to the Table 1.6 we can see that to calculate the Carry, we merely AND the two Input values. The Sum column in Table 1.9 is not so easy. Thinking about it in plain English instead of mathematical terms, "the Sum

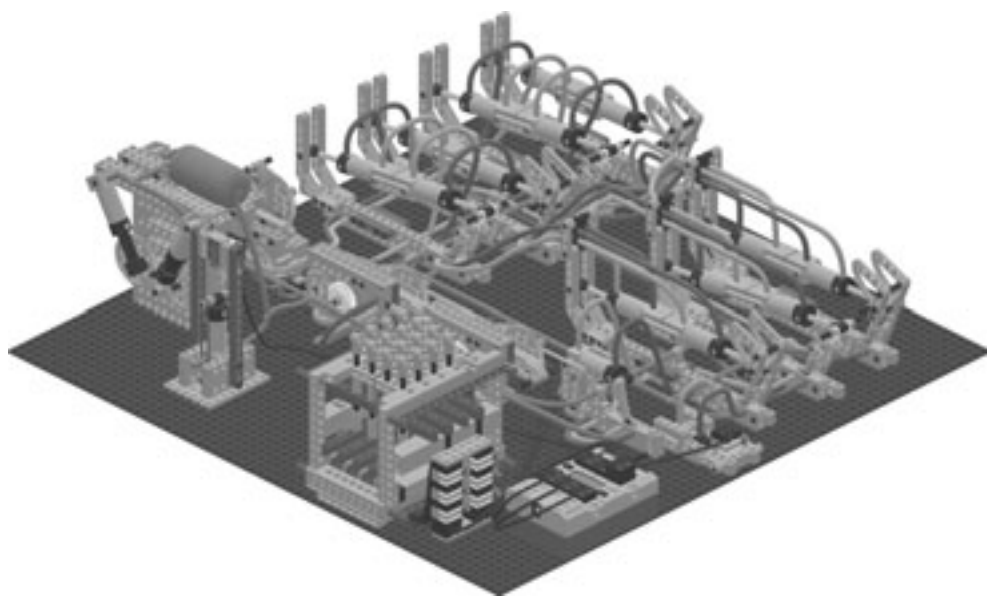
is 1 when either of the input values is a 1, but not when both of the inputs are 1." Mathematically this is called an *exclusive-or*, where you can choose one or the other of the 1's as the inputs, but not both. The technical term for the OR described in Table 1.7 is *inclusive-or*, but typically it is just called OR, because *exclusive-ors* are extremely rare.

In our case of adding two bits, the Sum column is saying "the sum is 1 if any of the inputs are 1, except when they are both 1." This matches the definition of the OR operator. The "except when they are both 1" part is the same as say "when there is not a carry". Table 1.10 shows how this description is mapped into AND, OR and NOT operators.

Table 1.10 Truth Table for the Two-Bit Adder

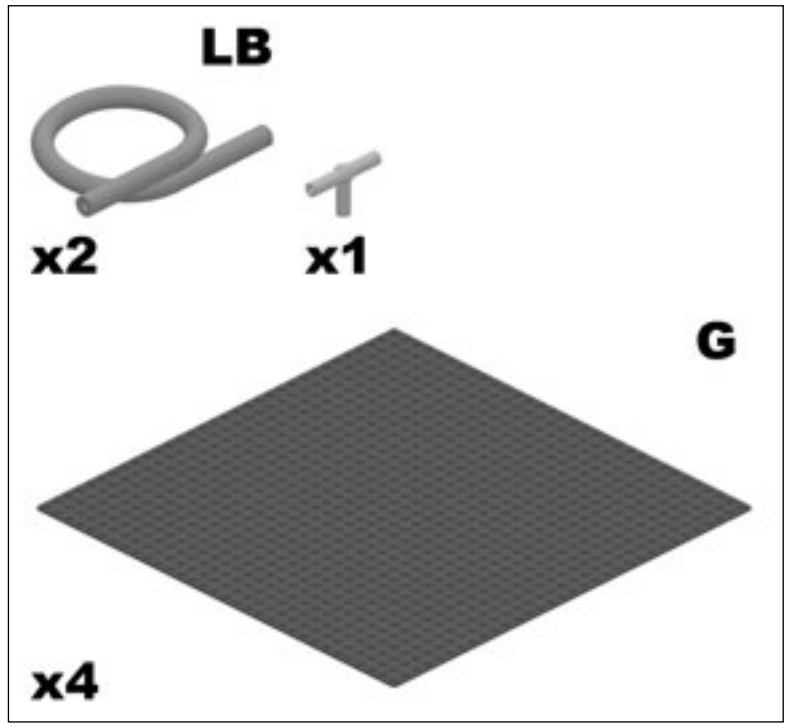
		GATE 1	GATE 2		GATE 3
Value 1	Value 2	Value 1 OR Value 2	Value 1 AND Value 2 (Carry)	NOT Carry	(Value 1 OR Value 2) AND NOT Carry (Sum)
0	0	0	0	1	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	1	1	0	0

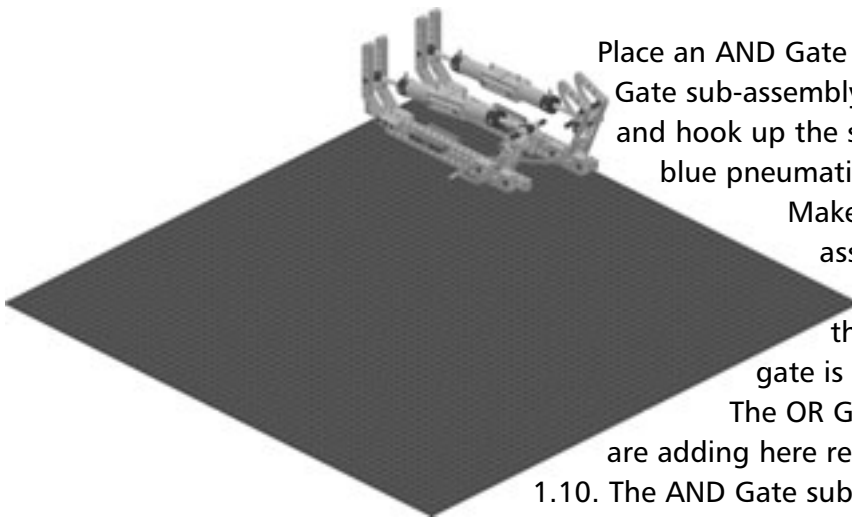
Putting It All Together



You are now ready to build the final assembly. The one thing that has not been explained yet is how the pneumatic adding circuit works. I will describe this while we are putting it all together.

Final Step 0



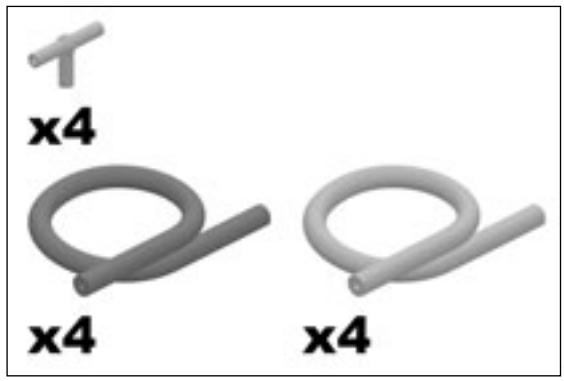


Place an AND Gate sub-assembly and an OR Gate sub-assembly on the large plate shown, and hook up the source pressure using a light blue pneumatic tube.

Make sure that the OR Gate sub-assembly (the one with the smaller switch handle) is the one in front and the AND gate is in the back.

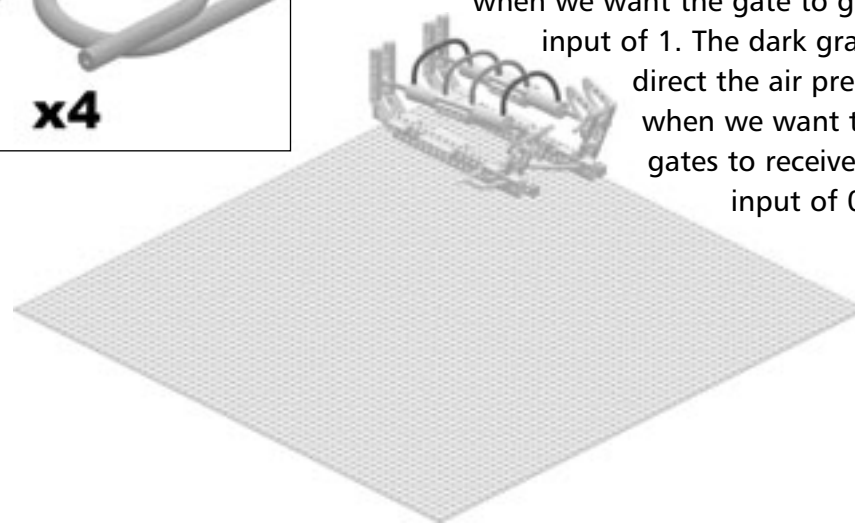
The OR Gate sub-assembly that you are adding here represents GATE 1 in Table 1.10. The AND Gate sub-assembly you are adding represents GATE 2.

Final Step 1

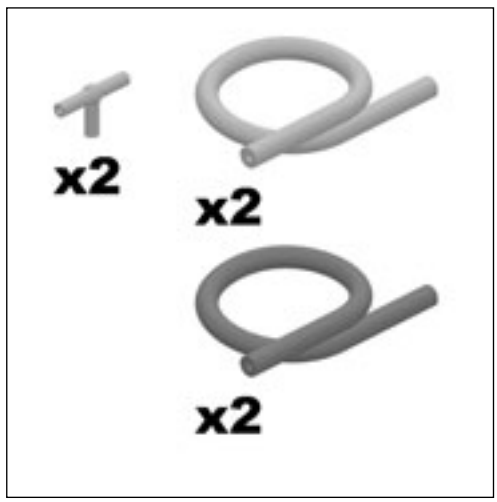


Hook the T-Connectors to the pneumatic pistons as shown. Both gates receive the same inputs. The inputs will be fed from the values that you type into the RCX from the keyboard.

The light gray hoses direct the air pressure when we want the gate to get an input of 1. The dark gray hoses direct the air pressure when we want the gates to receive an input of 0.



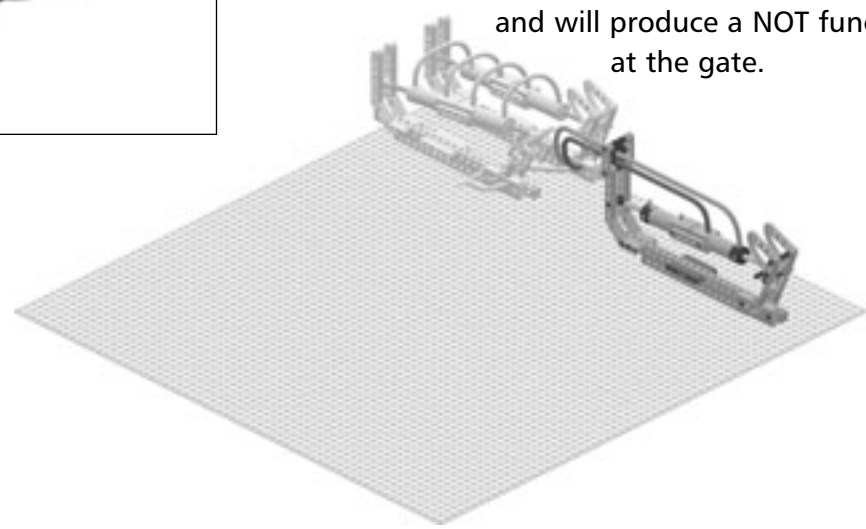
Final Step 2



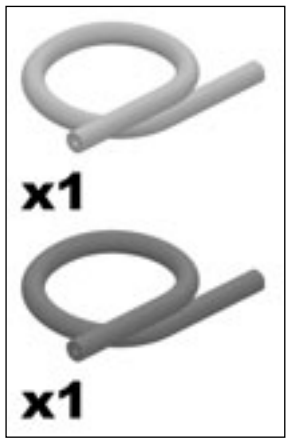
Add a second AND Gate sub-assembly and connect inputs to this new AND Gate sub-assembly as shown. This represents GATE 3 in Table 1.10.

Notice where the light gray and dark gray pneumatic tubes are hooked at the pneumatic switch and at the piston. The dark gray tube is hooked to the base and the light gray tube is hooked to the other T-Connector.

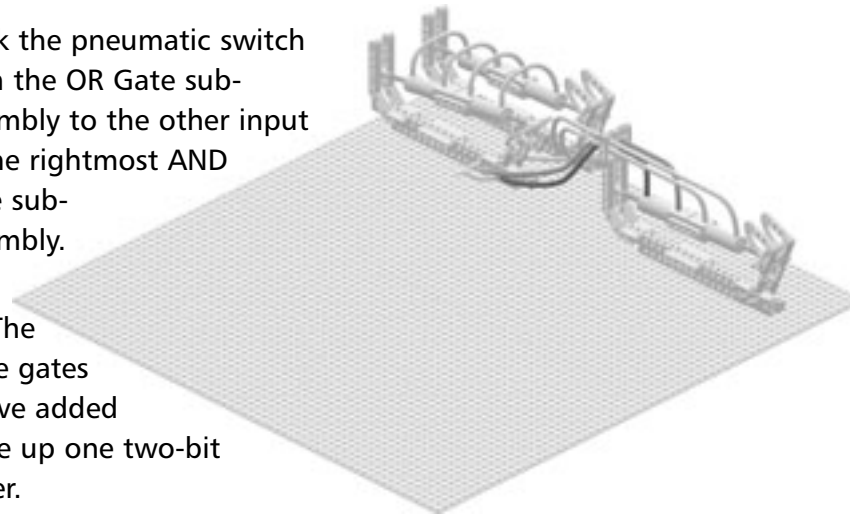
This is the opposite of the normal configuration and will produce a NOT function at the gate.



Final Step 3



Hook the pneumatic switch from the OR Gate sub-assembly to the other input of the rightmost AND Gate sub-assembly.

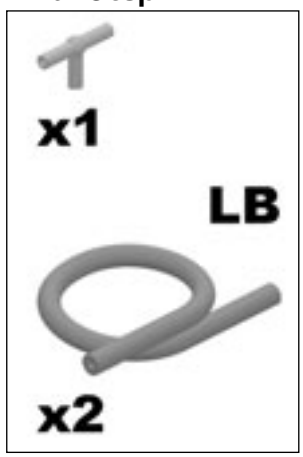


The three gates you've added make up one two-bit adder.

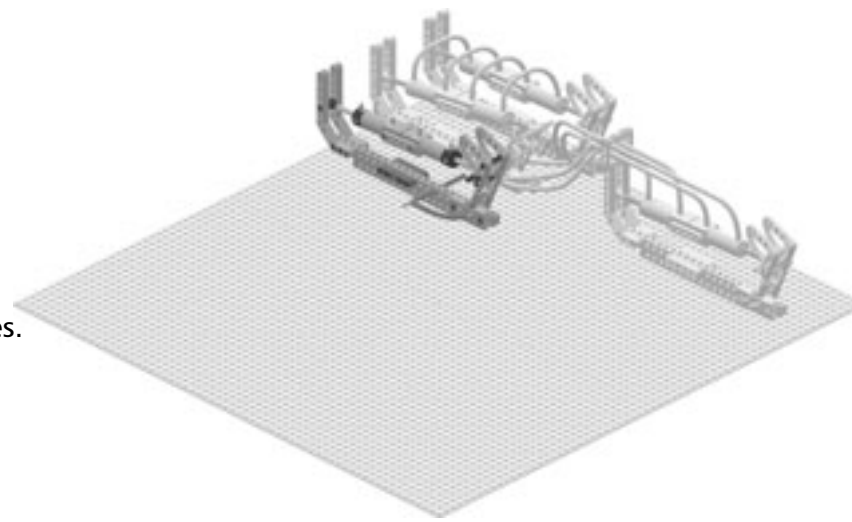
NOTE

The next sequence of steps builds a second two-bit adder that adds the sum from the first two-bit adder to the Carry Memory sub-assembly.

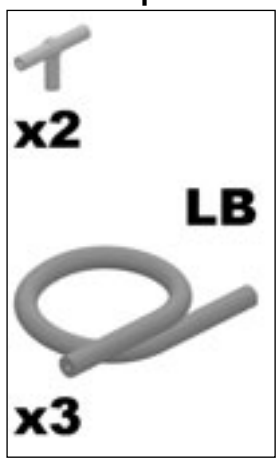
Final Step 4



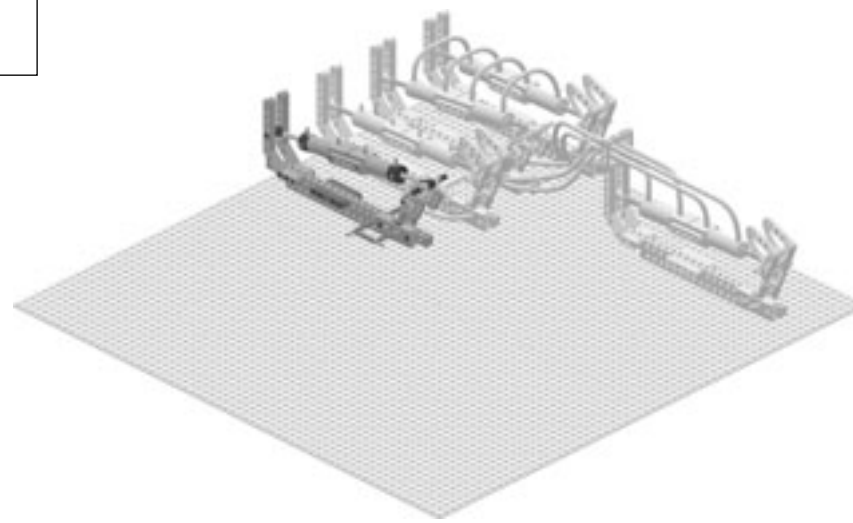
Add a third AND Gate sub-assembly and hook up its pressure source with blue tubes.



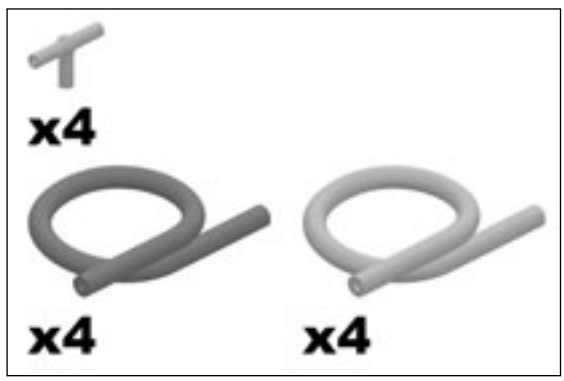
Final Step 5



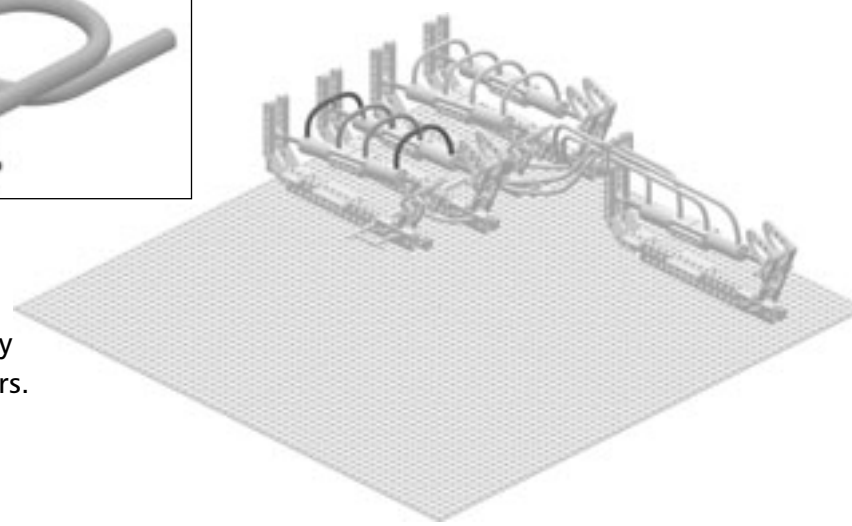
Add an OR Gate sub-assembly and hook up its pressure source with blue tubes.



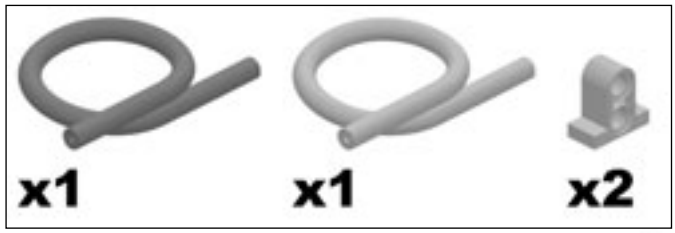
Final Step 6



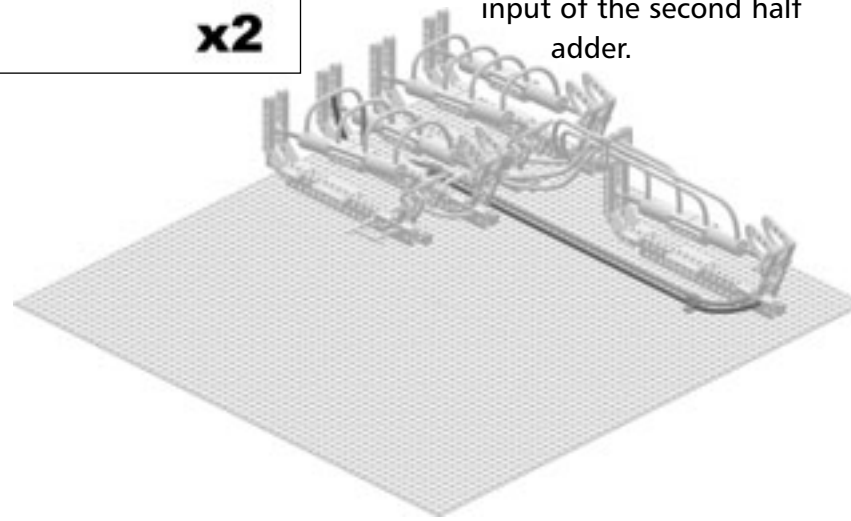
Connect the inputs of the AND and OR Gate sub-assemblies together using the light gray and dark gray tubing and the T-Connectors.



Final Step 7



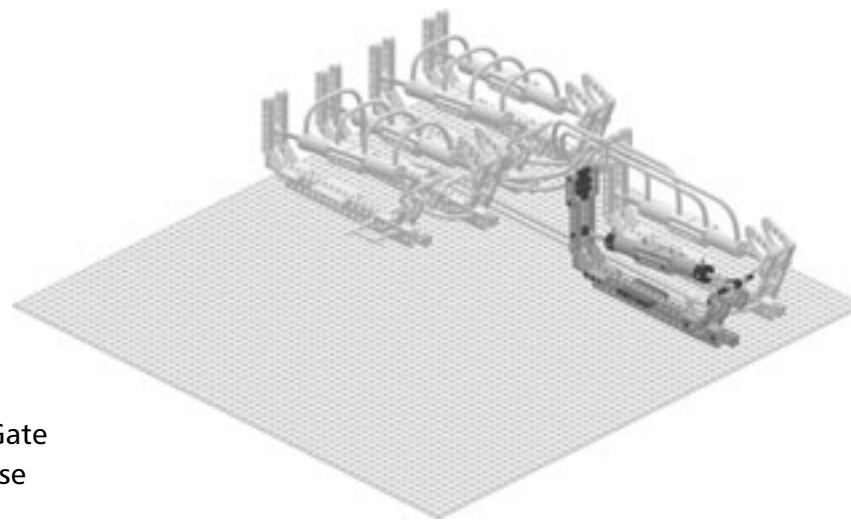
Hook the sum output of the first two-bit adder to an input of the second half adder.



NOTE

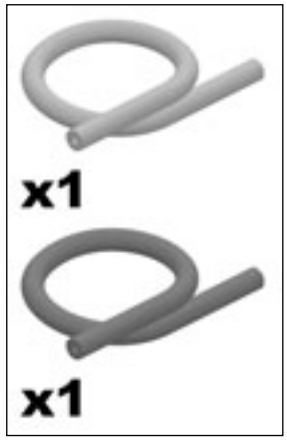
There is a Carry out from each of our two-bit adders. Only one of the carries is ever on at a time. We need to merge these two Carry outputs together into one Carry value that is the input to the Carry Memory. We merge the carries by OR-ing them together using an OR Gate sub-assembly.

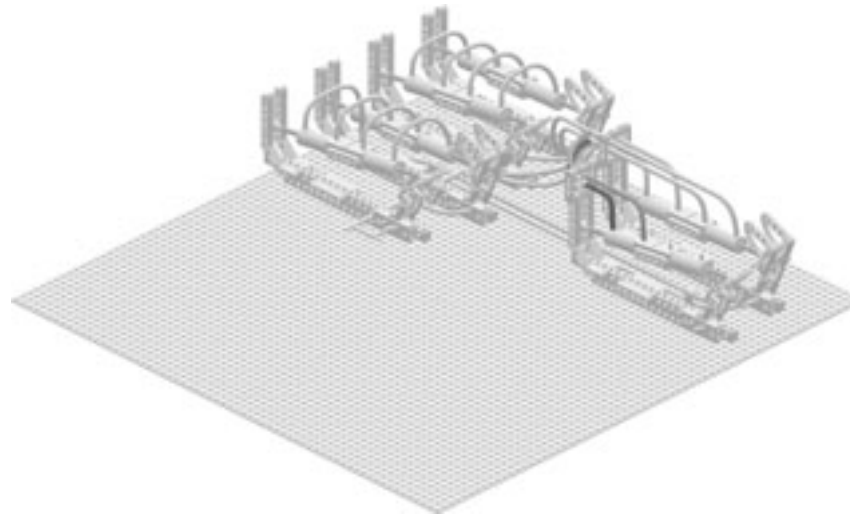
Final Step 8



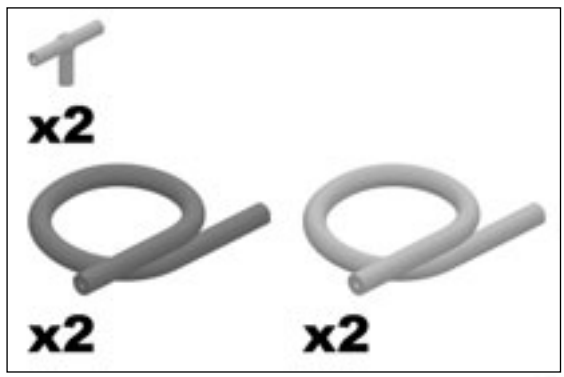
Add yet another AND Gate sub-assembly to the base plate as shown.

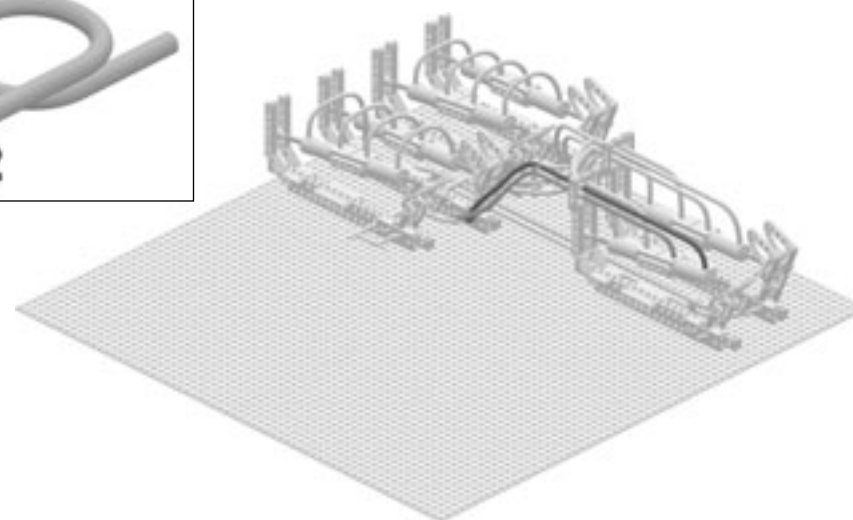
Final Step 9



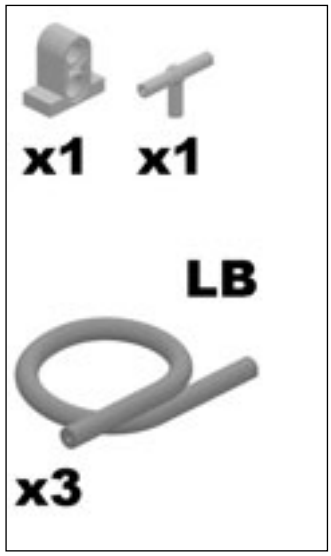


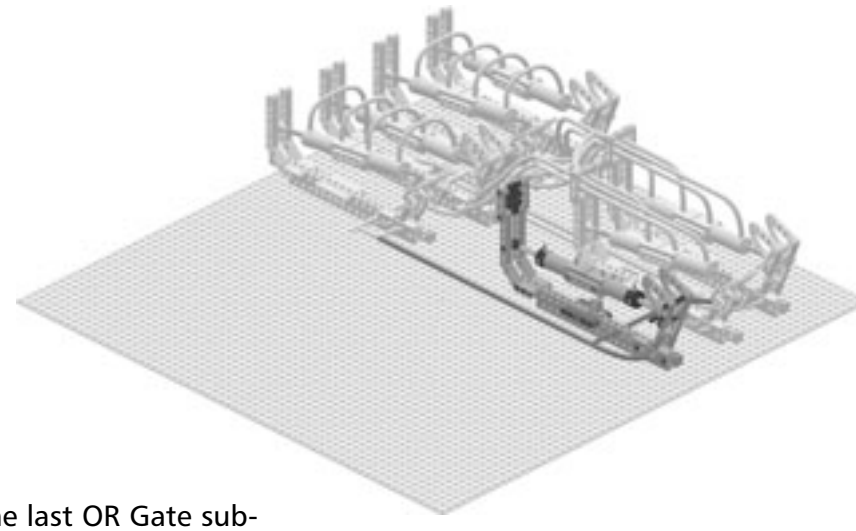
Final Step 10





Final Step 11

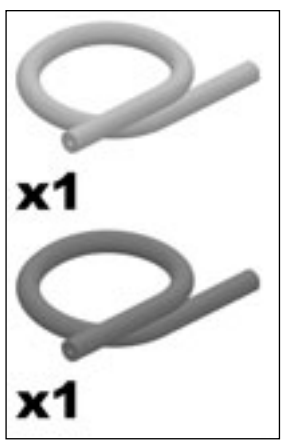


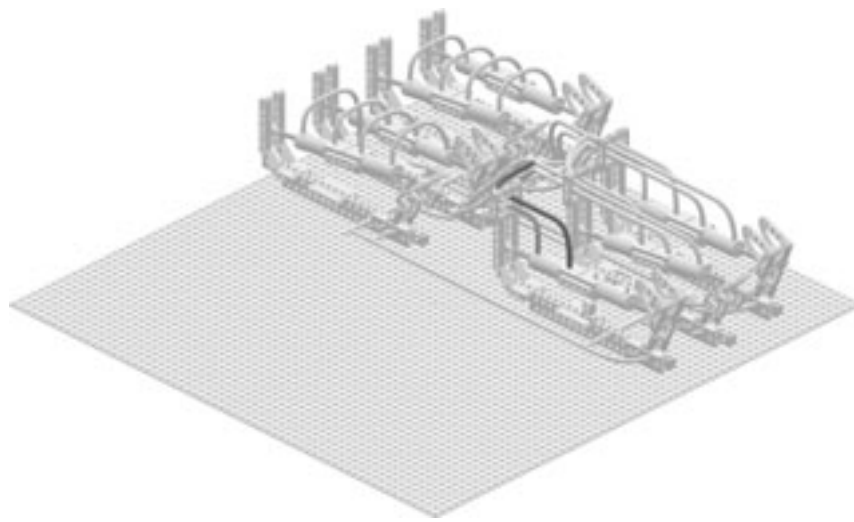


Here is where the last OR Gate sub-assembly is added to the pneumatic adding circuit. The pneumatic adding circuit calculates the final sum that is read by the RCX.

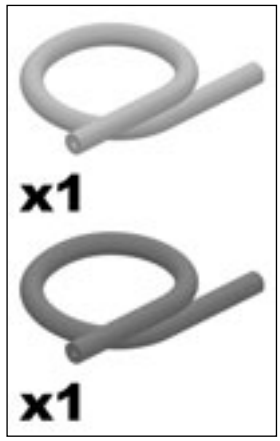
This gate plays the role of GATE 3 in Figure 1.10.

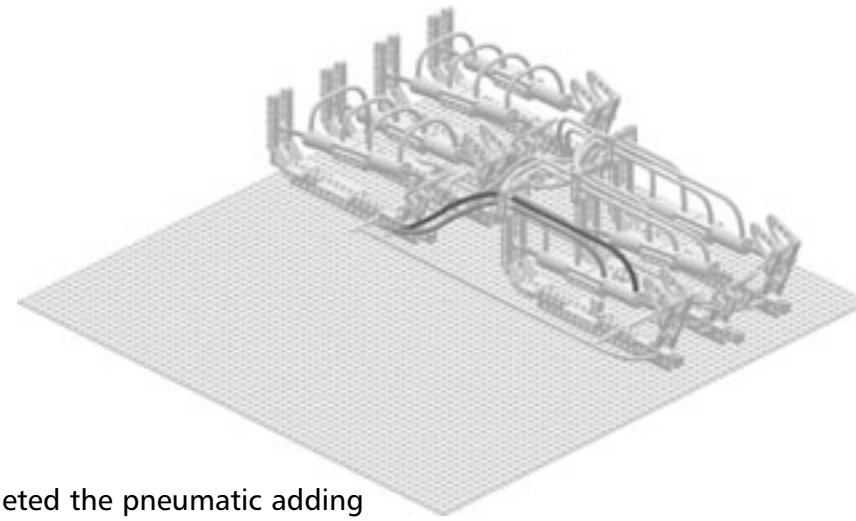
Final Step 12





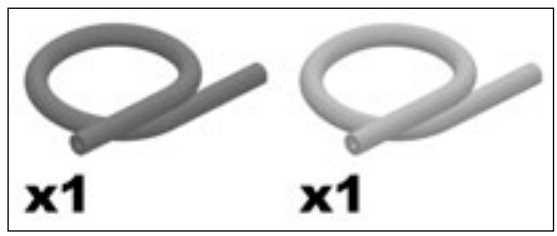
Final Step 13

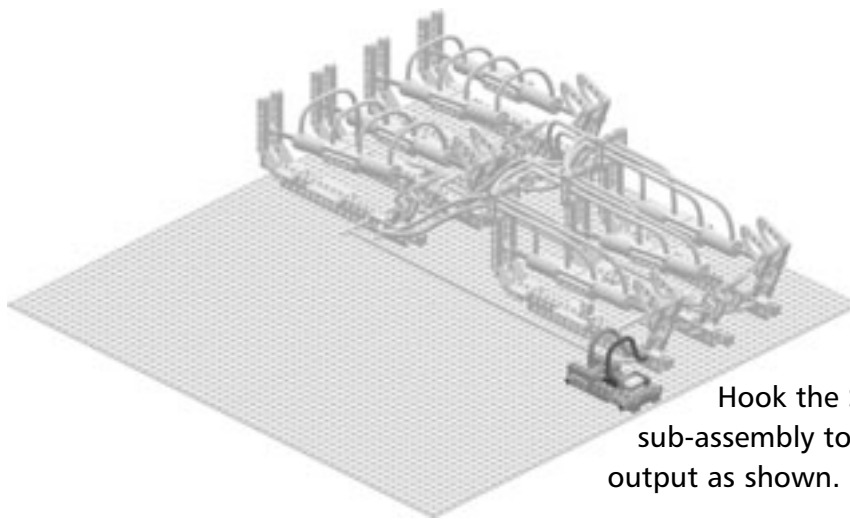




You have now completed the pneumatic adding circuit. Now we need to add the rest of the sub-assemblies. We'll start with the Sum Sensor and the Carry Memory sub-assemblies.

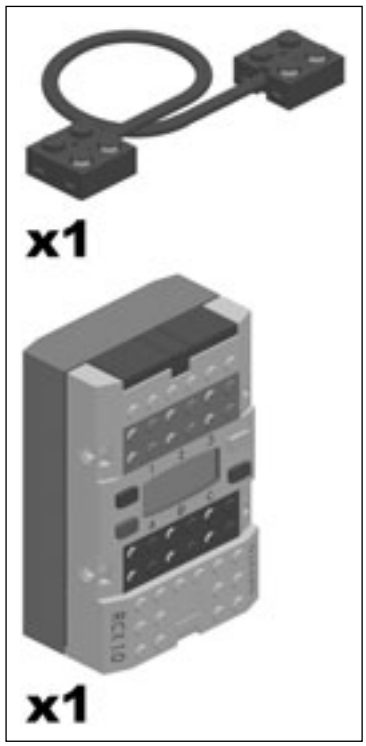
Final Step 14

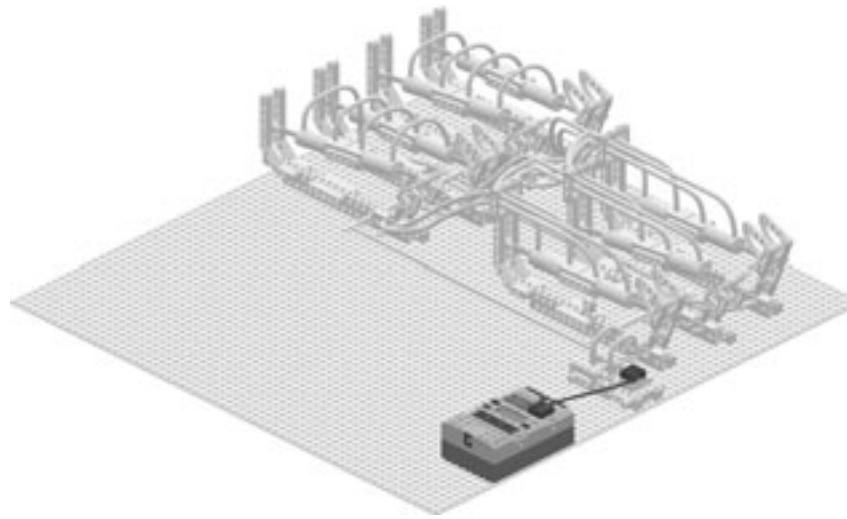




Hook the Sum Sensor sub-assembly to the sum output as shown.

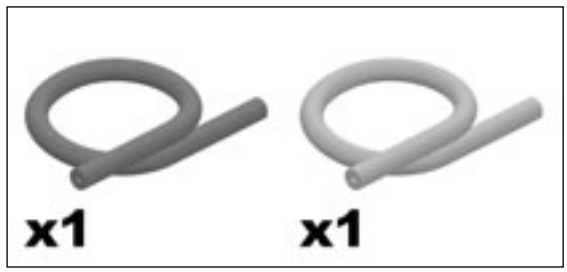
Final Step 15

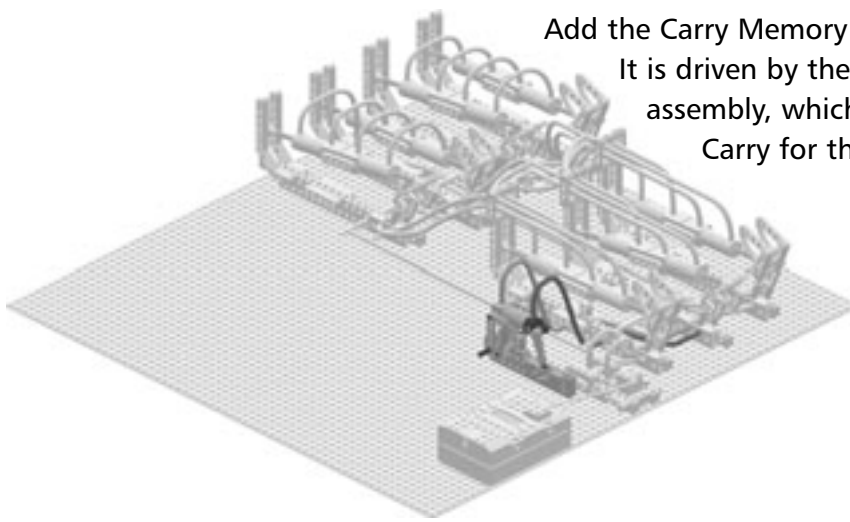




Connect the Sum Sensor sub-assembly to RCX Sensor Input 3.

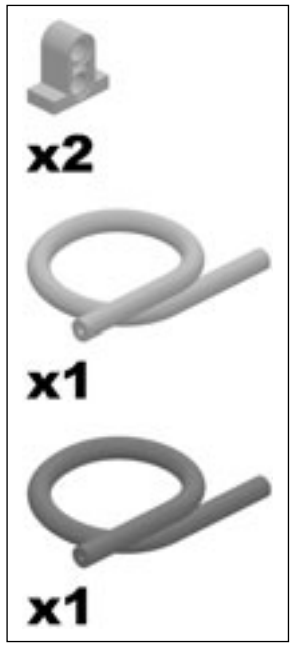
Final Step 16

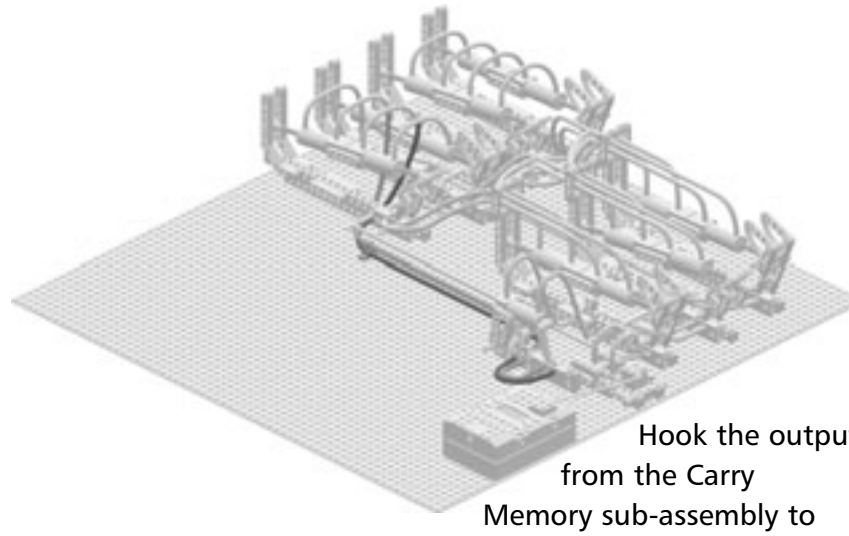




Add the Carry Memory sub-assembly. It is driven by the OR Gate sub-assembly, which calculates the Carry for the operation.

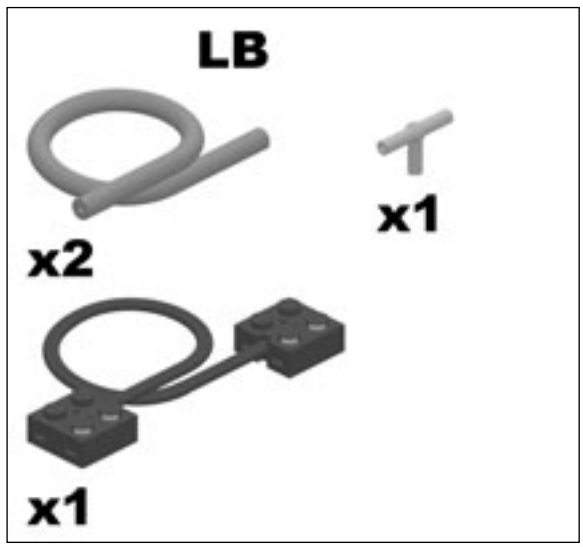
Final Step 17



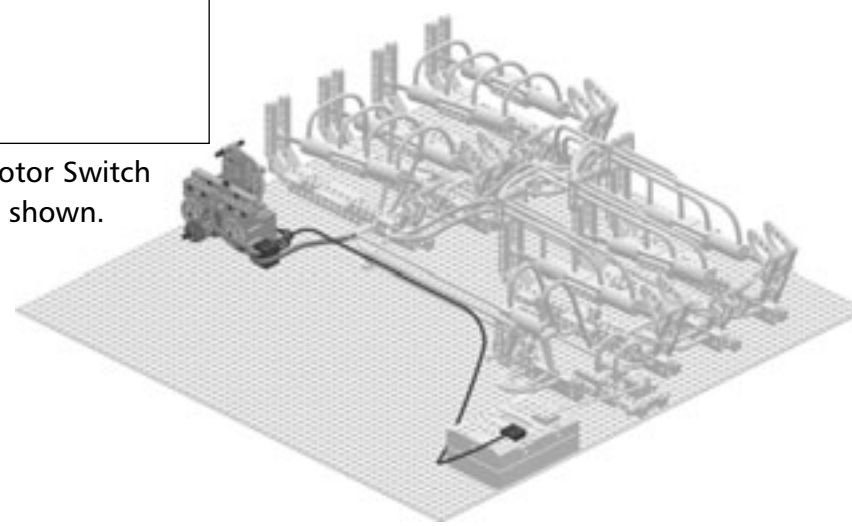


Hook the output from the Carry Memory sub-assembly to the second half adder.

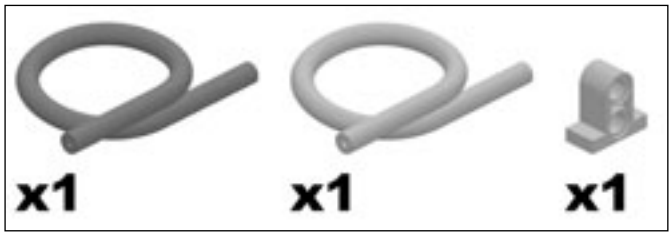
Final Step 18



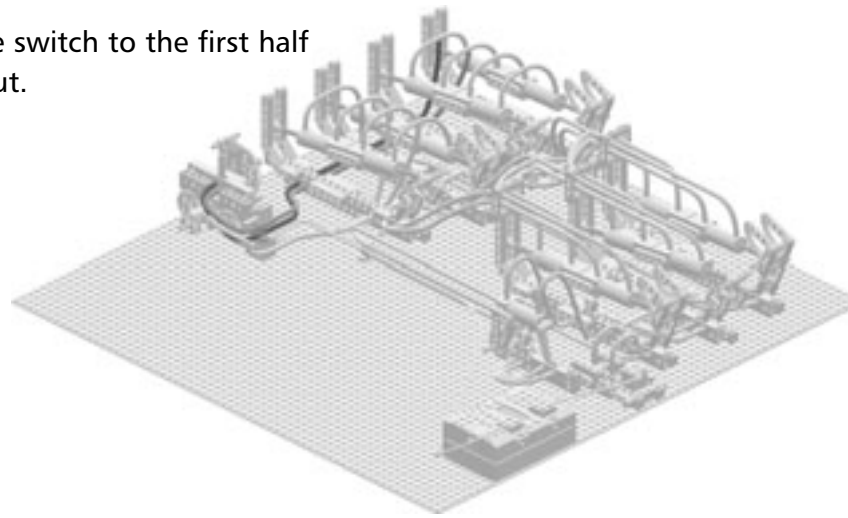
Add the Dual Motor Switch sub-assembly as shown.



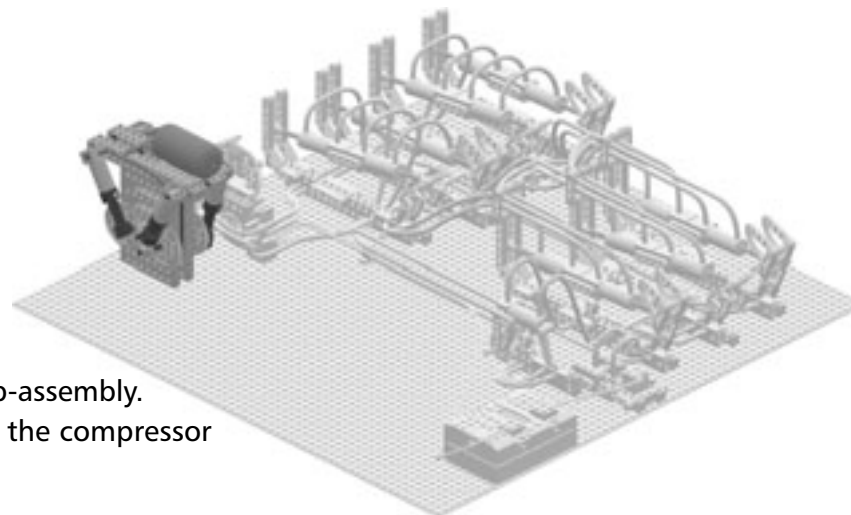
Final Step 19



Connect the switch to the first half adder's input.



Final Step 20

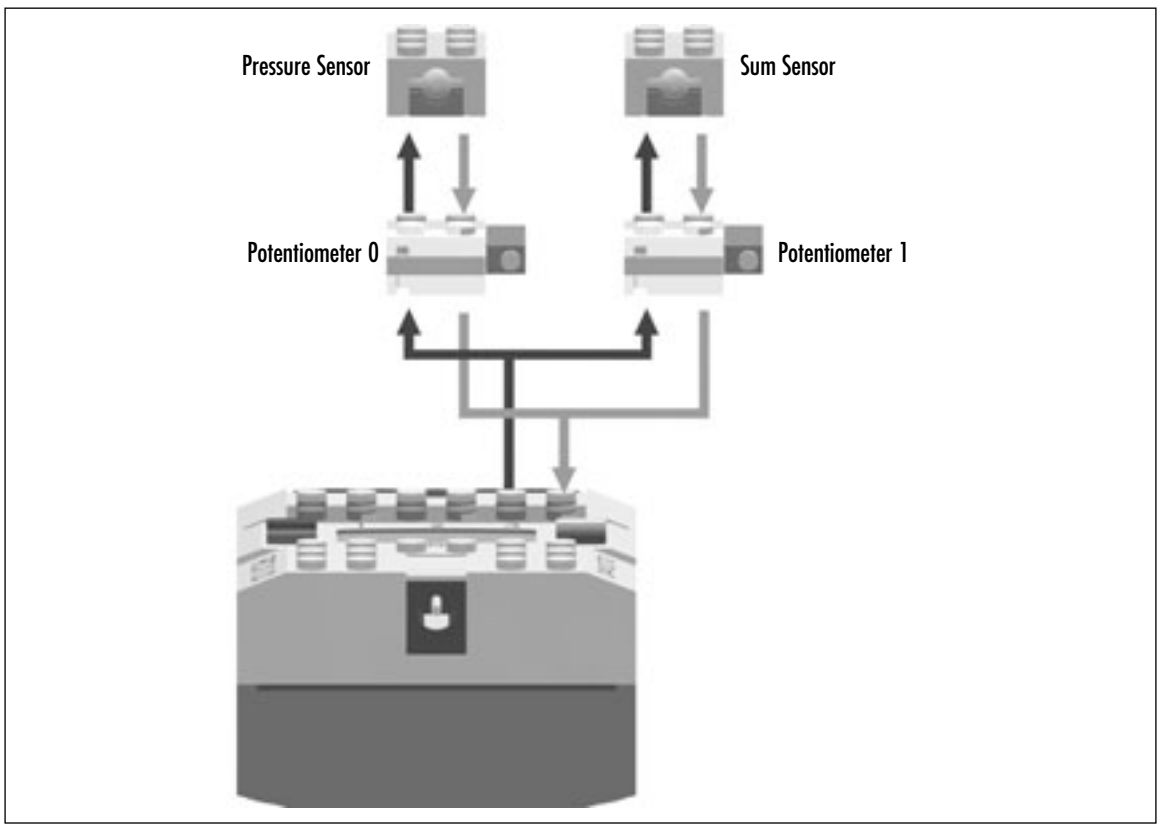


Add the Pump sub-assembly. This will complete the compressor module.

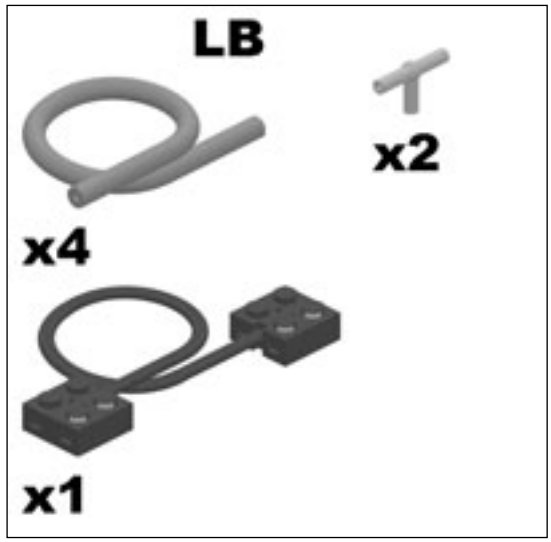
In the next series of steps, we will encounter the issue of the Digital Pressure Sensor and Sum Sensor sub-assemblies' touch sensor buttons, which can both be pressed at the same time. The Digital Pressure Sensor sub-assembly and the Sum Sensor sub-assembly share the same RCX input port using the same type of potentiometer bricks that was used in the keyboard row and column sensors. When a key is pressed on the keyboard, only one row touch sensor, and one column touch sensor is pressed. This is not the case for the Digital Pressure Sensor sub-assembly and the Sum Sensor sub-assemblies. They can both be pressed at the same time. This requires a different kind of resistor network so the RCX can tell what is going on.

Figure 1.11 describes the circuit used to hook the Sum Sensor sub-assembly and the Digital Pressure Sensor sub-assembly to a single RCX input port. Electricity travels from the RCX to the touch sensors along the black arrows. When the touch sensor for the Sum Sensor sub-assembly is pressed, electricity travels back to the RCX (following the gray arrows) through Potentiometer 0 which disposes of some of the electricity. The same thing happens with the Pressure Sensor sub-assembly, except its electricity travels back to the RCX through Potentiometer 1. Potentiometers are variable resistors, so Potentiometer 0 is adjusted to disperse more electricity that Potentiometer 1. This way the Sum Sensor sub-assembly gives a different reading than the Digital Pressure Sensor sub-assembly. When both sensors are pressed, the electricity has two paths returning to the RCX. The electricity from both paths is added together by the electrical cables that hook the sensors to the RCX. When both sensors are pressed, the RCX sees a reading that is different than if either of the touch sensors are pressed independently. Since the electricity can travel along two paths at the same time, this resistor network is called a *parallel resistor network*.

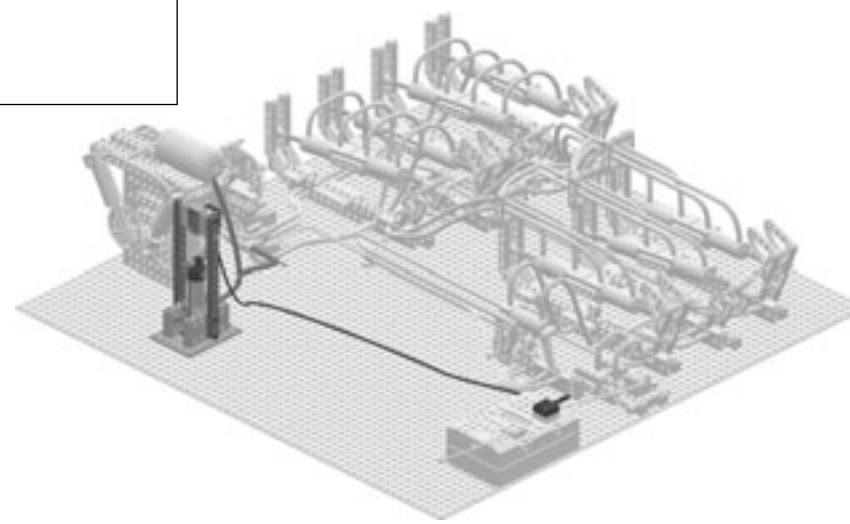
Figure 1.11 Parallel Resistor Network



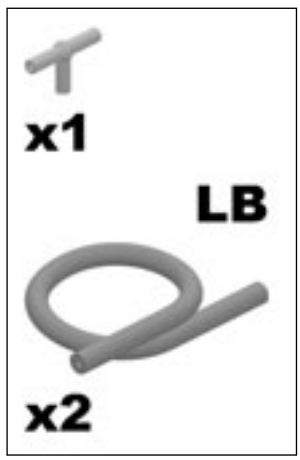
Final Step 21

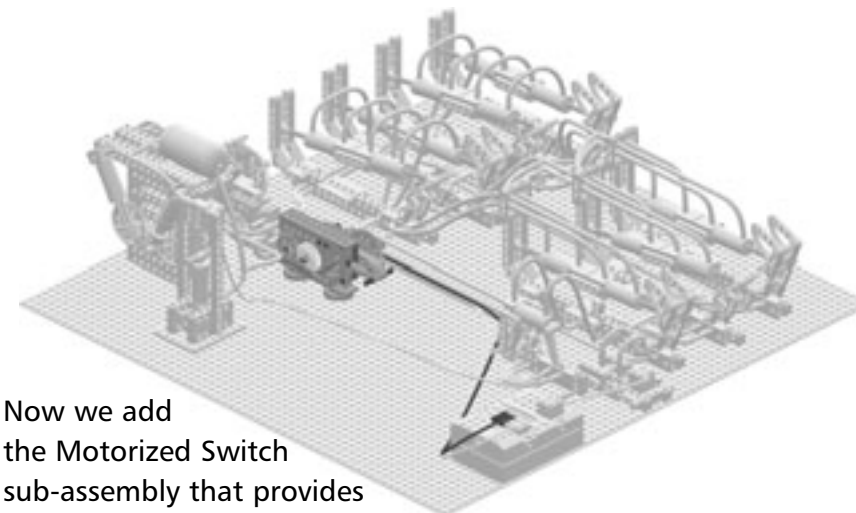


Add the Digital Pressure Sensor sub-assembly and hook it to RCX Sensor Input 3.



Final Step 22

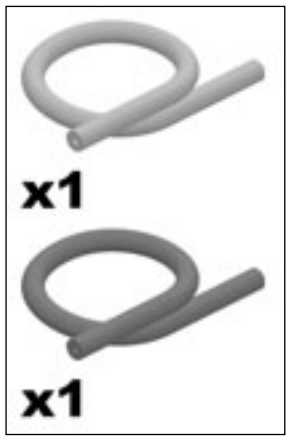




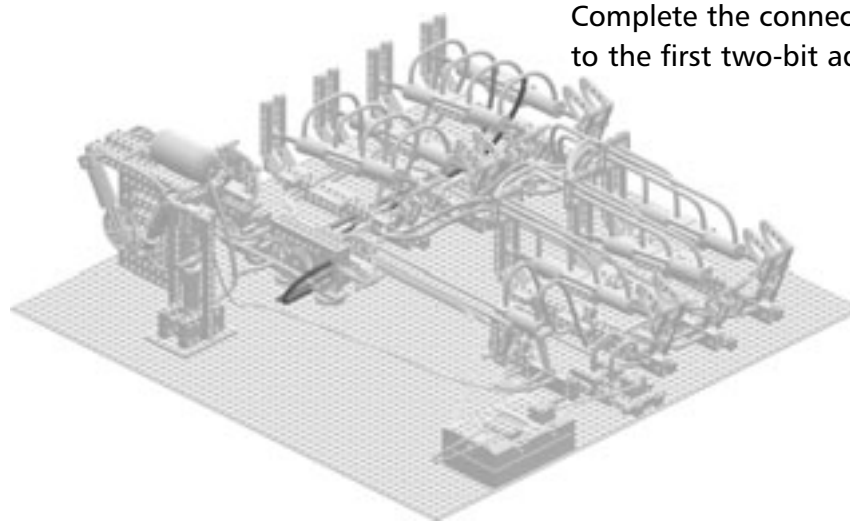
Now we add the Motorized Switch sub-assembly that provides the pneumatic inputs for Value 2 to the first two-bit adder.

Add the Motorized Switch sub-assembly and hook up the pressure source.

Final Step 23



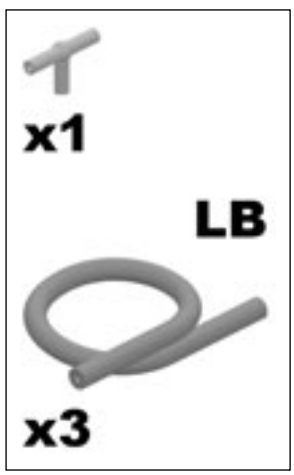
Complete the connections to the first two-bit adder.



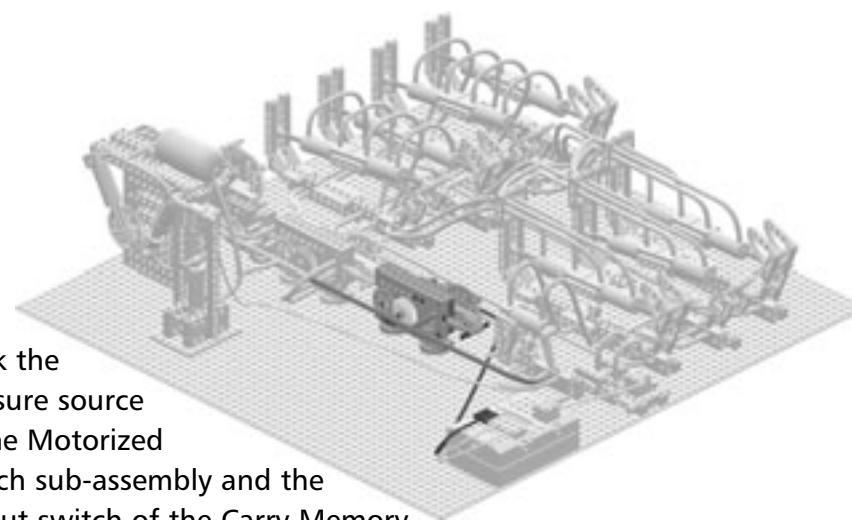
NOTE

The Carry Memory only gets pressurized when we want it to remember a new Carry value. We do this using a motorized switch that provides pressure to the OR gate that merges the carries from both two-bit adders.

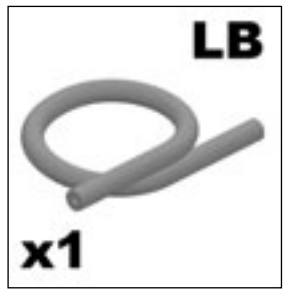
Final Step 24



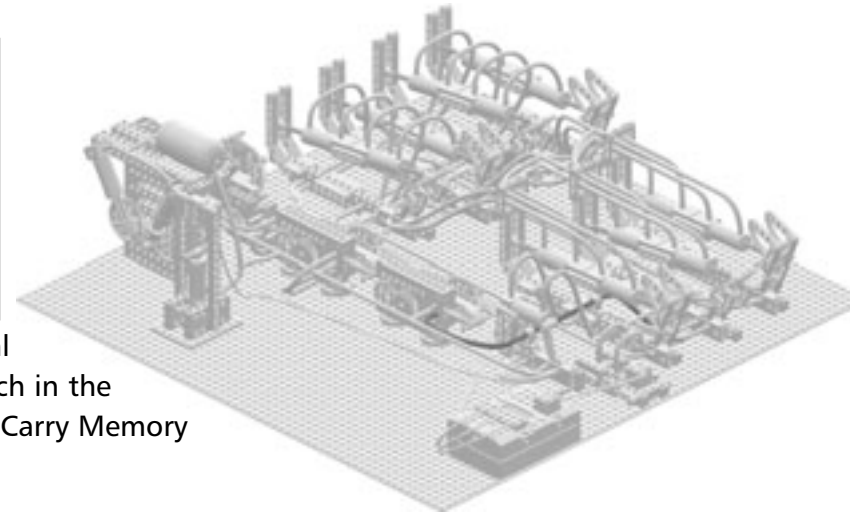
Hook the pressure source to the Motorized Switch sub-assembly and the output switch of the Carry Memory.



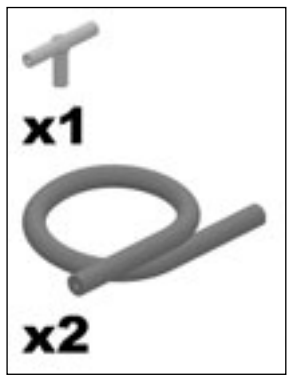
Final Step 25



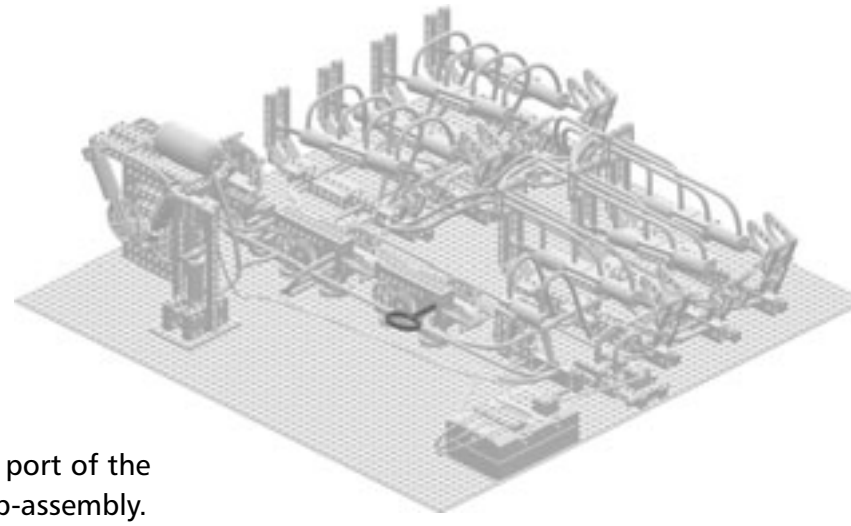
Hook the conditional pressure to the switch in the gate that drives the Carry Memory sub-assembly.



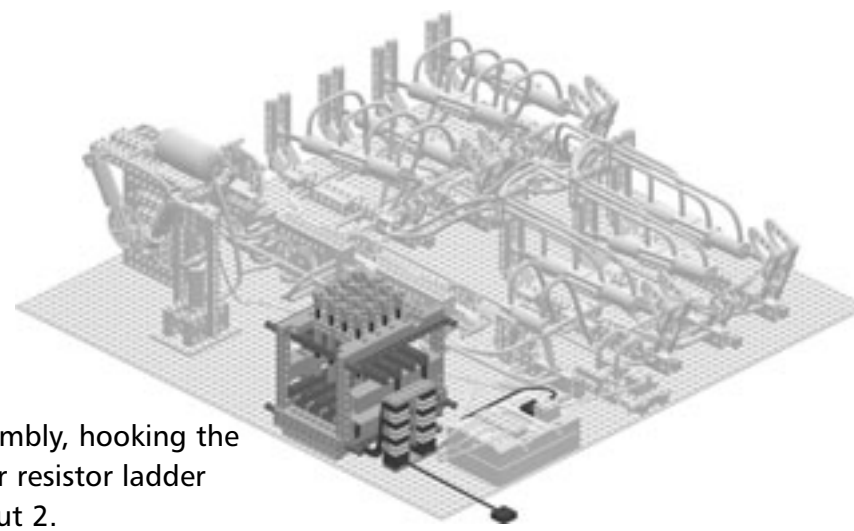
Final Step 26



Close off the unused port of the Motorized Switch sub-assembly.

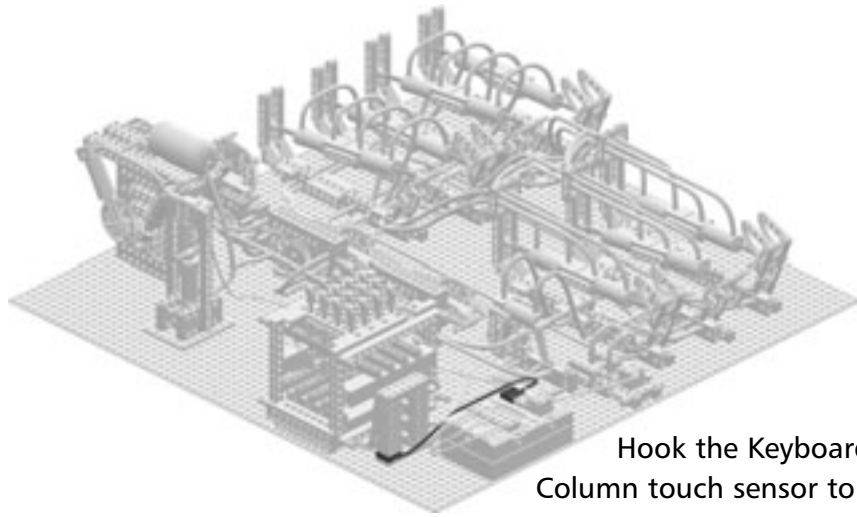


Final Step 27

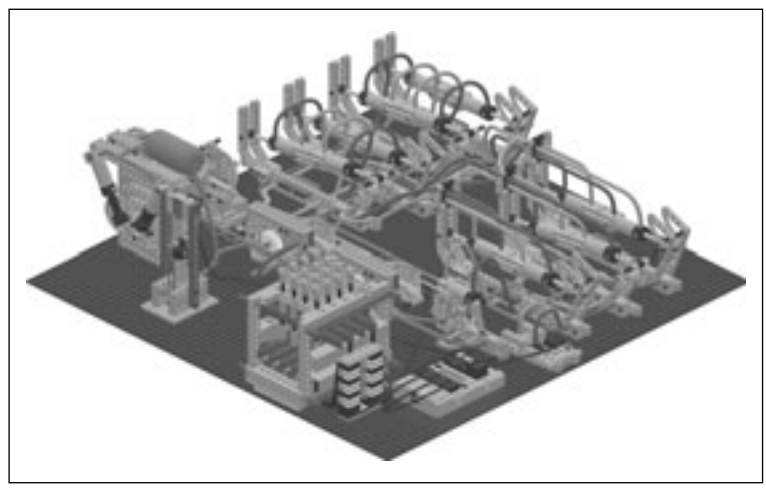


Add the Keyboard sub-assembly, hooking the Keyboard Row touch sensor resistor ladder network to RCX Sensor Input 2.

Final Step 28



Hook the Keyboard Column touch sensor to RCX Sensor Input 1.



Calibrating PneumADDic II

Before we can use PneumADDic II, we need to adjust the potentiometers associated with each of the RCX sensor ports. The RCX allows us to get readings directly from an RCX sensor port by setting the port to raw mode (directly from the RCX's analog to digital converter). The actual calculator program that runs on the RCX reads the raw values from each of the sensor ports and determines what touch sensors are depressed. We can use the following NQC program to adjust the potentiometers so they work with the calculator program.

NQC is a computer programming language for the RCX that is based on the syntax and programming capabilities of the popular C programming language. Dave Baum defined the NQC language as a replacement for LEGO's RCX graphics user interface programming methods. Dave's NQC language is a huge hit with people who may have learned other programming languages before they got their RCX. NQC is also a great way to learn how to program computers as well. Dave Baum's creation of NQC is one of the most significant contributions to the LEGO MINDSTORMS community to date.

To use an NQC program in an RCX, it must first be compiled using the NQC compiler. The compiler reads the NQC program, and then translates it into RCX code. We can then download the RCX code to the RCX, similar to the process used for RCX code created using LEGO's official programming languages. To learn about the definition of the NQC language and how to use the NQC compiler, go to the NQC Web site at www.baumfamily.org/nqc. You can find the compiled version of this program on the CD with the name raw_readings.rcx.

This program takes raw readings from Sensor Input 1 and displays the reading on the RCX's LCD display. You can use this while adjusting the potentiometers in PneumADDic II.

```
task main()
{
 int raw; // this is a variable that holds a single raw reading
          // these two statements configure sensor port 1 for raw readings
  SetSensorType(SENSOR_1, SENSOR_TYPE_TOUCH);
  SetSensorMode(SENSOR_1, SENSOR_MODE_RAW);
  while(1) {  // loop forever until you stop the program
    raw = SENSOR_1;        // record a raw sensor reading into the variable named
                           raw
    SetUserDisplay(raw,0); // display the raw reading on the RCX's display
  }
}
```

To adjust the potentiometers to the proper readings for the keyboard's column touch sensors, we must download raw_readings.rcx into the RCX (you may have to remove the RCX from your assembled PneumADDic II to take it to the computer).

The following steps explain how you should perform the calibration of PneumADDIC II:

1. After the program is downloaded to the RCX, return to your PneumADDic II design and attach the keyboard row touch sensors to Input Port 1, and press **Run** on the RCX. The number 1023 should show up on the RCX LCD display.
2. Then, press the key in the lower left corner of the keyboard. The 1023 should change to a smaller number (probably less than 60). If the number does not change from 1023, then you should check your electrical connections between the RCX and the leftmost Keyboard Column touch sensor sub-assembly.
3. Now, release the lower left key, and depress and hold the next key to the right. You should see the 1023 change to a different number. Adjust the bottom potentiometer of the Keyboard Column touch sensor sub-assembly potentiometer stack until you get a reading of 200(+/-4).
4. Move to the next key to the right, and the next potentiometer up, and adjust it until you get a reading of 400(+\-4). Finally, move to the rightmost key, and the top potentiometer and adjust it until you get a reading of 600(+\-4).
5. Remove the column sensor cable from Input Port 1, and replace it with the row sensor cable.
6. Press and hold the lower left key on the Keyboard sub-assembly, while adjusting the bottom potentiometer on the row potentiometer stack so you get a reading of 200(+\-4). Release the key.
7. Press and hold the upper left key on the Keyboard sub-assembly, and adjust the middle row potentiometer to 400(+\-4). Release the key.
8. Press and hold a key in the row second from the bottom while adjusting the top row potentiometer to 600(+\-4).
9. Remove the row sensor cable from Input Port 1.
10. Attach the Pressure and Sum Sensor cables onto Input Port 1.
11. Press and hold the touch sensor in the Sum Sensor sub-assembly, and adjust the Sum Sensor potentiometer Brick to a reading of 675. Release the touch sensor.
12. Press and hold the touch sensor in the Digital Pressure Sensor sub-assembly and adjust the Digital Pressure Sensor sub-assembly potentiometer brick to a reading of 400(+\-4).
13. Return the column sensor cable to Input Port 1, the row sensor cable to Input Port 2, and the Sum and Pressure Sensor cables back to Inut Port 3.

Using PneumADDic II

After you have calibrated the potentiometers for PneumADDic II, PneumADDic II is ready to use. Download the program calc.rcx from the CD-ROM into your RCX, and run the program. For an explanation of the mechanics of the calc.rcx program please refer to the section "Programming PneumADDic II" later in the chapter. The RCX is now waiting for you to use the keyboard to make it calculate. The format of the keys on the keyboard shown in Table 1.11. When you press keys on the keyboard, the RCX updates its display. When you press the equals key, PneumADDic II goes to work calculating your arithmetic at pneumatic speed!

Table 1.11 Key Assignments

7	8	9	Clear All
4	5	6	-
1	2	3	+
Change Sign	0	Clear Entry	=

Troubleshooting PneumADDic II

As with any complicated design, PneumADDic II has weak spots in the design. The first weak spot can be that at times there is not enough pressure to get accurate readings from the gates. If this is the case, make sure the rubber bands on the pressure sensor pneumatic piston are each wrapped three to four times around the bushings at the top of the piston. Please see **Digital Pressure Sensor Step 3** if you think you might be having this problem.

A second weak spot can be the rubber bands that connect the pneumatic pistons to the switch handle in the gates. Sometimes the rubber bands can fall off, or the axle/bushing turns and does not mesh properly with the handle. Wrapping the bands around the bushings attached to the pistons extra times can eliminate this problem.

A third issue of incorrect calculation can arise if your circuitry is not properly connected. Review the connection steps in the Final Assembly to make sure that you have all your connections correct. You can also use your knowledge of the circuit and Boolean logic to analyze the circuit by manually flipping the motorized switches and the Carry Memory switch, and predict the states of each of the gates in the design.

Programming PneumADDic II

The calc.rcx program is explained more fully in this section and contains comments to explain the program structure and how it works. This program models the behavior of a simple calculator that is able to perform addition and subtraction; it reads values from the

keyboard, and interfaces to the values to pneumatic addition logic created from LEGO TECHNIC pneumatics.

The program has consists of the following major actions:

- **sensor_decode** Reads the analog values of the RCX sensor inputs and determines the state (pressed or not pressed) of all the touch sensors in PneumADDic II.
- **set_switches** Controls the pneumatic switches that drive inputs to the pneumatic adding circuit
- **clock_carry** Controls the pneumatic switch that makes the carry memory remember a new carry value.
- **display** A task that displays numbers on the RCX's LCD display
- **calc** A subroutine that uses *set_switches()* and *clock_carry()* to control the pneumatic addition logic, using inputs (decoded by *sensor_decode()*) to get results.
- **main** The task that is run when you first run this program. It starts the *sensor_decode()* task that runs in parallel with *main()*, always reading and decoding sensor values. *main()* observes keystrokes from the keyboard, keeping track of the entered numbers and the mathematical operations desired. When needed, *main()* uses the *calc()* subroutine that makes the pneumatic adding circuit perform binary addition. Once the addition is complete, *main()* uses the *display()* task to display the answer.

In the NQC language, all variables, tasks, subroutines and functions must be declared *before* they are used. This means that subordinate tasks and subroutines must be declared before the superior routines use them. This means that the most superior task of all must be listed last. For this reason, the *main()* task is at the bottom of this program. To read the program, start with the *main()* task, and work your way backwards to the subordinate tasks, subroutines and functions.

```
/*************************************************************************
 * This task observes the RCX's analog sensor inputs and determines the
 * state (pushed or not pushed) for the pressure and sum sensors, and
 * records the state in global variables for the main task to use.
 *
 * It also observes the sensor for the keyboard row and keyboard column
 * sensors, and determines what row and column keys may be pressed.
 *************************************************************************/
#define COL_SENSOR SENSOR_1
#define ROW_SENSOR SENSOR_2
#define PRESSURE_SENSOR SENSOR_3
```

```
int rb, cb;         // encode the row and column pressed on the keyboard
int sumb = 0;       // the state of the sum sensor (0 - unpressed,1 pressed
int pressureb = 0; // the state of the pressure sensor
task sensor_decode()
{
  int raw;
  while(1) {
    // read the raw readings from the pressure/sum sensor input
    raw = PRESSURE_SENSOR;
    if (raw < 450) {
      pressureb = 1;
      sumb      = 1;
    } else if (raw < 550) {
      pressureb = 1;
      sumb      = 0;
    } else if (raw < 725) {
      pressureb = 0;
      sumb      = 1;
    } else {
      pressureb = 0;
      sumb      = 0;
    }
    // read the raw readings from the row sensor
    raw = ROW_SENSOR;
          if (raw < 100) {
      rb = 2;
    } else if (raw < 300) {
      rb = 0;
    } else if (raw < 500) {
      rb = 3;
    } else if (raw < 700) {
      rb = 1;
    } else {
      rb = 4; // no key pressed
    }
    // read the raw readings from the column sensor
    raw = COL_SENSOR;
          if (raw < 100) {
      cb = 0;
    } else if (raw < 300) {
```

```
      cb = 1;
    } else if (raw < 500) {
      cb = 2;
    } else if (raw < 700) {
      cb = 3;
    } else {
      cb = 4; // no key pressed
    }
  }
}
#define BIT_A            OUT_B
#define BIT_B            OUT_C
#define COMPRESSOR    OUT_C
#define CARRY            OUT_A
/*********************************************************************
 * These routines drive the motors which
 * control the pneumatic switches that
 * act as the inputs to the pneumatic
 * calculation logic.
 *********************************************************************/
#define TRUE_DIRECTION  OUT_FWD
#define FALSE_DIRECTION OUT_REV
#define BIT_WAIT 100
#define CALC_WAIT 500
/* this routine controls the values fed to to the pneumatic adder's
 * inputs. The binary values of a_dir and b_dir tell this routine which
 * way to turn the motors to provide the proper values to the pneumatic
 * adding machine.
 */
void set_switches(int a_dir, int b_dir)
{
  int outputs;
  if ( ! pressureb) {
    On(COMPRESSOR);
    until(pressureb);
    Off(COMPRESSOR);
  }
  outputs = BIT_A | BIT_B;
  if (a_dir) {
    SetDirection(BIT_A, TRUE_DIRECTION);
```

```
  } else {
    SetDirection(BIT_A, FALSE_DIRECTION);
  }
  if (b_dir) {
    SetDirection(BIT_B, TRUE_DIRECTION);
  } else {
    SetDirection(BIT_B, FALSE_DIRECTION);
  }
  OnFor(BIT_A|BIT_B,BIT_WAIT);
  if ( ! pressureb) {
    On(COMPRESSOR);
    until(pressureb);
    Off(COMPRESSOR);
  } else {
    Wait(CALC_WAIT);
  }
}
/* this routine makes the carry memory remember the carry of a
 * binary addition, by toggling the pneumatic switch one way, and
 * the other, temporarily applying pressure to the pneumatic
 * piston in the carry memory.
 */
void clock_carry()
{
  SetDirection(CARRY,TRUE_DIRECTION);
  OnFor(CARRY,BIT_WAIT);
  if ( ! pressureb) {
    On(COMPRESSOR);
    until(pressureb);
    Off(COMPRESSOR);
  }
  SetDirection(CARRY,FALSE_DIRECTION);
  OnFor(CARRY,BIT_WAIT);
}
/* These global variables are used to communicate between the
 * main task that manages observes the keyboard, the calc() routine
 * that is used by main() to perform pneumatic addition, and the
 * display task that displays numbers to the user using the RCX's
 * LCD display.
 */
```

```
int lhs; // value on left-hand side of arithmetic operator
int rhs; // value on right-hand side of arithmetic operator
int op;  // remembers the arithmetic operation to be performed
// task to update the display constantly
task display()
{
  while (1) {
    SetUserDisplay(rhs,0);
    Wait(10);
  }
}
task display_busy()
{
  while (1) {

SetUserDisplay(0,0);
    Wait(50);
    SetUserDisplay(0,1);
    Wait(50);
    SetUserDisplay(0,2);
    Wait(50);
    SetUserDisplay(0,3);
    Wait(50);
  }
}
// Keyboard encodings
    /*        0   1   2   3
          3    7   8   9   C
          2    4   5   6   -
          1    1   2   3   +
          0   CS   0 CE   =
     */
#define CHANGE_SGN    0
#define NUM_0         1
#define CLEAR_ENTRY   2
#define EQUAL         3
#define NUM_1        10
#define NUM_2        11
#define NUM_3        12
#define PLUS         13
```

```
#define NUM_4        20
#define NUM_5        21
#define NUM_6        22
#define MINUS        23
#define NUM_7        30
#define NUM_8        31
#define NUM_9        32
#define CLEAR        33
#define NONE         40
/*********************************************************************
   This is the calculation subroutine that performs binary addition
   of the values you key in at the keyboard.

   The task named main calls this routine when it wants to have to
   numbers added.  The task communicates the values it wants added
   using the global variables lhs (the value on the left-hand side
   of the plus sign), and the variable rhs (the value on the right-
   hand side of the plus sign
   calc walks its way through the binary places starting at the 1's
   place, then the 2's place, then the 4's place, up through the
   512's place, adding the bits from those binary places in lhs and
   rhs together, along with the carry.  After the sum output of the
   pneumatic circuit has been read, the calc routine makes the
   carry memory remember the carry, so that it can be used in the
   next binary place.
   As calc walks through the binary places, it keeps track of the
   total sum by accumulating the sum into the global variable rhs.
*********************************************************************/
sub calc()
{
  int value1,value2;         /* working copies of global variables
                                    lhs, and rhs */
  int binary_place;
  int get_carry;              /* remember if we need to get the last
                                    carry, because there are no more
                                    binary places to add.
                               */
  // make the display cycle through a repeating pattern so the
  // user knows addition is not complete.
  stop display;
```

```
start display_busy;
// O.K.  Here come the pneumatics calculations
if (op == MINUS) {
  rhs = -rhs;        // PneumADDic can perform subtraction by
                      // adding a negative number.
                      // This is kind of cheating because we're using
                      // the RCX's arithmetic electronics to create
                      // the negative number
}
// In the first binary place (the ones column), there is no
// carry in, so we must force the carry memory to 0.
set_switches(0,0);  // force inputs to the adder to zero
clock_carry();      // make carry remember the zeroes.
get_carry = 0;
value1 = lhs & 511; // copy values from lhs and rhs to working
value2 = rhs & 511; // variables

rhs = 0;            // put total sum result into rhs
// for each binary place
for (binary_place = 1; binary_place < 512; binary_place *= 2) {
  // if we have values to add, or we need to add the last carry
  if (value1 != 0 || value2 != 0) {

    // Add a bit from value1 and a bit from value2 using the pneumatic
    // adder

    set_switches(value1 & 1, value2 & 1);

    // if the sum sensor says we had a sum of 1, OR in a 1
                        // into the current binary place in lhs
    if (sumb) {
      rhs |= binary_place;
    }
    // remember the carry
    clock_carry();
    // when value1 and value1 and value2 run out of non-zero bits
                       // to add, we have to add the final carry into lhs.
    get_carry = 1;
  } else if (get_carry) {
    get_carry = 0;
```

```
      set_switches(0,0);     // extract carry by setting the value inputs
                             // to 0 and reading the sum sensor
                             // hint: 0 + 0 + carry = carry
      if (sumb) {
        rhs |= binary_place;
      }
    }

    value1 /= 2; /* throw away the bits we've already added */
    value2 /= 2;
  }
              // because of display limitations on the RCX, we can only display
              // 3 digit numbers, so we must limit the number of binary places
              // so that the largest number fits in 3 decimal digits. This means
              // the largest number supported is 511.
              //
              // When the pneumADDic sum ends up negative, we need to convert our
              // 9-bit integer to a 16-bit integer (the size of the integers the
              // RCX computer uses) by repeating the leftmost bit of our 9-bit
              // integer across the unused bits
  if (rhs & 256) {
    rhs |= -512;
  }
              // calculations have finished
              // display the results
  stop display_busy;
  start display;
              // beep so the user knows we're done
  PlaySound(SOUND_CLICK);
  PlaySound(SOUND_CLICK);
}
/*****************************************************************************
 * This is the main task.  It initializes
 * things, and does the calculator
 * user interface things.
 *****************************************************************************/
task main()
{
  int d;      // the digit value the user pressed on the keyboard
  int first;  // is the number being entered the first digit?
```

```
            // start the display task that shows the rhs variable to the user
  start display;
            // initialize all the sensor ports to be raw touch sensor ports
  SetSensorType(PRESSURE_SENSOR, SENSOR_TYPE_TOUCH);
  SetSensorMode(PRESSURE_SENSOR, SENSOR_MODE_RAW);
  SetSensorType(ROW_SENSOR, SENSOR_TYPE_TOUCH);
  SetSensorMode(ROW_SENSOR, SENSOR_MODE_RAW);
  SetSensorType(COL_SENSOR, SENSOR_TYPE_TOUCH);
  SetSensorMode(COL_SENSOR, SENSOR_MODE_RAW);
            // start the task that watches the sensors, decodes the readings and
            // determines what keyboard buttons where pushed, and if the
            // pressure sensor and sum sensor are pushed.
  start sensor_decode;
#if 1
            // pressurize the system
  if ( ! pressureb) {
    On(COMPRESSOR);
    until(pressureb);
    Off(COMPRESSOR);
  }
  lhs = 0;      // lhs and rhs are the values to be added
  rhs = 0;
  first = 1;    // We need to know when a digit key is the first digit of
               // a new number
  op = EQUAL;  // this keeps track of the operation we're performing

  while (1) { /* do this forever */
    int r,c;

    // wait for a keypad press
    r = 4;
    c = 4;
    while (r == 4 || c == 4) {
      r = rb;
      c = cb;
    }

    // audio confirmation of key press

    PlaySound(SOUND_CLICK);
```

```
d = 0;

// decode key and take proper action

switch (r*10+c) {
  case NUM_9:        // the user hit a number key
    d++;              // calculate the digit value
  case NUM_8:
    d++;
  case NUM_7:
    d++;
  case NUM_6:
    d++;
  case NUM_5:
    d++;
  case NUM_4:
    d++;
  case NUM_3:
    d++;
  case NUM_2:
    d++;
  case NUM_1:
    d++;
  case NUM_0:        // d now contains the value of the key
    if (first) {     // if this is the first digit of a new
      rhs = 0;       // number, zero out rhs
      first = 0;
    }
    // accumulate the value the user wants to add in rhs
    if (abs(rhs) < 99) {  /* allow 3 digits */
      rhs = rhs * 10 + d;
    }
  break;

  case PLUS:         // the + key was pressed
    lhs = rhs;       // transfer rhs to lhs
    op = PLUS;       // remember that the next operation is addition
    first = 1;       // next digit is the beginning of a new number
  break;
```

```
      case MINUS:       // The - key was pressed
        lhs = rhs;      // transfer rhs to lhs
        op = MINUS;     // next operation is subtraction
        first = 1;      // next digit is begining of new number
      break;

      case EQUAL:       // the = key was pressed
                        // If the previous operation was EQUAL, then the
                        // user is confused on how to run a calculator,
                        // because we only have one number to work with.
                        // Otherwise the operation to be performed is
                        // addition or subtraction
        if (op != EQUAL) {
          calc();       // perform the pneumatic calculation
        }
        op = EQUAL;     // we're down to one value and it is in rhs
        first = 1;      // next digit is the beginning of a new number
      break;
      case CLEAR:       // the user hit the clear key
        lhs = 0;        // zero out the values
        rhs = 0;
        first = 1;      // record that the next digit key is first in a
                        // new decimal number
        op = EQUAL;     // an operation of EQUAL means there is no
                        // arithmetic to perform
      break;

      case CLEAR_ENTRY: // the user hit the clear entry key
        rhs = 0;        // zero out the number on the display
        first = 1;      // next digit key is first in new number
      break;
      case CHANGE_SGN:  // change sign key was pressed
        rhs = -rhs;     // make the number negative
      break;

    } /* end case */

                        // wait until they've released they key
    while (rb != 4 || cb != 4);
  } /* end while */
```

```
#else
  while (1) {
    d = rb*10+cb;
    rhs = d;
  } /* end while */
#endif
} /* end main */
```

Summary

PneumADDic II has provided you examples of how to combine the disciplines of pneumatics and electronics. By combining electric motors with pneumatic pumps, we created an air compressor. Then by joining motors with pneumatic switches, the RCX can control pneumatic circuits. Finally combining pneumatic pistons with electric touch sensors, the RCX can sense what is happening in the pneumatic circuit.

You also got a glimpse into the world of electricity and electronics, and that potentiometers (adjustable resistors) affect electronic circuits by throwing away electricity by converting it to heat. You leaned about series and parallel resistive networks and how they can be used to attach multiple touch sensors to one RCX analog input port.

PneumADDic II is the most advanced and complicated LEGO pneumatic design I've ever created. It breaks new ground in the world of LEGO pneumatic logic gates, and introduces new technologies in the form of synchronous pneumatic design as can be seen in the implementation of the Carry Memory. PneumADDic II is an excellent educational tool in that it makes the world of Boolean logic and binary arithmetic accessible in a physical representation you can watch and comprehend.

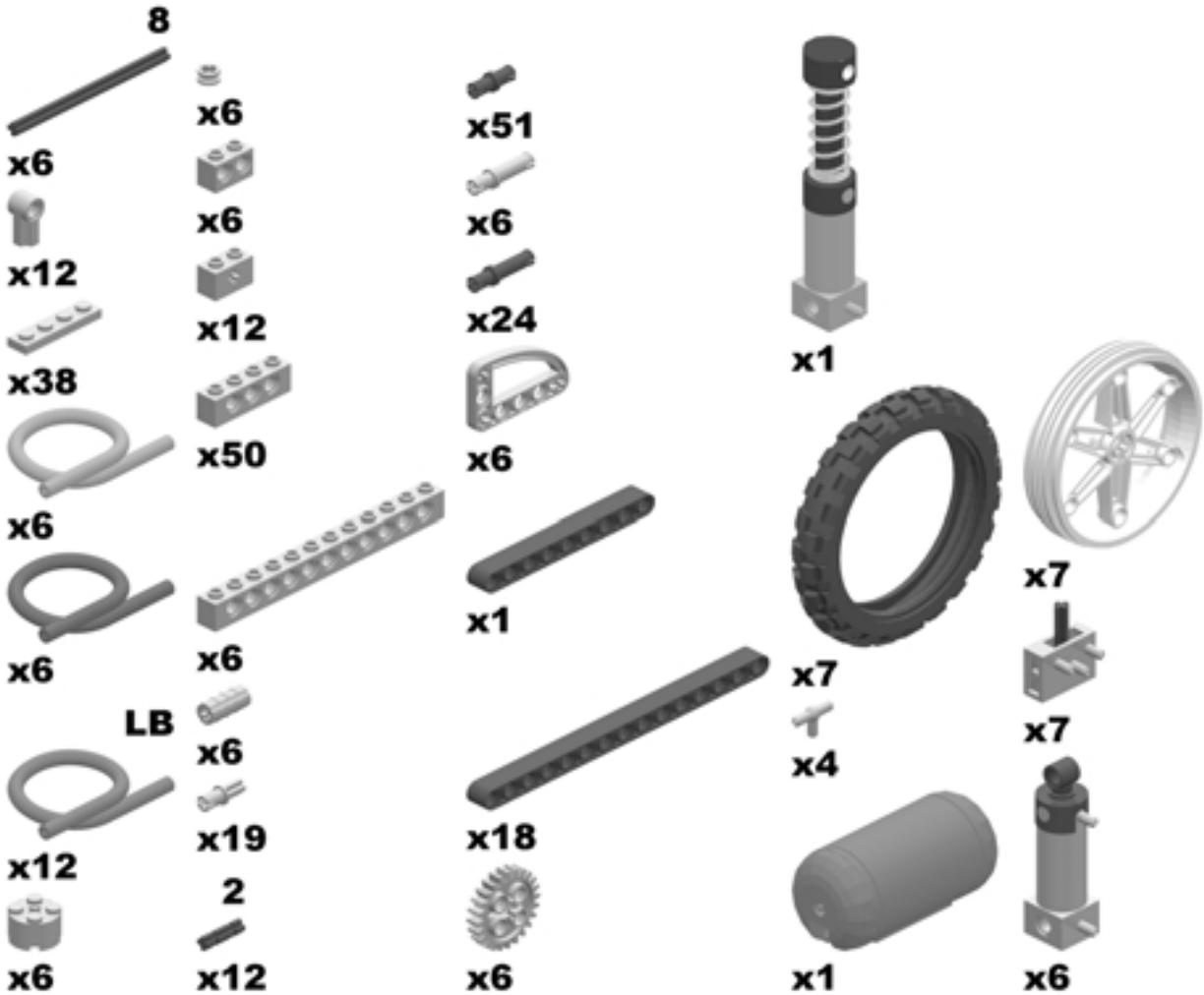
Masterpiece 5

Synchropillar

Kevin Clague

Bill of Materials

These are the parts necessary to build the Synchropillar as shown:



Introduction

In Masterpiece 4, "PneumADDic II," you learned that by using a single pneumatic pump and air pressure tank, combined with multiple pneumatic switches and pneumatic pistons, you could create pneumatic devices that can change shape using air pressure. PneumADDic II, however, had to use an RCX to control motors that flipped pneumatic switches. In this chapter, we will learn how to build pneumatic circuits that change on their own simply by applying air pressure. Instead of using motors to flip pneumatic switches, Synchropillar controls the pneumatic switches using pneumatic pistons, and compressed air.

One might question if Synchropillar really belongs in this book, because it has no RCX. Synchropillar is really more of a TECHNIC Masterpiece than a MINDSTORMS

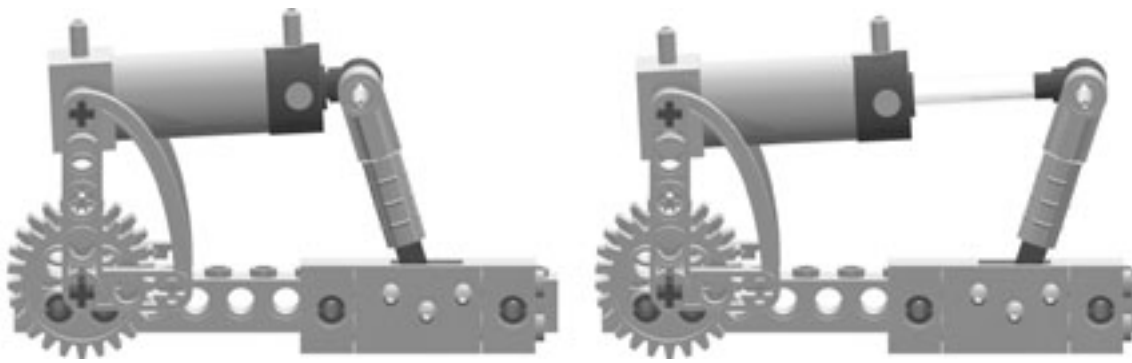
Masterpiece, because all of its parts can be purchased by buying TECHNIC sets (although if you own a MINDSTORMS 2.0 kit you already have many of these parts at your disposal as well). Nonetheless, Synchropillar is an advanced robot, because the way the pneumatic elements are coordinated and assembled eliminates any need for an RCX.

In this chapter, you will learn about Synchropillar, my pneumatic LEGO caterpillar. Synchropillar's body is comprised of seven segments, separated by six pneumatic pistons. Each segment of Synchropillar's body has a Foot sub-assembly that touches the ground. The robot travels by pushing its feet forward one foot at a time. The feet are forced to move forward by the pneumatic pistons that expand and contract in a coordinated fashion. To build Synchropillar with a minimum number of pneumatic pistons, I invented a new LEGO pneumatic design technique called *synchronous design*.

Autonomous Pneumatic Circuits

By controlling the position of a pneumatic switch using a pneumatic piston, we can create complex pneumatic circuits that repeatedly cycle through a set pattern without any need for human intervention (this could also be referred to as an *autonomous action*). Figure 5.1 shows a pneumatic piston controlling a pneumatic switch. Notice how the state of the pneumatic piston (expanded or contracted) controls the position of the pneumatic switch lever.

Figure 5.1 Pneumatic Pistons Controlling Pneumatic Switches



This pneumatically controlled switch is the basic building block for many simple and advanced pneumatic circuits. Figure 5.2 shows two pneumatically controlled switches linked together to create a pneumatic timing circuit. Erik Brok's Web page (http://homepages.svc.fcj.hvu.nl/brok/legomind/insights/other/pneutime.html) first introduced me to this type of pneumatic circuit.

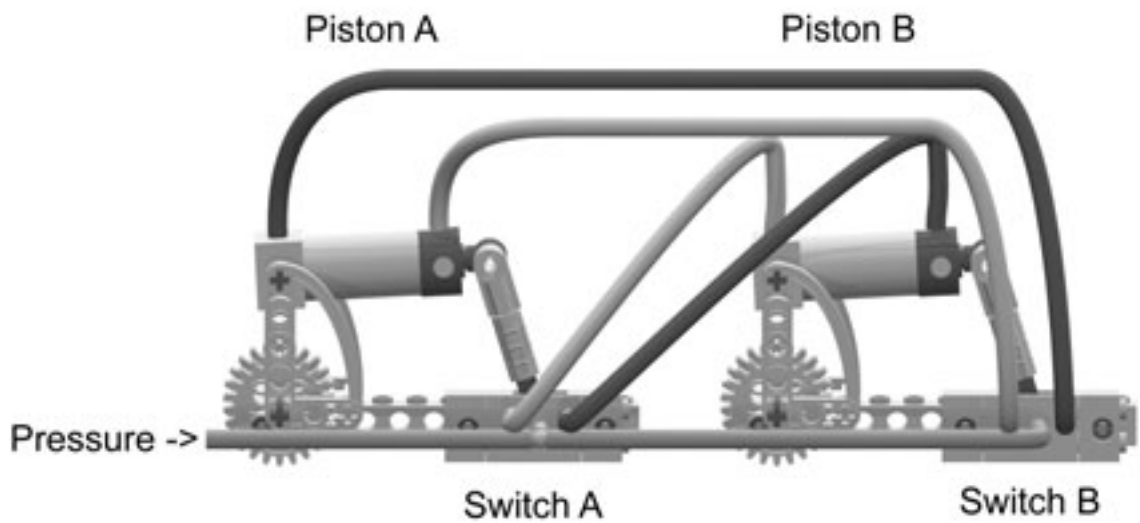
NOTE

You can find additional information on Erik Brok and his Web site (http://home.zonnet.nl/ericbrok/legomind), "LEGO on my Mind," in Masterpiece 6.

When pressure is applied to this circuit, one of the two pneumatic pistons is always either expanding or contracting. When Piston A contracts, Switch A flips to the left, which in turn causes Piston B to contract. When Piston A expands, Switch A flips to the right, which in turn causes Piston B to also expand. Simply put: Piston B always tries to mimic Piston A.

However, the relationship between Piston B, Switch B and Piston A is slightly different. When Piston B contracts, Switch B flips to the left. However, because the pneumatic tubing connections to Piston A are reversed in comparison to Piston B, Piston A

Figure 5.2 Pneumatic Timing Circuit

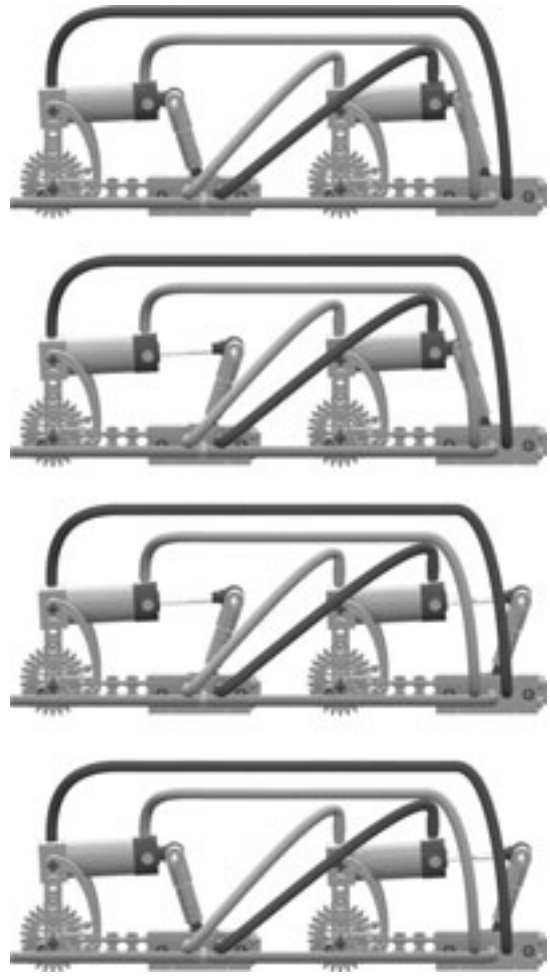


expands. When Piston B expands, Switch B flips to the right, which causes Piston A to contract. Simply stated: Piston A always tries to be in the opposite state of Piston B.

The pneumatic timing circuit in Figure 5.2 can also be referred to as a *pneumatic feedback circuit*, because Switch B feeds back to Piston A (where we started our initial analysis of the circuit.) Piston A alters the state of Piston B, which in turn alters the state of Piston A. Over time this circuit goes through four distinct states. Each state is illustrated in Figure 5.3. From the fourth state, the circuit returns to the initial state of

contracted/contracted. All you need to do is apply air pressure and the circuit will continually cycle through all four states.

Figure 5.3 Pneumatic Timing Circuit States



The rate of change in the timing/feedback circuit depends upon the amount of pressure that is applied to the pneumatic pistons, the friction of the pneumatic pistons, and the friction in the pneumatic switches. The higher the pressure applied to the circuit, the faster the pneumatic pistons expand and contract, causing the entire circuit to cycle through the four states. This timing/feedback circuit is a fundamental building block of complex pneumatic designs.

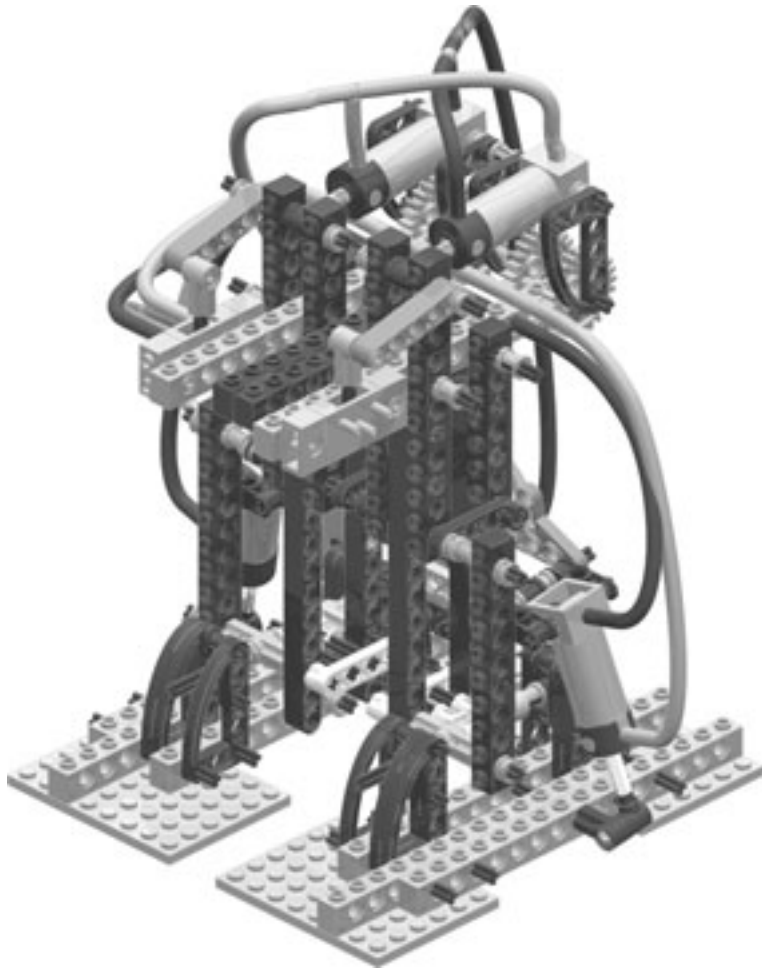
Another Model That Uses Pneumatic Timing/Feedback Circuits: PneumaPed

In addition to the Synchropillar, another great example of the pneumatic timing/feedback circuit can be seen in practice in my pneumatic biped walker, aptly named PneumaPed. An illustration of PneumaPed is shown in Figure 5.4. PneumaPed uses four pistons and two switches. Two pistons, one on each of the PneumaPed's hips, drive the legs forwards and backwards. Two pistons, one located on each of the ankles, cause the PneumaPed to lean toward the right and the left. The pneumatic piston and its pneumatic switch for the right hip, combined with the pneumatic piston and the pneumatic switch for the right ankle, create the pneumatic timing circuit shown in Figure 5.2.

The pneumatic piston for the left hip is driven by the pneumatic switch for the right ankle. The difference here is that the pneumatic tubing for the pneumatic piston in the left hip is connected in the opposite manner of the pneumatic piston for the right hip. When the pneumatic piston for the right hip expands, the pneumatic piston for the left hip contracts, and vice versa. This makes PneumaPed's legs stride forwards and backwards similar to ours. While the pneumatic piston of PneumaPed's right ankle is driven by the right hip pneumatic switch, the pneumatic piston of the left ankle is driven by the right hip pneumatic switch. The left ankle's pneumatic tubing is connected in reverse from the right ankle piston, so that both ankles lean right and left in a coordinated fashion.

You can find the LDraw building instructions for PneumaPed on my web site www.users.qwest.net/~kclague

Figure 5.4 PneumaPed



Asynchronous Designs

If you build two of the pneumatic timing/feedback circuits, shown in Figure 5.2, and apply equal pressure to both, they will both follow the sequence of Figure 5.3 initially. However, it won't take long before the two timing circuits fall out of synchronization. This happens because the time to complete the four phases of the sequence depends in the friction of the pneumatic pistons and pneumatic switches. No two pneumatic pistons or pneumatic switches possess the exact same amount of friction; therefore, the time it takes to complete the sequence will be different for the two timing circuits. This is not to say that the two pneumatic timing circuits may not stay in sync for a while, but one of the timing circuits is bound to change faster than the other, and sooner or later they fall out of step with each other. The fact that the two timing circuits are not coordinated makes them *asynchronous* (meaning not synchronized). As a computer-engineering

student, I learned that designing electronic asynchronous circuits was hard and more of an art than a science. During my studies, I was also taught a much easier alternative design technique called *synchronous design*. Almost all of the digital electronics in your personal computer are designed using synchronous design techniques.

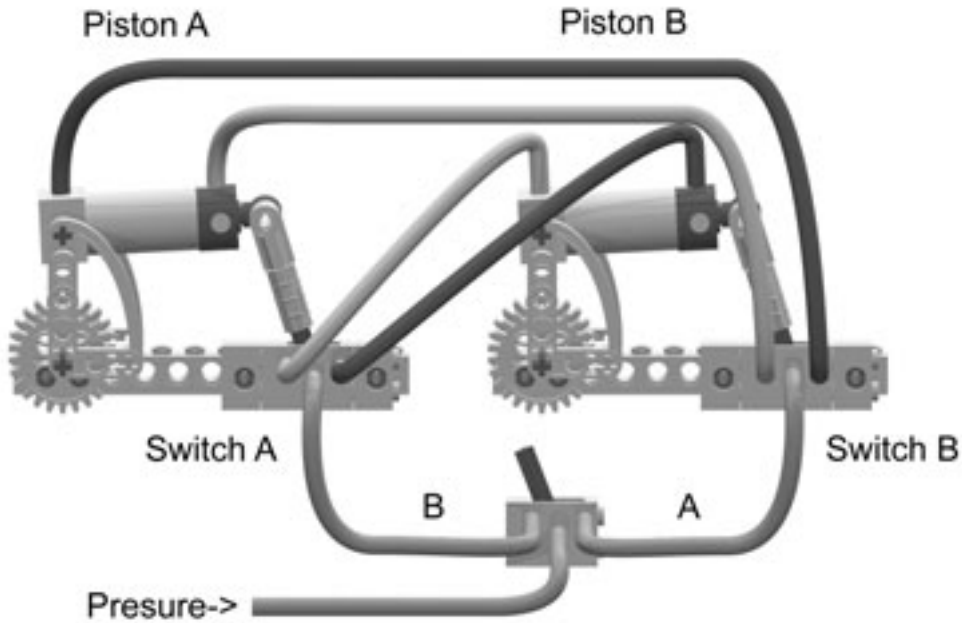
Pneumatic Memory

As we saw in Masterpiece 4, "PneumADDic II," a pneumatic switch controlled by a pneumatic piston can be used to create pneumatic memory device. PneumADDic II used a device like the one in Figure 5.2 for the Carry Memory sub-assembly. When the pneumatic piston is contracted, the pneumatic switch flips to a specific position. The pneumatic piston and pneumatic switch will stay in that position until an outside force acts upon it. Another way to explain this is that the pneumatic piston/switch *remembers* its current position until an outside force acts upon it. By applying pressure (outside force) to the pneumatic piston, we can change the position of the pneumatic piston, and therefore, change the position of the pneumatic switch. When pressure is removed, the pneumatic piston/switch remembers the new position. The technique of applying pressure can be a very powerful pneumatic circuit design technique: applying pressure to force the pneumatic piston to remember a new value, and then removing the pressure, thus allowing the pneumatic piston/switch to remember the position until additional pressure is applied, which will cause the piston/switch combination to change state. Only controlled application of pressure to the circuit can control when the piston changes positions. This technique has a parallel in the world of electronic circuit design, called *synchronous* design methodology.

Synchronous Designs

Figure 5.5 shows a circuit much like our asynchronous timing circuit. However, in this example, Switch A and Switch B have separate pressure input lines (pneumatic tubing). Pressure is applied only to one pressure input line at a time. When pressure is applied to Pressure Line A, there is no pressure in Pressure Line B. Pressure Line A drives Switch B, and changes the position of Piston A (and its corresponding pneumatic switch). While this occurs, there is no pressure in Pressure Line B; therefore, Piston B cannot change its position. When the application of pressure toggles from Pressure Line A to Pressure Line B, Pressure Line B is pressurized but Pressure Line A is not; therefore, Piston B is able to change position while Piston A remains unchanged. By alternating pressure from Pressure Line A to Pressure Line B, we can regulate when the circuit is able to expand and contract its corresponding pistons.

Figure 5.5 Synchronous Pneumatic Circuit



NOTE

This alternating pressure is similar to the "clocking" of processor speeds used in synchronous computer designs. When you go out and buy a new computer with a 2GHz processor, you are buying a processor that can be clocked at 2 billion times in a second. This clock in synchronous designs acts like a conductor of a band or orchestra. The conductor is the external time reference that everyone uses to stay coordinated. During warm ups, before the conductor comes out, the orchestra sounds like a bunch of strange unrelated sounds, because the musicians are warming up without coordinating with each other. They are warming up asynchronously. When the conductor directs a musical piece, the conductor synchronizes the musicians so they can make beautiful music. The conductor plays the role of the clock in an orchestra.

The alternating pressure acts as a synchronizer in synchronous pneumatic timing circuits. Using synchronous design techniques here allows us to create pneumatic circuits that are otherwise impossible to create using only asynchronous design techniques.

If you build two of the circuits as shown in Figure 5.5, and connect Pressure Line B to both Switch As and Pressure Line A to both Switch Bs, the two circuits will stay coordinated as you toggle pressure between Pressure Line A and Pressure Line B. The pistons in the two circuits may not expand or contract at exactly the same rate, but as long as the

pressure is not toggled faster than it takes to expand or contract the slowest piston, the circuit will stay synchronized.

Now that we understand the concept of synchronous pneumatic design, we can talk about Synchropillar's design.

Building Synchropillar

The inspiration for this design comes from memories of my childhood when I watched a caterpillar move by sending waves of expansion down the sequence of muscles that lead from its head to its tail. When the expanded muscle wave hit the end of the tail, it started over at the head. The same thing that happens in our six-stage synchronous circuit.

Synchropillar has a body segmented into seven parts, each with a LEGO wheel and tire as a foot. The seven segments are separated by the six pistons, as shown in Figure 5.6. The flowing steps show you how to create each segment, and how to hook them together using pistons and switches. Synchropillar contains six pneumatic memory elements (pneumatic pistons combined with pneumatic switches), where three memory elements are controlled by Pressure Line A, and three memory elements are controlled by Pressure Line B. Figure 5.6 shows the synchronous circuit used to make Synchropillar walk. Notice that this circuit looks much like the two-stage sequence you would see if you built two of the circuits shown in Figure 5.5. However, there are two primary differences:

- Six memory elements are used instead of four.
- More importantly, the last stage (Piston/Switch 3B) is not connected in the opposite to the first stage (Piston/Switch 1A). This causes Piston 1A to try to match the position of Piston 3B, instead of trying to be in the opposite position of Piston 3B.

Each pneumatic piston in Figure 5.6 tries to match the position of the pneumatic piston to its left (when pressure is applied to it). After you toggle the pressure and the pneumatic pistons move, two of the pneumatic pistons are always expanded, while the other four are contracted (Figure 5.6). This is slightly different than the pneumatic timing circuits described above, where the number of pistons that are expanded and contracted changes as you toggle pressure from Line A to Line B. The circuit in Figure 5.6 creates a sequence that is of *length 6*, as shown in Table 5.1.

Figure 5.6 Synchropillar Schematic

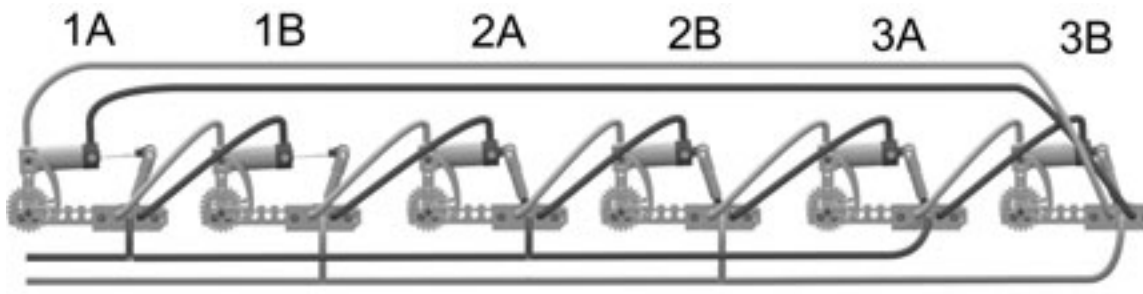


Table 5.1 Synchropillar Sequence

Time	1A	1B	2A	2B	3A	3B
1	Expand	Expand	Contract	Contract	Contract	Contract
2	Contract	Expand	Expand	Contract	Contract	Contract
3	Contract	Contract	Expand	Expand	Contract	Contract
4	Contract	Contract	Contract	Expand	Expand	Contract
5	Contract	Contract	Contract	Contract	Expand	Expand
6	Expand	Contract	Contract	Contract	Contract	Expand
7	Expand	Expand	Contract	Contract	Contract	Contract

The *length 6* sequence shown in Table 5.1 is hard to produce with an asynchronous pneumatic circuit. I created a similar caterpillar using asynchronous design, and it required twice as many pneumatic pistons and twice as many pneumatic switches. The whole idea of using synchronous design techniques is to produce sequences that are difficult, if not impossible to produce in an asynchronous design.

Building the Feet

The design of Synchropillar's feet is incredibly simple. Each foot is constructed out of a minimal amount pieces: a wheel assembly, TECHNIC bricks, TECHNIC liftarms, and connector pins.

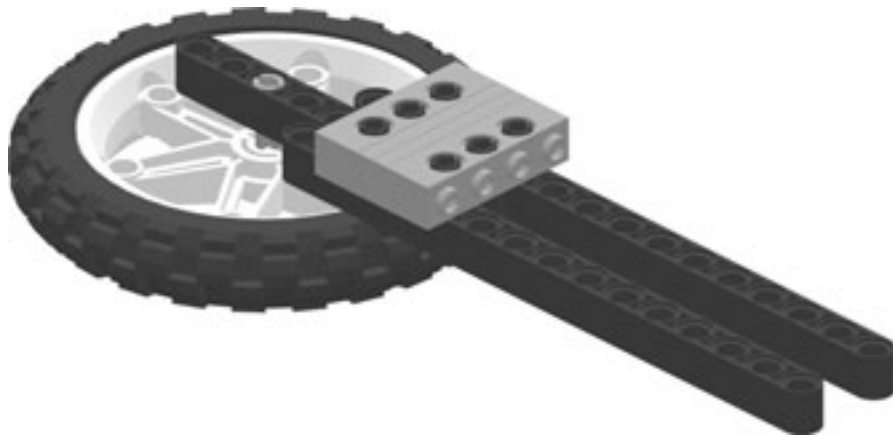
The tires on each of the wheels provide a gripping function that allows the other wheels to grip a flat surface as the Synchropillar propels itself forward. You will note that each foot contains a wheel that touches the ground, as well as TECHNIC liftarms that interconnect the feet along Synchropillar's central line. 1x4 TECHNIC bricks and 1x4 plates are used to interlock the feet together into a single body. The liftarm and the brick interlock mechanism allow the feet to slide in sequence, but still maintain the shape of the body.

The middle feet provide for the mountings of a pneumatic piston and switch, as well as the connections to the trailing neighbor's pneumatic piston. This allows the body

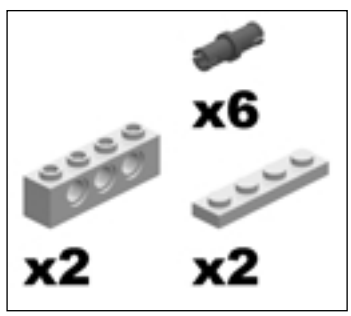
segments (feet) to separate when the interconnecting piston expands, and slide together when the interconnecting piston contracts. The front foot has no mountings for pistons or switch, because there is no piston or switch associated with the front foot. The front foot provides for connections to the trailing neighbor foot. The back foot has mountings for piston and switch, but does not provide for connecting to a trailing neighbor foot, because the back foot is the last foot.

The Front Foot

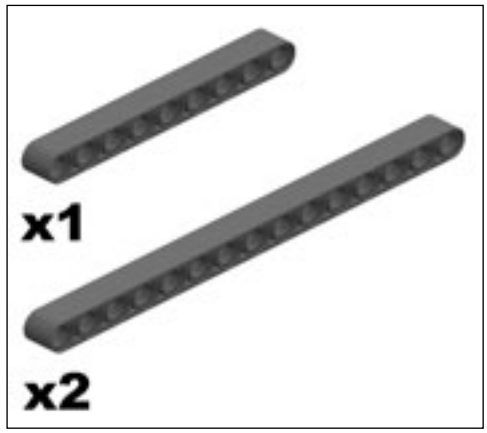
In this series of steps, you will build the Front Foot sub-assembly. Because the front foot is only connected to the feet behind it (rather than being connected to feet on either side or ahead of it), this sub-assembly is a bit different from the feet you will construct later in the chapter.

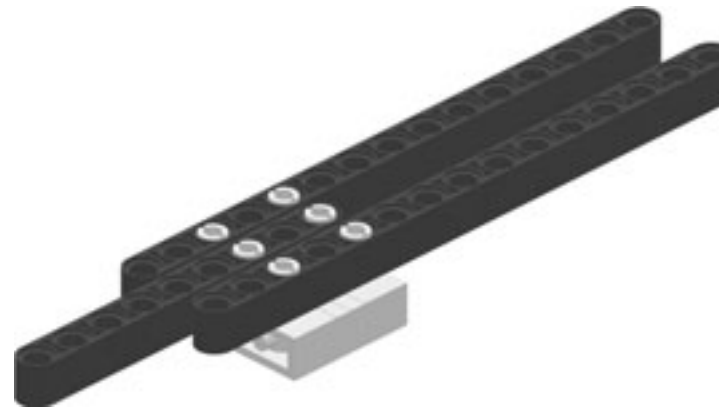


Front Foot Step 0

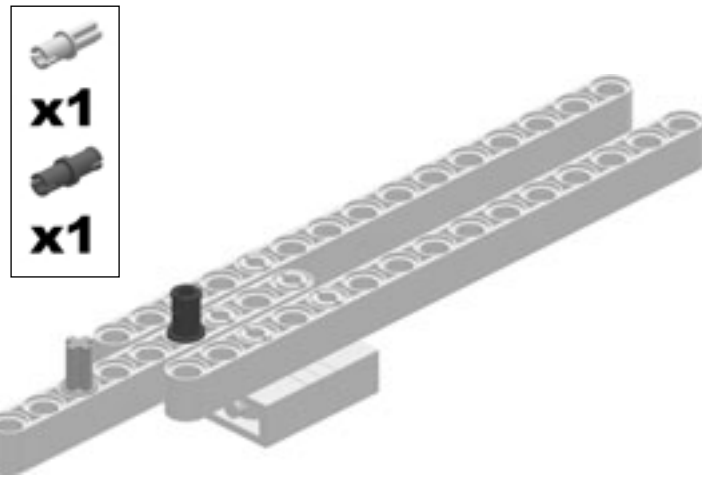


Front Foot Step 1

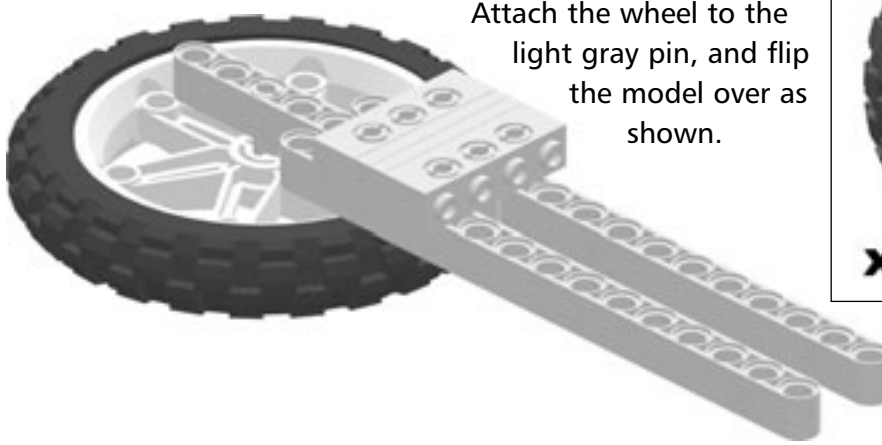




Front Foot Step 2

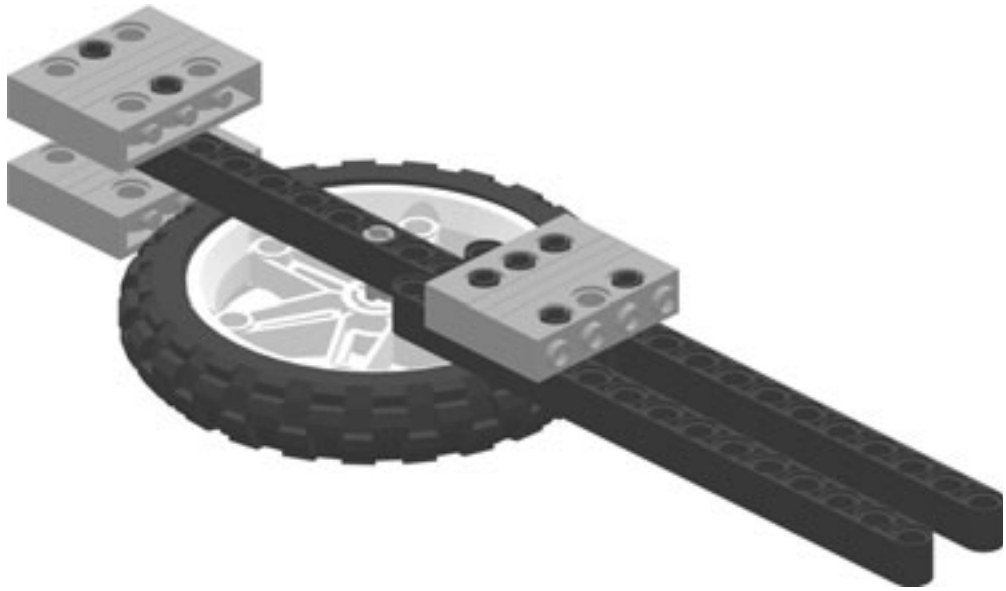


Front Foot Step 3



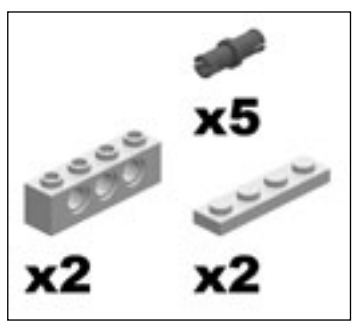
Attach the wheel to the light gray pin, and flip the model over as shown.

The Middle Feet

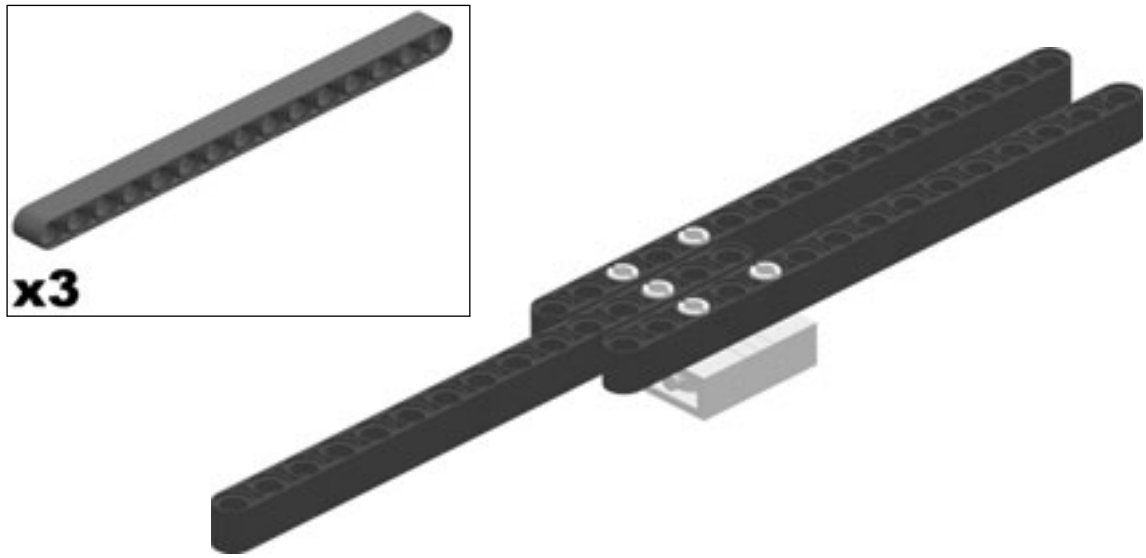


Moving on, we will now build the Middle Foot sub-assemblies. You will need to build **five** of the Middle Foot sub-assemblies. Notice that these constructions contain longer liftarms than those that were used in the Front Foot sub-assembly. This liftarm construction, combined with the TECHNIC bricks attached to the model in **Middle Foot Step 5**, provides the ability for the feet to interlock, but still slide independently of each other.

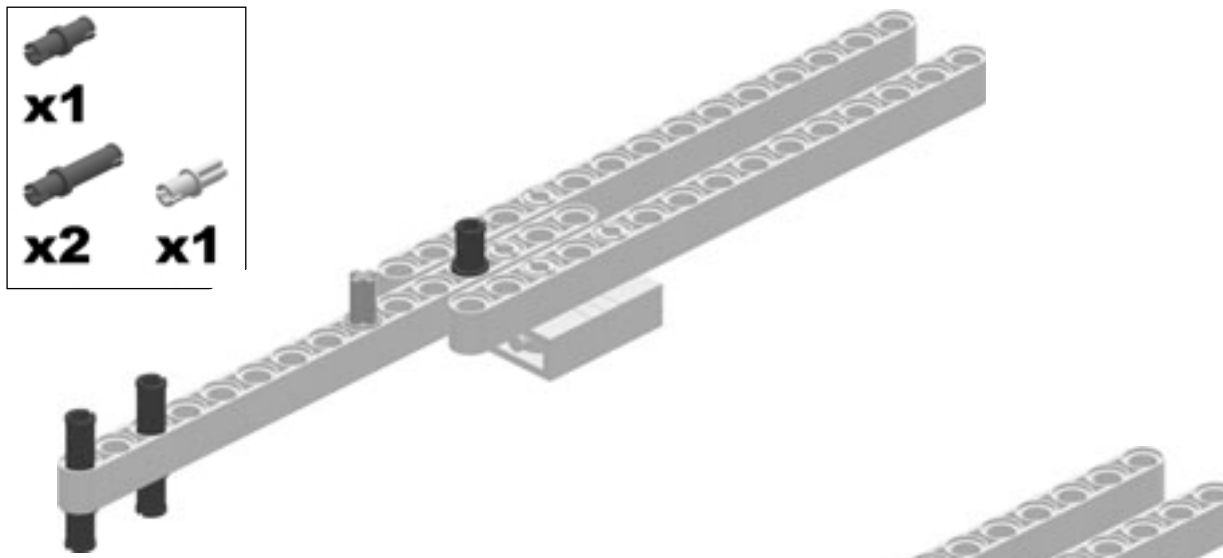
Middle Foot Step 0



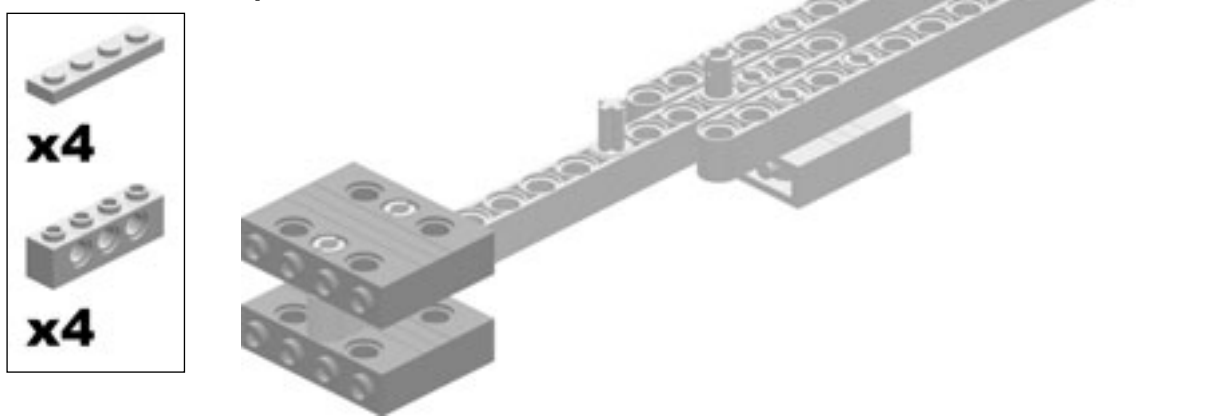
Middle Foot Step 1



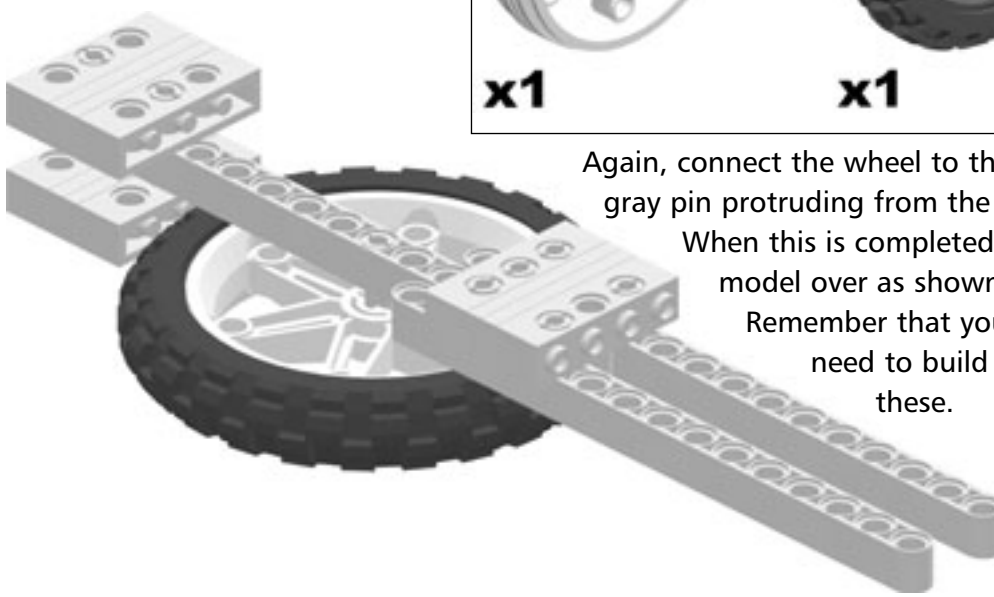
Middle Foot Step 2



Middle Foot Step 3



Middle Foot Step 4

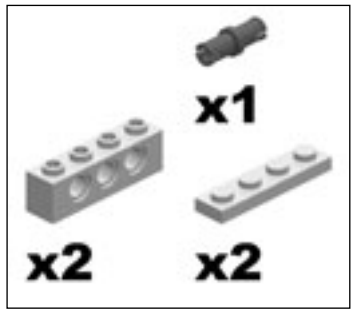


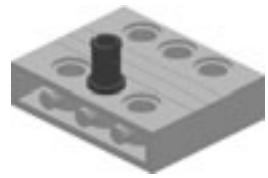
Again, connect the wheel to the light gray pin protruding from the liftarm. When this is completed, flip the model over as shown. Remember that you will need to build **five** of these.

The Back Foot

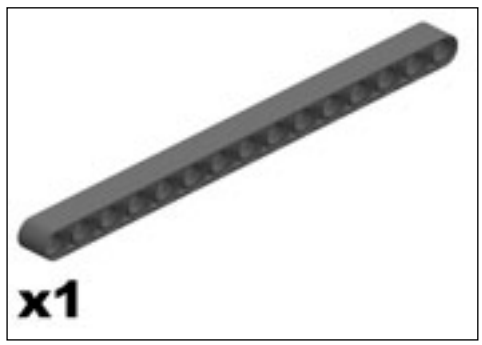
Finally, let's build the Back Foot sub-assembly. The back foot does not have another foot trailing it; therefore, it requires only a single liftarm, rather than the three liftarms that the Front and Middle Foot sub-assemblies require.

Back Foot Step 0

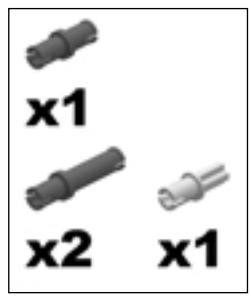


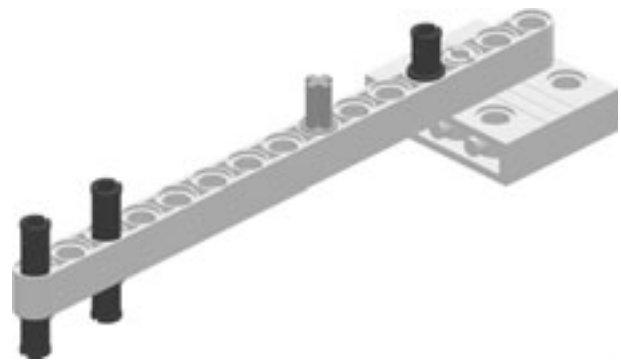


Back Foot Step 1

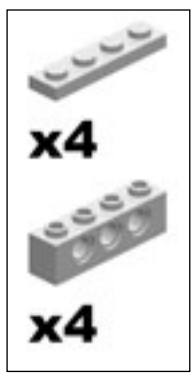


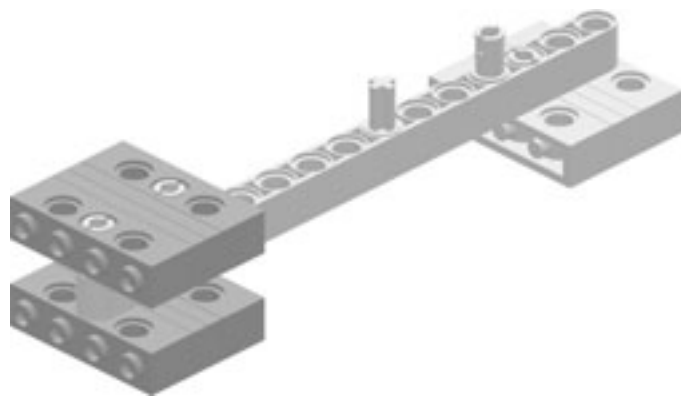
Back Foot Step 2





Back Foot Step 3





Back Foot Step 4

x1

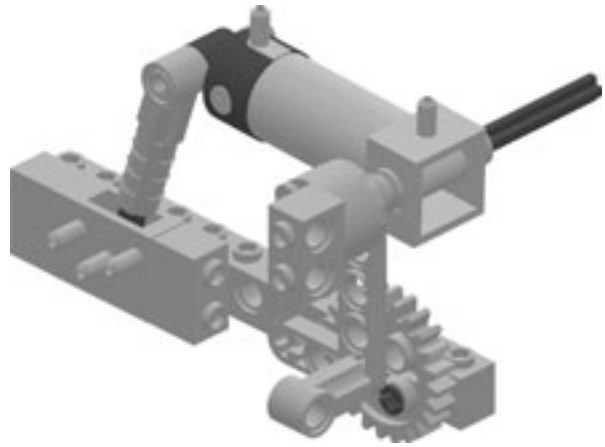
x1

Connect the wheel to the light gray pin, and rotate the model as shown.

Building the Pneumatic Memories

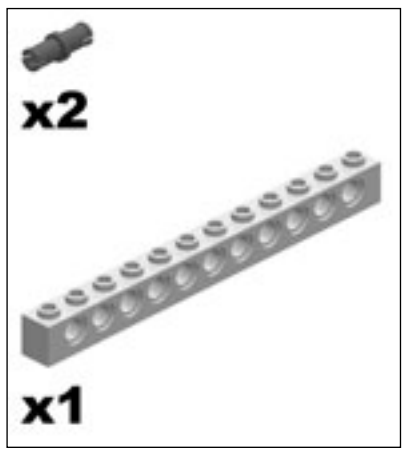
In total, Synchropillar contains six pneumatic memories. The memories tend to protrude from what you would consider the body of Synchropillar (its center line). In hopes of streamlining the design of Synchropillar, and making it more aesthetically pleasing and balanced, I decided to connect three pneumatic memories to the left side of Synchropillar's (these are the Left-hand Memory sub-assemblies), and connect three pneumatic memories to the right-hand-side of Synchropillar (these are the Right-hand Memory sub-assemblies).

The Right-hand Memory

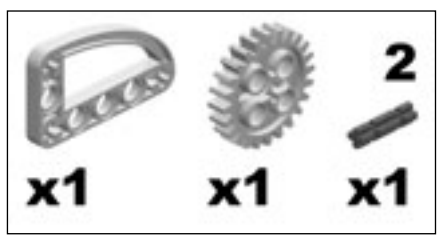


We will start with building the Right-hand Memory sub-assemblies. As stated previously, there are three of the pneumatic memories; therefore, you will need to build **three** of these.

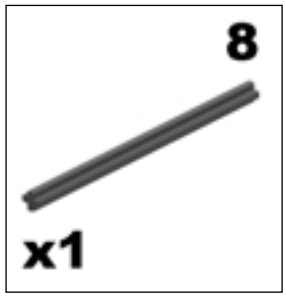
Right-hand Memory Step 0



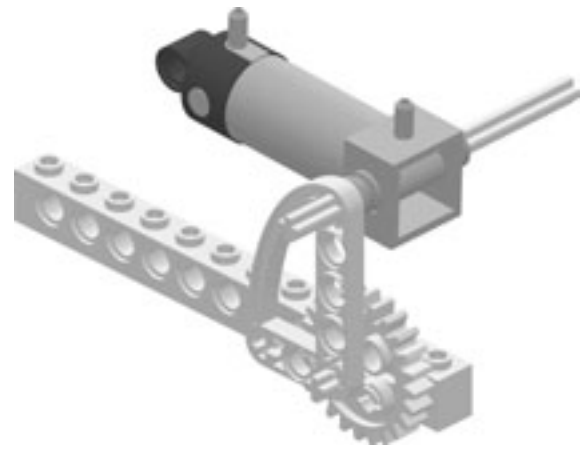
Right-hand Memory Step 1



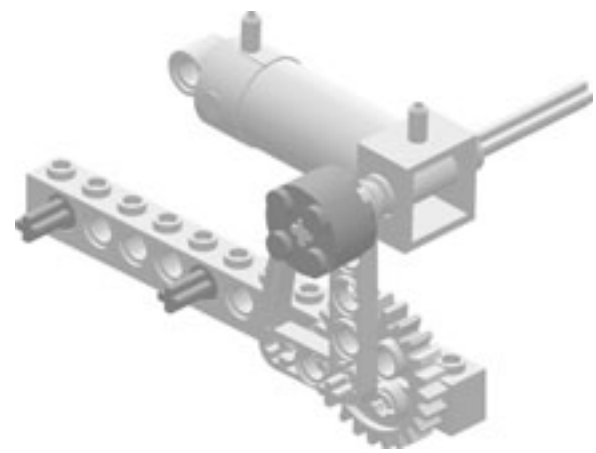
Right-hand Memory Step 2



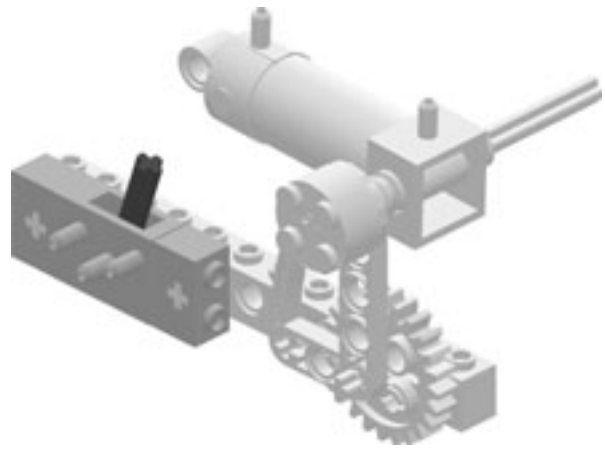
Right-hand Memory Step 3



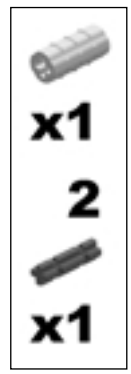
Right-hand Memory Step 4

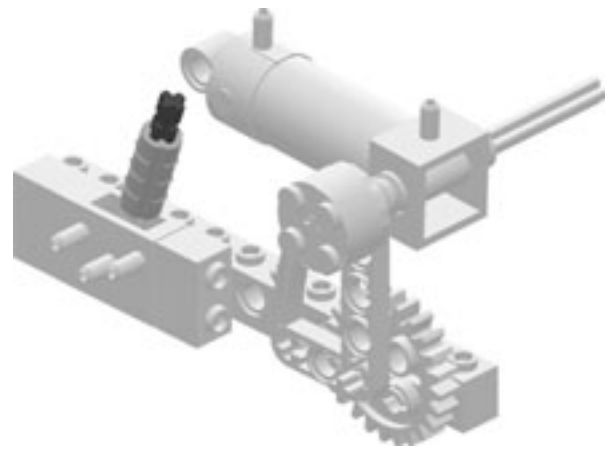


Right-hand Memory Step 5

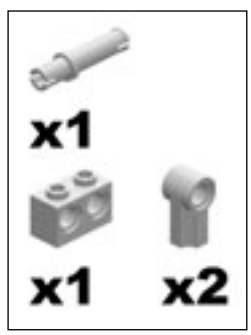


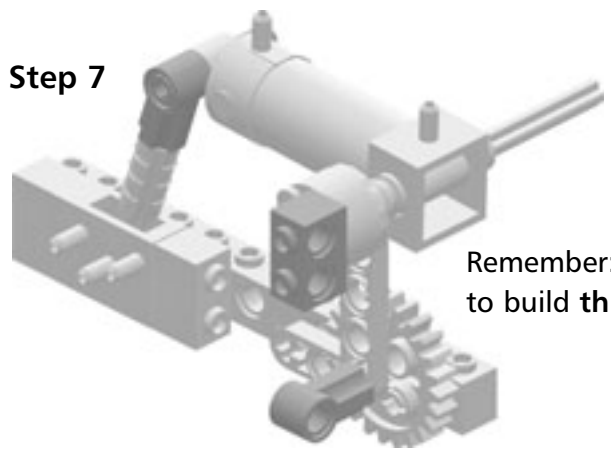
Right-hand Memory Step 6





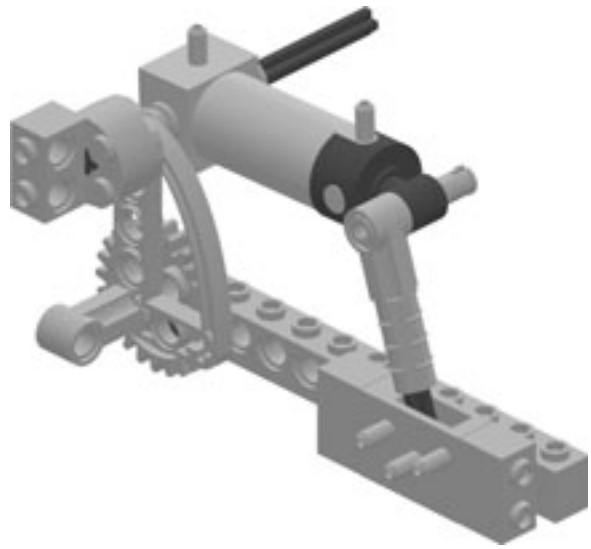
Right-hand Memory Step 7





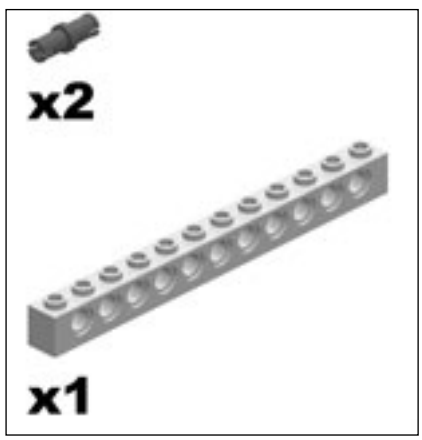
Remember: you will need to build **three** of these.

The Left-hand Memory

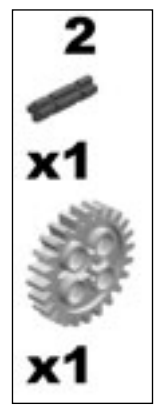


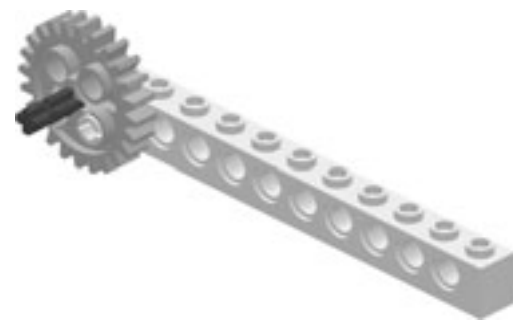
The Left-hand Memory sub-assembly is built on the opposite end of the TECHNIC brick than the Right-hand Memory sub-assemblies. Therefore, if you run into any problems during this next series of steps, refer to the corresponding step in the Right-hand Memory sub-assembly building instructions for a different point of view. You will need to build **three** of these constructions.

Left-hand Memory Step 0

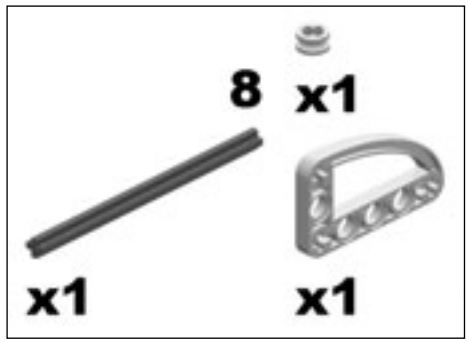


Left-hand Memory Step 1

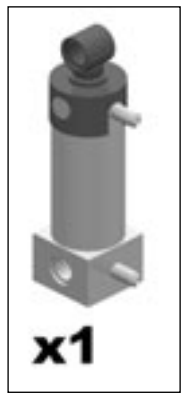


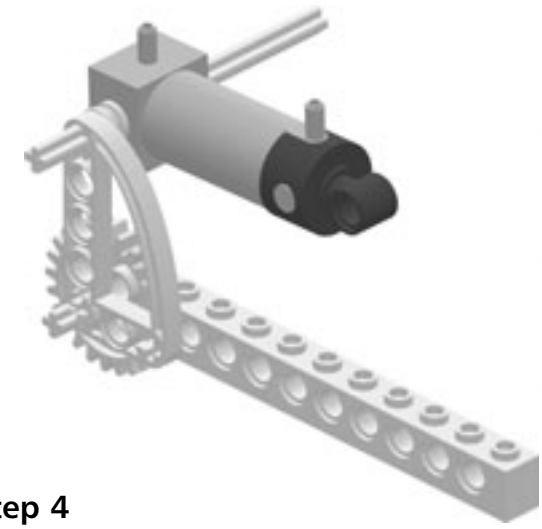


Left-hand Memory Step 2

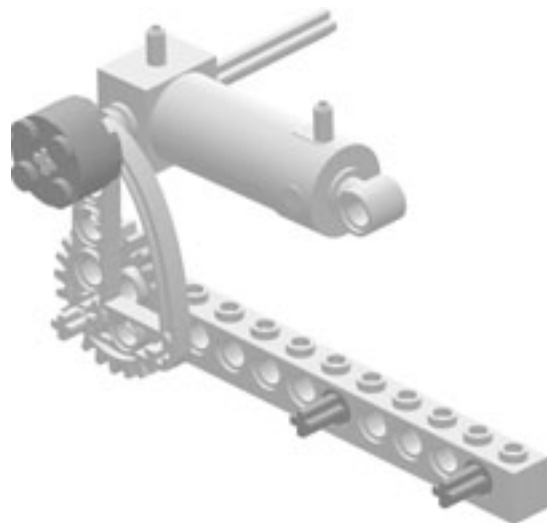


Left-hand Memory Step 3

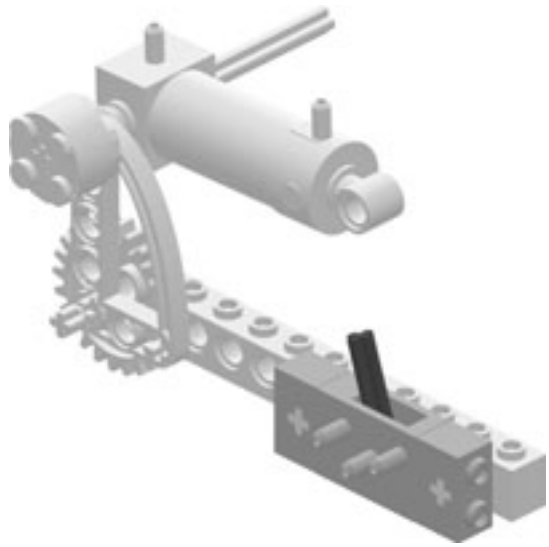




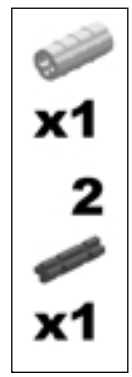
Left-hand Memory Step 4

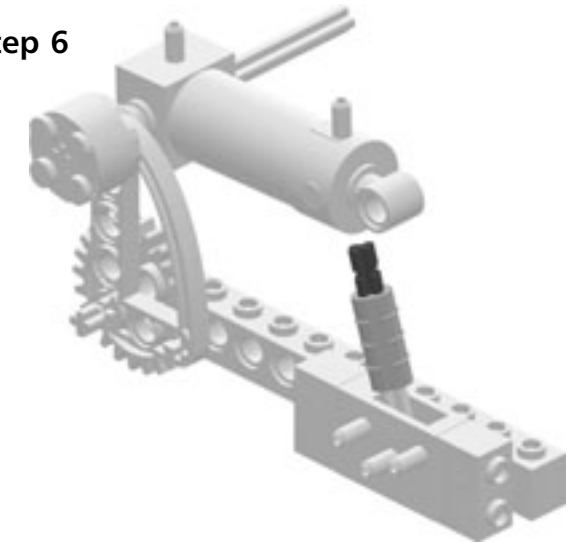


Left-hand Memory Step 5

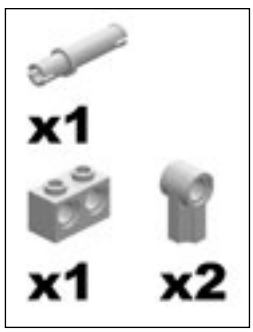


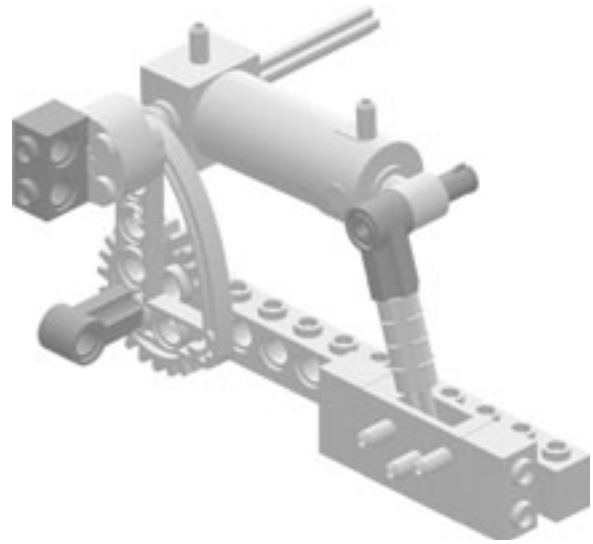
Left-hand Memory Step 6



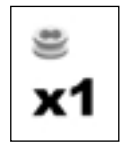


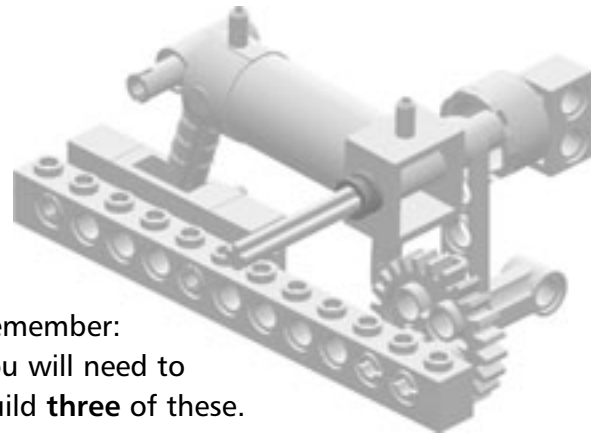
Left-hand Memory Step 7





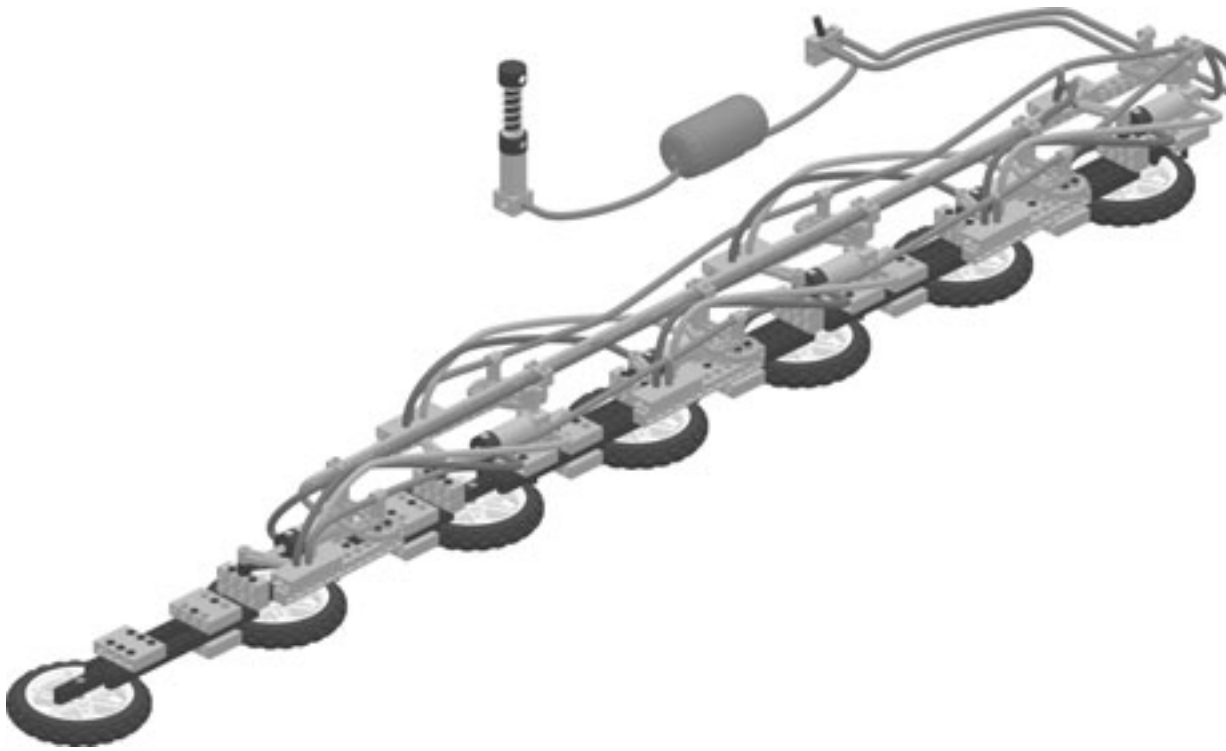
Left-hand Memory Step 8





Remember:
you will need to
build **three** of these.

Putting it All Together



At this point, you have created all the sub-assemblies needed to complete the assembly of Synchropillar. Synchropillar's overall design is a linear combination of a Front Foot sub-assembly, Right-Hand Memory sub-assembly, Middle Foot sub-assembly, Left-Hand Memory sub-assembly, and so forth.

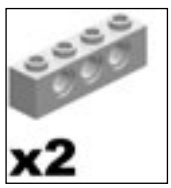
When you assembled all the memories, the instructions had you assemble the memories with the pistons contracted, but according to Table 5.1 that describes the pneumatic sequence of Synchropillar's circuit, two of the memories need to start in the expanded position. In the final assembly, I've arbitrarily chosen to have the first two pistons (counting from the front) expanded, and the remaining pistons contracted. I could just as easily chosen the middle two pistons, or the last two pistons.

You can either expand the memories before you assemble Synchropillar (as I did in the Final steps), or wait until Synchropillar is completely assembled, and then expand the first two memories.

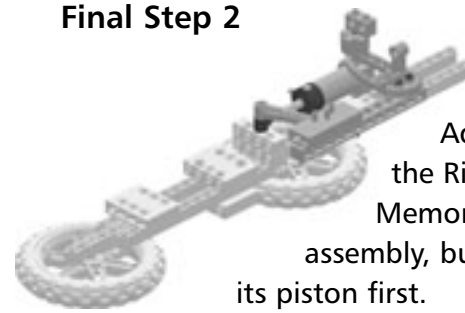
Final Step 0

Locate the Front Foot sub-assembly and one of the Middle Foot sub-assemblies. Add the pins to the Middle Foot Sub-assembly before connecting these two sub-assemblies as shown.

Final Step 1

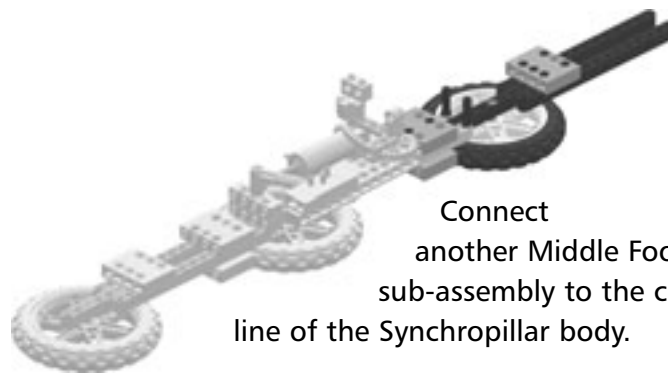


Final Step 2



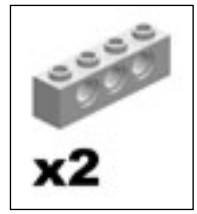
Add one of the Right-hand Memory sub-assembly, but expand its piston first.

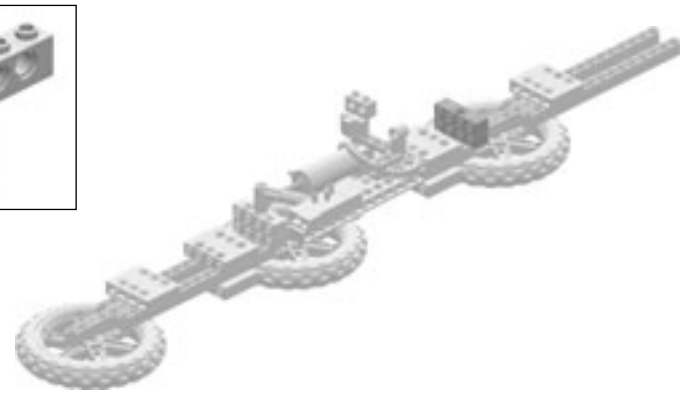
Final Step 3



Connect another Middle Foot sub-assembly to the central line of the Synchropillar body.

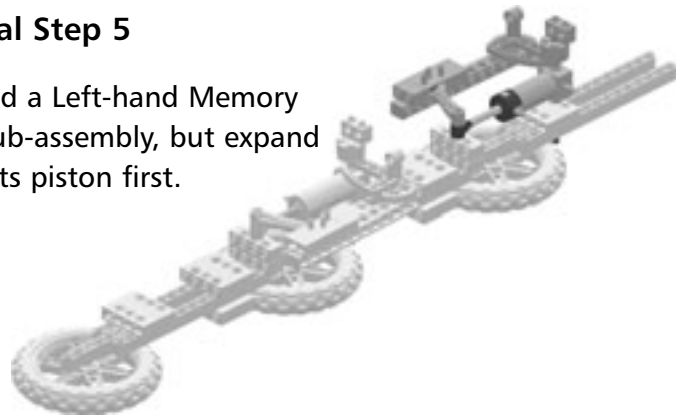
Final Step 4



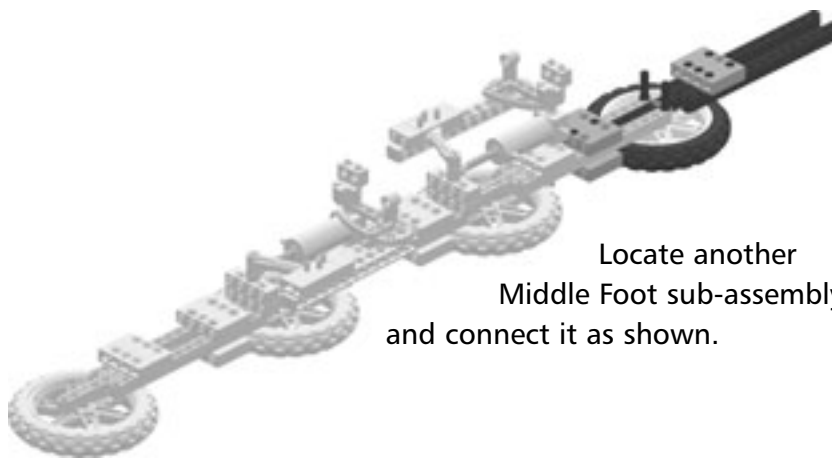


Final Step 5

Add a Left-hand Memory sub-assembly, but expand its piston first.

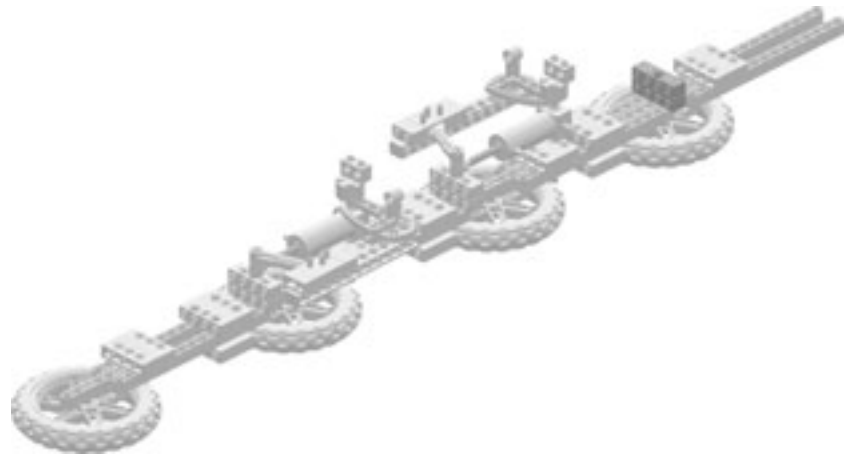


Final Step 6

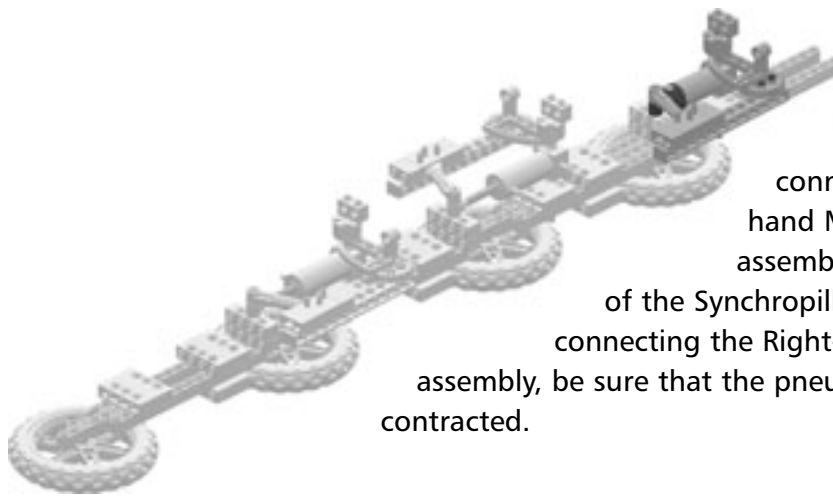


Locate another Middle Foot sub-assembly and connect it as shown.

Final Step 7



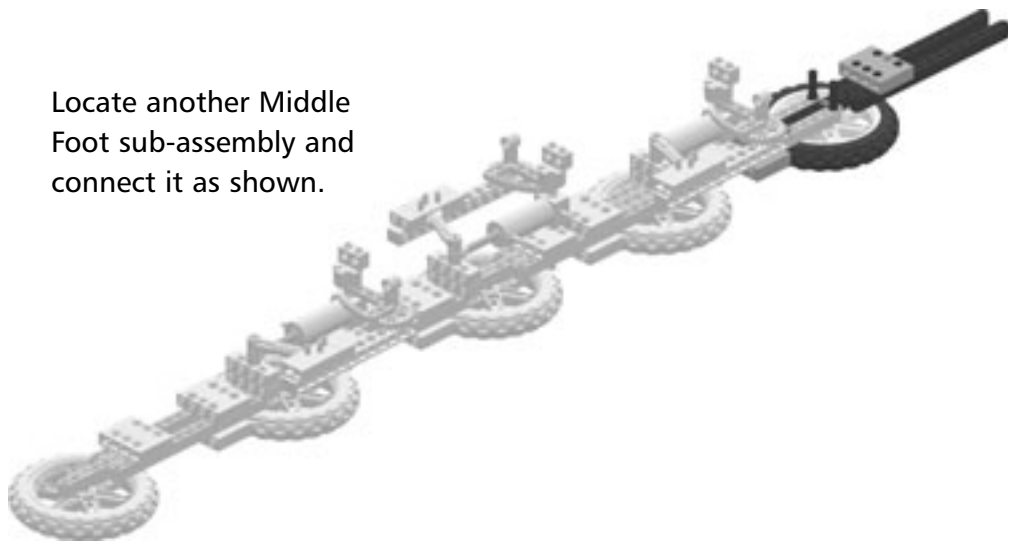
Final Step 8



In this step, we will connect a second Right-hand Memory sub-assembly to the central line of the Synchropillar body. When connecting the Right-hand Memory sub-assembly, be sure that the pneumatic piston is contracted.

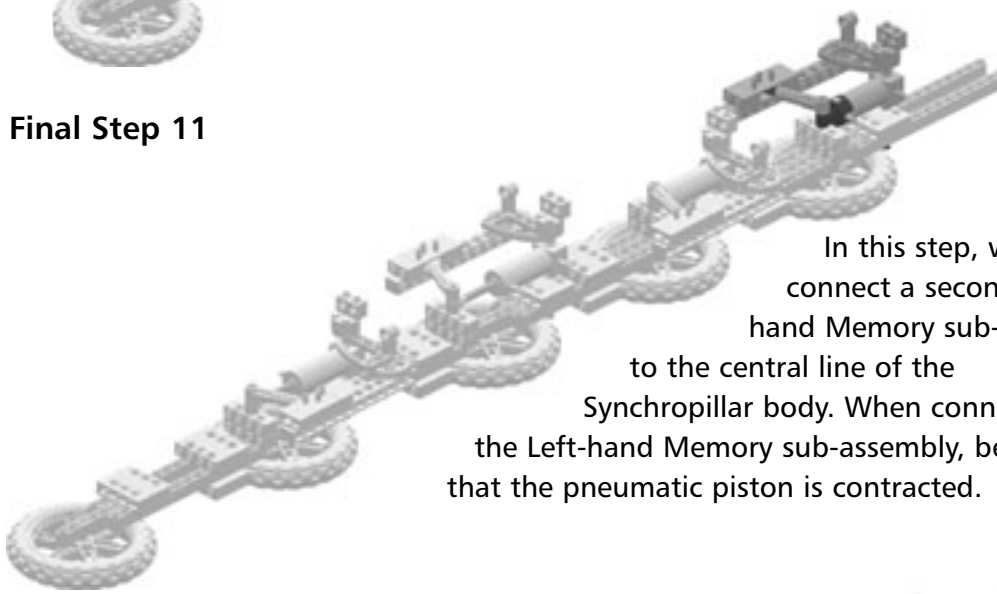
Final Step 9

Locate another Middle Foot sub-assembly and connect it as shown.



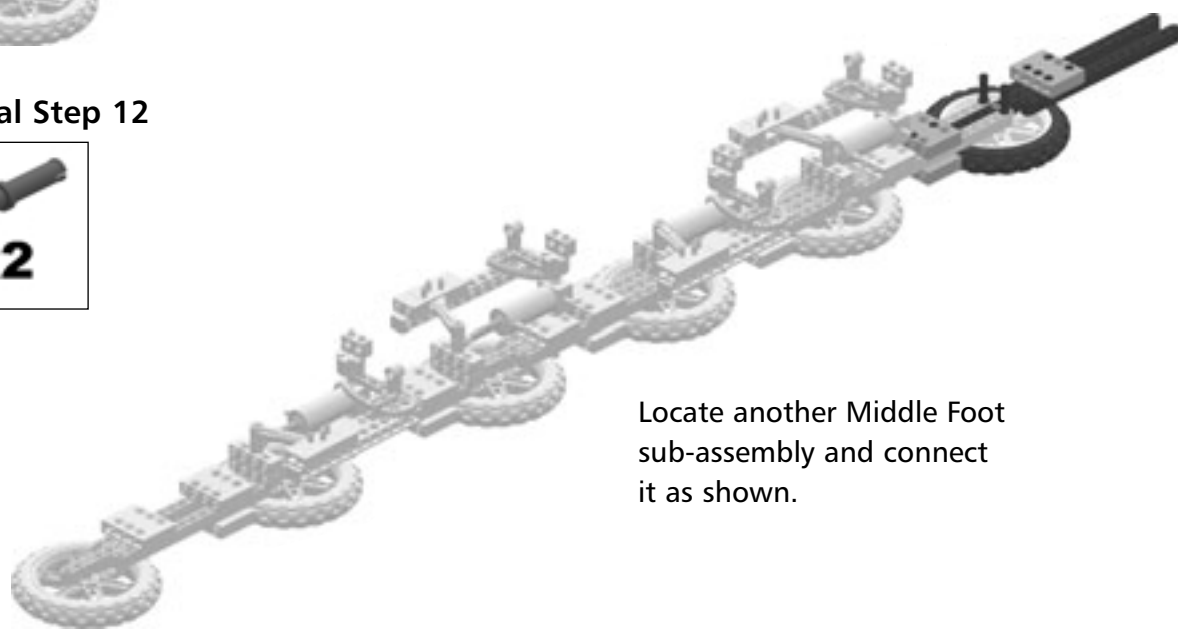
Final Step 10

Final Step 11



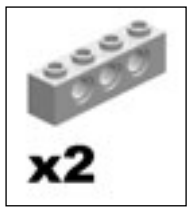
In this step, we will connect a second Left-hand Memory sub-assembly to the central line of the Synchropillar body. When connecting the Left-hand Memory sub-assembly, be sure that the pneumatic piston is contracted.

Final Step 12

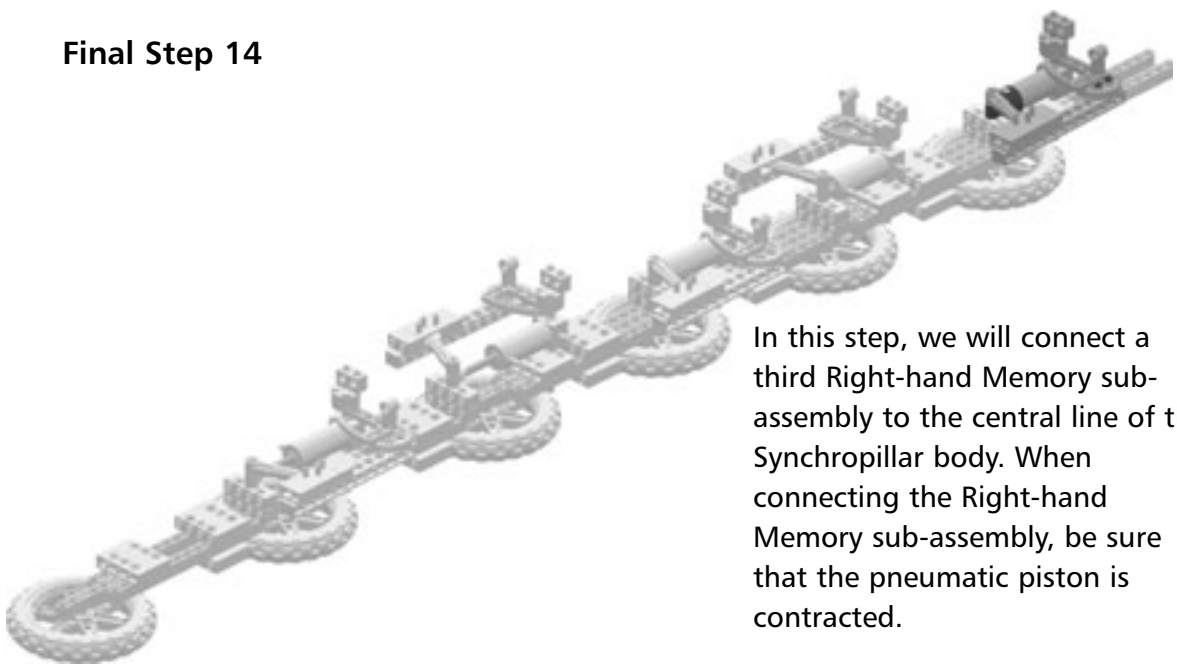


Locate another Middle Foot sub-assembly and connect it as shown.

Final Step 13



Final Step 14

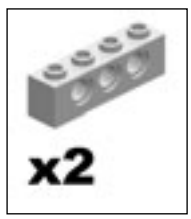


In this step, we will connect a third Right-hand Memory sub-assembly to the central line of the Synchropillar body. When connecting the Right-hand Memory sub-assembly, be sure that the pneumatic piston is contracted.

Final Step 15

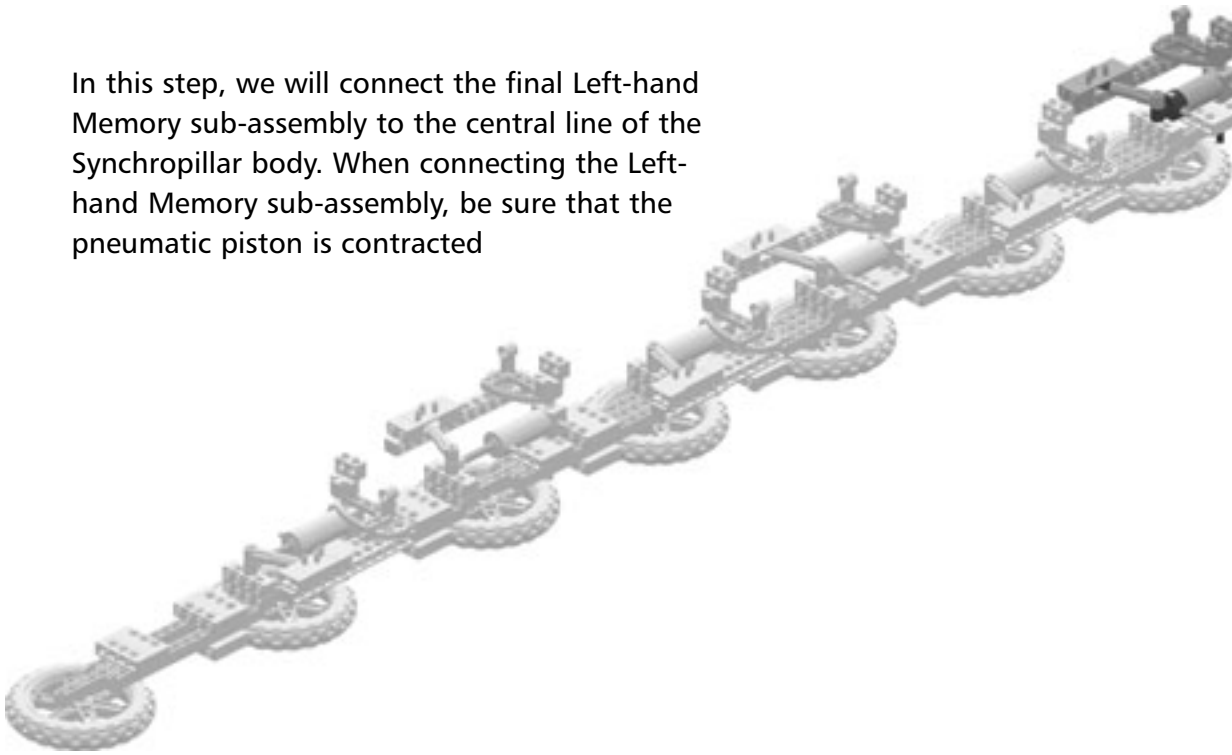
Locate
the Back Foot sub-assembly,
and connect it as
shown.

Final Step 16

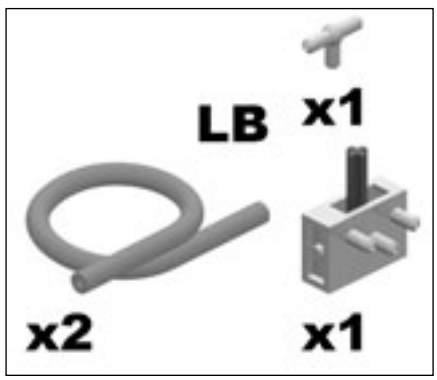


Final Step 17

In this step, we will connect the final Left-hand Memory sub-assembly to the central line of the Synchropillar body. When connecting the Left-hand Memory sub-assembly, be sure that the pneumatic piston is contracted



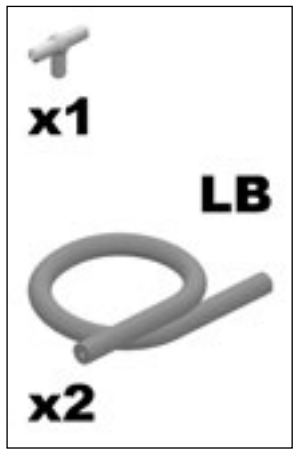
Final Step 18



Use light blue pneumatic tubing to connect the pneumatic switch and the T-connector as shown.

Use the second piece of shorter light blue pneumatic tubing to connect the T-connector to the Left-hand Memory sub-assembly as shown.

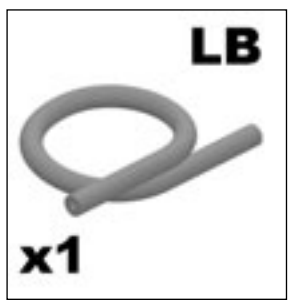
Final Step 19



Use another piece of light blue tubing to connect the T-Connecter (from **Final Step 18**), to a second T-Connector as shown.

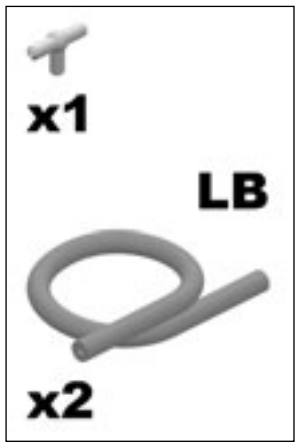
Using a second piece of shorter light blue tubing, connect the T-Connector and the middle Left-hand Memory sub-assembly.

Final Step 20



Connect the T-Connector from the middle Right-hand sub-assembly to the pneumatic port on the front Left-hand Memory sub-assembly using another piece of light blue tubing.

Final Step 21

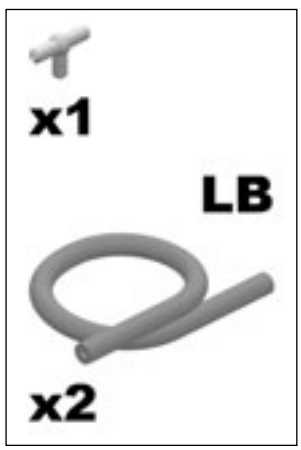


Connect a piece of light blue tubing to the other port of the pneumatic switch, and run the tubing to a T-connector as shown.

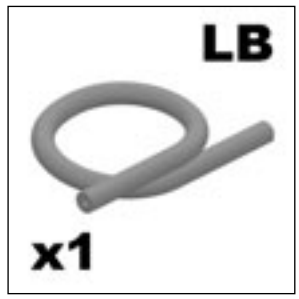
Connect the T-Connector to the rear Right-hand Memory sub-assembly as shown.

In the next few steps, you will connect the Right-hand Memory sub-assemblies in the same way that the Left-hand Memory sub-assemblies were linked together.

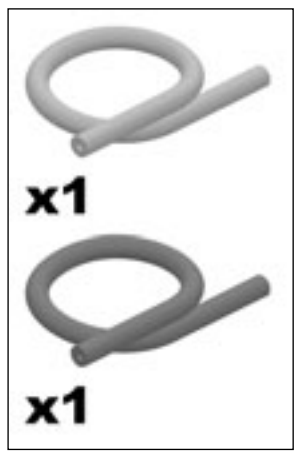
Final Step 22



Final Step 23

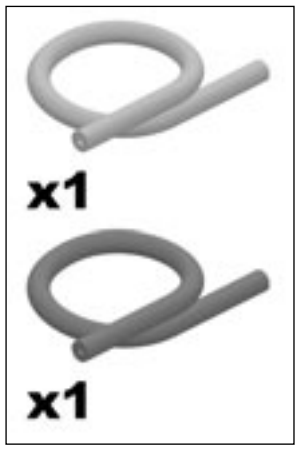


Final Step 24



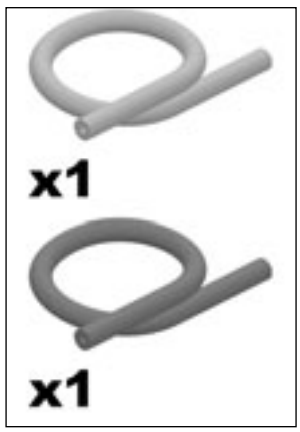
Use a piece of light gray and dark gray tubing to connect the pneumatic switch to the pneumatic piston as shown.

Final Step 25



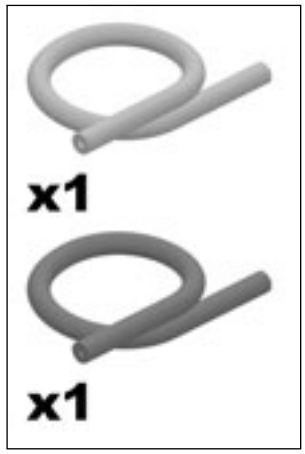
Connect the light gray and dark gray tubing as shown.

Final Step 26



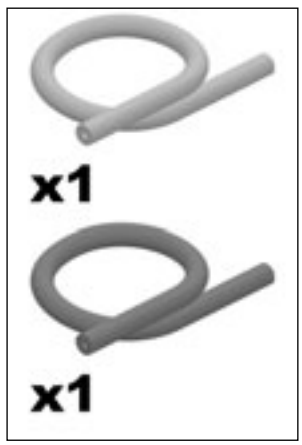
Connect the light gray and dark gray tubing as shown.

Final Step 27



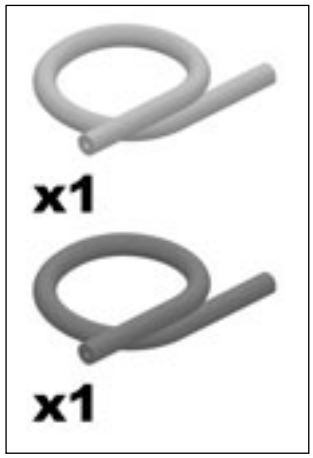
Connect the light gray and dark gray tubing as shown.

Final Step 28

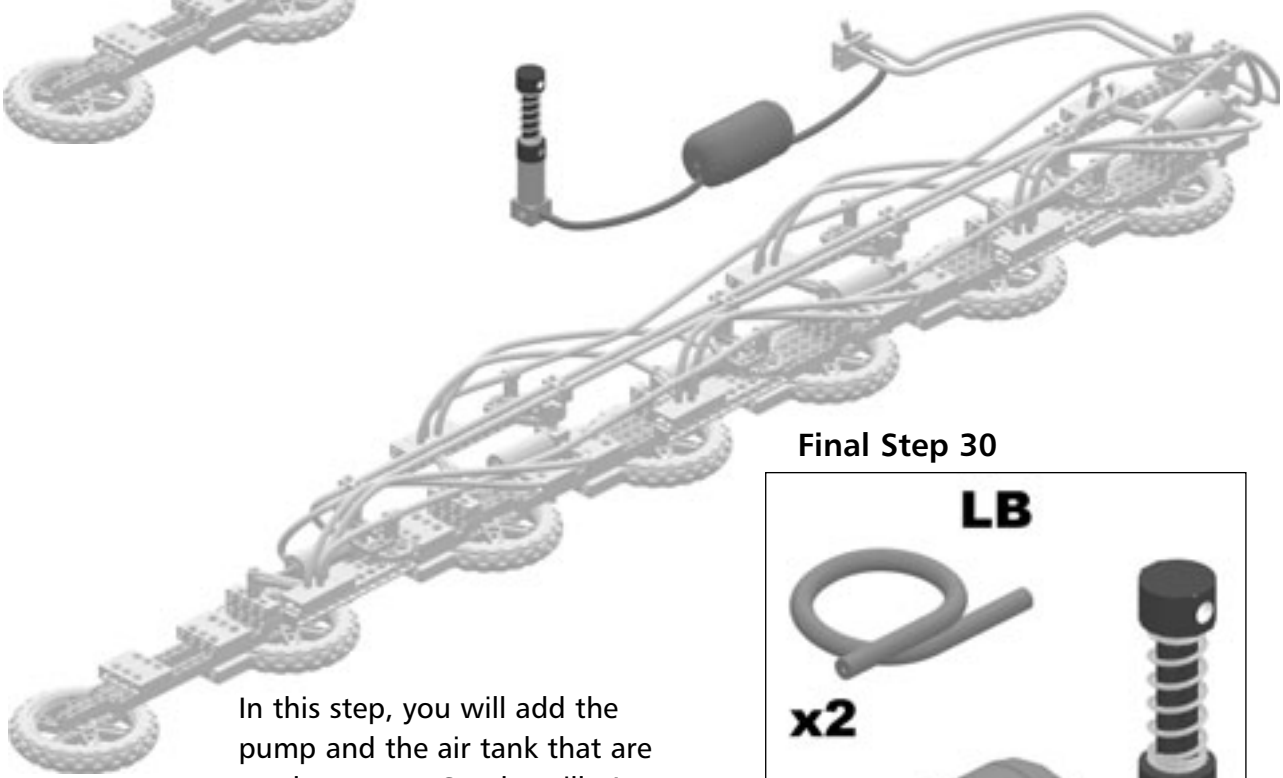


Connect the light gray and dark gray tubing as shown.

Final Step 29



Connect the light gray and dark gray tubing as shown.



Final Step 30

In this step, you will add the pump and the air tank that are used to power Synchropillar's movements. Connect the tubing, large pneumatic pump, and pressure tank as shown.

Experimenting with Synchropillar

To make Synchropillar walk, pump the large pneumatic pump with your hand to build up pressure in the blue pressure tank. Flip the pneumatic switch attached to the pressure tank, and watch Synchropillar's feet move! By repeating the pumping and flipping the pneumatic switch actions, you will be able to see Synchropillar cycle through all four states of the pneumatic timing circuit.

Notice that if you continuously pump the large pneumatic pump, but do not flip the pneumatic switch, you will feel a lot of pressure in the pressure tank (the pump will become rigid and it will take more effort from you to depress the pump plunger). When you flip the switch hooked to the pressure tank, the feet will move very quickly. If you repeatedly flip the switch back and forth without pumping, the feet will move more slowly, and eventually stop. This is attributed to the fact that when you flip the switch, it releases pressure on half the pistons, the pressure that remains in the pressure tank will decrease with each expansion and contraction action that causes the pneumatic pistons to change states and move the feet.

Pump the pneumatic pump, and flip the switch attached to the pump, until Synchropillar gets back to the start of sequence where the front two pistons are expanded and the remaining pistons are contracted. Now pump the pump and flip the switch back and forth more, and count how many times you have to flip the switch to make Synchropillar restart the sequence. It should take six flips of the switch to go through the complete sequence.

You can also try a different experiment. Reverse the connections of the hoses on the front piston, so the light gray hose is connected to the black base piston, and the dark gray hose is connected to the yellow top of the piston. Contract all the pistons. You have now created a six-piston timing circuit similar to the two-stage timing circuit in Figure 5.5. Synchropillar won't walk as well, but will still be interesting to watch. After you pump the pneumatic pump and flip the pneumatic switch attached to the pressure tank, count how long it takes Synchropillar to get back to its initial state of all pistons contracted. It should take 12 flips of the switch. This is twice as long as the sequence you counted before you switched the hoses on the first piston.

Summary

Einstein said, "Make everything as simple as possible, but not simpler." I believe I have achieved the ultimate simplicity in this design by inventing an advanced technique for creating synchronous LEGO pneumatic designs.

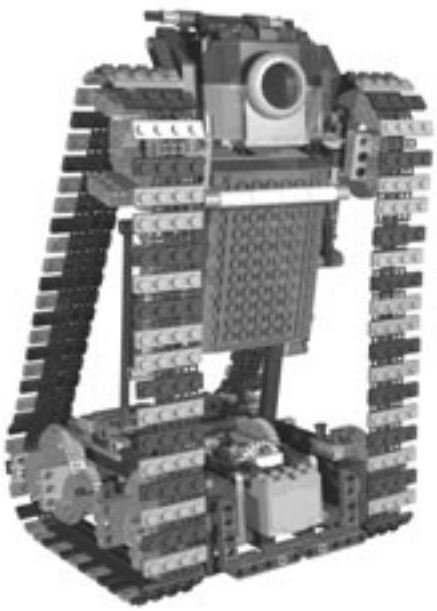
The Synchropillar's design centers on the ability to control pneumatic switches with pneumatic pistons. To reiterate, by combining pneumatic piston/switch pairs into a larger circuit, we can create a pneumatic timing/feedback circuit that repeats sequences of patterns. The first example of pneumatic feedback circuit within this chapter was the timing circuit made built with only two pneumatic pistons (Figure 5.3). We learned that if we built two of these pneumatic timing circuits, then controlled the pressure released into the circuits using the same pressure source, nothing would keep the two timers coordinated with each other over time. The term to describe this inevitability of the circuits becoming unsynchronized is the core feature of an asynchronous circuit—because we cannot control how rapidly they change state.

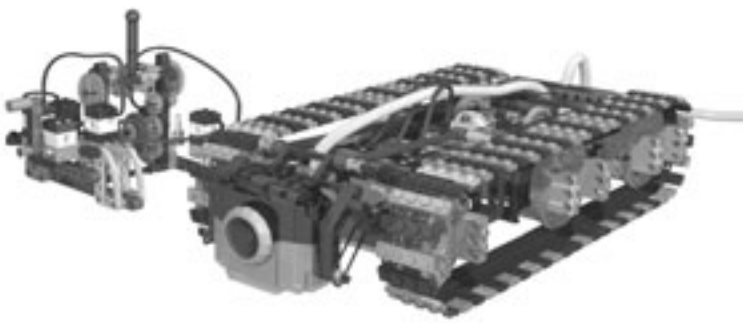
By modifying the pneumatic timing circuit, we created a synchronous design, where alternating pressure between the pneumatic pistons causes a change in one piston, but does not affect the next piston—until pressure is applied to that next piston directly. This synchronous design concept is not limited to the Synchropillar, as it can be used in a variety of other robots of your own creation.

Masterpiece 6

The Shape-Shifting Camera Tank (SSCT)

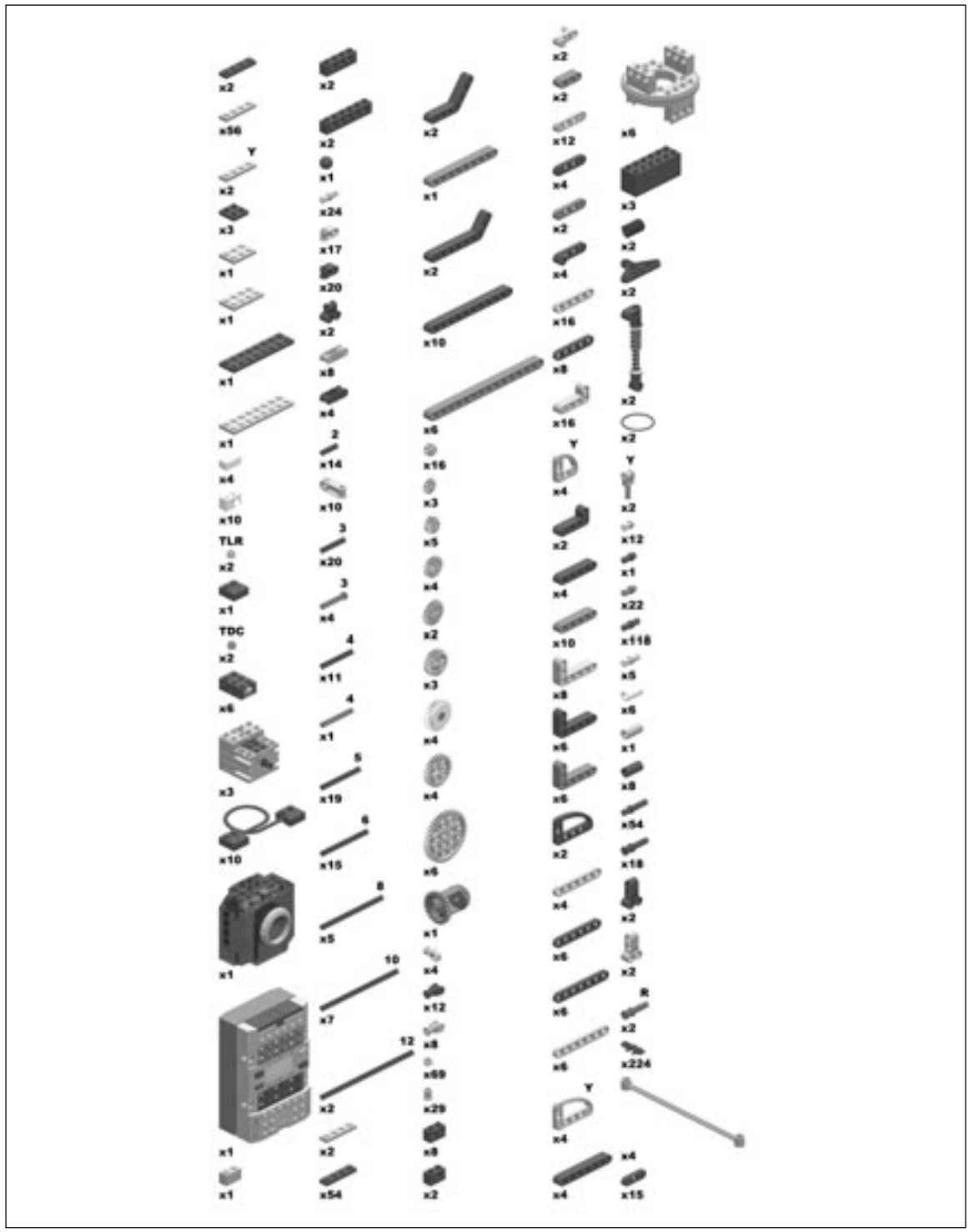
by Miguel Agulló





Bill of Materials

These are the parts you will need to build the SSCT as shown.



Introduction

The Shape-Shifting Camera Tank (SSCT) is a very interesting piece of machinery: a tracked vehicle that can change its shape, a foldable tank of sorts. The SSCT can lay low and flat, and thus creep into tight spaces, but in this position it gives up maneuverability, as it requires a quite a bit of space to turn. By raising the end that carries the camera, it can greatly reduce its footprint to less than half. With less of its track in contact with the ground, the SSCT can almost turn in place. At the same time, it gains height, allowing the camera to peer over obstacles on its path. The SSCT can adopt any of the intermediate positions between these two extremes, adapting its shape to the terrain around it.

The overall idea is brilliant: pushing the envelope of caterpillar-style vehicles (first seen in Masterpiece 5) and one step closer to actual caterpillars, which can raise part of their tracks off the ground. The SSCT is based on a real-life search-and-explore robot, the Variable Geometry Tracked Vehicle (VGTV). The VGTV is a search-and-rescue robot deployed, among other places, at the World Trade Center disaster site in New York City. The SSCT is a LEGO rendition of such an apparatus. The VGTV is one of several products produced by Inuktun Sevices Ltd. (www.inuktun.com), a company whose product catalog any reader of this book should consult at some point. Inuktun produces many intriguing real-life robotic products that I am sure will inspire every single LEGO MINDSTORMS aficionado. There is always something to learn from the professionals, and Inuktun's products offer a glimpse into many inspiring machines and the field of applied professional robotics.

NOTE

For movies of the VGTV in action we recommend that you visit the following URL: www.csee.usf.edu/robotics/USAR/RobotInformation/InuktunVGTV.htm.

LEGO renditions come in many flavors. The first time I saw a picture of the real VGTV, I knew I wanted to build a LEGO masterpiece replicating its basic functionality. The concept of building a tank-like vehicle that was not taller than its tracks, and could carry an RCX and a camera on a shape-shifting chassis was intriguing. I didn't necessarily set out to replicate the exact mechanisms or the appearance of Inuktun's robot, but I soon found out that these simple specs translated incredibly well into LEGO MINDSTORMS elements. The RCX has three current outputs, which happen to match the total number of motors that the SSCT needs to operate; and most importantly, the vehicle integrates a camera at no extra cost to the motor, greatly enhancing the Vision Command kit.

While it took me several months of intense work (as well as many others full of playful consideration) to put together the SSCT model presented in these pages, it is really just a starting point. Perhaps the first lesson to learn from the SSCT is that this is a great way to introduce robot builders to the mechanical design of vehicles. For instance,

the SSCT is not far, in scale, from the actual VGTV, and thus many of the challenges posed by its construction are equally realistic(that is, it should be able to cope with at least some types of real-world terrain). I have chosen a way to recreate this vehicle that explores certain LEGO and non-LEGO design laws and philosophies, but what is really important is the myriad of possibilities and variations hidden inside each of the different components and in the overall design.

Rather than just a vehicle, I see the SSCT as a *platform*. The simple idea of a six-wheeled tank that can change shape is significantly more important than specific implementation of a camera carrying motorized vehicle. In this chapter, I will guide the reader through several important details, and the step-by-step instructions will allow any builder to recreate my particular version of the SSCT. Ultimately, the key message is that it is possible to build the SSCT in a variety of different ways. At the end of the chapter, you will find a handful of suggestions for alternative designs and features.

NOTE

If you do intend to build a complete SSCT, you should make a point of becoming acquainted with the real VGTV, since the SSCT uses some very different approaches to perform the same tasks. One of my main priorities was to make the design as compact and modular as possible. Replicating the exact VGTV mechanisms in LEGO would have probably meant ending up with a bulkier, less manageable model.

How it Works

Robotics is still considered a futuristic science; even hobbyists often overlook the commercial fabric underlying the discipline. But the fact of the matter is, while we ponder how to improve our line followers, there are plenty of companies out there trying to make a profit by *selling* robots. Some robotic diehards will roll their eyes and point out that what these companies create and sell are not autonomous machines, and thus they do not qualify as robots. To me, the emphasis here is on *real world*. When built, the SSCT is a cool toy that allows us a peek into a very practical application of real life robotics: *telepresence*.

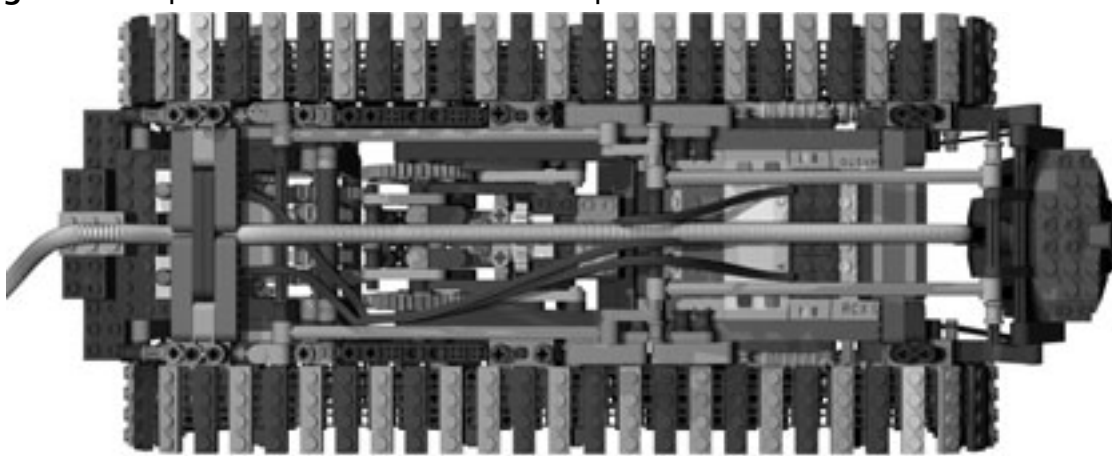
Introducing Telepresence

Telepresence is a multidisciplinary technology that has been quietly gaining popularity for a long time. Models similar to Hideaki Yabuki's CyberArm IV (Chapter 7) or the SSCT point to the developments that hold more interest to robot builders: developing remote controlled machines with the capacity to interact with their immediate environment with as much flexibility as possible. Think of the surgical operations where the patient is in Switzerland and the surgeon operates in real time from California. Think of the deep-sea

diving devices that provided viewers with images of the sunken Titanic. Think of the last mission to the surface of Mars relayed to the Internet almost in real time. All of these creations operated on the principle of telepresence. The trick to telepresence is balancing the capabilities of our creations to the environment they operate in. We want to equip our creations with as much power and intelligence as possible, yet not encumber them to the point of becoming unusable in real (and remote) environments. This is the philosophy behind the SSCT: to cram as much MINDSTORMS power as possible into the smallest shape-shifting chassis allowable.

A project of this scope requires a modular approach to achieve a successful end. Although the layout in Figure 6.1 might look tight, it can be easily dismembered into several sub-assemblies that connect together in a few key points. Through a careful division of functions among those sub-assemblies, the SSCT allows for the easy extraction of any of its component for redesign or replacement.

Figure 6.1 Top View of the SSCT: Not Much Space Left!



For now, let's consider some of the overall characteristics of the SSCT as a LEGO MINDSTORMS model.

As an RCX-carrying vehicle, the SSCT efficiently uses the three Output Ports of the RCX. Two Output Ports connect to a pair of motors that couple together to propel the vehicle backward or forwards forcefully, but disengage to steer the SSCT using tank-style skid steering (moving the tracks in opposite directions). The remaining output connects to a third motor that lifts or lowers the front end of the tank, changing the shape of the tracks and the vehicle contained within them. Adding another machine, the Vision Command camera, to this set-up offers extra functionality without tying up precious resources. Since the camera is integrated in the shape-shifting vehicle, it has three degrees of freedom:

- It can move up and down.
- It can move right and left.
- It can move forward and backward.

Incidentally, the camera element is one of the reasons why I set out to build the SSCT. While many outstanding creations that combine the RCX with the Vision Command camera, there is a noticeable absence of vehicles that incorporate a camera.

NOTE

For a great example of a robot that includes a static Vision Command Camera, I recommend that you review the LEGO Rubik Cube solver J.P. Brown. Images, instructions and commentary for this robot can be found at: http://jpbrown.i8.com/cubesolver.html.

One of the main impediments to using the Vision Command camera in a moving vehicle is that it is a tethered camera that needs to be hooked to a PC computer to work. This presents more challenges for a moving model than for a stationary unit. Luckily, the SSCT is a device designed precisely to relay information about an inaccessible environment to a remote command post. Our job is to make sure the explorer gets there and back, and helps us as much as possible throughout the process.

The actual VGTV is a tethered robot, which arguably serves its purpose well. As a search and rescue robot, one of its requirements is that it should cope successfully with unexpected situations. If you are releasing a robot into unknown crevices that might or might not contain disaster survivors, you want to make sure you can retrieve the device effectively in order to explore as many crevices as possible. Ideally, we would have a multitude of cheap, deep-burrowing, life-sniffing robots that allowed rescue crews to quickly pinpoint life signals over a wide area. For now, we must deal with today's disasters with today's tools—and the VGTV dragging a cable behind it as it transmits visual or other type of signals from deep into unknown territory is one of the best we have.

NOTE

The passionate essay, "Rats, Robots and Rescue," offers insight into the role of robotic telepresence in rescue operations. Written by Robin R. Murphy, the director of the Center for Robot-Assisted Search and Rescue (CRASAR™) of the University of South Florida, it can be found at the following URL: www.csee.usf.edu/robotics/crasar/roborats.html.

Apart from integrating three motors linked to the three output ports of the RCX and a Vision Command camera, the SSCT model has some other noteworthy LEGO characteristics. If you look closely at the Bill of Materials for the model, you will notice how the model uses very few classic LEGO TECHNIC bricks (the kind that have studs on

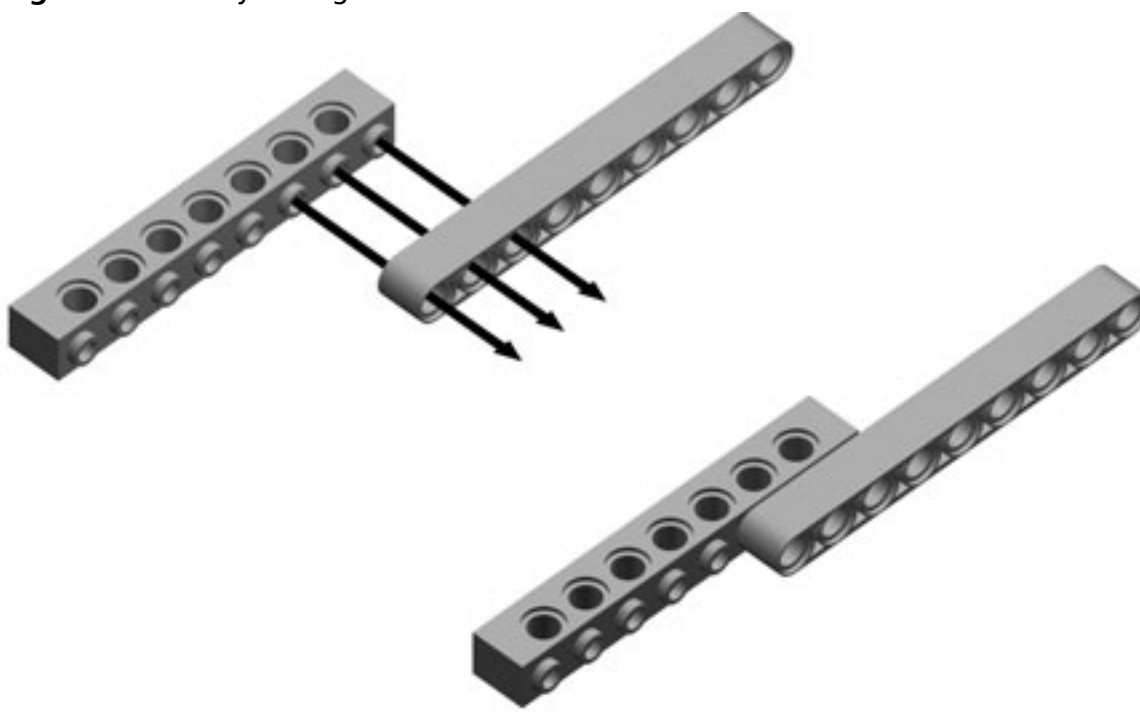
top, sockets at the bottom, and holes in the middle). Instead, it uses lots of the newer liftarm type of parts. Figure 6.2 shows both of these parts.

Figure 6.2 A Classic TECHNIC Brick (Left) and a Newer Type of Liftarm (Right)

The classic TECHNIC brick is a winner when it comes to building simple and solid structures. Most large official LEGO TECHNIC Vehicle kits are built around a chassis made out of this type of bricks. On the other hand, LEGO has been employing more and more of the liftarm elements in their releases (as well as developing lots of new part shapes based around it). These newer elements, which are very compatible with the older type of bricks, allow greater building flexibility, but at the cost of more complex building techniques.

There is a bit of a learning curve involved in changing between building only with TECHNIC bricks and using liftarm elements. Mainly, the familiar and reassuring stud-and-socket attachment is now gone… or is it? Actually, Figure 6.3 shows one of the most unusual yet strong attachments between LEGO elements.

Figure 6.3 A Very Strong Attachment!



Building with liftarms requires extra elements such as pins, connectors, axles, and so forth to simply connect *two* liftarms together. Therefore, this method of building is not as straightforward as simply stacking bricks. On the other hand, constructions built with liftarms tend to be lighter in weight and often smaller than similar ones made out of classic TECHNIC bricks. An additional advantage of the liftarm part family is that it comes in two widths, which allows builders to scale down the subcomponents of their robots. So, while it is harder (but not impossible) to design a solid structure using liftarms, the advantage is that such structures are lighter and attach to one another in a variety of compact ways, offering great advantages for model construction.

With all this in mind, I advise you to take another closer look at the Bill of Materials for the SSCT that can be seen at the beginning of the chapter. Notice how the model uses 118 friction pins and 54 long friction pins! I have divided the building instructions for the SSCT into individual sections for each sub-assembly. This way, even if you do not intend to build the whole vehicle, you can experiment building some of the sub-assemblies with the purpose, perhaps by attaching them to other designs. Each section also includes the techniques and science applied in each of the components, so you can also create your own experiments based on them. As you saw above with the liftarms example, I have also included suggestions on alternative parts to use whenever possible.

Moving and Balancing the Beast: The Propulsion Unit

This first sub-assembly is what makes the SSCT a vehicle: a propulsion unit. Tracked vehicles are confined to using one type of steering, specifically skid steering. Since the tracks are affixed along the side of the vehicle, the only way to turn the vehicle is to stop one track while the other one continues moving. A possible variation is to reverse the movement on each track, which further reduces the radius of the turn. Clearly, using a pair of electric LEGO motors and the RCX (or even a couple of LEGO TECHNIC electric pole reversers) driving and steering a tracked vehicle is a matter of switching the motors on and off in the correct combinations.

Ironically, many tank builders spend a lot of time figuring out how to make tanks go in a *straight line*. One disadvantage of dedicating a separate motor to each track is that motors never have the exact same speed of rotation. Which means that one track will eventually overcome the other and the vehicle will start to turn. (This concept was also covered in Masterpiece 5 Synchropillar.) A primitive method of avoiding this problem is powering both tracks with a single motor and steering the vehicle by disengaging one of the tracks from the power supply. A better method to avoid this problem is to use a dual-differential drive, also called an adder-subtractor mechanism. An adder-subtractor mechanism uses two differential drives attached to a pair of motors:

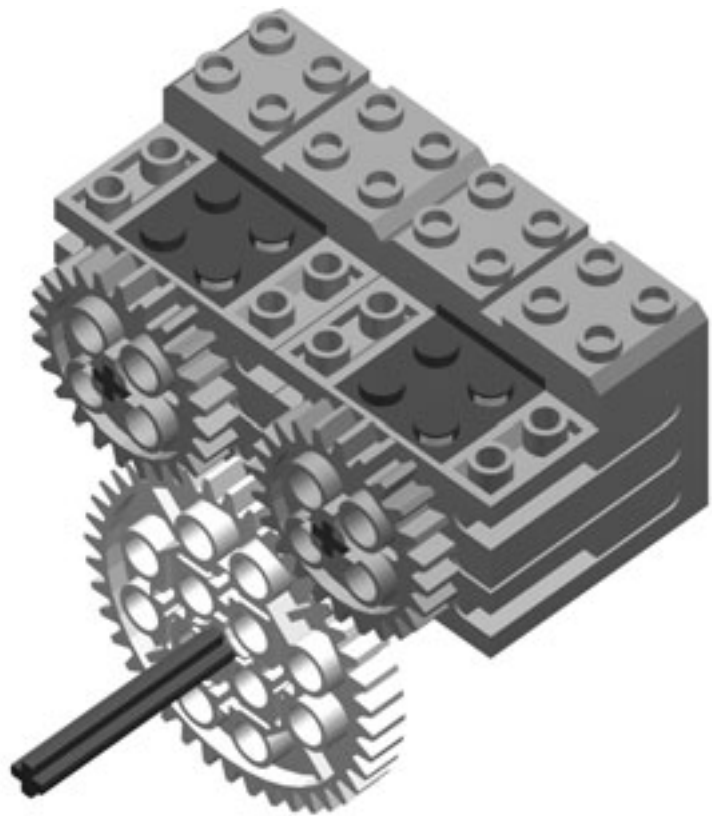
- One motor drives the vehicle backwards and forwards by moving both tracks in the same direction.
- The other motor moves them in opposite directions to turn the vehicle.

The book, *Building Robots with LEGO MINDSTORMS,* by Giulio and Mario Ferrari, (Syngress Publishing, ISBN: 1-928994-59-8), offers a very detailed explanation of this type of system. Later in this chapter, we will use some of the properties of the adder-subtractor mechanism when assembling the differential gears in the Joystick sub-assembly.

The main disadvantage of a dual-differential drive for a vehicle like the SSCT is that one LEGO motor is hardly enough to move a relatively heavy load with any reasonable speed. The size and weight of the SSCT (over two pounds) place it almost beyond what is reasonable to expect from a child's toy (which was the initial intent of LEGO after all). I wanted a solution that used the combined power of both motors to forcefully drive and steer the SSCT.

The solution proposed here uses two clutch gears to couple or disengage the motor outputs, depending on the direction that they turn in. To explain the role of this mechanism in the SSCT, let's start with the basics. If we connect two electric motors to the same gear (Figure 6.4), they become mechanically coupled. As long as the motors are turning in the correct direction, they will share the load (even if they are connected to the same power source). This wonderfully simple, self-adapting mechanism of electrical motors comes in very handy when extra power is needed.

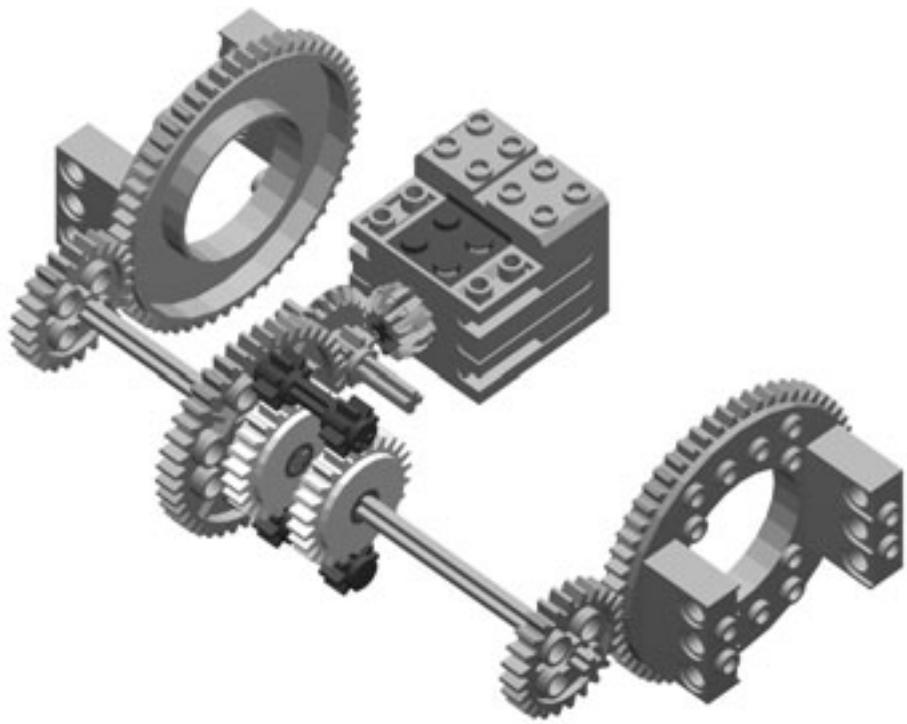
Figure 6.4 Two Motors Connected to the Same Gear become Mechanically Coupled



To power the SSCT, we will gear down the output from both motors and couple them together using two LEGO clutch gears. These gears remain fixed to their axle until a certain opposing force is applied; once the right pressure is achieved, they become loose from the axle. In other words, if both motors are turning in the same direction, they remain coupled and provide equal speed to both tracks, making the tank move in a straight line. However, if the motors rotate in opposite directions, the coupling breaks and they transfer the movement independently to each of the tracks, making the vehicle turn. Obviously, since the two motors have to be able to turn in opposite directions, they have to be connected to two different RCX outputs.

Figure 6.5 shows the gear train that drives the SSCT. Only one motor is shown, but the layout is symmetrical. Each motor supplies power to a gear train that ultimately transfers power to the tracks through turntables. Gear trains are also connected to each other via the parts colored in black (two sets of pairs of 8T gears mounted on a common axle). Since these are connected to the motor outputs via clutch gears (in white), the motor outputs will be coupled if both turn in the same direction. However, if they turn in opposite directions, they will overcome the resistance of the clutch gears and decouple. This does mean that part of the motor force is spent on countering the resistance of the clutch gears when decoupling the gear trains; then again—that is precisely the reason why we use two motors: to provide extra power, so the trade-off works in our favor.

Figure 6.5 Two Clutch Gears Connecting the Gear Trains



Balancing the SSCT

Remember that the reason why we need two motors to power the SSCT is primarily due to its weight. The vehicle is designed to carry an RCX loaded with batteries and a Vision Command camera, this combination weighs over one pound, they are also mounted on the foldable end of the robot. Because the vehicle changes shape and moves around, we need to carefully consider the weight distribution on the foldable end of the robot at all times or the vehicle might raise the wrong end!

Balancing loads on moving objects might seem like a complicated task for untrained eyes, but in the case of the SSCT, all we need to do is employ a simple and basic law of mechanics to quickly figure it out. When not completely flat, the SSCT balances its weight using the model of a first class lever. An old fashioned scale is a great example of the same type of levers. The load and effort are place at both ends, with the fulcrum, or axle in between them. The scale gives it all away: when placed at an equal distance, equal weights will balance themselves. If we move one of the weights closer to the fulcrum, the scale will tip to the other side. The actual formula is:

L/l=R/E

- L is the length of the effort arm (one arm of the scale).
- *l* is the length of the resistance arm (the other arm).
- R is the resistance (weight or force applied on one arm).
- E is the effort (weight or force applied on the other arm).

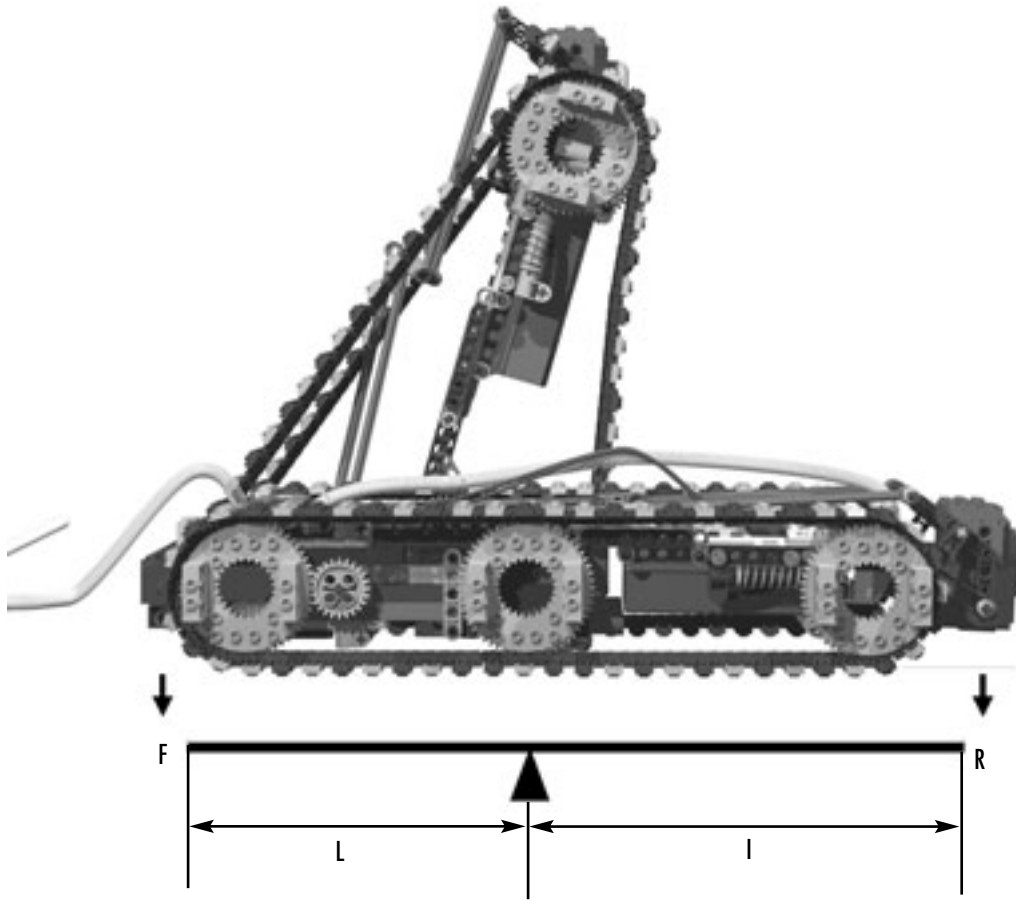
To calculate the weight needed to keep the tracks at the back of the vehicle in contact with the ground at all times, we need to find out how much weight to apply on this end of the SSCT and where to apply it. What the previous formula states is that the difference between the lengths of the arms is our ratio, or mechanical advantage. Thus, a tuned scale has a mechanical advantage of zero. As the location of the resistance varies, the force required to overcome it also changes. How does this apply to the SSCT? Let's look at Figure 6.6.

When we raise the camera end of the SSCT, its weight exerts a downward force that, unless it is countered, can force the SSCT to raise its bottom rather than its head. As we will see in the next section, the SSCT has a mechanism that simply pulls on the camera end to lift it, but that mechanism itself does not ensure that the front end will rise. To make sure that the correct end raises, we need to make certain that when the SSCT starts balancing its weight in the first class lever fashion illustrated in Figure 6.6, the weight of the bottom end overcomes the weight of the camera end at all times. Keep in mind that L and l are measured as a line extending from the fulcrum point in perpendicular lines that represent effort, resistance, or simply weight. In Figure 6.6, we have sketched a parallel line below the imaginary one, which works just as well.

What this means is the higher the camera end is raised, the closer it will be to the fulcrum, and thus the resistance required to overcome its effort (weight) will also

decrease. Thus, the most critical situation is when the camera end is almost touching the ground. Because the middle wheel is closer to the back wheel than to the front wheel, even if we take out of the equation the weight of the rest of the vehicle, this set-up presents a mechanical disadvantage for the back end. If l is longer than L, placing equal weights on both ends will always result in the front end remaining in contact with the ground. Uh-oh!

Figure 6.6 Weight Distributions in the SSCT



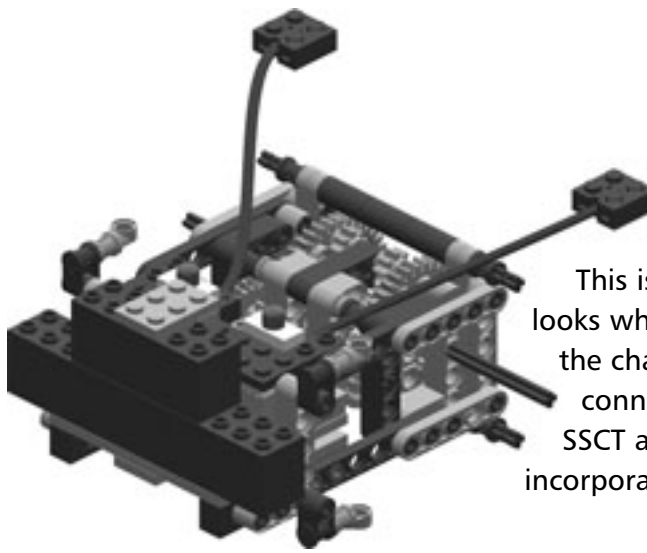
Of course, we do have to take into account the weight of the two halves of the vehicle, and more importantly, where the weight is placed. The weights give us the Force and Resistance on each end of L and *l*, and the center of the weight tells us precisely where those ends are. Where is the center of the weight? The center is the point where each half remains balanced. Because the SSCT is extremely modular, it is very easy to measure both the weight of its subassemblies, as well as where the center of the weight is located. Use a scale to weigh the elements that form each half of the vehicle. When assembled, and not counting parts for the tracks, the sub-assembly carrying the camera

and RCX is easily detached from the bottom half. Place each part in the scale and note the weight for each half.

To find where the center of the weight is located in an individual part, sub-assembly or completed vehicle, take advantage of the holes of TECHNIC parts and use an axle as a fulcrum to simulate a makeshift scale. The point where the part or sub-assembly remains balanced is the center point of the weight. The goal is not to figure out the exact weight requirements and specific measurements of L and l in the SSCT, but rather to learn how to make educated guesses. Remember to allow some margin for bumpy terrain.

At the end of the chapter, you will find several suggestions on how to alter the SSCT for different tasks. Nevertheless, the basic shape-shifting chassis remains, with its mechanical disadvantage for the back half of the vehicle. This is, of course, by design, as it is a by-product of a smaller and more agile wheelbase and a higher camera height when the vehicle is upright. However, the core design of the SSCT is designed in such a way to allow flexibility when counter-weighting the vehicle to keep the correct end upright.

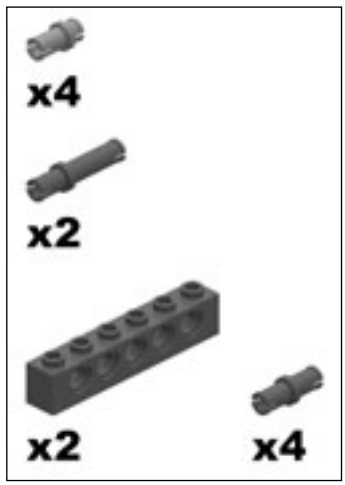
Building The Propulsion Unit



This is how the Propulsion unit looks when finished. At the end of the chapter, you will find how it connects to the rest of the SSCT and will be able to incorporate it into your own tanks.

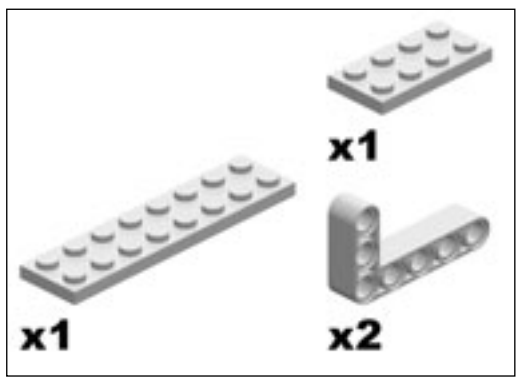
The propulsion unit incorporates a space behind the motors to accommodate counter-weights. In the instructions provided here, we use three LEGO weights; these weights are approximately 53 grams and can be found in official boat and train kits. I recommend any MINDSTORMS builders to acquire some, but since they are not very common, we can turn to other solutions within the LEGO realm. A small battery box (with battery) weighs less than 53 grams and is bulkier, but might work in a different version of the SSCT and can provide extra battery power to lights or other devices on the vehicle. By placing weight further away from the fulcrum, we move the center of the weight, extend L and ultimately gain mechanical advantage. To aid this, simple replace the two classic TECHNIC bricks in step one with longer ones.

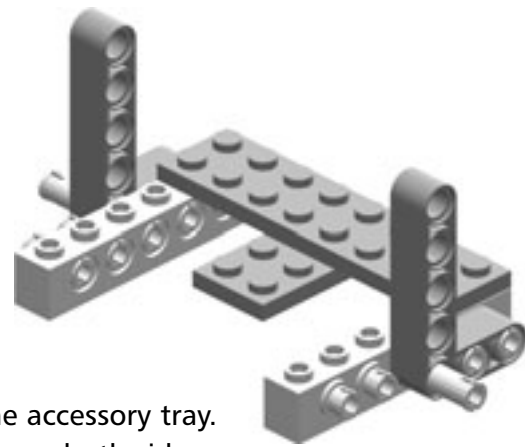
Propulsion Unit Step 0



In this assembly, we use two classic 1x6 TECHNIC bricks. Two of the studs will be used as arms of a tray to place counterweights. Using bricks longer than 1x6, we can counterbalance the SSCT with battery boxes that also provide power to lights or other elements onboard the SSCT.

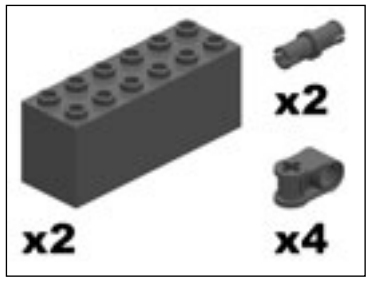
Propulsion Unit Step 1

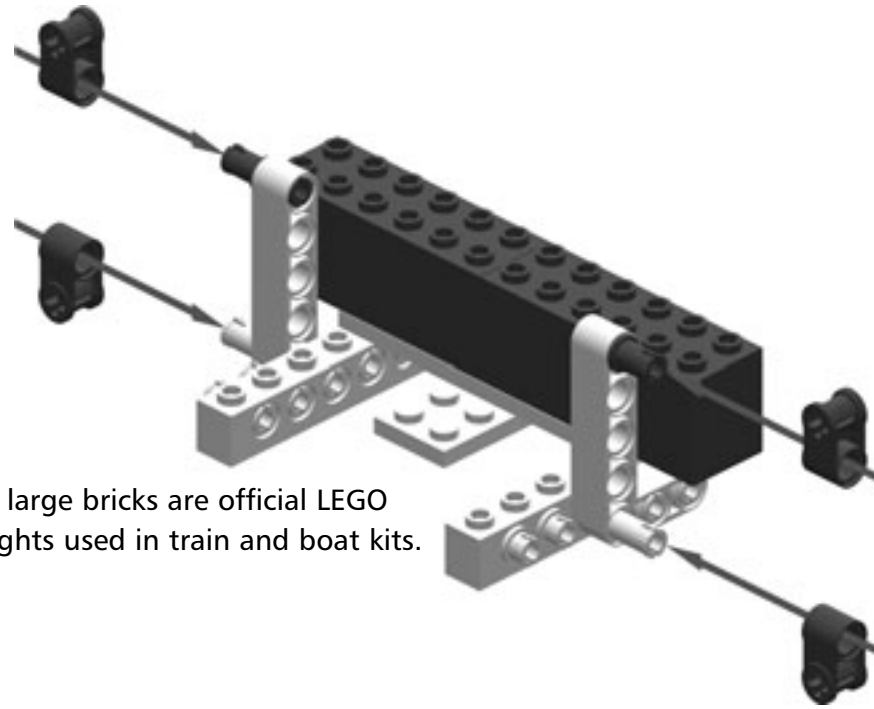




The 2x8 plate will be the bottom of the accessory tray. Connect both 5L liftarms symmetrically on both sides.

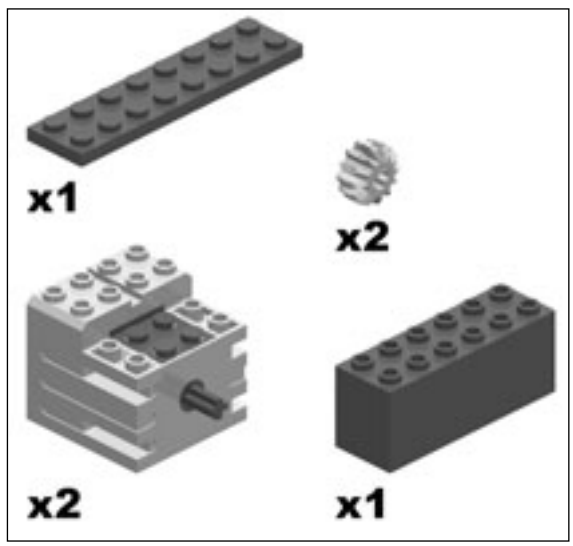
Propulsion Unit Step 2

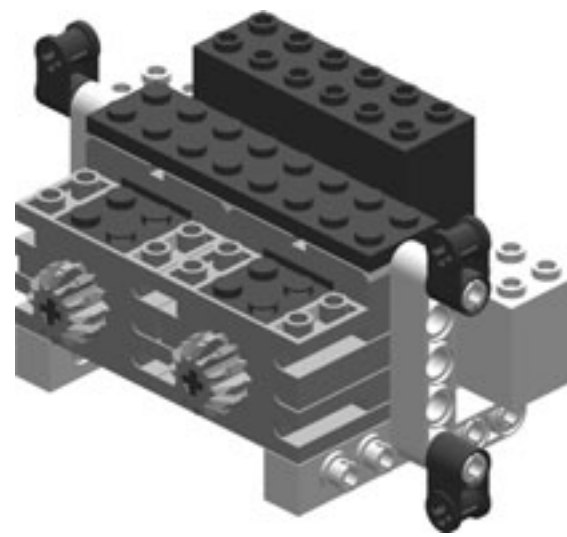




The large bricks are official LEGO weights used in train and boat kits.

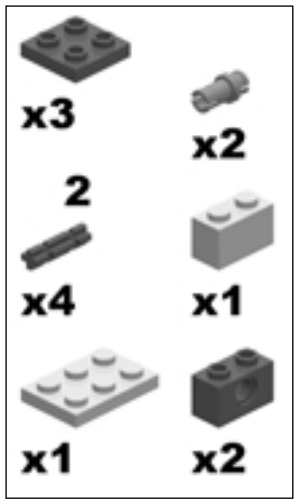
Propulsion Unit Step 3

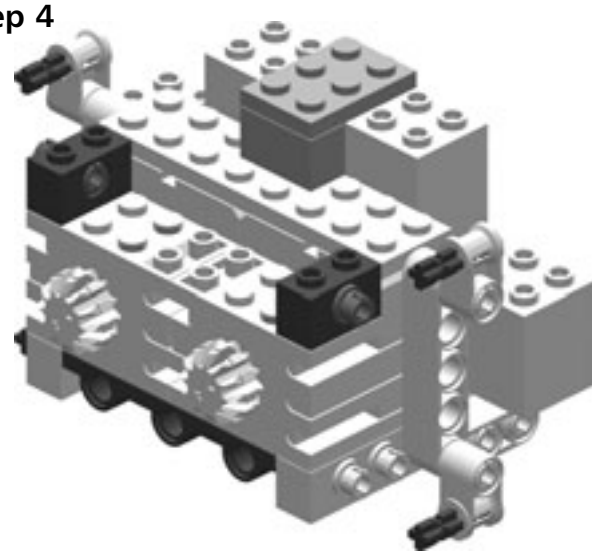




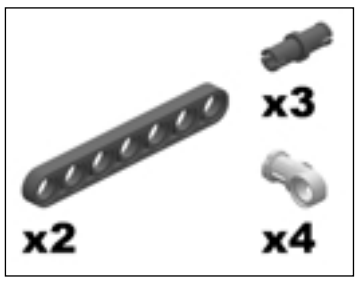
The motors also add weight to the base of the SSCT, but unlike the weights placed in the tray behind them, they are not removable!

Propulsion Unit Step 4

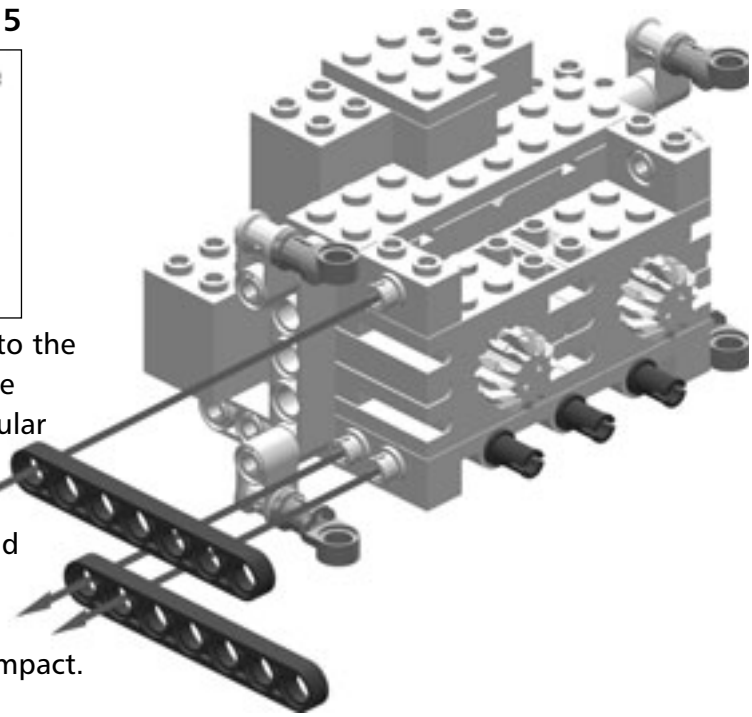




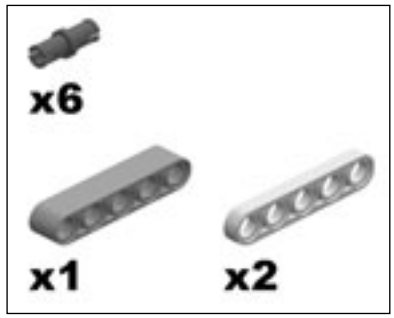
Propulsion Unit Step 5

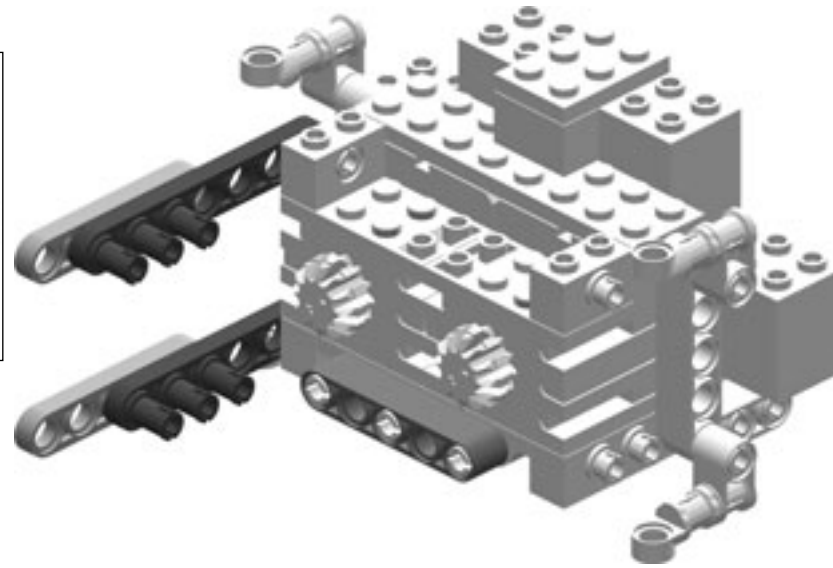


Attach the thin liftarms to the three-quarter pins. Unlike structures built with regular bricks, the ones made with liftarms often gain sturdiness as we add more elements. The finished sub-assembly is remarkably solid and compact.

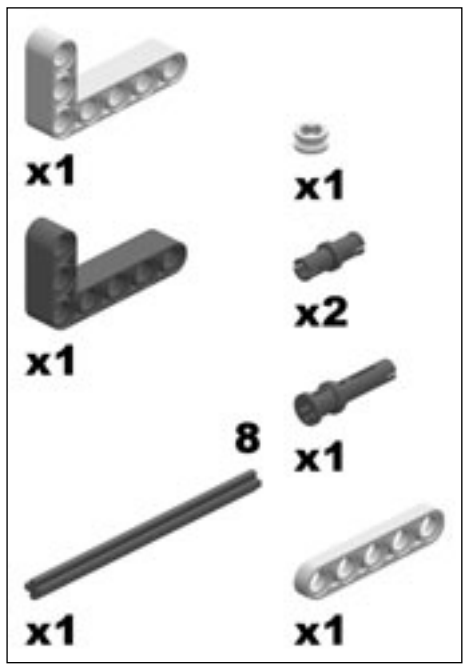


Propulsion Unit Step 6

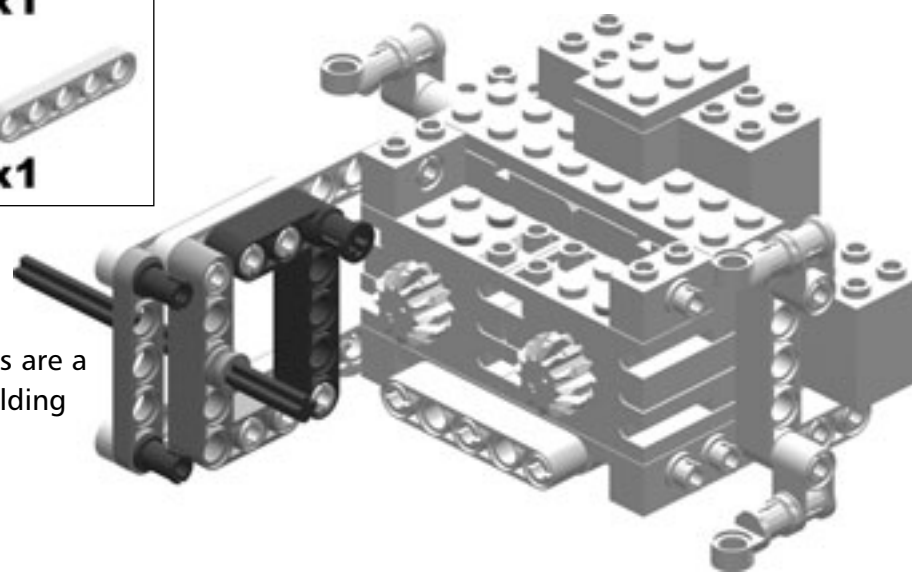




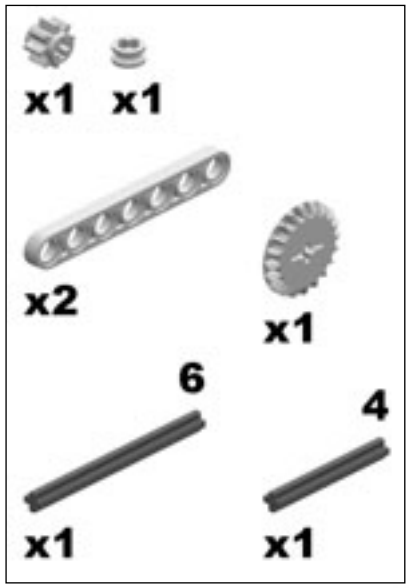
Propulsion Unit Step 7



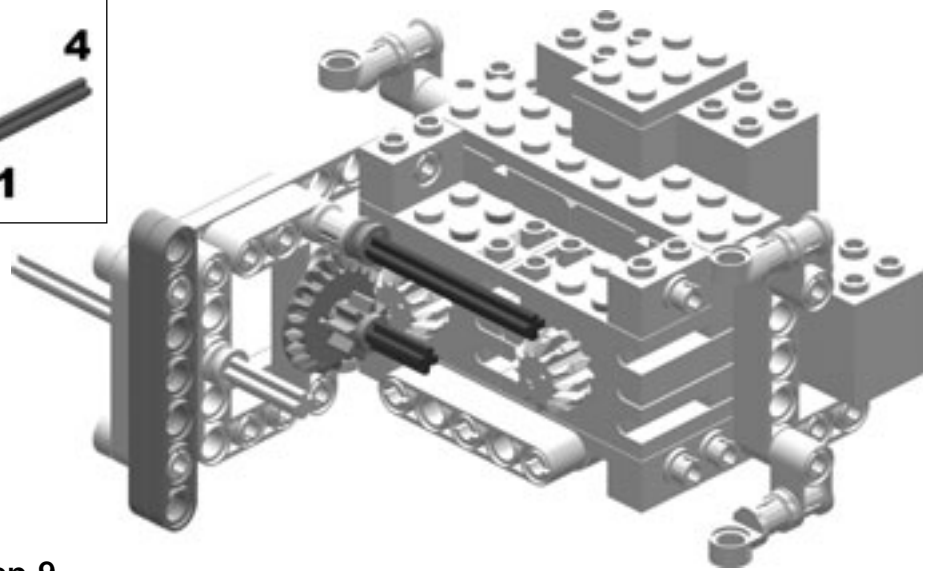
The L-shaped liftarms are a great help when building strong structures.



Propulsion Unit Step 8

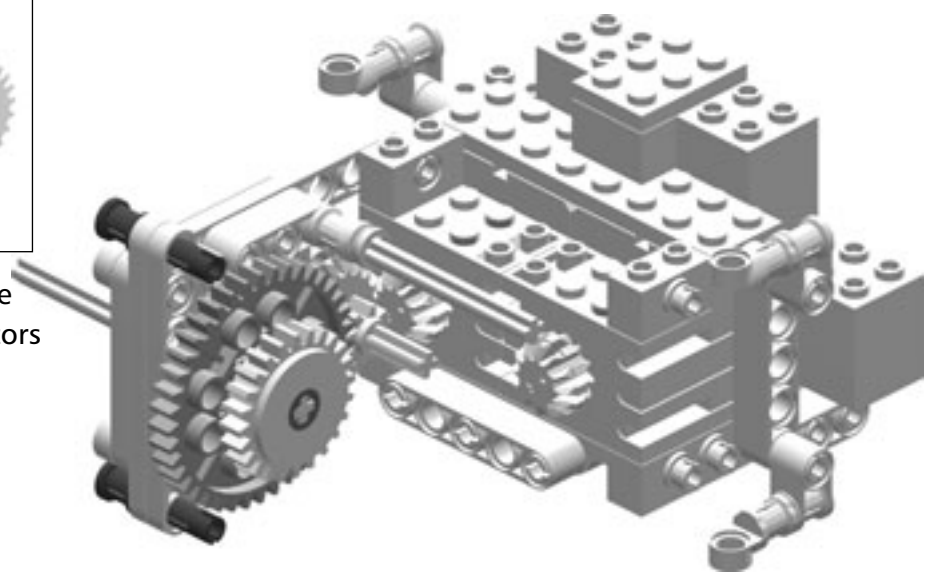


Once part of the structure is up, we start to build the gear train. The gear train reduces the motor output speed approximately 19 times, increasing the torque in the same amount (and thus less friction).

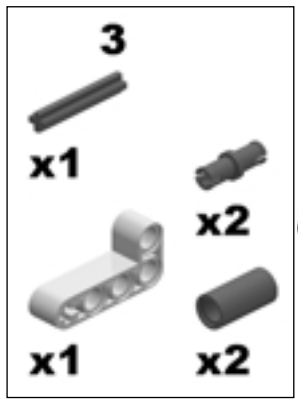


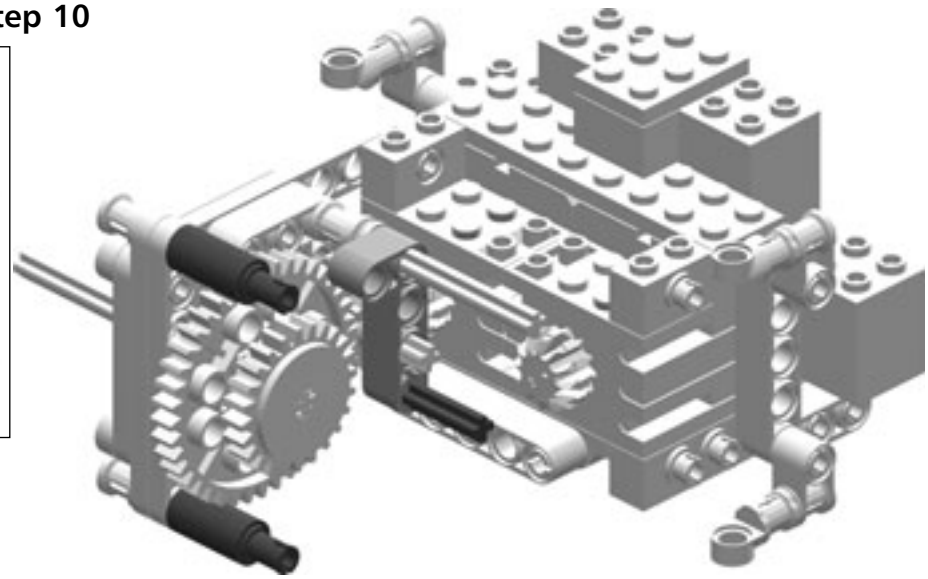
Propulsion Unit Step 9

As discussed earlier, the output of the two motors is connected with dual clutch gears.

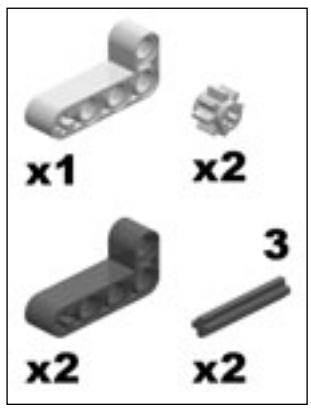


Propulsion Unit Step 10

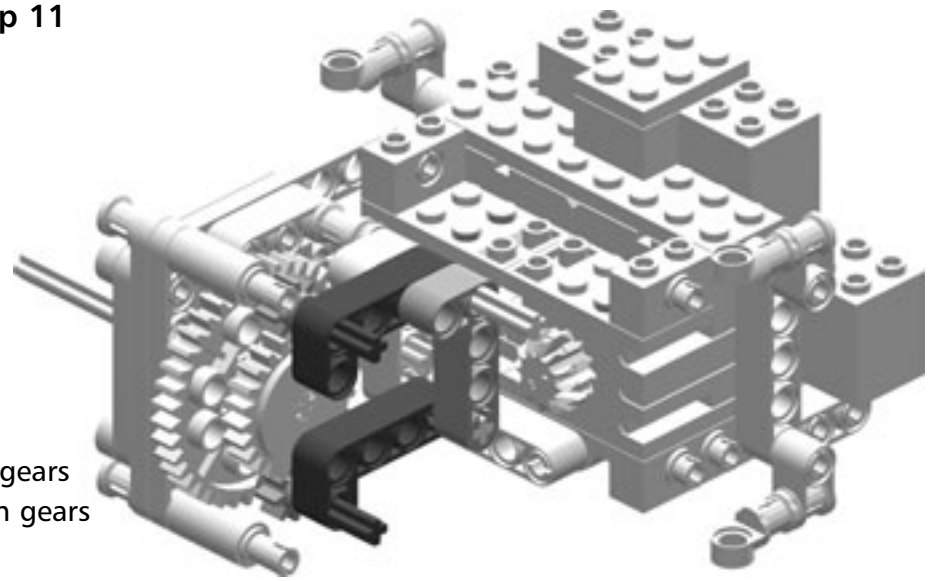




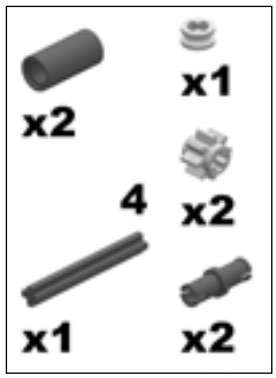
Propulsion Unit Step 11

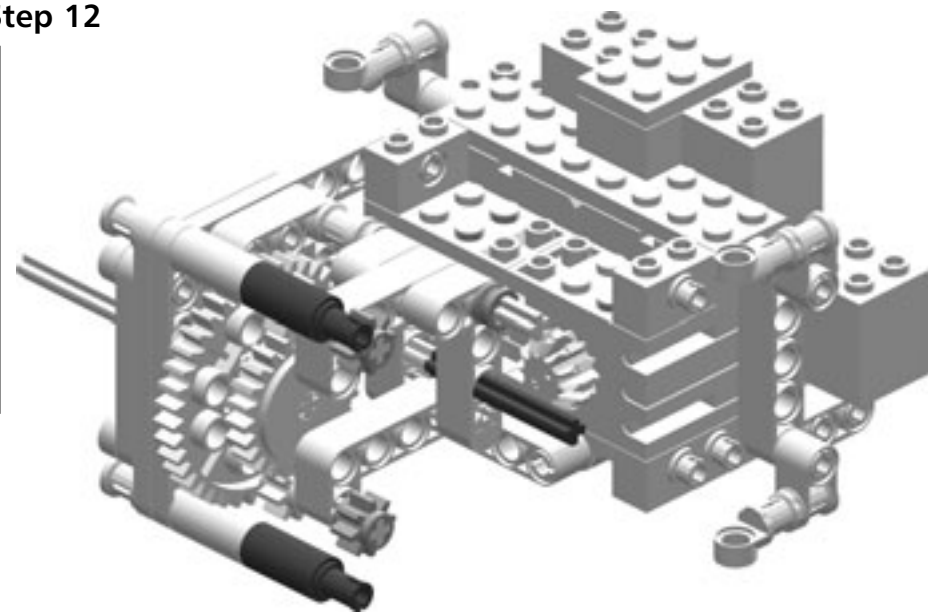


We now add the idler gears that connect the clutch gears of each gear train.

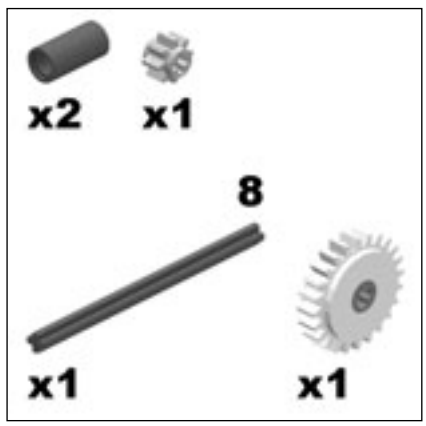


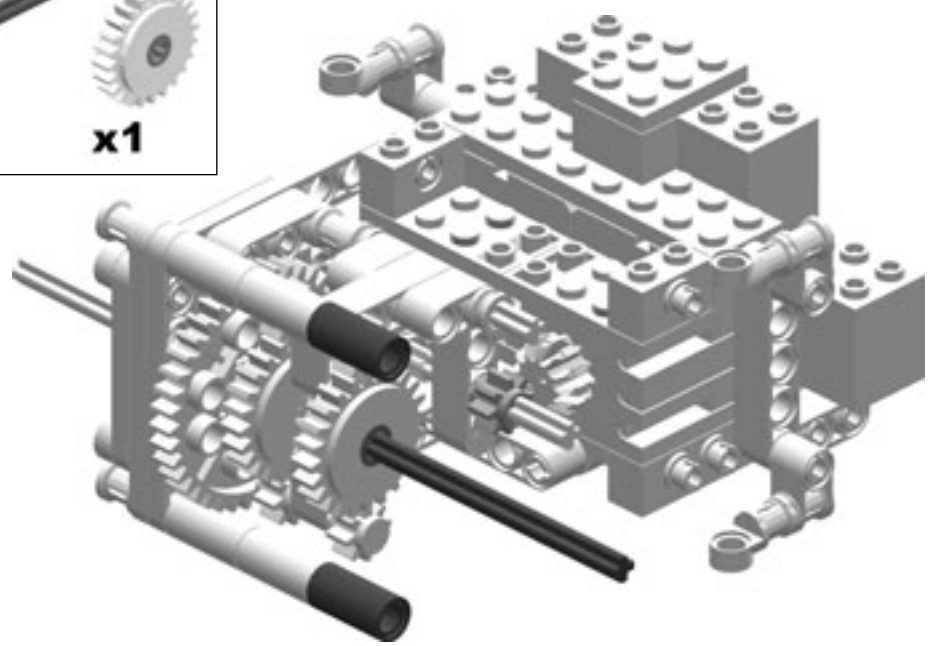
Propulsion Unit Step 12



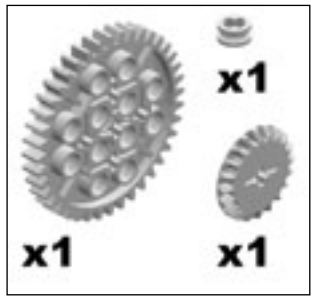


Propulsion Unit Step 13

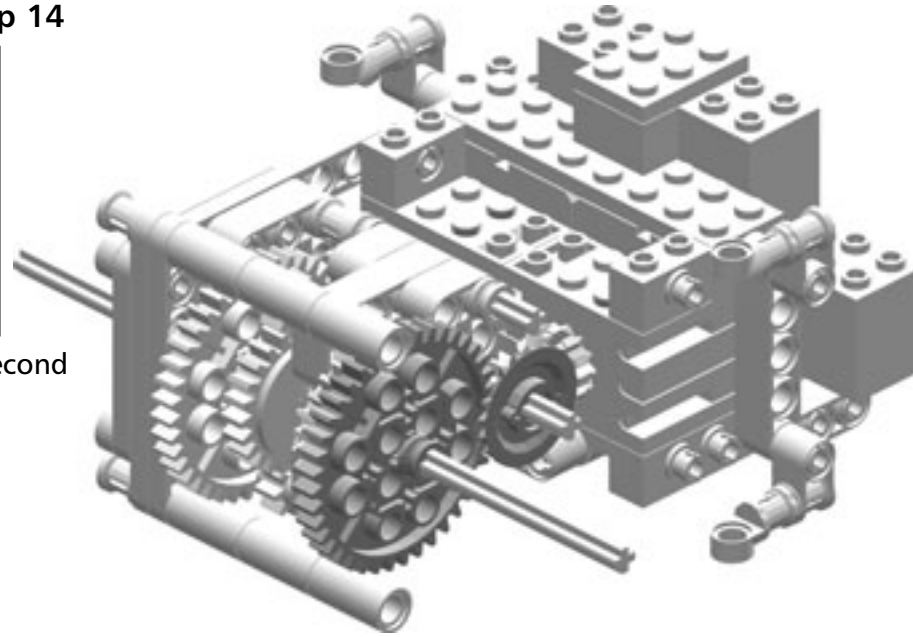




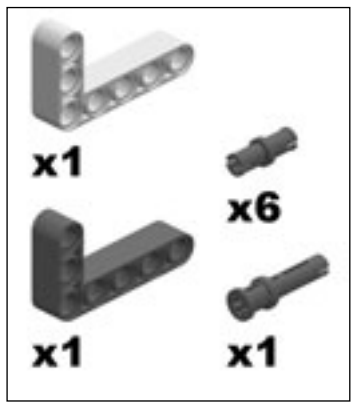
Propulsion Unit Step 14

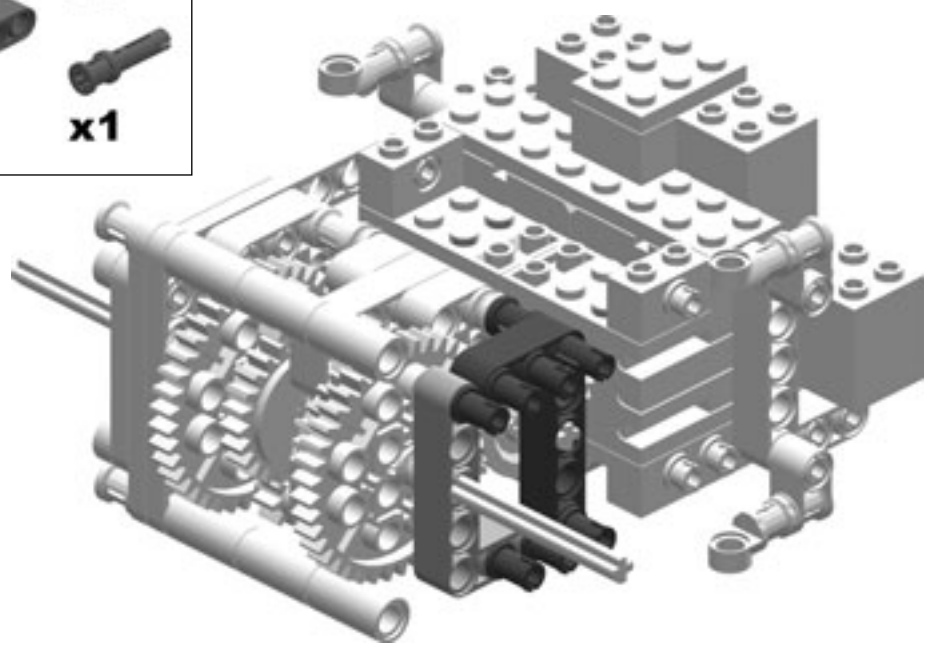


In this step, add the second gear train.

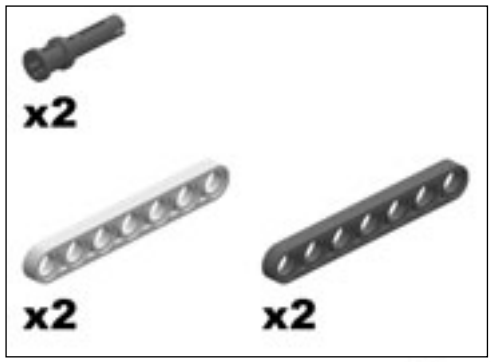


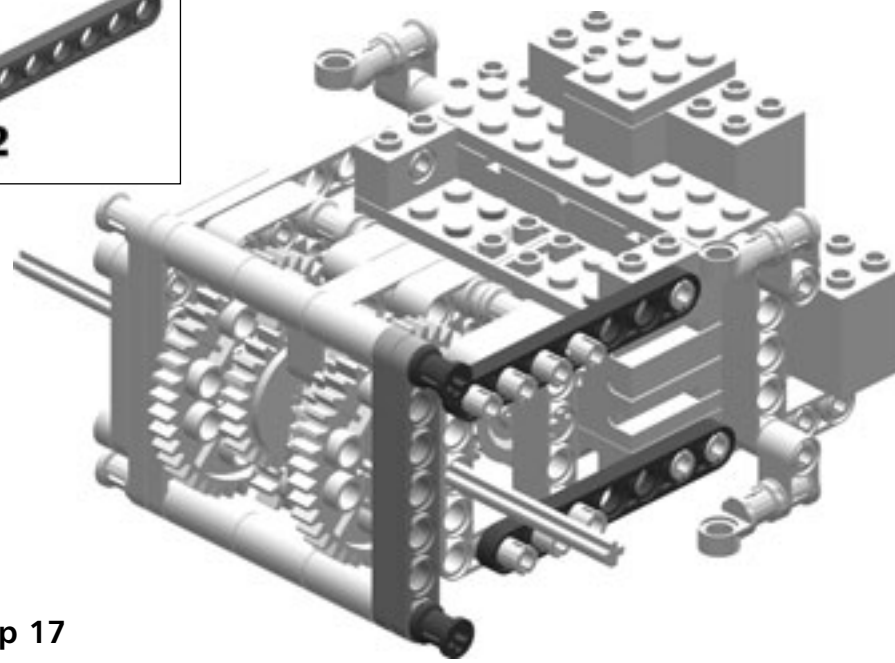
Propulsion Unit Step 15



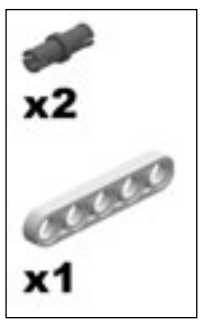


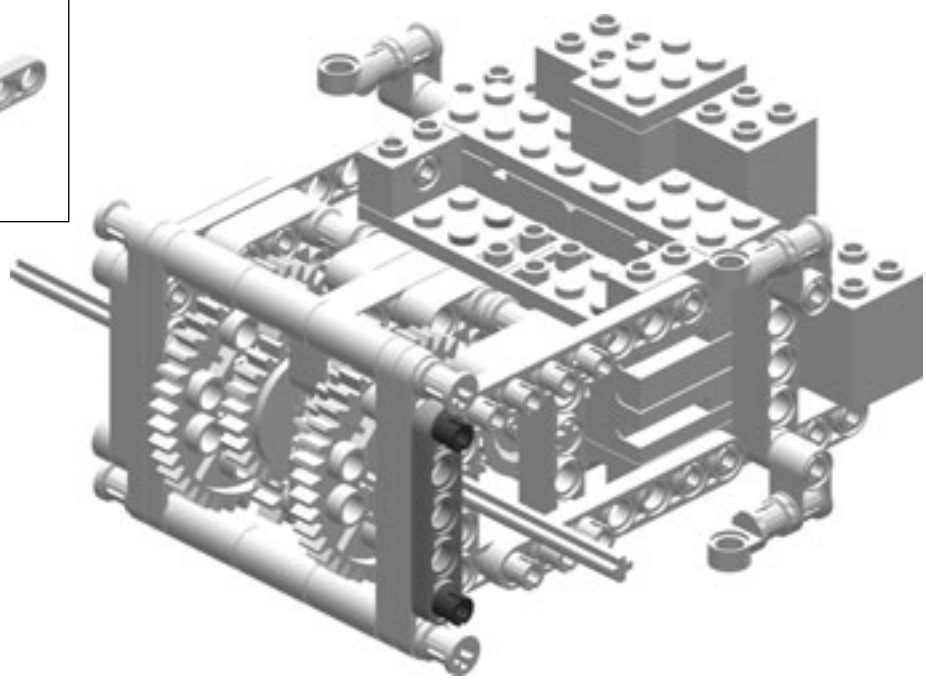
Propulsion Unit Step 16



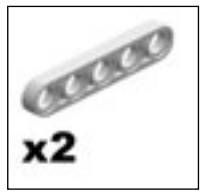


Propulsion Unit Step 17

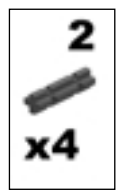




Propulsion Unit Step 18

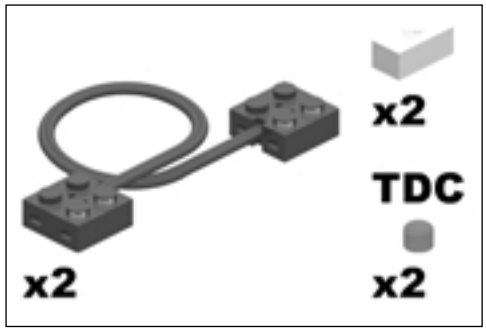


Propulsion Unit Step 19



Add one #2 axle to each pin with friction and stop bushing. These axles are the anchor points of the unit to the vehicle's chassis.

Propulsion Unit Step 20



NOTE

The light bricks are not necessary for the operation of the vehicle, but do give you a visual cue of the motor status, which can come handy when building large models.

The Folding Muscle Unit

The shape-shifting mechanics of the SSCT involve two sub-assemblies. One is the mechanism that pulls together or pushes apart the two halves of the chassis. This in turn changes the wheel alignment and, consequently, the shape of the track as well. We will talk about the science behind the changes in the shape of the track in the next section. In this section, we will build the Folding Muscle Unit sub-assembly that causes those changes.

Unlike the traction unit with its clutch-based coupling/decoupling feature, the mechanics of the lift are rather ordinary. We simply proceed to gear down the output of a motor 600

times, increasing its torque in the same amount, minus friction. This significant reduction is necessary to lift over 600 grams with a single motor needed for the operation of the SSCT. The front section weighs approximately 600 grams, including the RCX loaded with batteries, the vision command camera, and a number of other pieces.

Dealing with the force generated by such a gear reduction is not something LEGO is designed to withstand, so we must be extra careful when building the structures that houses such power. We must be fully aware of several issues at play, and have ready several tips on how to deal with them or we might end up hurting our fingers or breaking our precious LEGO parts, which to some of us is a painful as breaking a digit.

Pressure is defined as the amount of force within a specific area. A well-designed structure must take into account the areas where the pressure is higher, and incorporate measures to fight the stress. A good option is to dissipate the force over a larger area, thus reducing the pressure. Even if we are shooting for a small-sized mechanism, there are very imaginative ways to incorporate such physics.

Bricks & Chips…

Mecha Building: Eric Sophie

The lift mechanism of the SSCT, although compact, takes advantage of the method suggested by LEGO builder extraordinaire Eric Sophie. Eric is the creator of a body of work unparalleled in theme and scope. He is best known as a "Mecha" builder. Mecha building is an unofficial LEGO theme with a wide acceptance among fans. Mechas are creations rooted in Japanese pop culture, from Manga TV series (especially the earlier ones featuring large robotic characters) to the Transformers™ line of toys. Like the official LEGO Model Team kits, the emphasis is on looks, but most Mecha creations also incorporate some sort of mechanical functionality. Mechas come in all sizes and many of them push the envelope of LEGO biomechanics to remarkable extents.

This is the terrain that Eric covers with admirable success. Starting with his series of little robots (Little Red Bot, Little Blue Bot, and so forth), which instantly make the most diehard gearhead's heart twitch, and ending with his enormous automatons, Eric has broken down and replicated gracefully once and again natural movements and expressions, often bearing enormous loads in the process. Eric's work can be found at his Web site: www.biomechanicalbricks.com.

One of the unavoidable effects of stretching LEGO beyond its tolerance is that the system will start to behave inconsistently. Some elements will happily bear extra load while others give in much earlier, quickly overwhelmed by the laws of physics. The gear train of the folding muscle (or lift) sub-assembly supports the weight of the front part of the vehicle, but this force needs to be transmitted through a suitable, equally strong medium. The output axle of the muscle mechanism is able to lift over 600 grams, but if

we simply attach a liftarm to it, the pressure on a small part of the axle will be too much and the material will give in to *torsion*. Instead of having a solid arm that raises the camera, we have a wobbly one. This wobbly camera mount is almost as embarrassing as having the robot fall face first on its camera lens.

Figure 6.7 shows three different ways to output the power of the gear train. The one at the left is the weakest because the weight supported by the liftarm is transmitted to a small section of the axle: the section where it is braced by the end of the liftarms. The one at the right is the strongest, in part because four friction points provide twice the surface than the two points that are used in the center image – but there is more to it.

Figure 6.7 Avoiding Axle Torsion

Eric Sophie has suggested this method of attaching beams directly to 40T gear, which he uses extensively, to reduce the pressure. To see how it works, let's start with the soles of our feet. By dividing our weight by the total area which our feet have contact with the ground, we will get an idea of how much weight each square centimetre or inch of sole supports (which is probably more than you think if you have never calculated it before). As you may well know, this happens to be too much pressure for fresh snow. With snowshoes on, our weight doesn't significantly change, but is spread across a larger area, thus decreasing the weight per square unit, or pressure.

Now applying this snowshoe theory to LEGO, how does it work, since the load on the axle is radial? What we use here is another application of the same formula that we used when distributing the weights of the SSCT. When a lever's fulcrum is located between the force and the resistance, similar to Figure 6.6, we call it a *first class lever.* However, many levers have the fulcrum at one end while one of the loads is located at the other end, with the second load in the middle. Depending on whether the resistance or the force is located closer to the fulcrum, we call those gear *second* or *third class levers.* In our tank, the resistance is the weight of the front end of the vehicle, but where is the force that keeps it aligned with the axle applied? We apply force to the minuscule arms of the axle. Talk about brutal torque reductions! That is a one-stage reduction of maybe 1:60. This means that with a construction similar to the image on left in Figure 6.7, a very small vertical movement of the front end (which weighs over 600 grams) is actually applying a force of close to 40 kilos on a tiny section of the axle. It's no wonder why the plastic gives.

Using the 40T gear, we move the force further away from the fulcrum. This reduces the mechanical advantage (ratio of L to l, or the distances between the fulcrum and the location two forces) of this third class lever set-up to a more tolerable level of maybe 1:10. The load is applied over a wider area, which includes more contact points between the critical elements of the structure. In other words, the vertical movement of the front end carries a lot less punch at the axle, which is dissipated by the large gear. This is truly a win-win situation, and worth creating the rightmost construction in Figure 6.7. With the axle secured into a fixed position, compare how much radial force you need to introduce any deformation in the axle using your own muscles.

While I had originally thought of using MINDSTORMS touch and/or rotation sensors to detect where to stop the lifting mechanism at its maximum top and bottom positions, there are several reasons why these ideas were discarded. For instance, one of the specifications for the SSCT was to make it a useful base vehicle, thus using a simpler way that bypassed the need for sensors makes sense. Additionally, while ideally we only need a single input port to control the vehicle, this might turn out not to be possible (in fact, we might need all three, as we will see later in the chapter). And even if only one sensor port of the RCX is used, I actually wanted to leave the other two input ports free for users to be able to create specialized devices connected to them (some ideas for this will be provided at the end of the chapter). So, the SSCT incorporates mechanical means to limit the movement range of the muscle mechanism.

This is one of the original purposes meant for the clutch gear employed in the propulsion sub-assembly: as a way to avoid disasters in gear trains. Well-placed clutch gears will not allow our mechanism to bind (which can end up breaking stuff, be it LEGO parts or other elements in the model). When extremely strong forces come into action, the clutch gear's internal mechanism will give to pressure and become a sort of limited slip idler gear, hopefully breaking the flow of force between the source and the rest of the train.

As we just saw, when dealing with large gear reductions, weights and levers, we must place our parts very carefully, lest they be affected by overwhelming forces. The problem of using a clutch gear to break the connection between a motor source and a high-torque gear train involves a similar approach to the one above. However, instead of countering the force of the Front sub-assembly's weight on the axle that connects it to the gear train using a simple lever scheme, we are looking to counteract the brutal force of the complete gear train, which is, in effect, a collection of levers combined together.

Due to the same mechanical advantage law we have already seen, when we reduce the speed of the output of a motor with a gear train, we increase its torque. The Folding Muscle Unit sub-assembly is driven by a rapidly rotating LEGO motor. The output has little force: it is very easy to stall this motor with your fingers, but be aware that it is not healthy for the machine. By reducing its speed, we gain torque. Thus, we can assume the further away from the motor that a gear is, the less speed and more torque it carries. Thus, if we place the clutch gear too close to the output of the gear train, where the

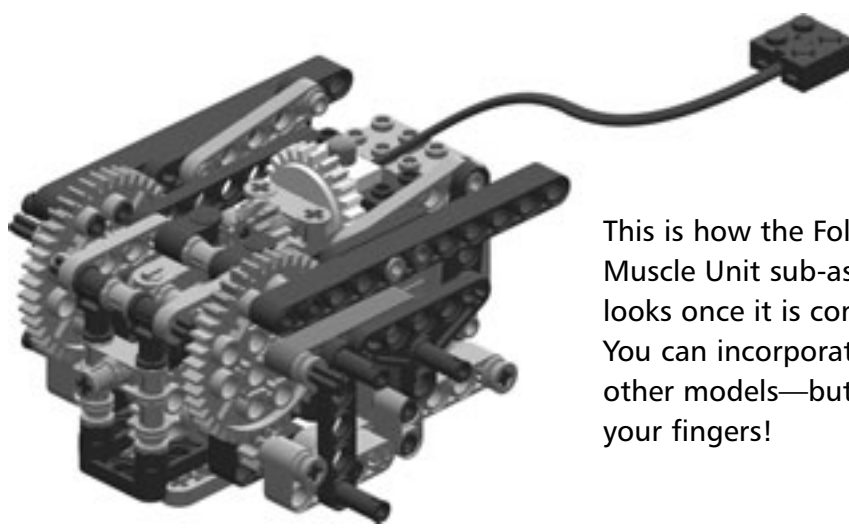
forces at play are larger, it will prematurely give. In the case of the SSCT, any attempt to fold the vehicle will cause the weight of the front end to overwhelm the mechanical sensors and the linkage with the motor will be broken. Yet another potential source of embarrassment: a shape-shifting robot without enough force to actually change its shape.

We can assume the opposite is also true: the closer the location of the clutch sensors to the motor's output, the less torque the clutch gear carries, spinning instead at a greater speed. In order to stop it, we need to introduce a lot of resistance into the system before the mechanical sensors actuate. The problem is that we have to send that overwhelming resistance inversely through the gear train, which is designed to precisely counter it.

In other words, right after we spent some time figuring out how to best maximize the mechanical process of lifting and lowering the camera, we are forced to think how to completely overwhelm that process in order to mechanically stop the mechanism at the desired limits. The process is not very different. With the clutch gear in place at the beginning of the gear train, I created the upper and lower brakes by applying overwhelming force to the mechanism at places that take advantage of the mechanical properties of levers. For instance, when trying to stop an arm connected to an axle, place the brake as far away as possible from the rotating axle fulcrum, and you will be increasing its mechanical advantage.

In the SSCT, the brakes are collapsible links that allow a certain degree of rotation to the arm, but prevent it from rotating outside a certain range. The upper brake is included in this sub-assembly: two liftarms attach to the front assembly arms as far as the compact design can spare. The lower limit brings into action several contact points in the structure, but ultimately the long collapsible links at the top of the robot (that also maintain the camera upright) provide the stopping power. Notice how far away from the axle they are. They attach to the camera at one end of the robot, and we will build them in the next section.

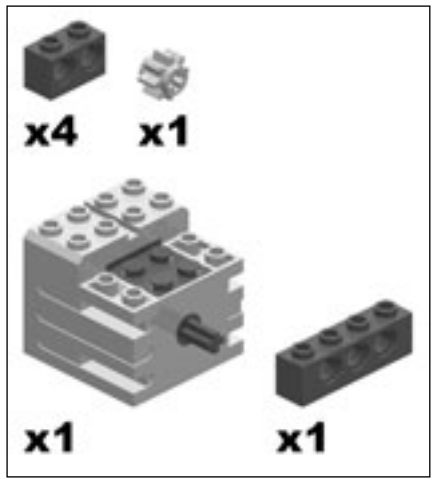
Building the Folding Muscle Unit



This is how the Folding Muscle Unit sub-assembly looks once it is completed. You can incorporate it in other models—but watch your fingers!

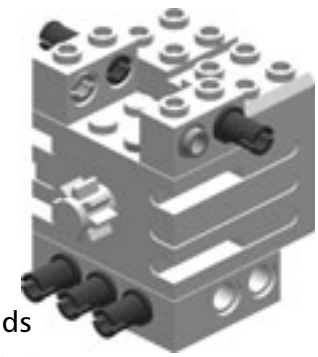
We start with a motor, with the objective to build a small but powerful muscle of sorts around it. Gearing down the motor in a compact space is only half the problem. Making sure that the structure can withstand the forces generated is equally important, or it will come apart.

Folding Muscle Unit Step 0



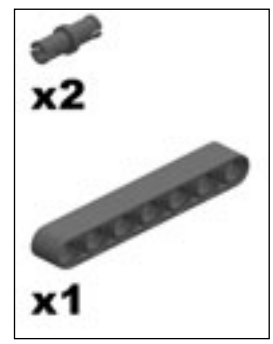
In general, whenever a design calls for attaching liftarms to a motor, we do so using classic TECHNIC bricks as intermediate parts.

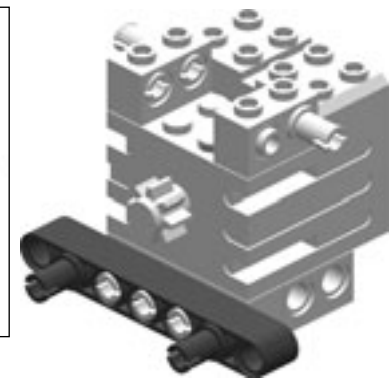
Folding Muscle Unit Step 1



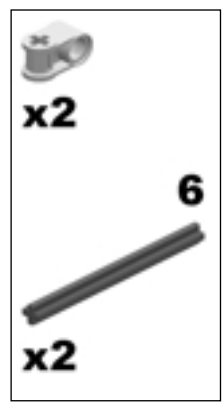
The pins are the studs for the liftarms.

Folding Muscle Unit Step 2



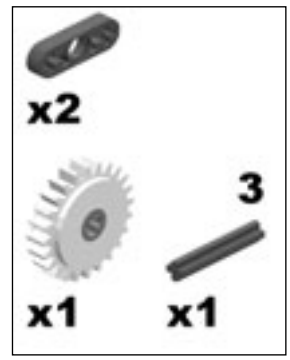


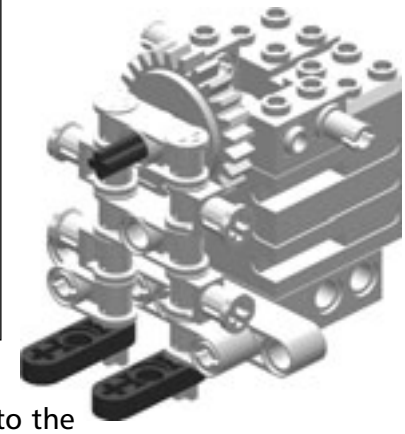
Folding Muscle Unit Step 3



Folding Muscle Unit Step 4

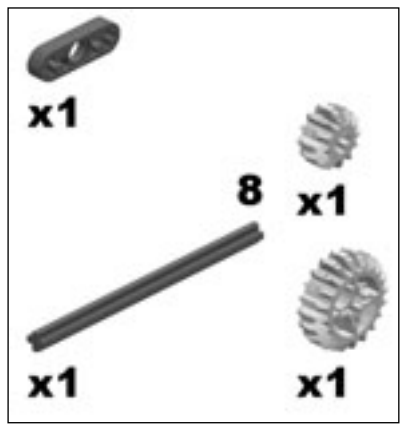
Folding Muscle Unit Step 5

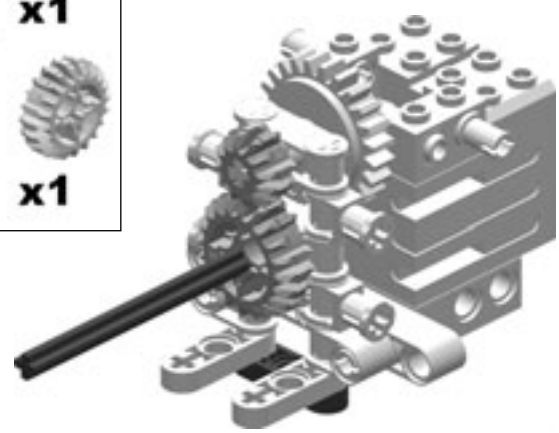




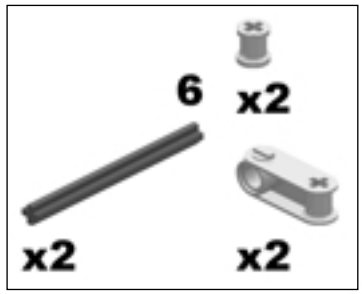
By connecting the clutch gear directly to the motor output, we ensure that it will disengage the gear train only in the event of binding forces, such as those activated by the mechanical brakes. In fact, the clutch gear can be considered part of the limiting mechanism of the Folding Muscle Unit sub-assembly.

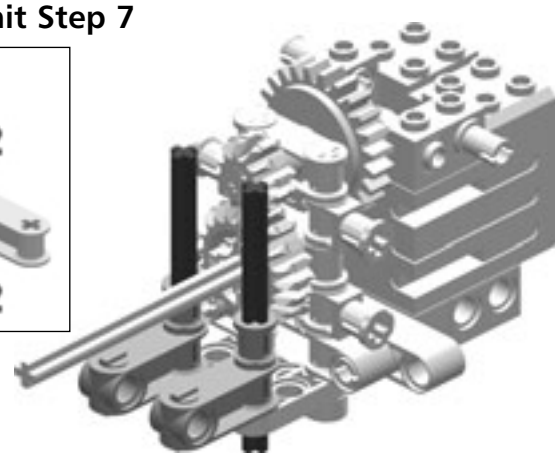
Folding Muscle Unit Step 6



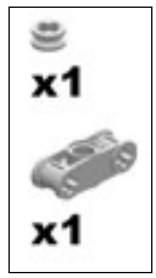


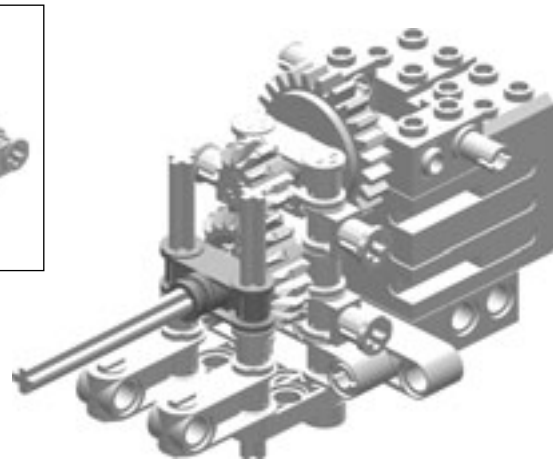
Folding Muscle Unit Step 7



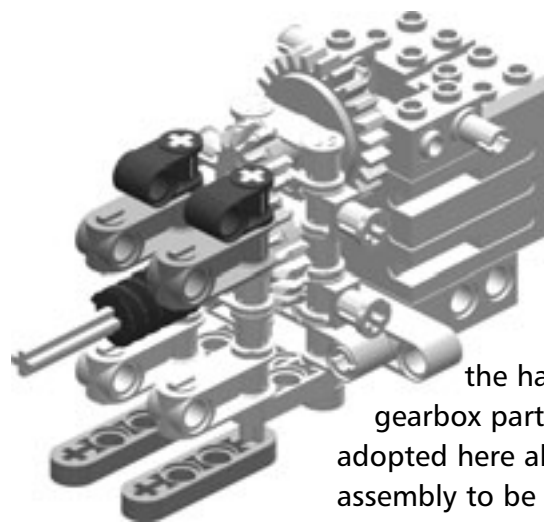


Folding Muscle Unit Step 8



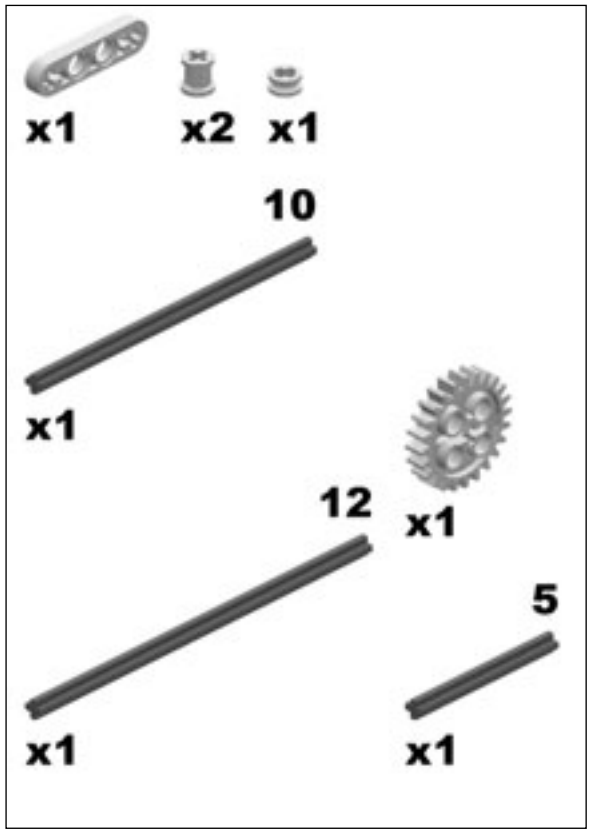


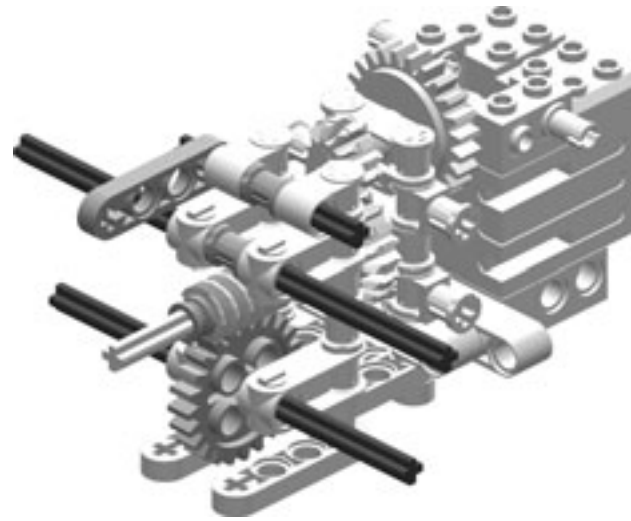
Folding Muscle Unit Step 9



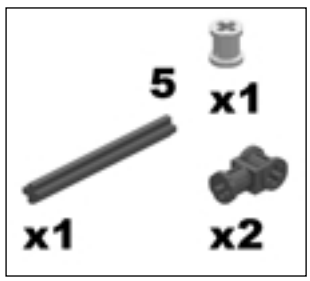
Like many high torque LEGO gear trains, this assembly uses a worm gear combined with a 24T gear (which provides a 1:24 mechanical advantage). Many such designs include the handy space-saver LEGO gearbox part. However, the solution adopted here allows the sub-assembly assembly to be even smaller!

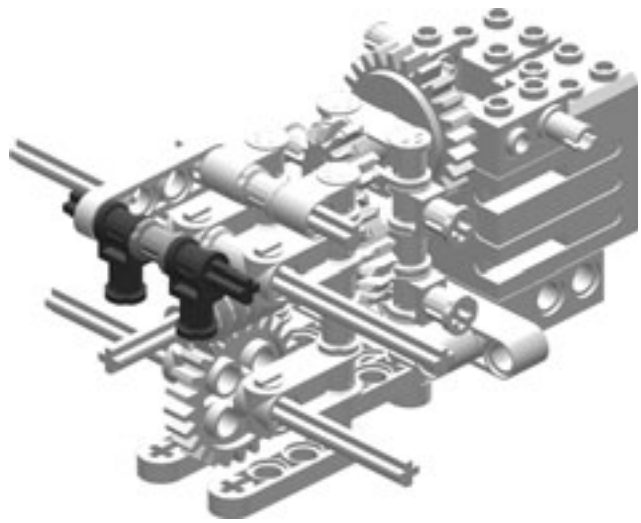
Folding Muscle Unit Step 10



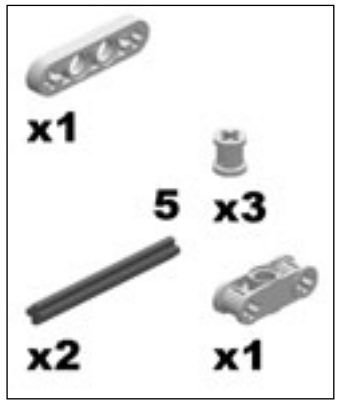


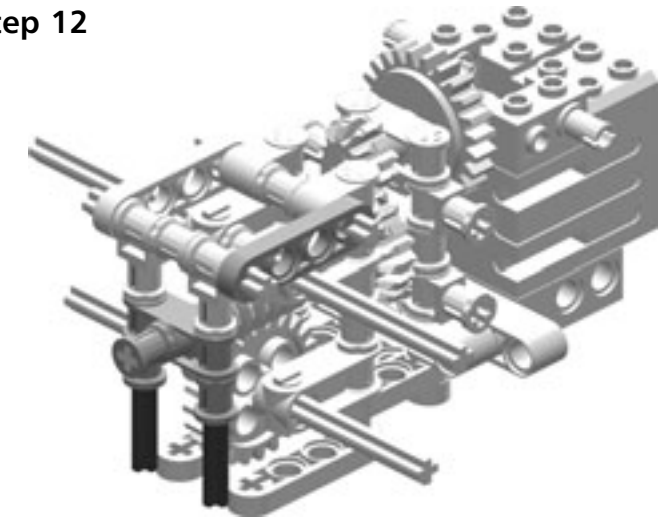
Folding Muscle Unit Step 11



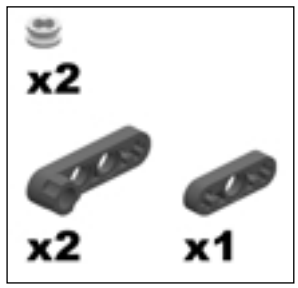


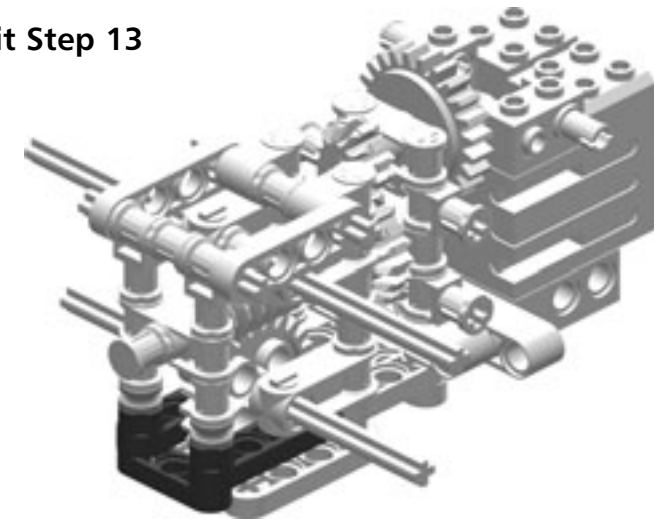
Folding Muscle Unit Step 12



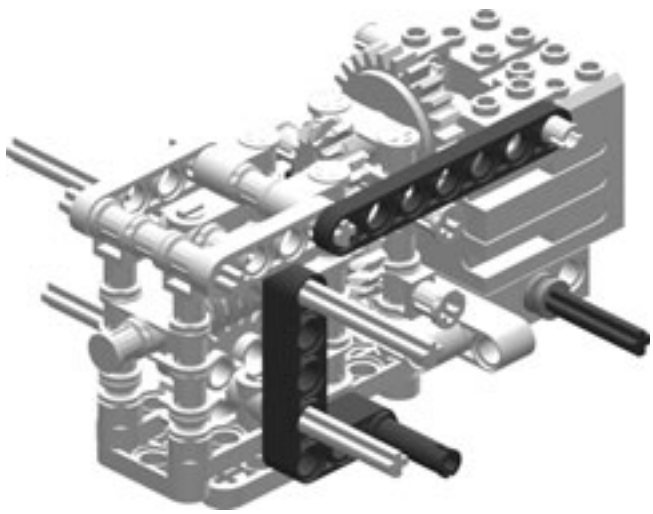


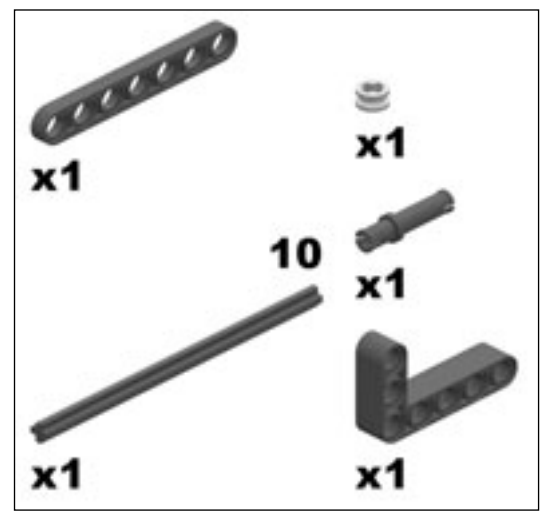
Folding Muscle Unit Step 13



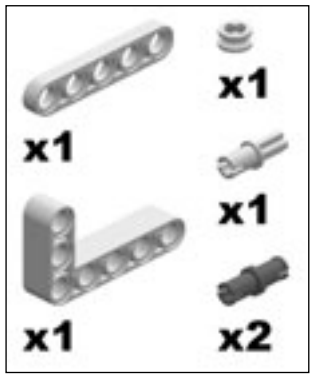


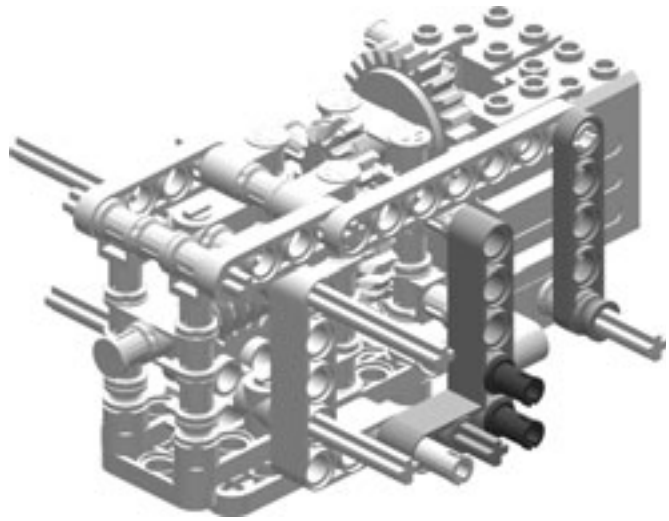
Folding Muscle Unit Step 14



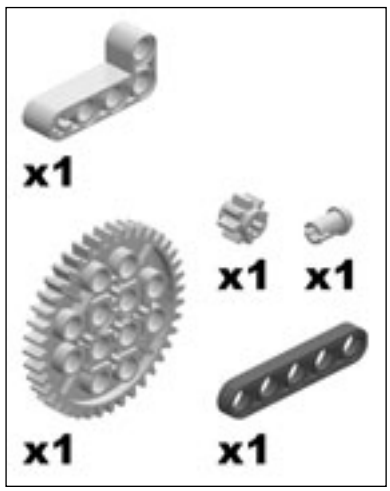


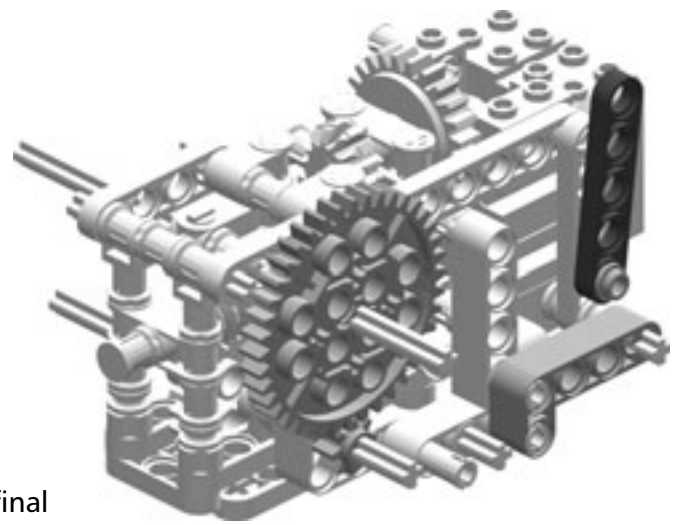
Folding Muscle Unit Step 15





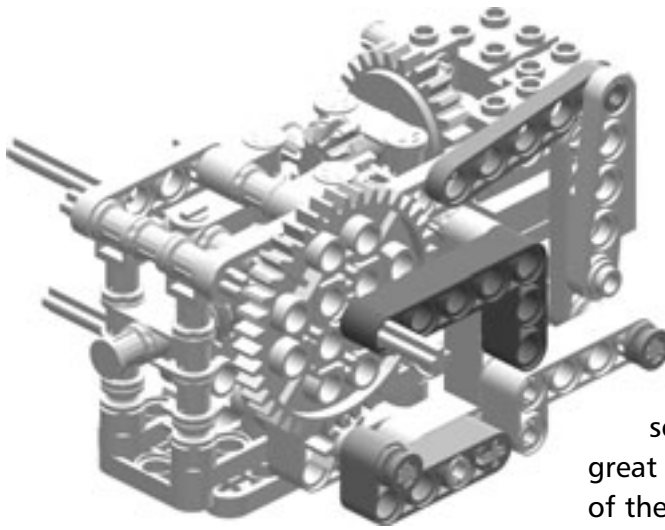
Folding Muscle Unit Step 16

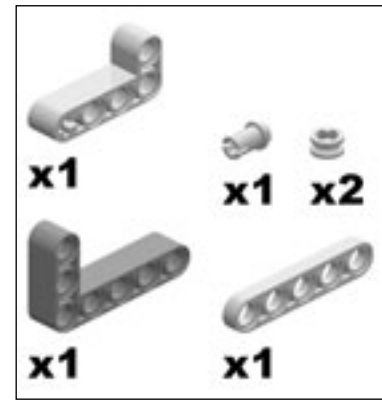




In this step, we add the final gear of the gear train, the large 40T gear. We also add the first two parts of the upper-movement limiting mechanism, the black liftarm and the half-length pin that attaches it to the structure. The brake linkages are collapsible. So, until attached to the rotating arm in **Folding Muscle Unit Step 19**, they will rotate freely around the half-length pin.

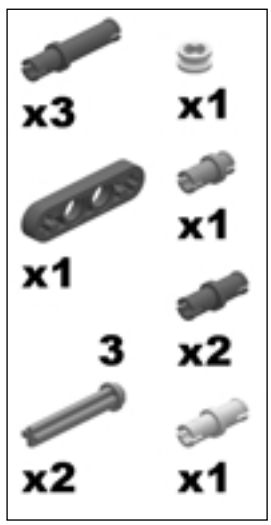
Folding Muscle Unit Step 17

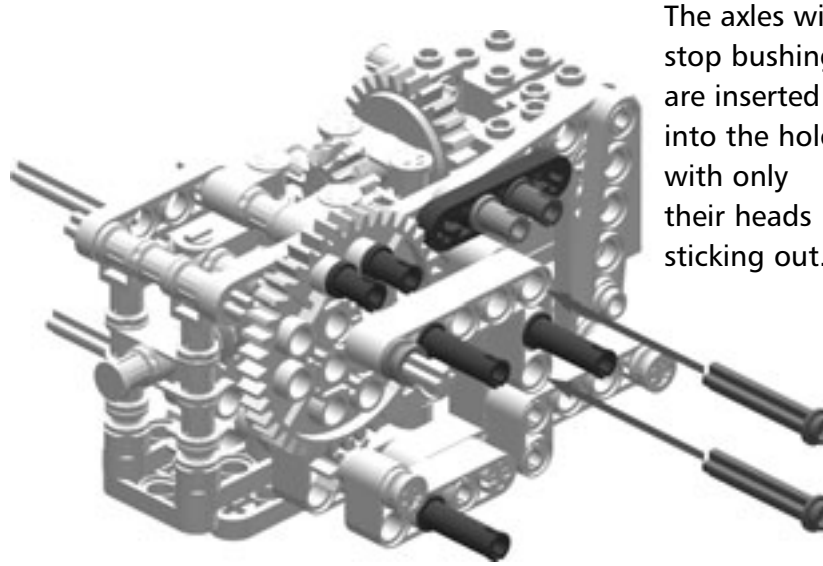




The straight gray liftarm is the second part of the brake. This is a great example of how to take advantage of the half-stud width of liftarms.

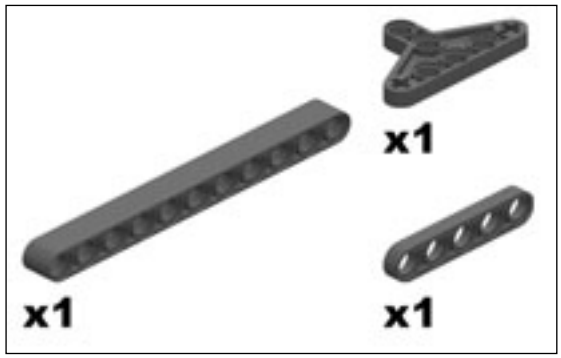
Folding Muscle Unit Step 18

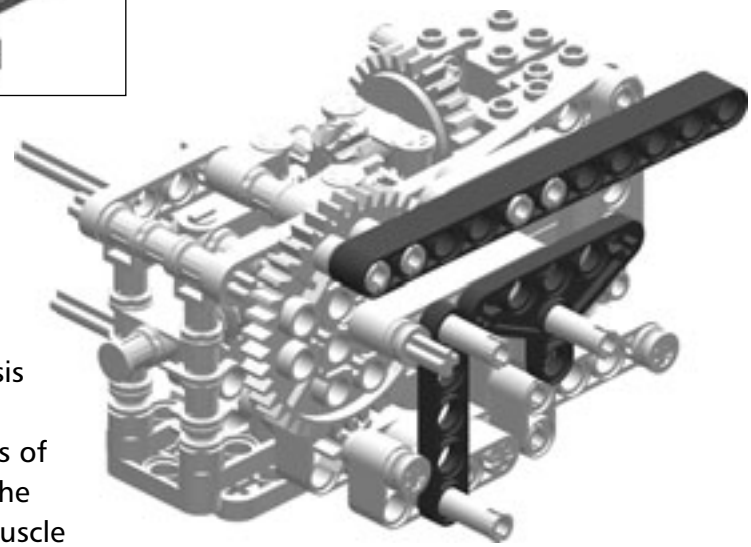




The axles with stop bushings are inserted into the holes, with only their heads sticking out.

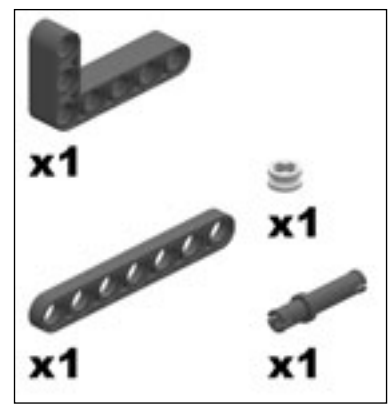
Folding Muscle Unit Step 19

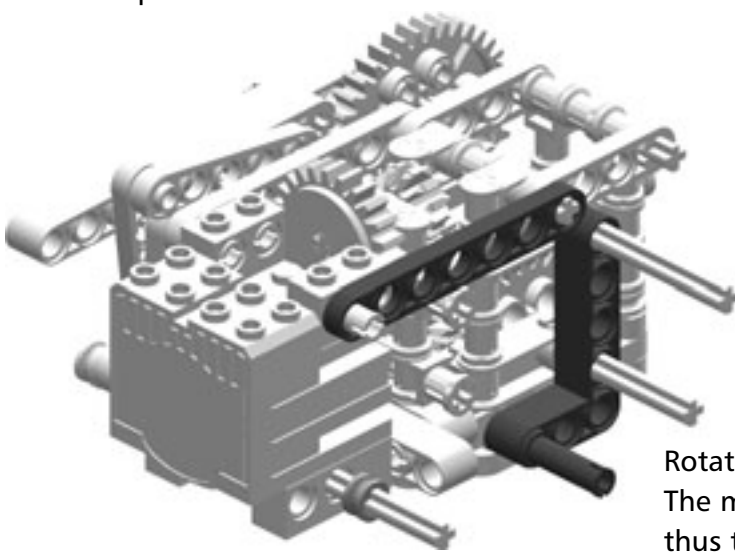




In this step, we add the long liftarm, which connects the muscle with the front sub-assembly that carries the camera and the RCX. Note how the long friction pins that connect the muscle to the chassis are not as integrated into the structure as the attaching points of the previous step. Fear not, as the design of the chassis and the muscle attachments combine forces to keep the design compact but secure. This is achieved in the subsequent section "Putting It All Together" later in the chapter.

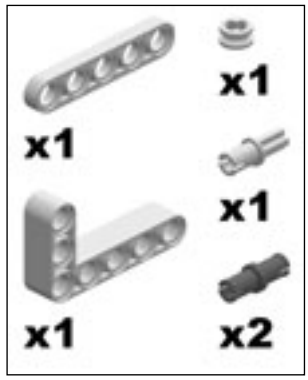
Folding Muscle Unit Step 20

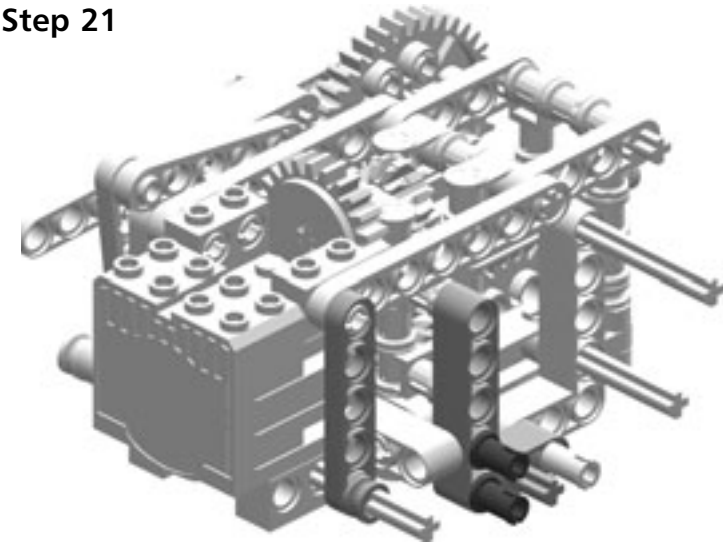




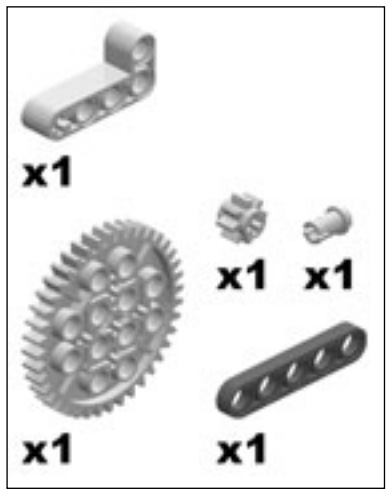
Rotate the assembly as shown. The mechanism is symmetrical, and thus the next steps replicate the other side.

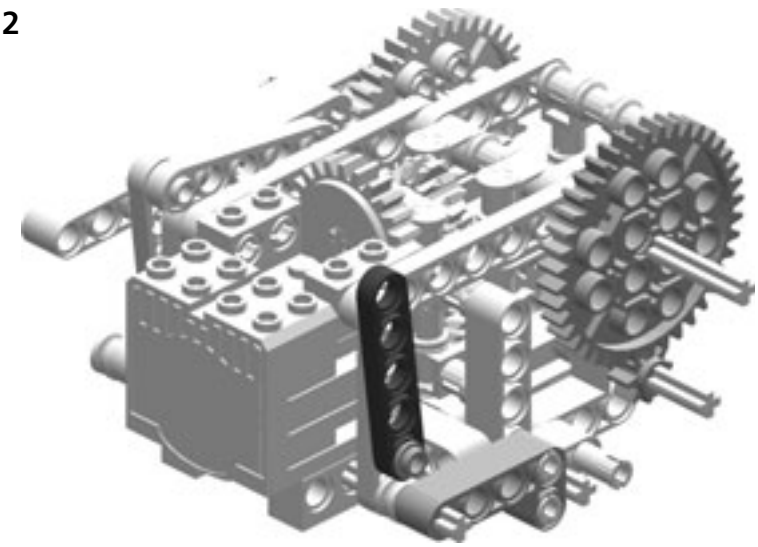
Folding Muscle Unit Step 21



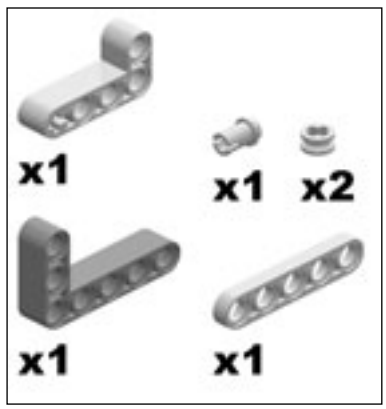


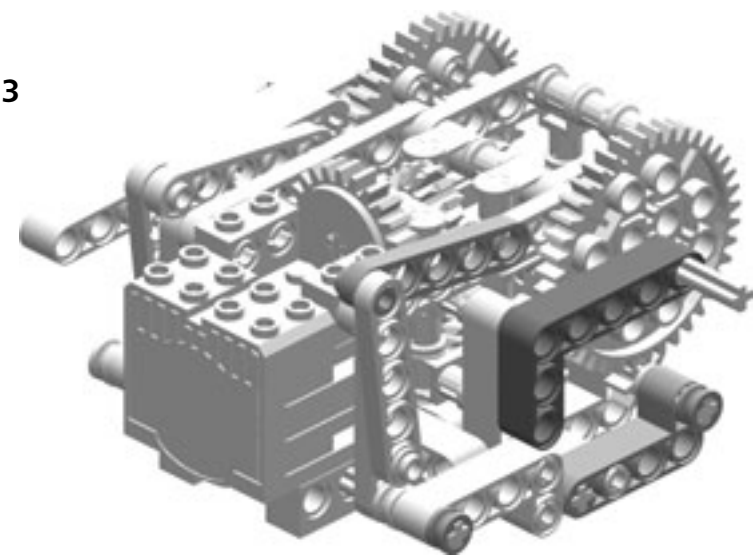
Folding Muscle Unit Step 22



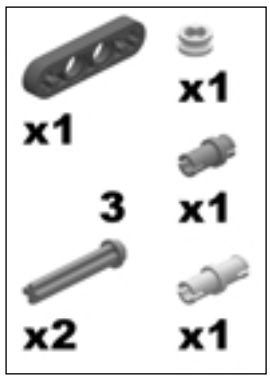


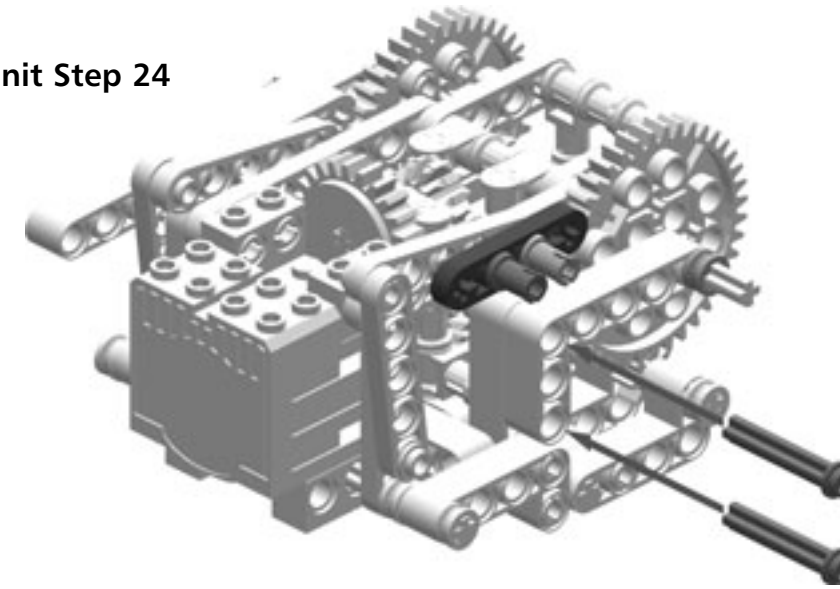
Folding Muscle Unit Step 23



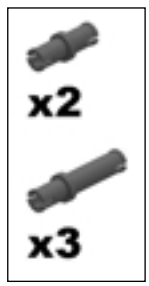


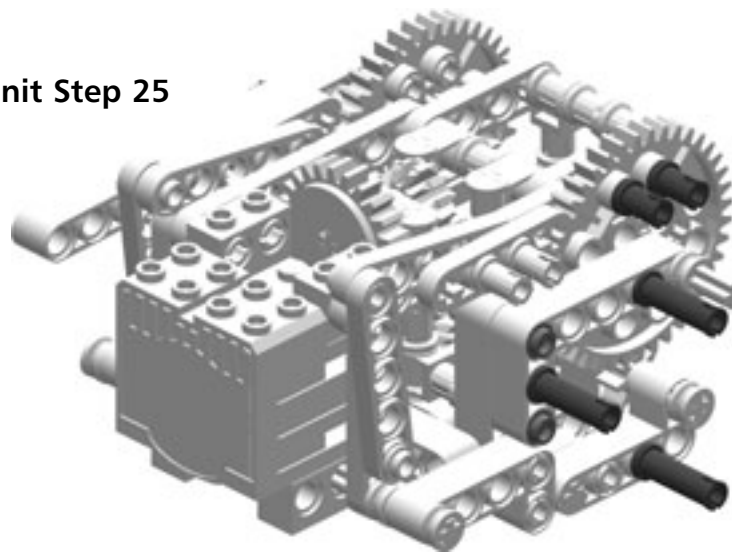
Folding Muscle Unit Step 24



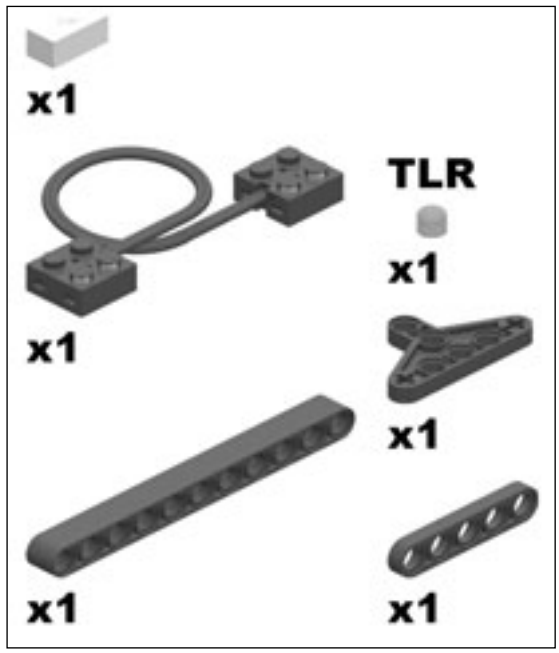


Folding Muscle Unit Step 25

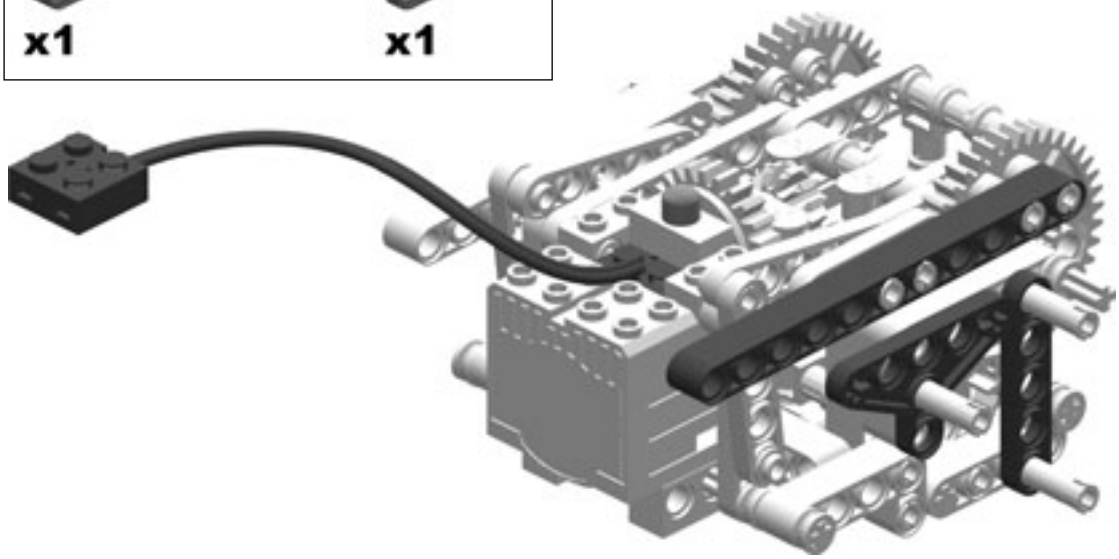




Folding Muscle Unit Step 26



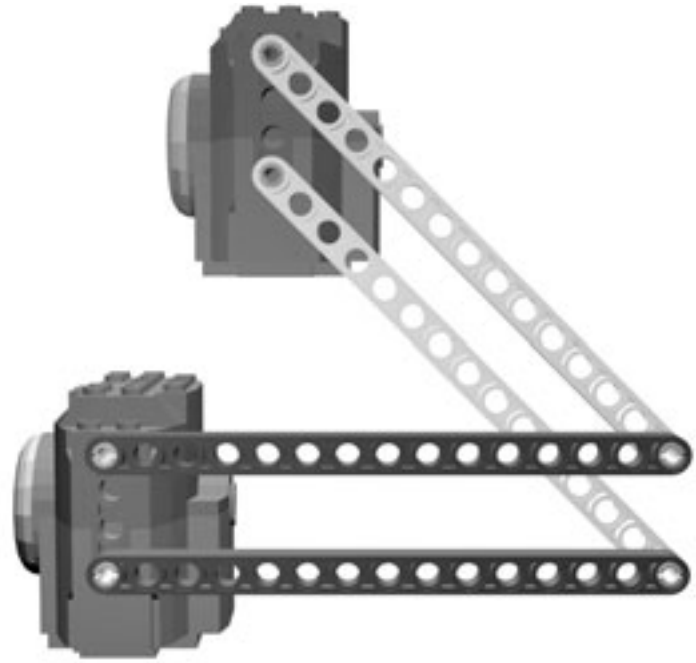
When assembling the complete robot, connect the motor of to Output Port C of the RCX. Since this sub-assembly is strong but slow, the light brick connecter to the power wire is very useful at least while building the robot. The light will flash if the motor is rotating one way and will remain on if it is turning the other. If the brakes are not yet attached and the mechanism binds, the light always tells us right away which way the motor is turning before things start to break.



Shape-Shifting Science: The Front Sub-Assembly

While the Front sub-assembly carries two of the most interesting apparatus on the SSCT, the RCX and the Vision Command camera, the mechanical devices on it are rather simple, but they make up for it by being spring-loaded. One spring-loaded device keeps the camera pointed forward at all times, independent of the shape of the robot. This is accomplished using a variation of the parallelogram linkage. This linkage uses two independent arms to raise and lower a platform while keeping it level. As shown in Figure 6.8, if the two arms are parallel to each other, the platform at the moving end (the camera in this case) is kept pointing forward while the arms rotate, elevating it.

Figure 6.8 A LEGO Parallelogram Linkage



In the SSCT, the lower arms of the parallelogram linkage have been substituted with rubber bands, while the top arms are collapsible (these also act as a mechanical brake for the muscle mechanism). The effect can be best appreciated once you put the whole model together: look laterally at the camera while the vehicle raises and lowers its front end and refer to Figure 6.8 to clearly see how the implementation works.

The other spring-loaded mechanism present on the Front sub-assembly is a *track tensor*, a device that keeps the track tense. Large tracked LEGO vehicles always benefit from this type of mechanism for a number of reasons. The larger and heavier the vehicles become, the harder is to find the right length of track that will provide the right traction over uneven terrain, without skipping gears or breaking apart the tracks by decoupling links. This becomes paramount in the case of the SSCT where the tracks not only propel and drive the vehicle, but also change shape.

The building instructions for the Tread sub-assemblies can be found later in the chapter, but since the Front sub-assembly plays an enormous role in the shape-shifting characteristics of the SSCT, I will now introduce the science that governs the shape changes of the vehicle.

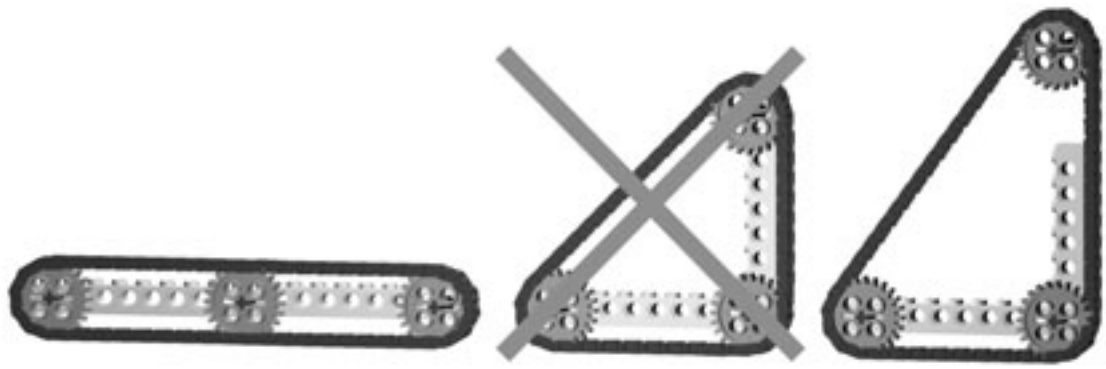
NOTE

Incidentally, a lot of the credit for the following material goes to my dad, Miguel Agulló Sr., who helped me make sense of it.

The shape-shifting mechanics of the SSCT and the original VGTV are significantly different. In the VGTV, all of the wheels occupy different positions relative to each other depending on the shape of the track. Among other things, this takes further advantage of the previously explained lever laws, allowing a better weight distribution that in turn enables a tiny wheelbase when the vehicle is upright. However, translating a similar mechanism to LEGO would have resulted in a bulkier robot; therefore, I took a completely different (and simpler) approach. In the SSCT, the back and middle wheels remain in the same position and at the same distance from each other. What about the front wheel? This is where the mathematics behind the shape-shifting tracks kicks in.

Figure 6.9 illustrates why mathematics play a paramount role when designing shape-shifting tracks. In the leftmost image, we have a simple set-up. A chain connects three gears that are attached to two 1x8 TECHNIC beams. The size of the track is 50 links. If we rotate the front beam 90 degrees, the model will look like the one in the middle image of Figure 6.9… or will it? The fact is that the chain of the model in the middle has only 44 links. A 50-link chain actually allows us to place the wheel, when in this position, significantly higher. The chain of the figure at right has 50 links, and is as long as the one at left.

Figure 6.9 A Simple Shape-Shifting Scheme



Thus, in order to design a shape-shifting chassis, we need to take into account, from the beginning, the length of the track that will be used with it. Later in the chapter, we will briefly discuss what alternatives exist when it comes to tracks, but for now we will assume that the track is make of standard LEGO TECHNIC tread links, which create non-elastic but highly customizable chains (in terms of number of links used).

To design a successful shape-shifting tracked vehicle, we must be able to model the behavior of the tracks in order to adjust the chassis accordingly. As we can see in Figure 6.9, a simple folding operation of the chassis calls for further adjustments in other parts of the system to keep up with the changes in the shape of the tracks.

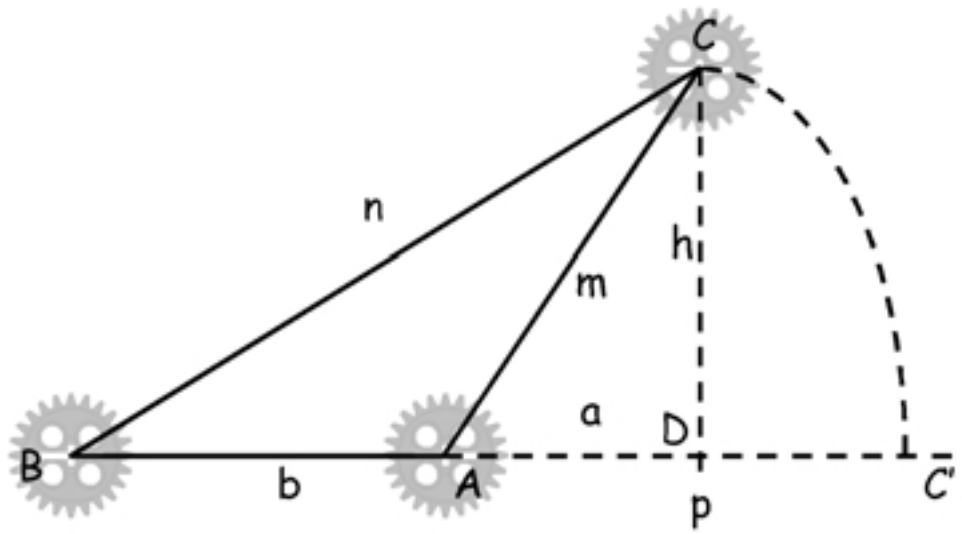
Figure 6.10 shows how we can calculate the position of the front wheel at any given rotation angle of the front assembly for a pre-set length of track. The triangle ABC is the

shape of the track (to compute the actual length, add the necessary links to complete encircle one gear to this figure). Since the position of the back and middle wheel is fixed, it is known. We also know that the length of the tracks equals:

$2b + 2p$

When the front wheel is down, the track goes from gear A to gear C (segment p) once at the bottom and returns again at the top (p+p = 2p); and then from gear A to gear B (segment b) and back again (b+b=2b).

Figure 6.10 Shape-shifting Math



Once we start lifting the front gear C off the ground, we need to adjust its position to keep the track in tension. The optimal position is equal to the lengths of a and h – how do we find them? We do know the total sum of b+m+n (the length of the track) and we also know the length of b and p. Using that data, we can find a and h by applying the Pythagorean Theorem to the triangles defined by BCD and ACD.

$$n^2 = (a+b)^2 + h^2$$

or

$$n = \sqrt{(h^2 + (a+b)^2)}$$

and

$$m^2 = a^2 + h^2$$

or

$$m = \sqrt{(a^2 + h^2)}$$

If the length of AB does not change, we know that n+m = 2p+b, thus:

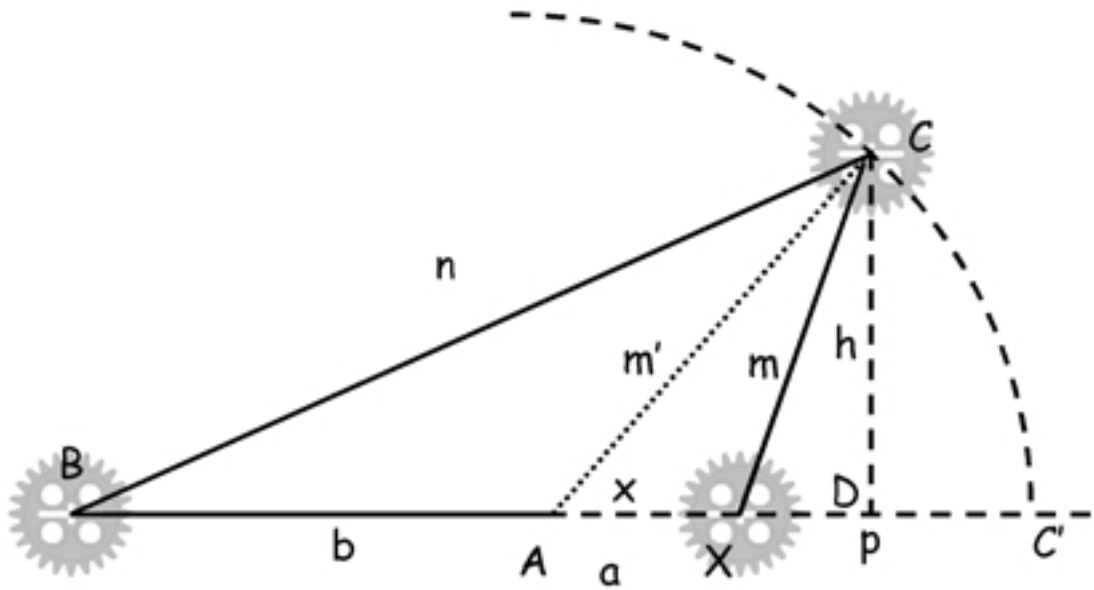
$$\sqrt{a^2 + h^2} + \sqrt{h^2 + (a+b)^2} = b + 2p$$

Since b and p are known, for each value of a, we will get a corresponding value of h. While this might seem a bit steep for the non-mathematically inclined, it can be coded into an spreadsheet program that will calculate the values for h and a. A sample spread sheet can be found on the companion CD-ROM. Plotting them on a graph will produce an accurate rendition of the trajectory of the front wheel for any track length and wheel distance.

From a more practical point of view, one alarming by-product of changing the shape of the tracks this way is that the distances between the wheels need to vary quite a lot in order to keep the tracks in tension, as we saw in Figure 6.9. If the tracks become too loose, the tank will cease to move. One way to shorten that distance is to move the rotation axis of the lifting arm between the back and middle wheels. As we can see in Figure 6.11, this transforms the above formula into the following:

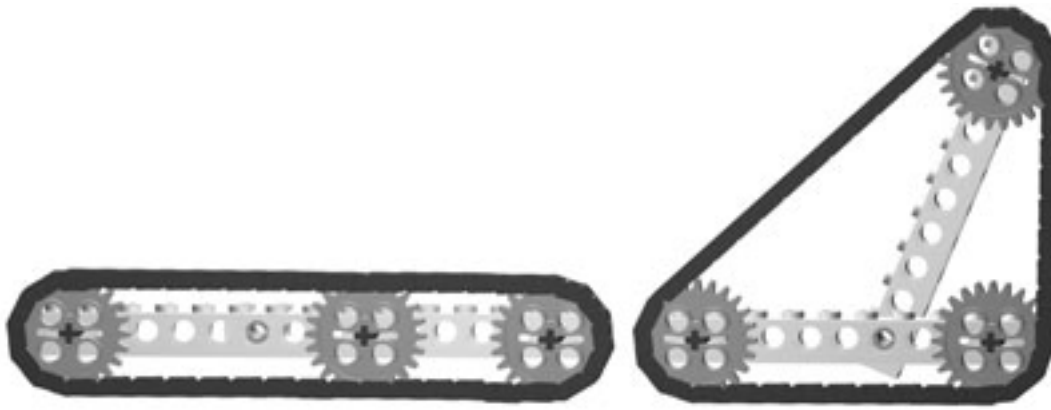
$n+m = 2p+b-x.$

Figure 6.11 Keeping the Chain Tight

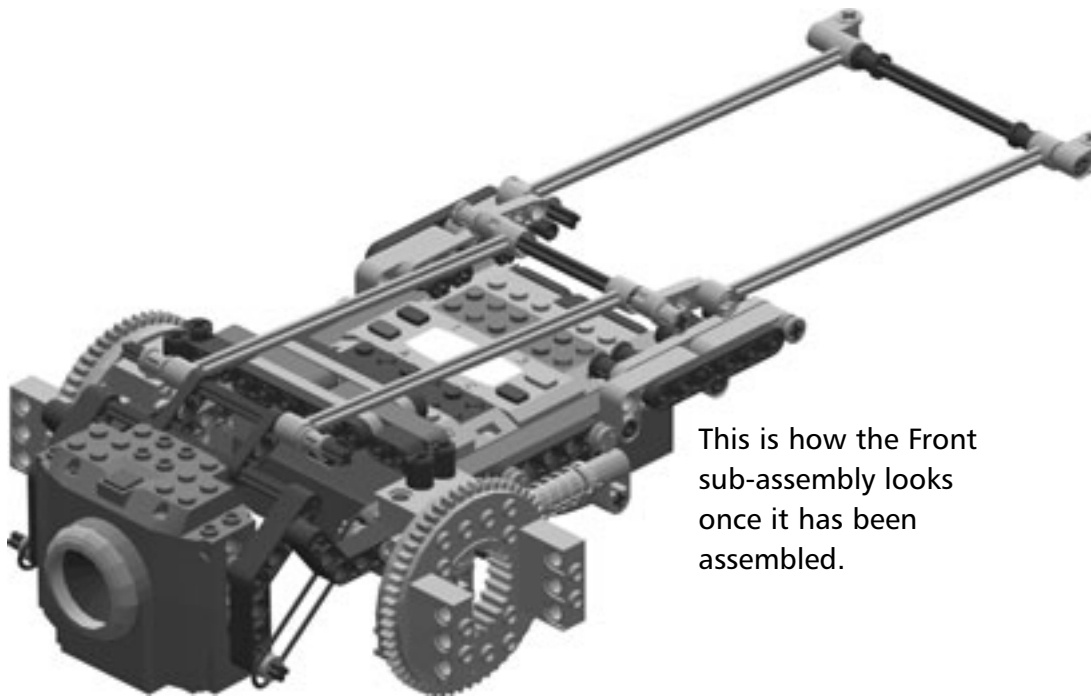


In effect, the distance introduced between the middle wheel and the rotation axis of the lifting mechanism translates directly into the available length of the chain. Figure 6.12 shows a simplistic scheme of such a solution, which is similar to the one used in the SSCT. Both set-ups use exactly 43 chain links, and do not require readjusting the front wheel with respect to the chassis as the vehicle folds and the track changes shape.

Figure 6.12 The SSCT Solution



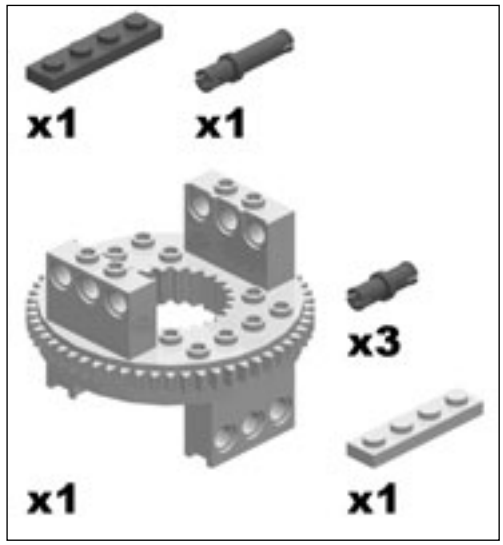
Building The Front



This is how the Front sub-assembly looks once it has been assembled.

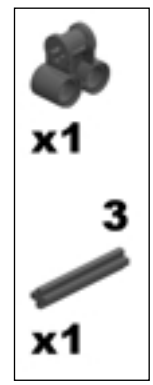
As we will see in more detail in the next section, the larger the elements we use, the easier it is to create strong structures with them. They also allow us to add plenty of details to the assembly. Once completed, watch how the track tensioning system and the camera linkage work. The simplicity of the mechanisms allows us to sneak them in without much trouble. This sub-assembly is perhaps the most customized part of the vehicle and thus harder to use as-is in other vehicles. An alternative mentioned at the end of the chapter is to build a structure that keeps the RCX and the camera (or a light sensor) level with the floor as the robot changes shape. In this design, only the camera is kept leveled.

Front Step 0



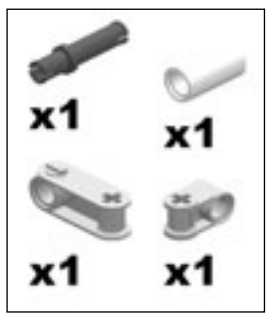
The following section will address how the turntables play a key role in the chassis of the SSCT. In the current sub-assembly, the RCX provides a very stable base, and the turntables and camera are an equally strong base for the mechanisms attached to it.

Front Step 1

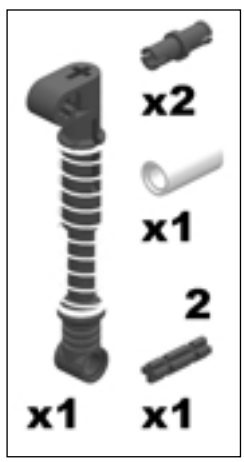


The turntables are designed so that it is very easy to attach both the more traditional elements (stud-based) and the newer ones (pin-based) as well.

Front Step 2

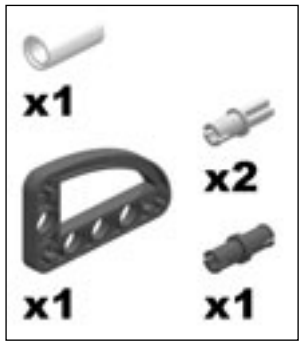


Front Step 3



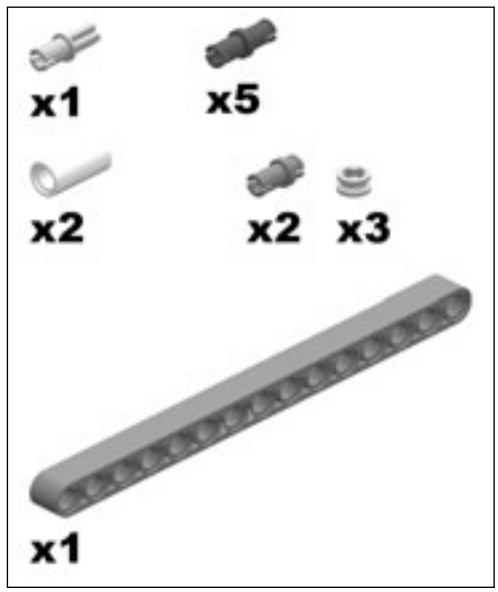
You can use either of the two types of LEGO long shock absorbers for the track tensor. The older black ones are softer, and you might want to consider making the track slightly shorter than indicated to compensate for this. The gray shock absorbers (available in more recent kits), offer more resistance. Attach the long friction pins—don't worry about the extra space left. Once the sub-assembly is complete, the shock absorber will stay in place.

Front Step 4



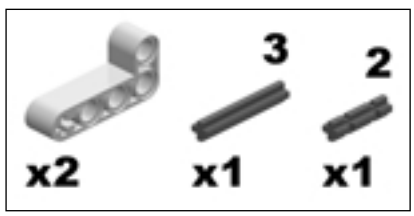
The L-shaped liftarms with quarter oval are part of the newer LEGO releases and come half-stud width!

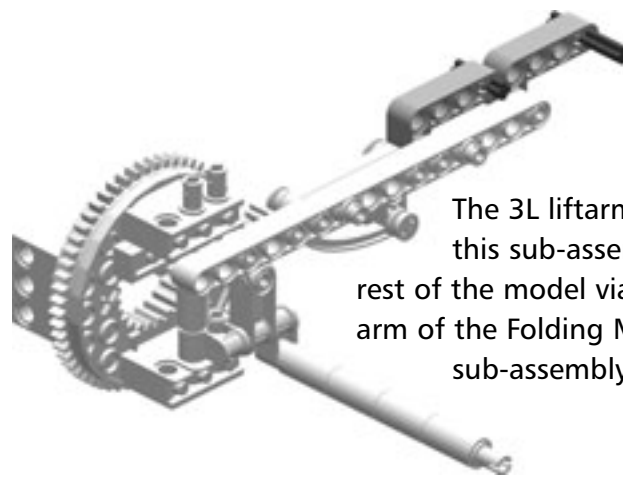
Front Step 5



Notice how the round hollow parts often used to cover axles are actually pin-joiners.

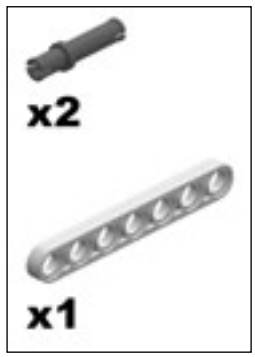
Front Step 6

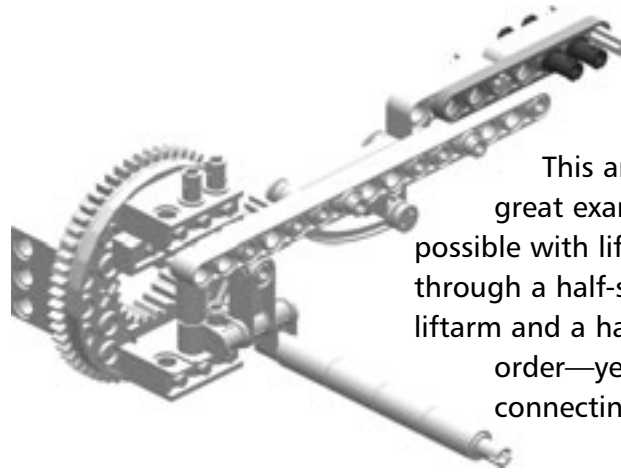




The 3L liftarms connect this sub-assembly with the rest of the model via the rotating arm of the Folding Muscle Unit sub-assembly.

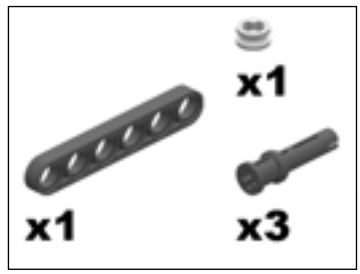
Front Step 7

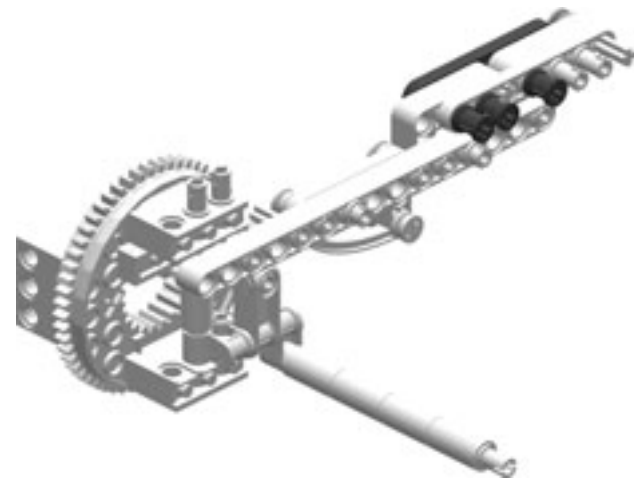




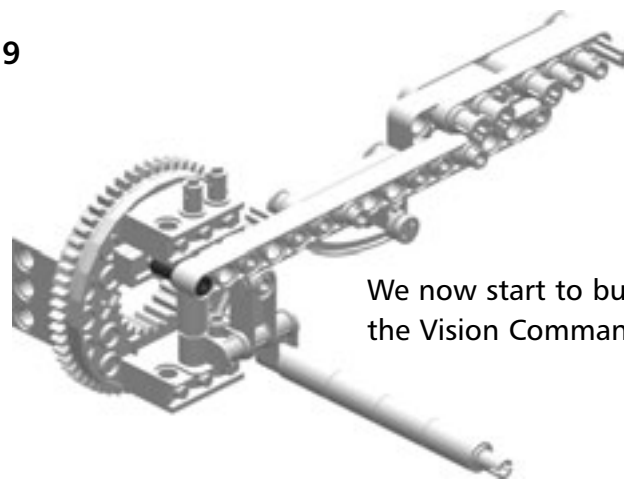
This and the next step are a great example of an assembly only possible with liftarms. Insert a long pin through a half-stud liftarm, a one-stud liftarm and a half-stud liftarm, in that order—yet another way of connecting liftarms.

Front Step 8



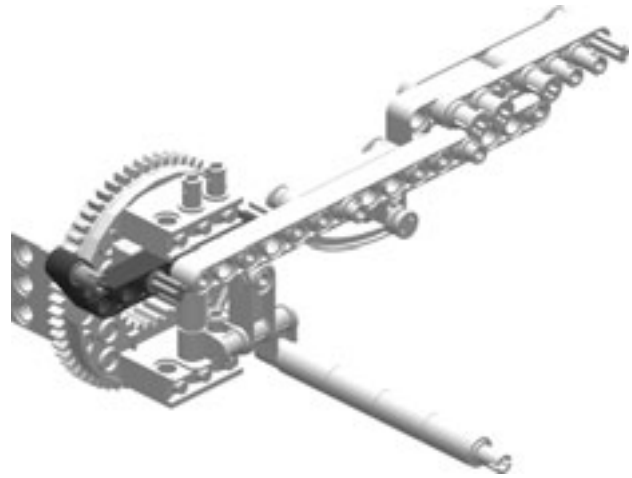


Front Step 9



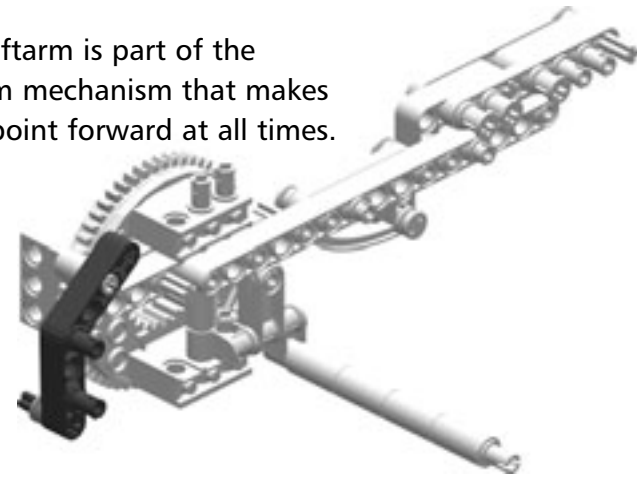
We now start to build the cradle that holds the Vision Command camera.

Front Step 10

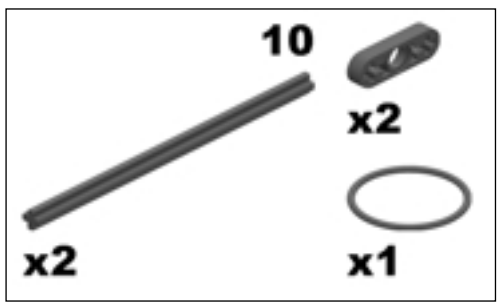


Front Step 11

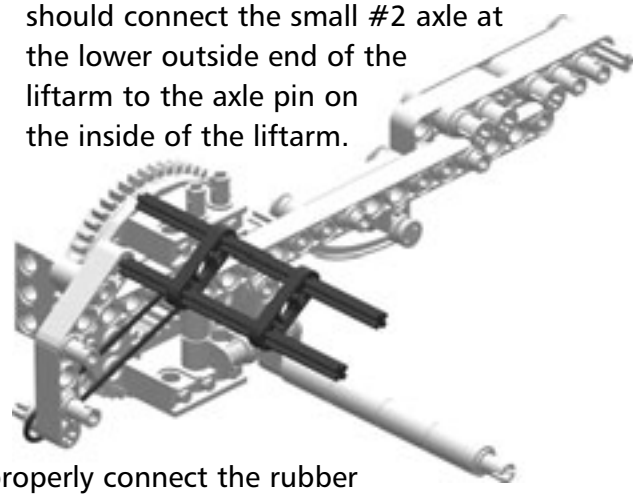
The angled liftarm is part of the parallelogram mechanism that makes the camera point forward at all times.



Front Step 12

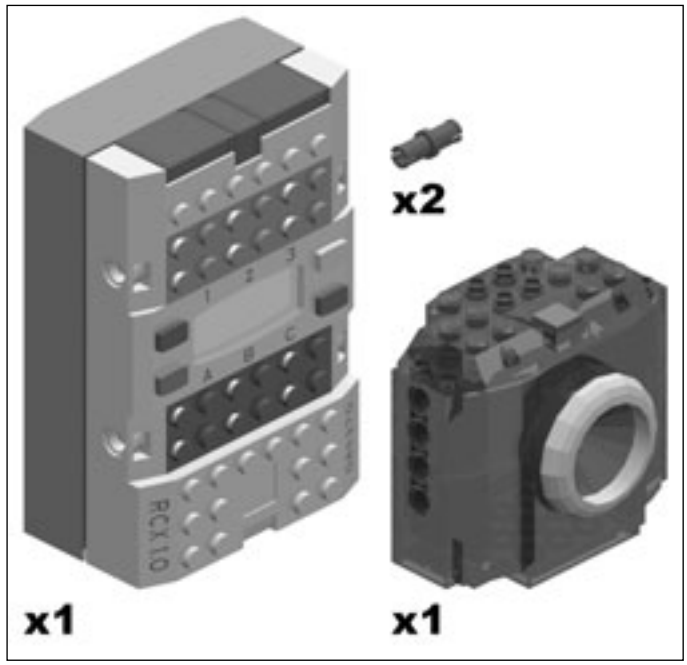


In this step, you should use a small black rubber band from LEGO. This rubber band should connect the small #2 axle at the lower outside end of the liftarm to the axle pin on the inside of the liftarm.

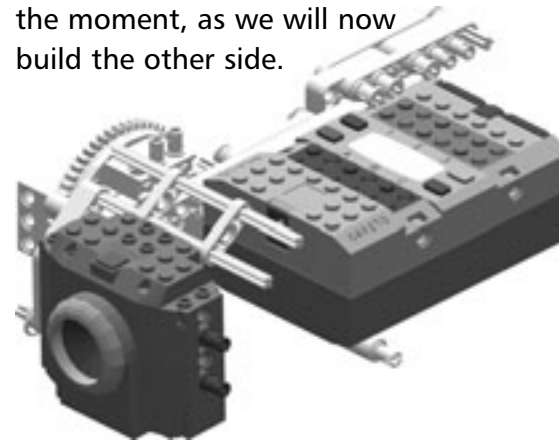


Front Step 22 shows a view of a similar assembly from the opposite angle. If you are unsure on how to properly connect the rubber band, please review that step.

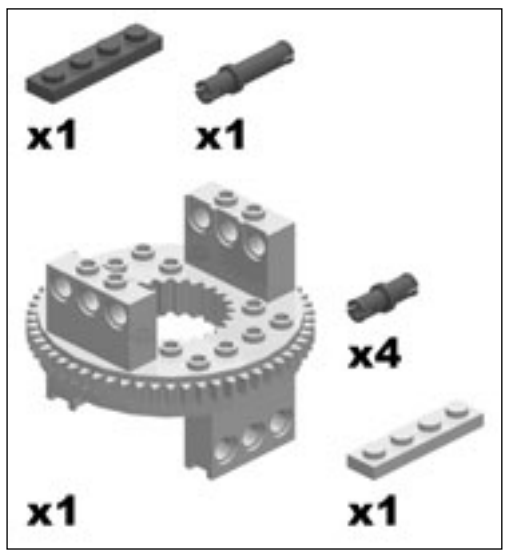
Front Step 13



Add the RCX and the Vision Command camera. Set this half of the sub-assembly aside for the moment, as we will now build the other side.

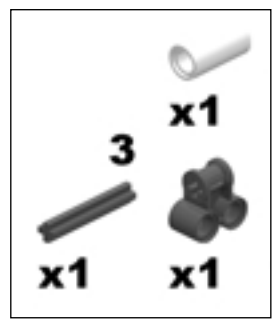


Front Step 14

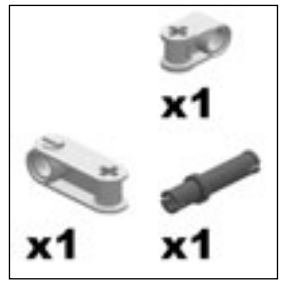


You will now build the other side of the Front sub-assembly. This side is symmetrical to the side you just finished building.

Front Step 15



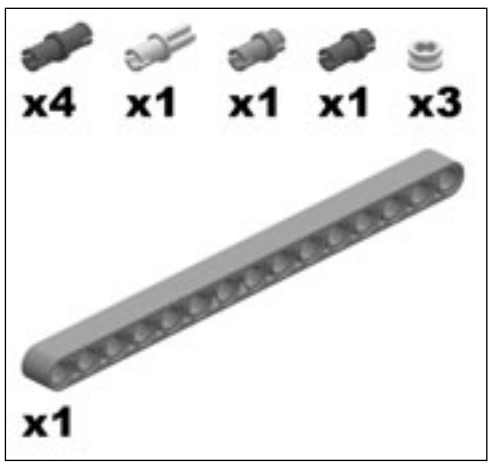
Front Step 16

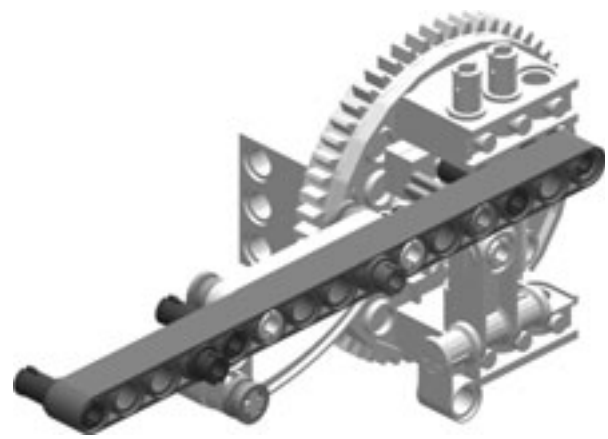


Front Step 17

Front Step 18

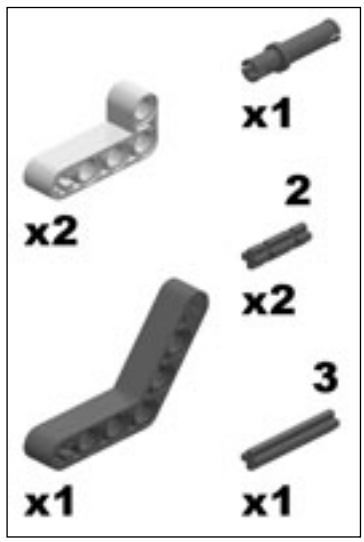
Front Step 19

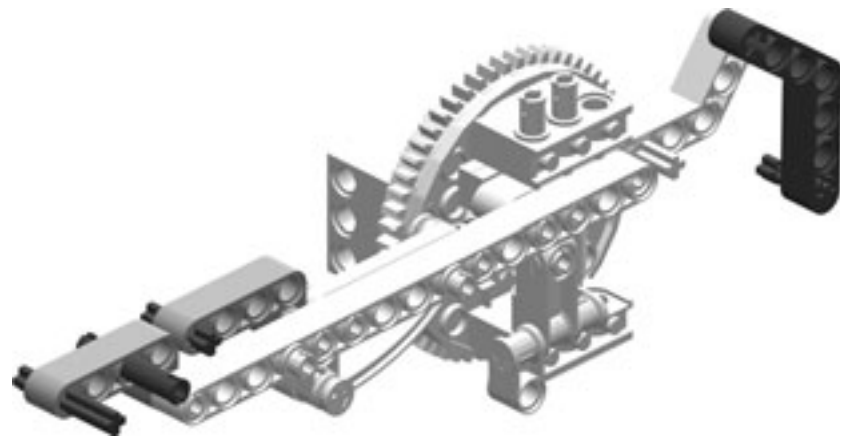




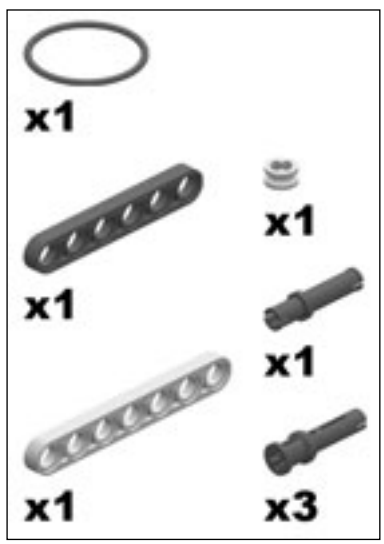
Front Step 20

Front Step 21

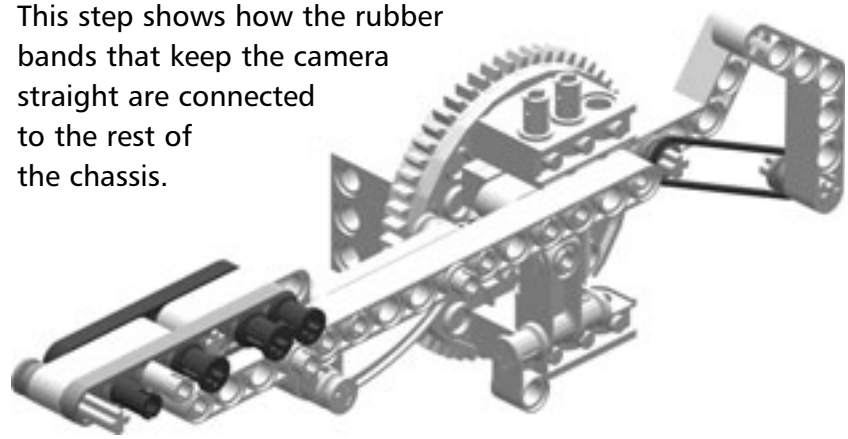




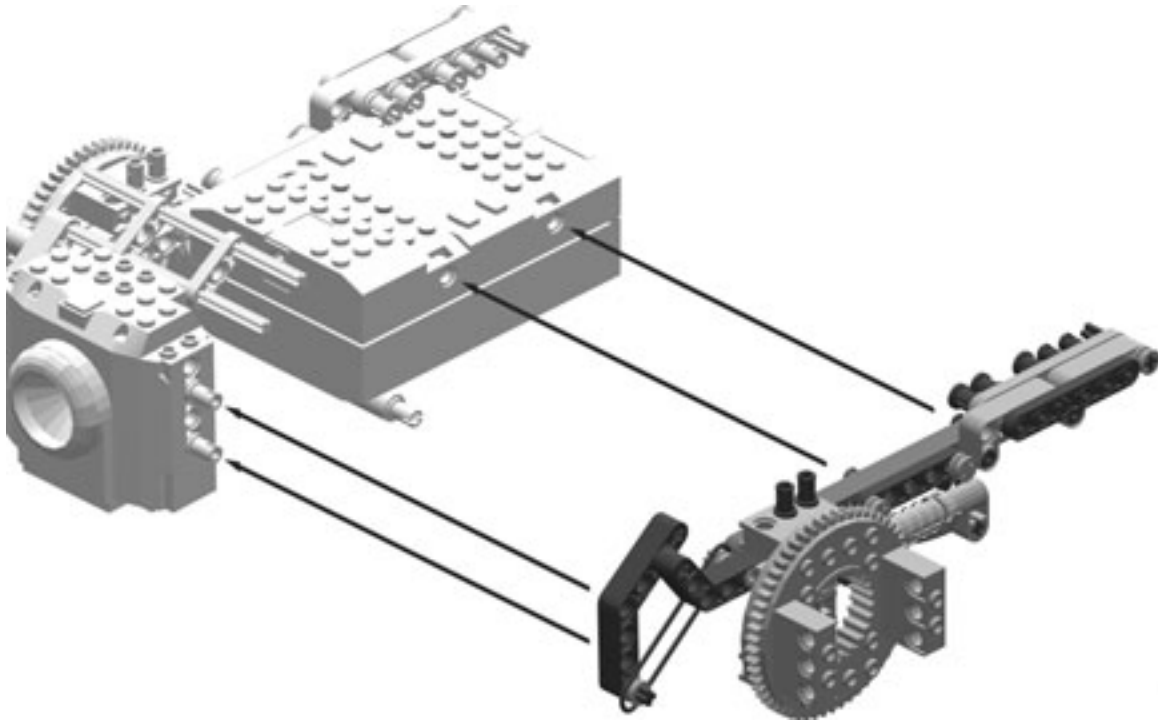
Front Step 22



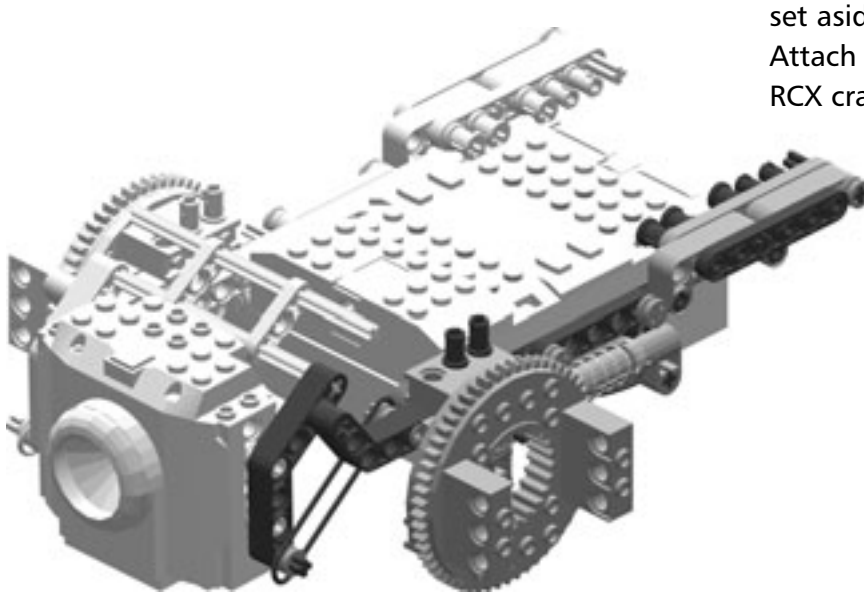
This step shows how the rubber bands that keep the camera straight are connected to the rest of the chassis.



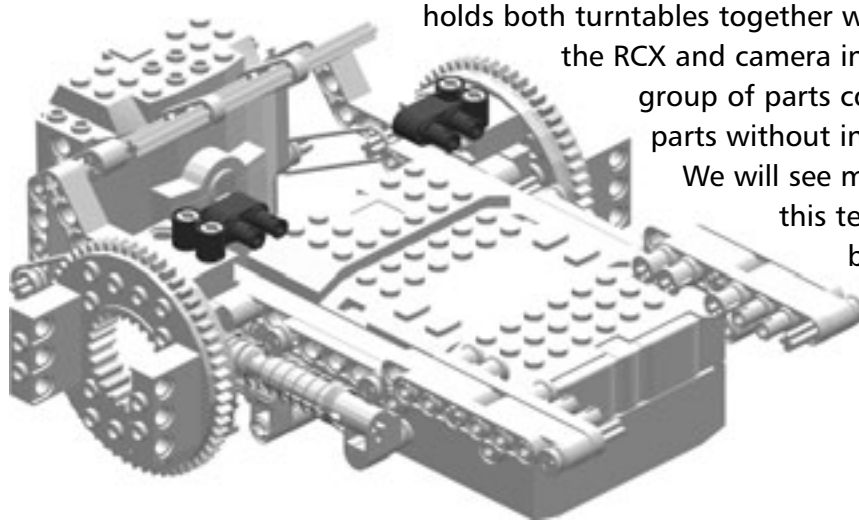
Front Step 23



Retrieve the portion of the Front sub-assembly that you set aside in **Front Step 13**. Attach the other side of the RCX cradle as shown.

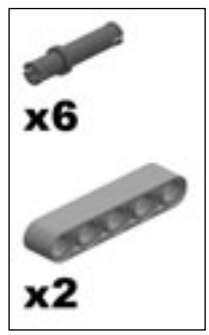


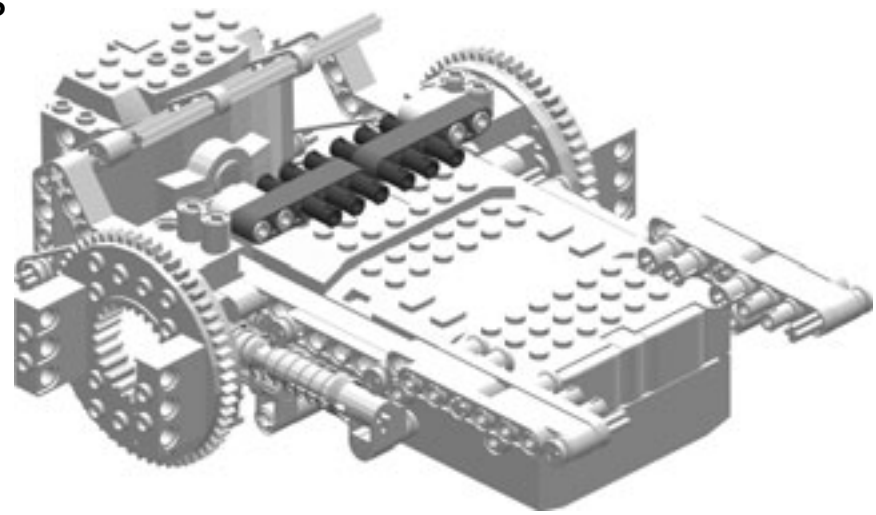
Front Step 24



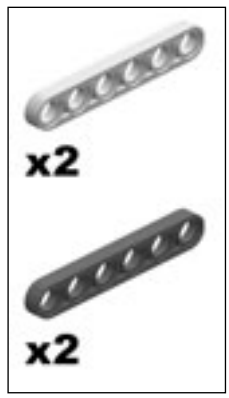
In the next few steps, we will build a lock that holds both turntables together while sandwiching the RCX and camera in between. This group of parts connects with other parts without interlocking them. We will see more examples of this technique when building the chassis in the next sections.

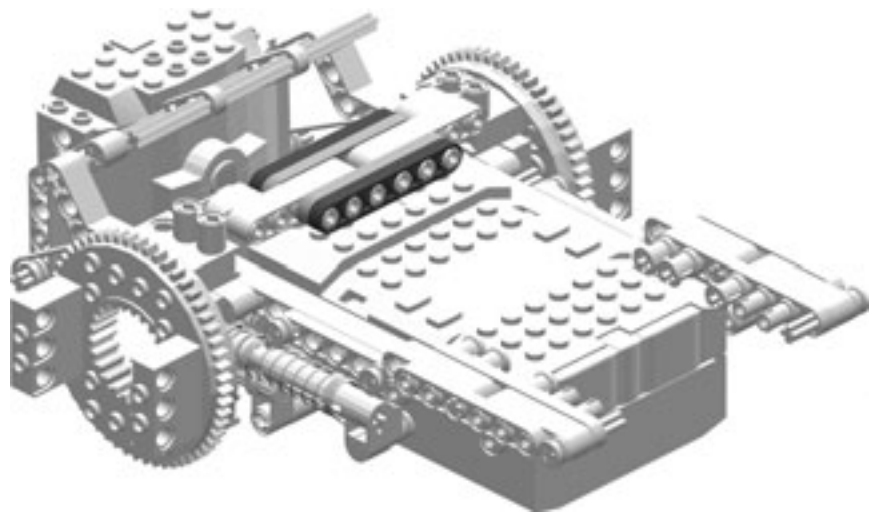
Front Step 25



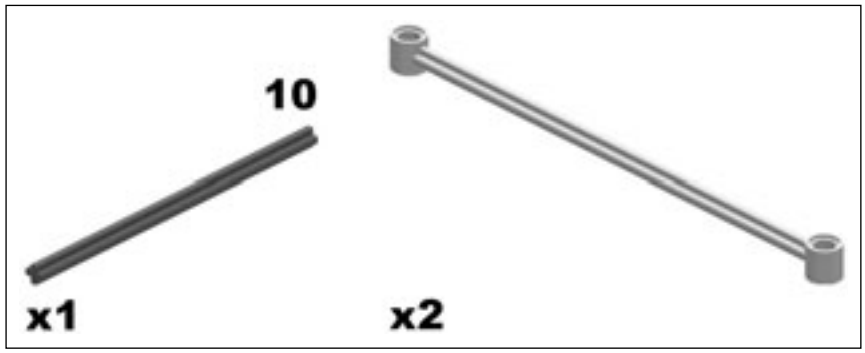


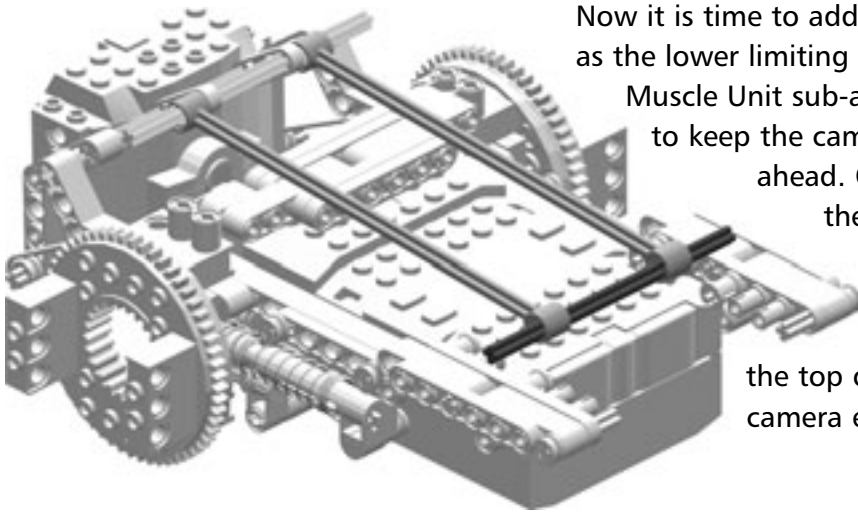
Front Step 26





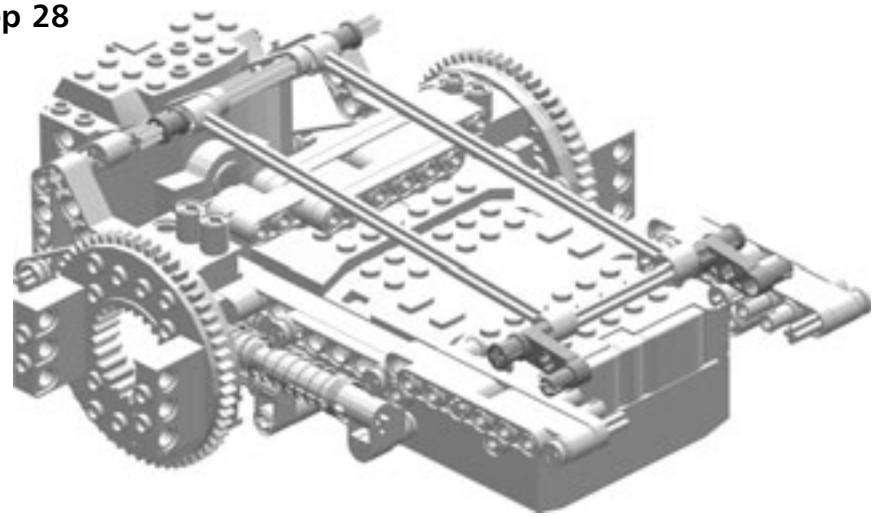
Front Step 27



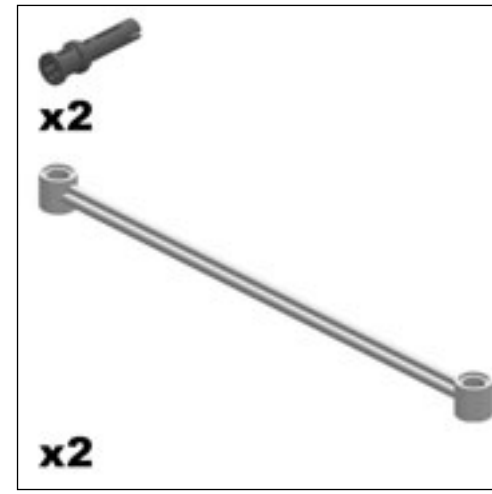


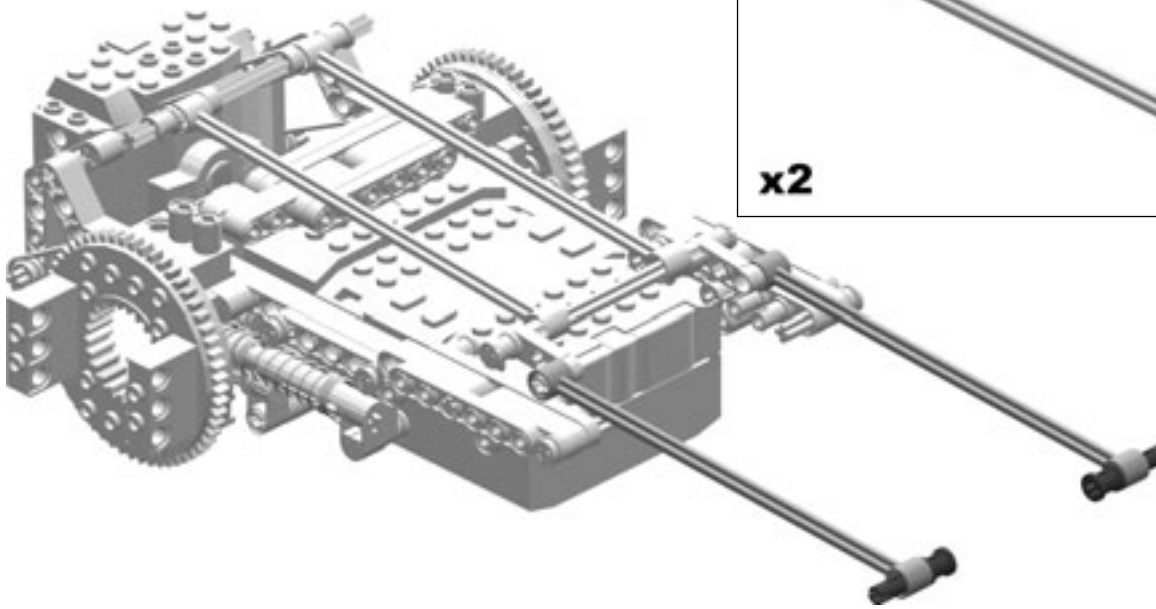
Now it is time to add the long linkages that act as the lower limiting mechanism for the Folding Muscle Unit sub-assembly as well as linkages to keep the camera pointing straight ahead. One advantage of using these links is that they are less rigid that other LEGO parts. This allows them to adapt well to the top of the vehicle when the camera end is down.

Front Step 28

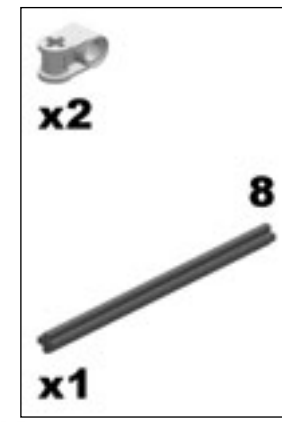


Front Step 29





Front Step 30



The Sides of the Vehicle: A Turntable-Based Chassis

Once we have built the sub-assemblies that will create the main body, we need to bring them together. You'll notice that there are still four turntables missing, not to mention a chassis to hold everything together. The fact is that the turntables configure the actual chassis. These elements attach to the sides of the chassis (two for each side). In this particular model, these two sides are merely an afterthought, which is a wonderful way to deal with them. Using turntables as wheels was born out of necessity, as they are they only current TECHNIC gears with appropriate height to enclose the RCX and camera. A side benefit is that using them for wheels adds a tough look to the vehicle, which goes well with the character of an exploring machine. However, their best advantage is that they are a world apart from the rest of the LEGO gears when it comes to integrating them into a vehicle chassis.

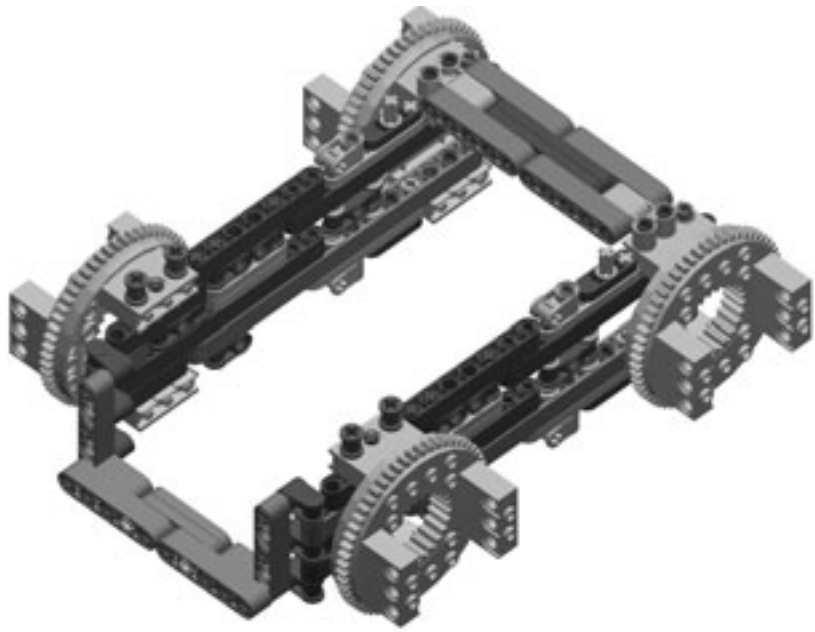
Most gears, like most wheels, attach to the chassis. The turntables are both the chassis *and* the wheels. To quantify what this means, simply imagine the common gear attachment: A gear connected to an axle that goes into one of the holes of a liftarm or TECHNIC brick. Now imagine how many ways you could torture that axle with your hands: don't underestimate the power of the Folding Muscle Unit sub-assembly as it looks for ways to skip the heavy job of lifting the Front sub-assembly. Next, attach a brick or liftarm to a turntable with a couple of friction pins and try torturing that structure using your bare hands! There are ways, but it won't be easy and you won't get far. The reason you will not be successful is sheer bulk: examine the four thick protuberances (more than arms) that stick out of the massive gear, and the axle (so thick, that it can afford to be hollow in the middle). All this makes most assemblies that include turntables equally as strong.

The pressure and lever mechanisms are familiar: you are simply pushing and pulling against more material. Using equal force, the pressure is smaller because the area is larger. An important thing to remember is that when you oversize an element to withstand pressure, you might simply be shifting the load further down the chain. The makers of LEGO cranes and other mechanisms designed to lift extraordinary loads are familiar with the stripping of 8T LEGO gears. Stripped gears are those that lose their teeth. If a high torque mechanism binds, pressure can quickly build up where a small area has to deal with strong forces: the teeth of the small 8T gear (with an intrinsic low mechanical advantage) fit that description particularly well. Even the bracing of the gear with the axle is actually larger than the surface of contact between gear teeth. The next thing you know is the teeth are popping off the 8T gear as the binding forces strip it. That is a fatal injury, and the gear cannot be repaired.

If you use turntables as the track's wheels of a heavy vehicle like the SSCT, simply figure out where the turntables need to be situated with respect to the rest of the components. Rest assured that you will find any number of convenient ways to solidly attach two turntables to create the sides of the vehicles and then sandwich the rest of the

subassemblies between them. As you can see in Figure 6.13, the turntables *define* the chassis of the SSCT, providing basic support to the rest of the elements.

Figure 6.13 The Base Chassis of the SSCT is Built around the Turntables

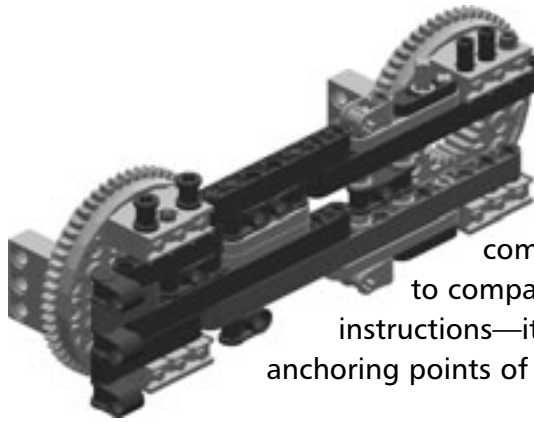


Bricks & Chips…

A Note About the Left and Right Sides of the Chassis…

The Right and Left Side sub-assemblies do not use any classic TECHNIC bricks, but this is the one area where I recommend using them over liftarms. Since you are building the basis of the vehicle, you might as well make it as strong as you can. This is easier to achieve with the classic bricks. Since liftarms are easy to attach to turntables, you might want to incorporate both elements into an alternative design and take advantage of the attachment shown previously in Figure 6.3.

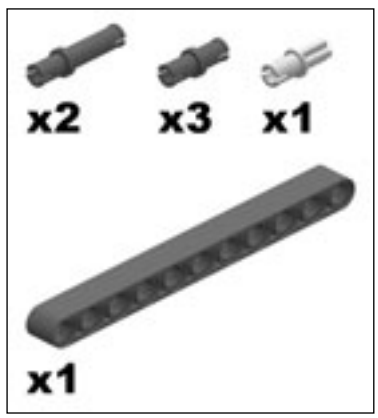
Building The Right Side

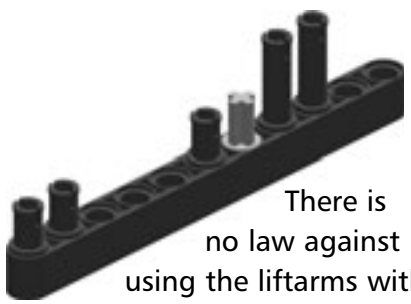


This is how the sub-assembly looks when completed. You might want to compare it to the final assembly instructions—it can help you locate the anchoring points of the other subassemblies.

The Right Side sub-assembly and the Left Side sub-assembly are simple constructions that will form the sides of the chassis. It is a matter of attaching the very strong turntables with the parts of the correct length to match the anchoring points of the previous sub-assemblies. Keep in mind that the following series of instructions are suggestions only. I recommend that you experiment with these suggestions and attempt to optimize the chassis on your own. This is especially true if you are lacking some of the parts required—experiment with what you have before acquiring more for the sake of a simple chassis.

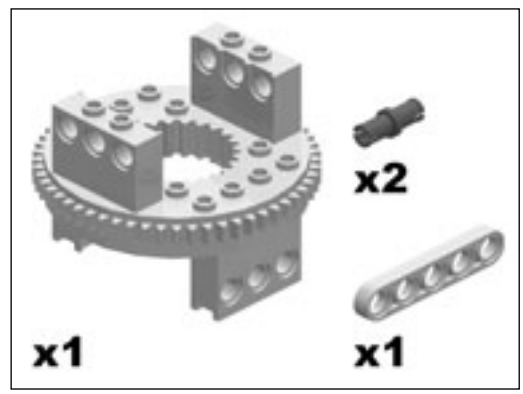
Right Side Step 0

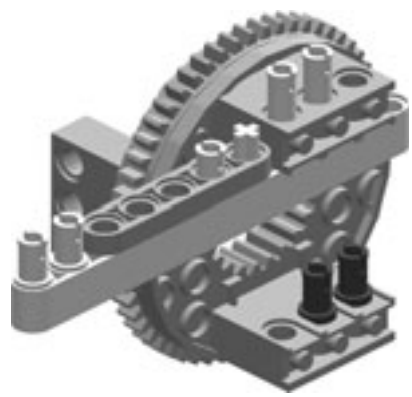




There is no law against using the liftarms with the holes looking up.

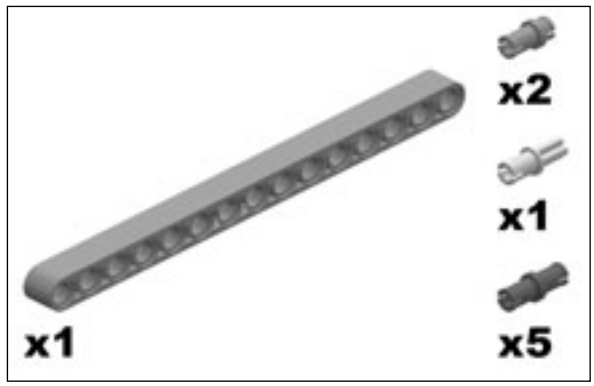
Right Side Step 1

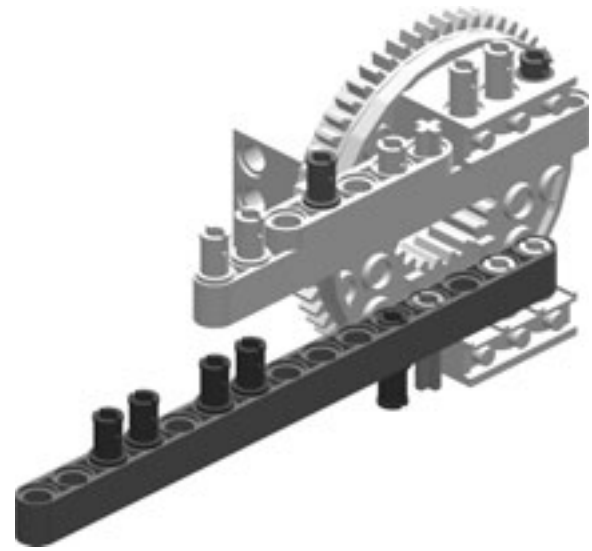




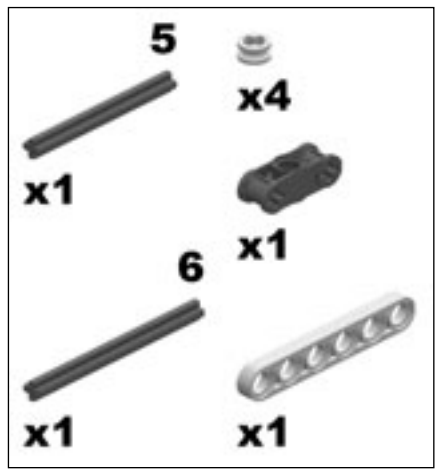
Using this orientation, they are easy to attach to the turntable.

Right Side Step 2

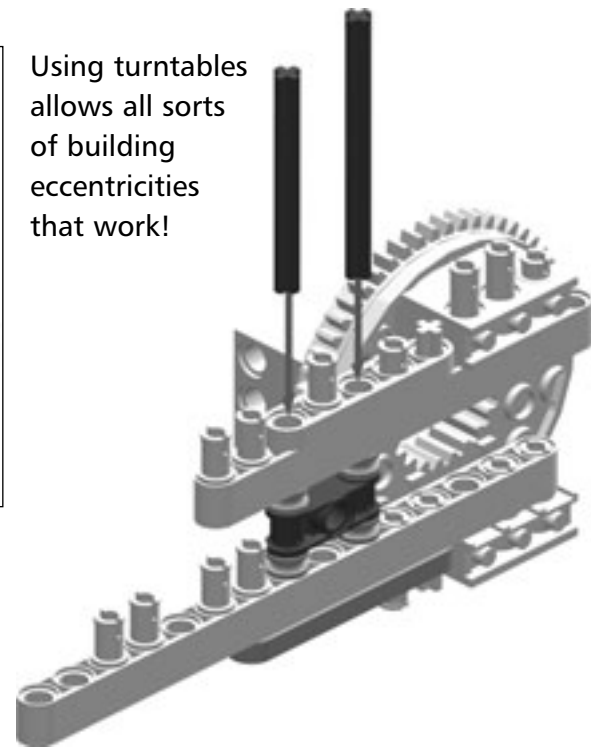




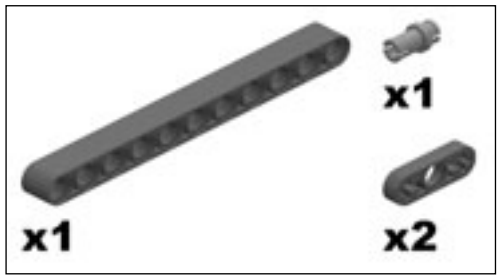
Right Side Step 3



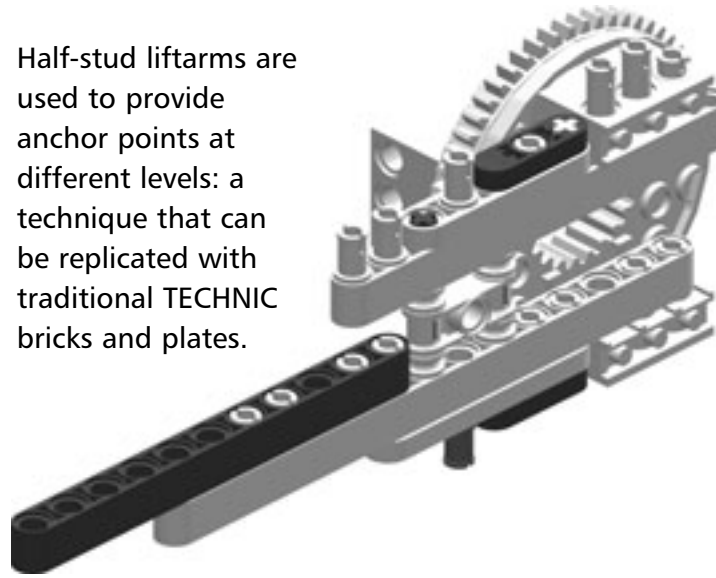
Using turntables allows all sorts of building eccentricities that work!



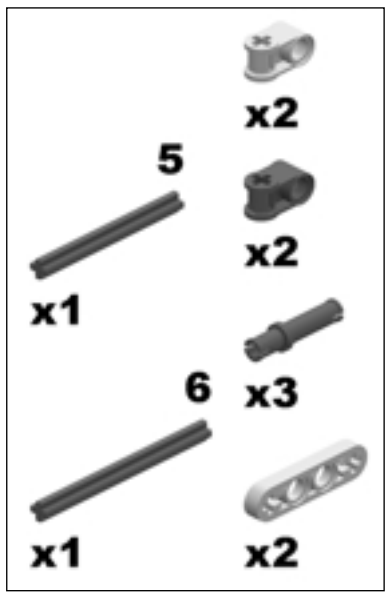
Right Side Step 4

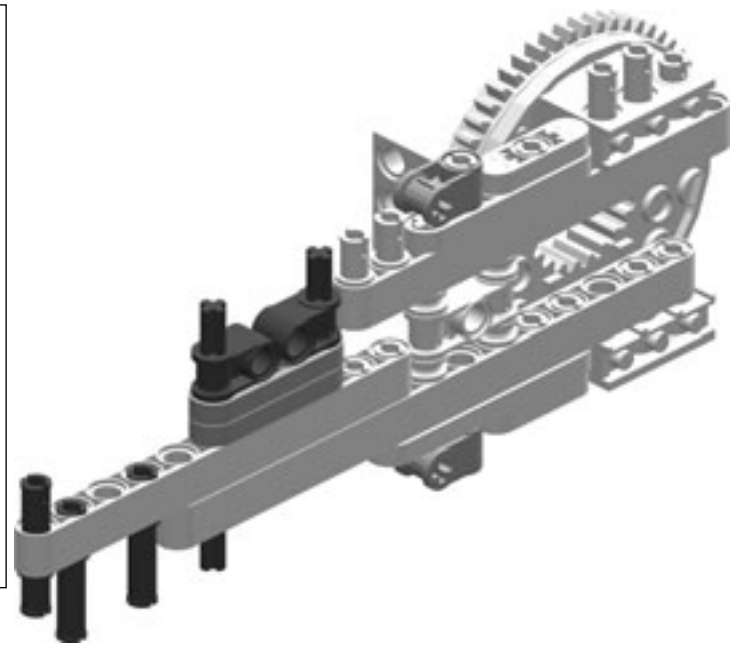


Half-stud liftarms are used to provide anchor points at different levels: a technique that can be replicated with traditional TECHNIC bricks and plates.

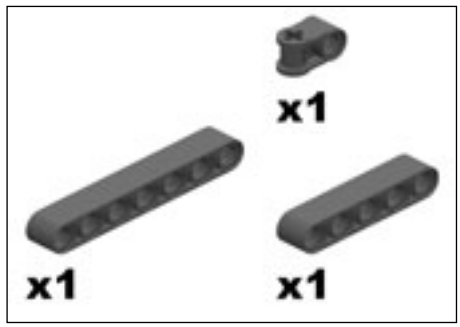


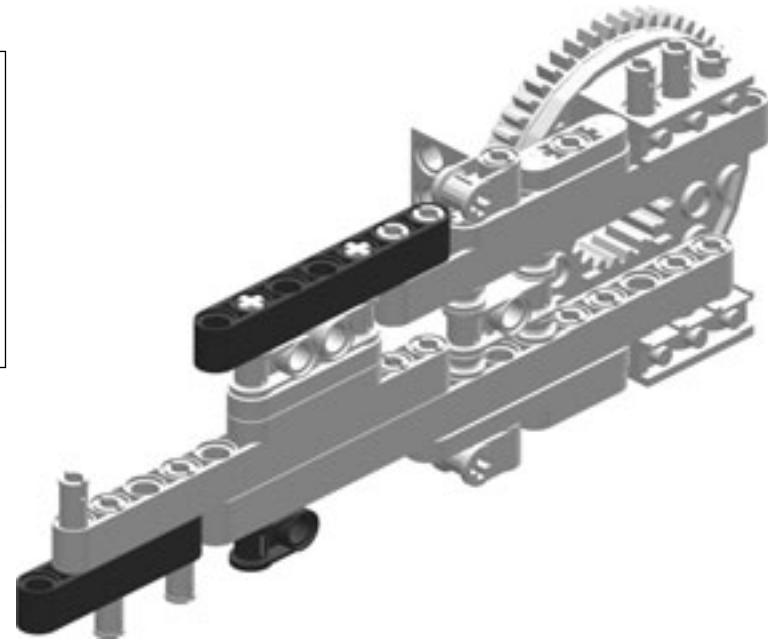
Right Side Step 5



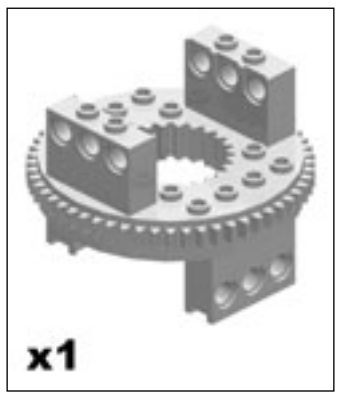


Right Side Step 6

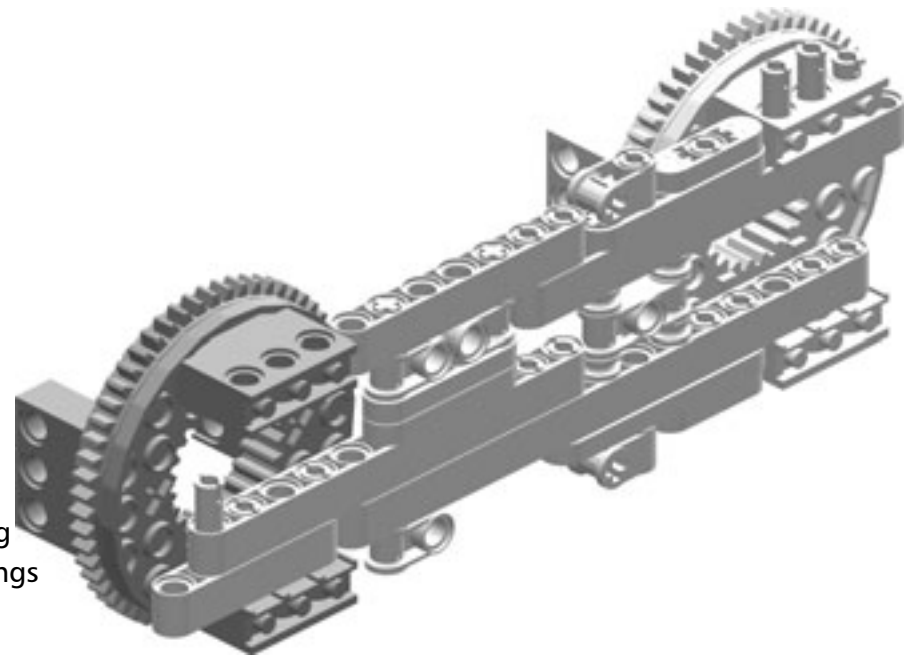




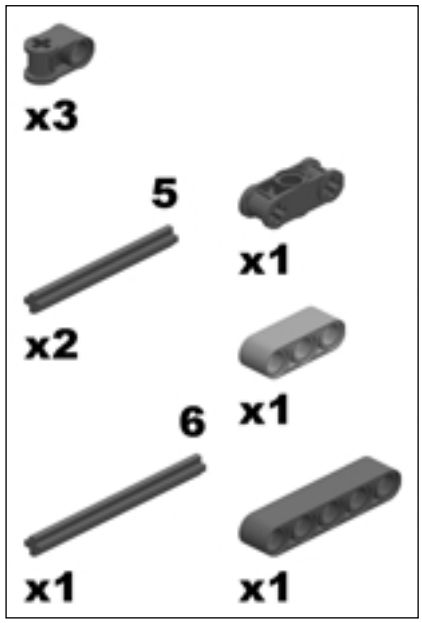
Right Side Step 7



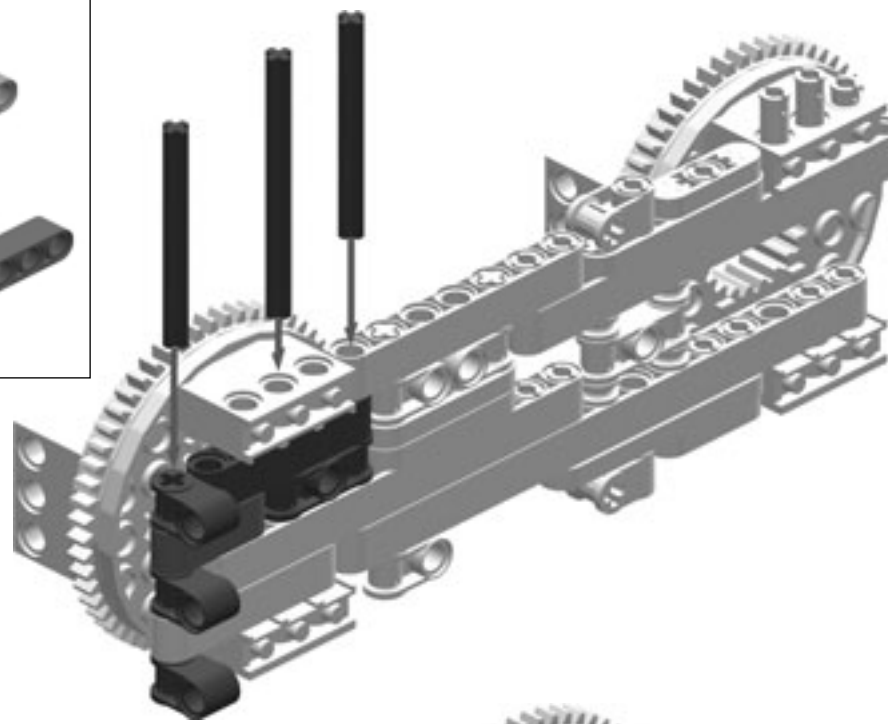
While the structure might seem a bit weak in the step above, adding the second turntable brings consistency back.



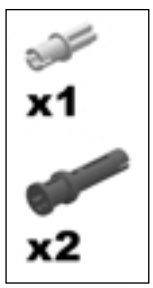
Right Side Step 8

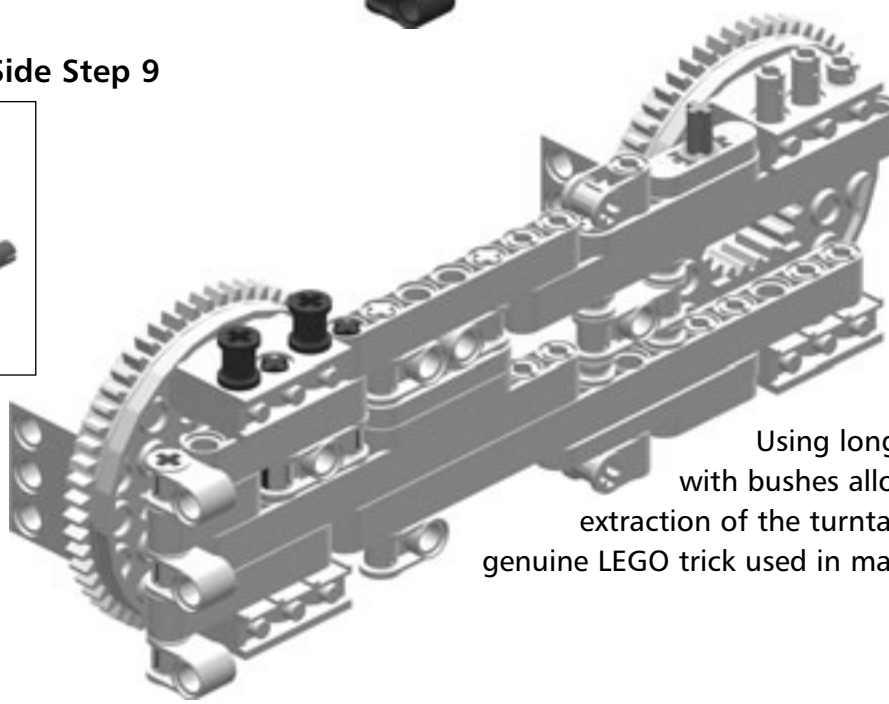


The connectors at the front will rotate at the axle, but once we add the lock in the final assembly, they will stay in place.



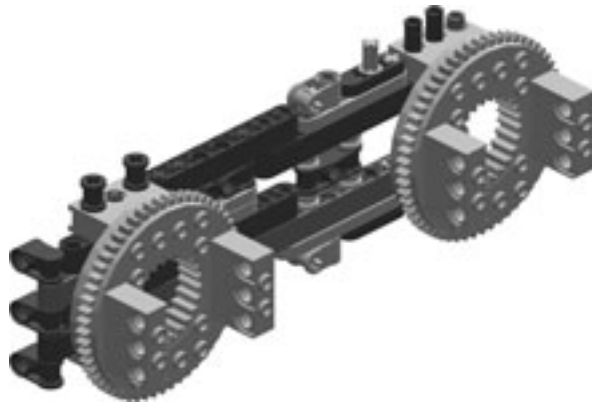
Right Side Step 9





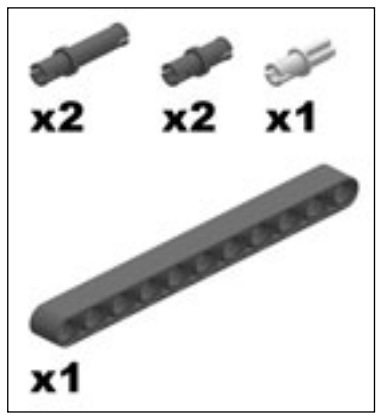
Using long friction pins with bushes allows easy extraction of the turntable. This is a genuine LEGO trick used in many official kits.

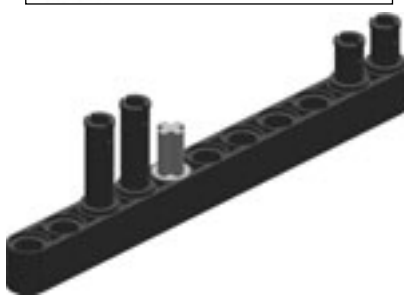
Building The Left Side



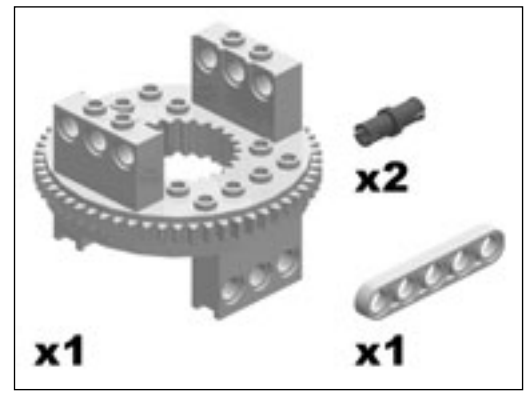
The Left Side sub-assembly is similar to the Right Side sub-assembly. The functional difference between these sub-assemblies is the side to which the turntables are attached.

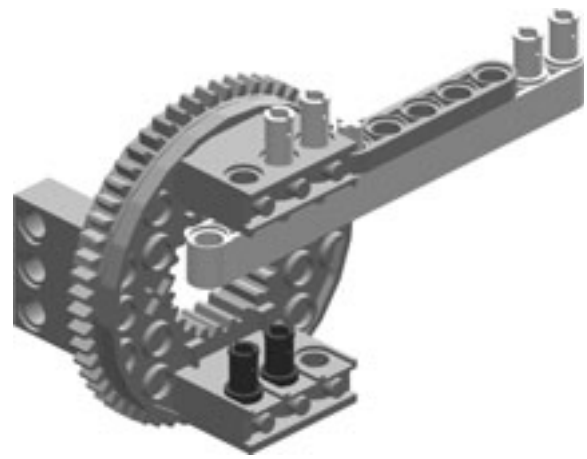
Left Side Step 0



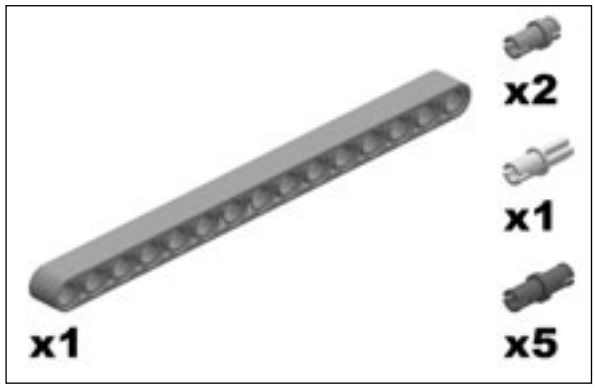


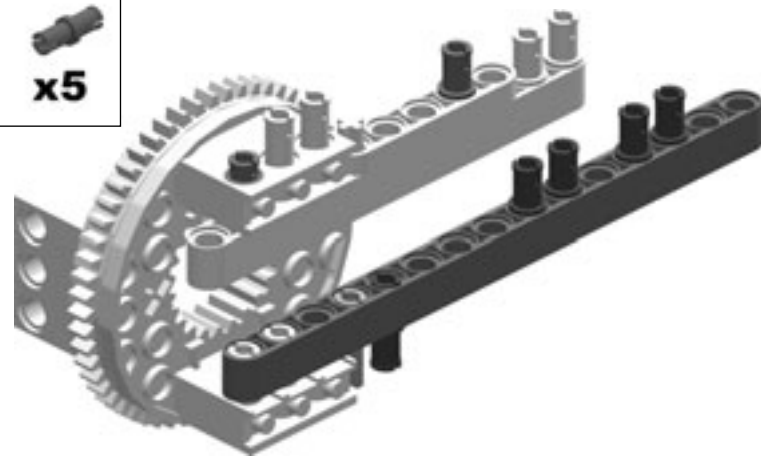
Left Side Step 1



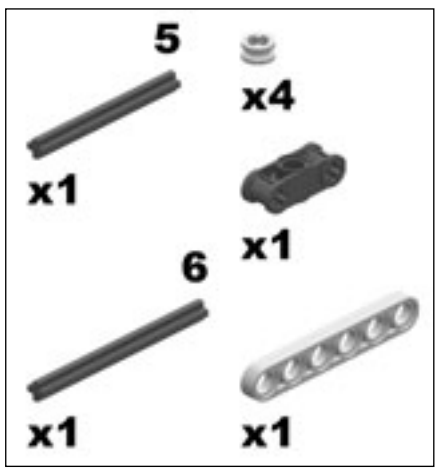


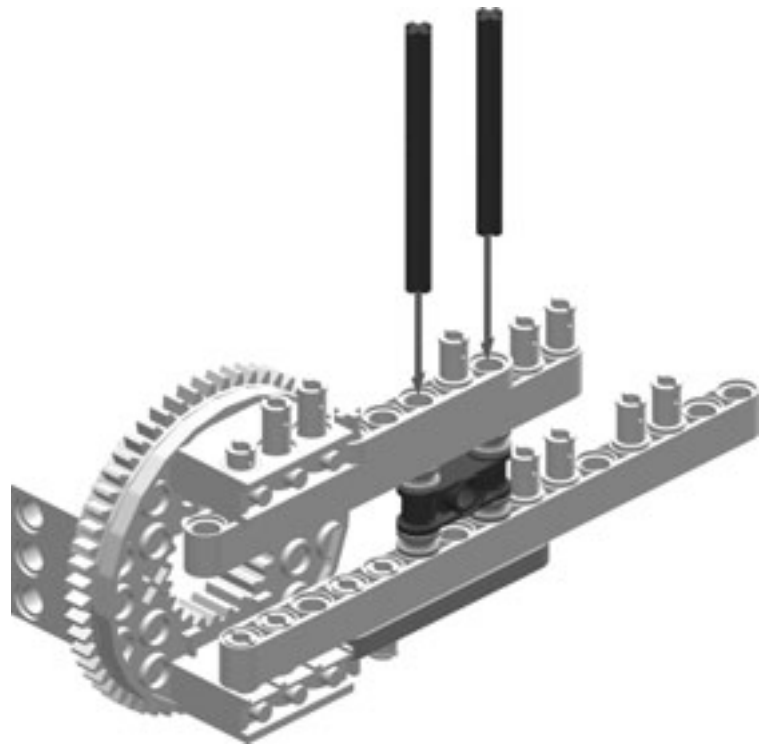
Left Side Step 2



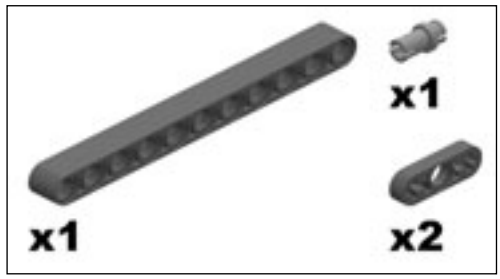


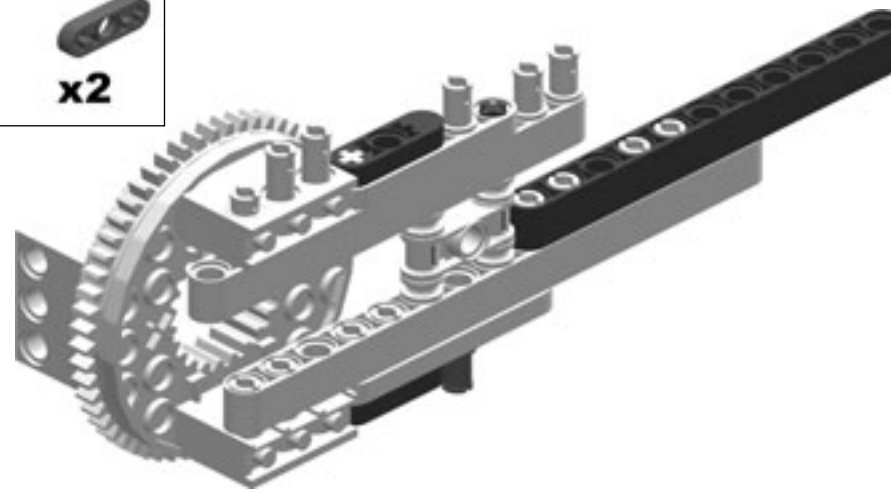
Left Side Step 3



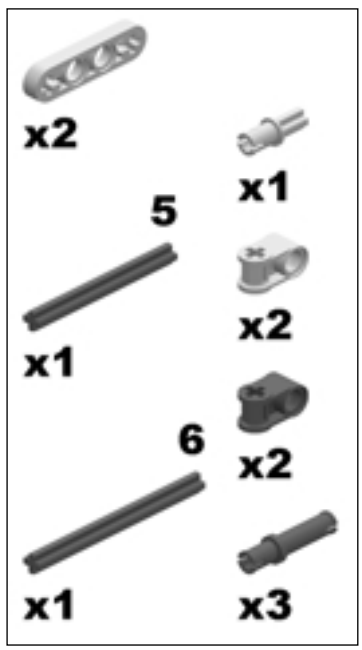


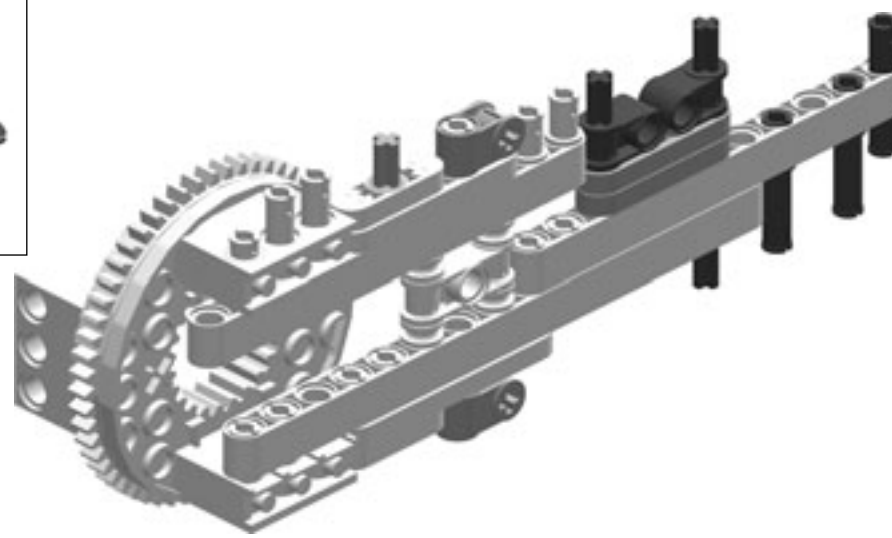
Left Side Step 4



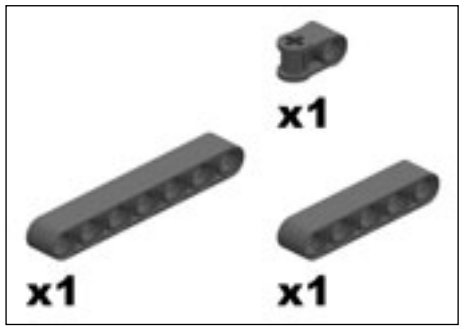


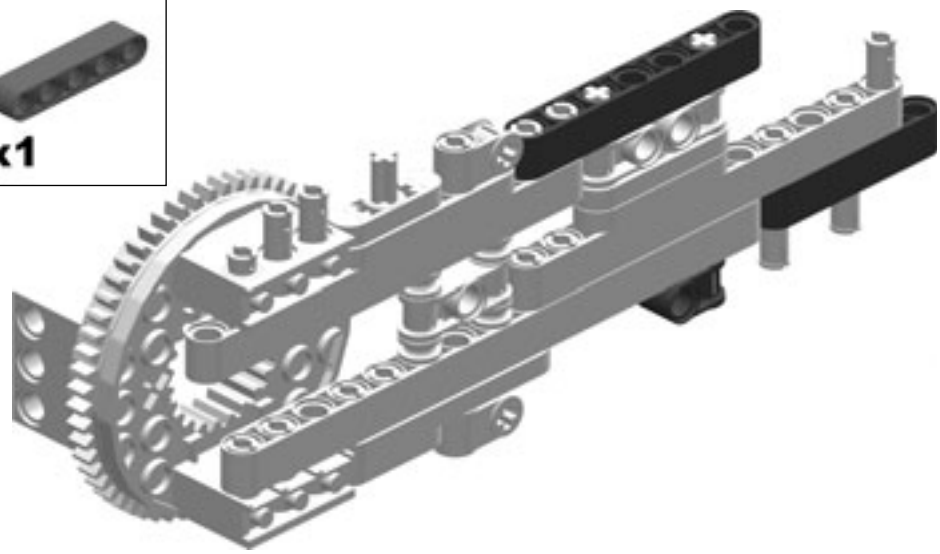
Left Side Step 5



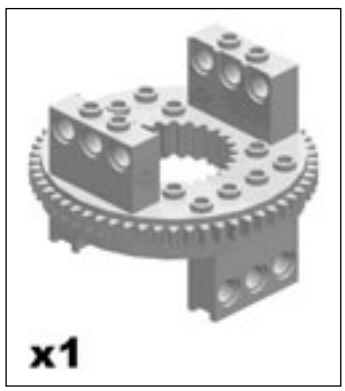


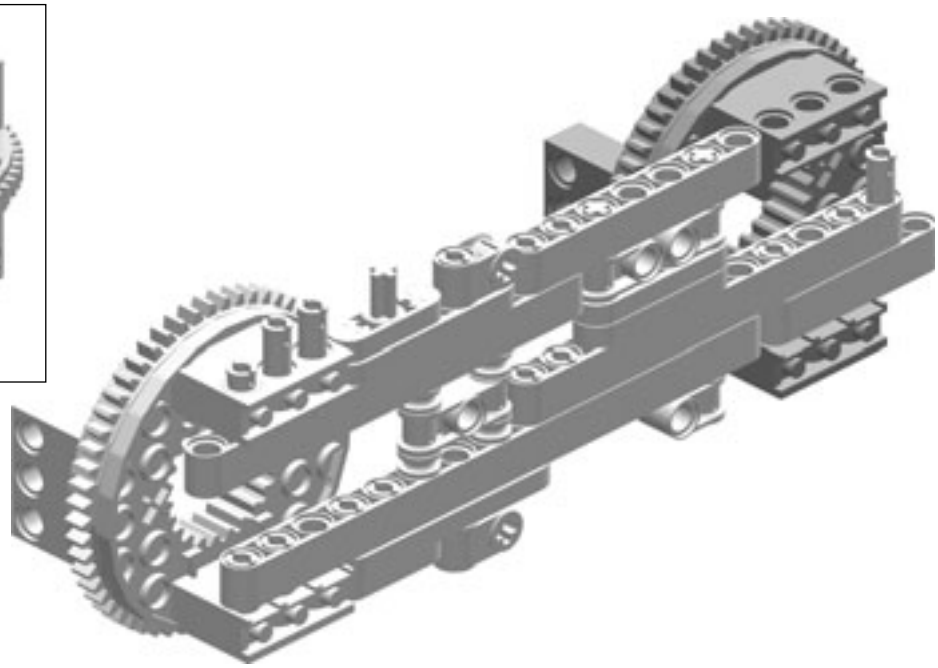
Left Side Step 6



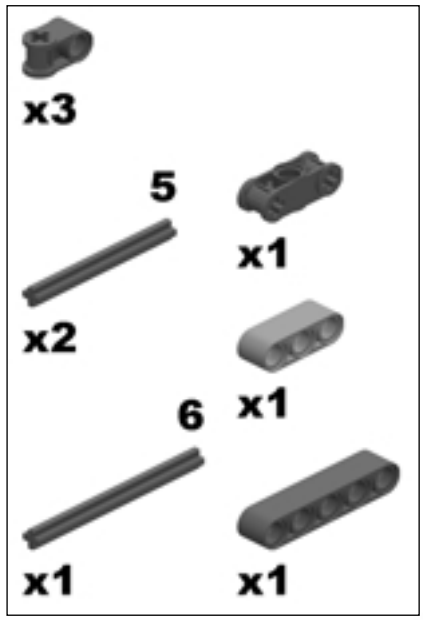


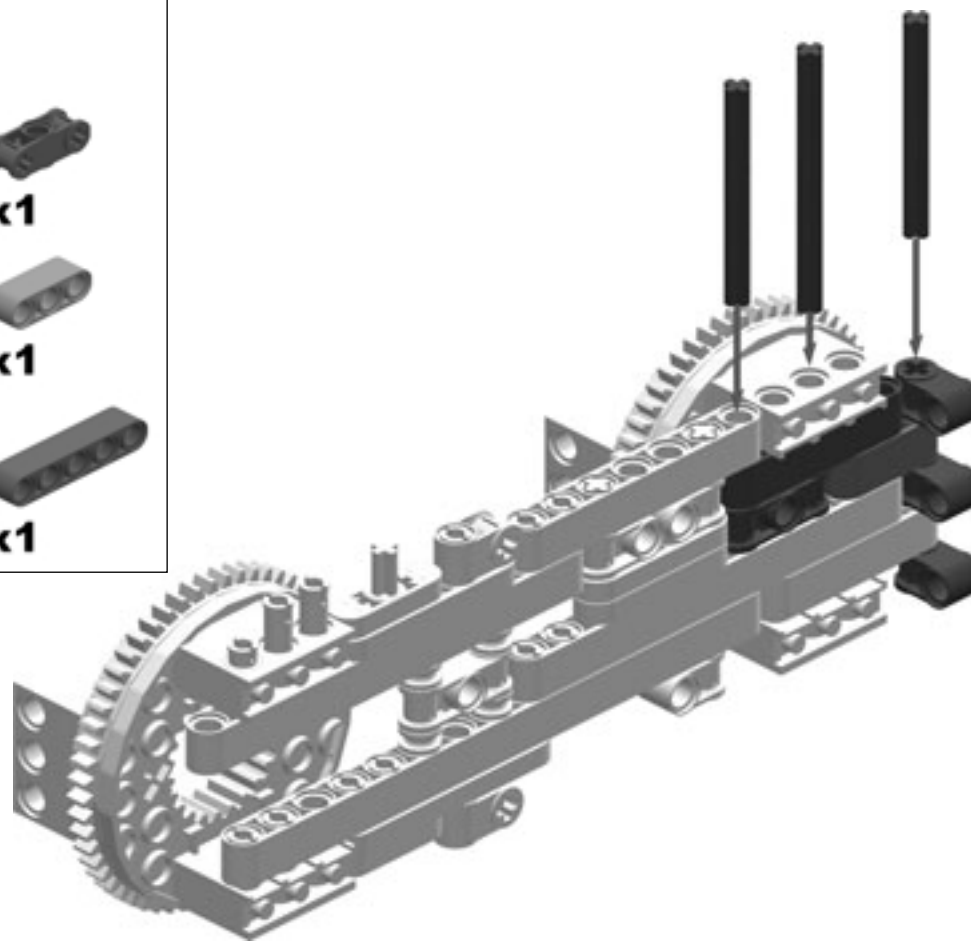
Left Side Step 7





Left Side Step 8





Left Side Step 9

The Locks

Building with liftarms is quite different from building with classic bricks. We already mentioned the half-stud width measurements, which should be a radical concept for LEGO TECHNIC and MINDSTORMS builders who are used to dealing with bricks of a width no less than 1 stud.

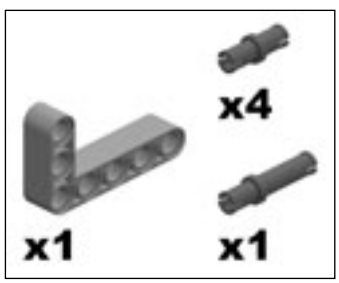
Another interesting building technique, which not only applies to liftarms but becomes apparent when dealing constantly with them, is that while we generally build LEGO by interlocking parts, there is a lot to be gained by simply *locking* them—especially if locking is as resistant as interlocking.

The following two elements lock the two sides of the SSCT's chassis and sandwich the propulsion and muscle mechanism in between them. Thus, they are an integral part of the chassis. As we will see at the end of this chapter, these fundamental elements attach and detach with amazing ease, and thus are key in keeping the vehicle together—and taking it apart. They are similar to the structure that connects the two turntables together above the RCX and behind the Vision Command camera in the Front sub-assembly that we saw earlier (**Front Steps 24**, **25**, and **26**).

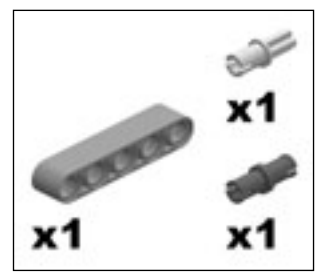
Building the Front Lock

This is where we will build the Front Lock sub-assembly. This sub-assembly attaches to both sides of the chassis, like the Back Lock sub-assembly—which we will build next—to complete the chassis. The benefit of having separate locking mechanisms is that the locks will detach easily for modifications to the chassis or the whole vehicle.

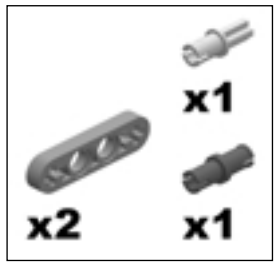
Front Lock Step 0



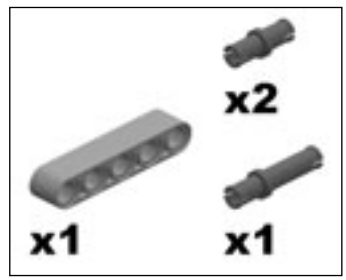
Front Lock Step 1



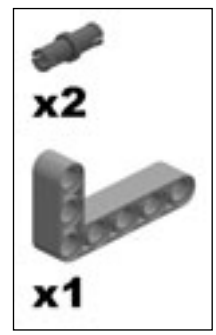
Front Lock Step 2

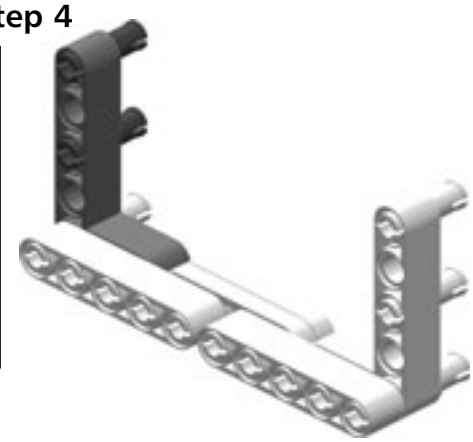


Front Lock Step 3



Front Lock Step 4



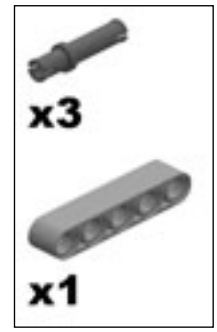


Building the Back Lock

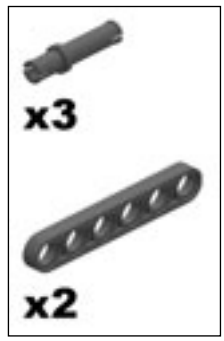
As you can see the construction of the Back Lock sub-assembly is very simple, neat and functional; however, it takes advantage of some of the newer members of the liftarm part family.

Back Lock Step 0

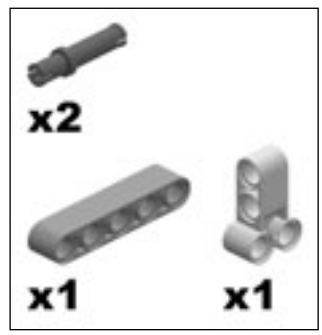
Back Lock Step 1



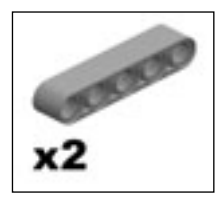
Back Lock Step 2



Back Lock Step 3



Back Lock Step 4



The Tread

For this book, I have strived to provide a vehicle entirely made out of official LEGO elements, as I tend to be a purist: a builder who does not incorporate non-LEGO elements to his models. The main reason for this is that models may not be easily replicated if the non-LEGO elements are not readily available. Following this philosophy, I have fitted the vehicle with over 200 TECHNIC tread links, and some plates to add traction.

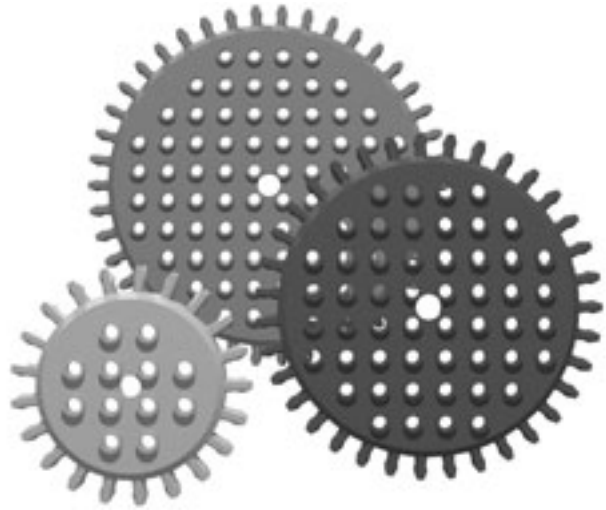
Unfortunately, the quality of the TECHNIC tread links is not nearly as good as that of the rest of LEGO system, due to some important issues with its design. Other than the unnecessarily smooth surface of the tread links, one of the least severe issues is the tendency of the links to break loose from each other in certain situations. This does actually not affect the SSCT much, due in large part to the track tensor. In the case of the SSCT, the most annoying problem is that the plates to tend to come loose from the treads.

If you have tread links and plates, it is easy to see why the plates tend to pop out (the ones with the solid studs underneath hold onto the links slightly better). There are several ways around this issue, but none of them are entirely satisfactory. The most straightforward solution is to glue the plates to the tread. This way of altering LEGO parts is something to think seriously about before proceeding, however, due to the price of each part.

Nonetheless, there are exceptions, and if you find the SSCT's functions useful for further experimentation, consider investing in good quality glue. A cheaper option can be to try to find a non-LEGO track that matches the turntables' teeth so that they are able to drive it. As co-authors Mario and Guilio Ferrari (The Learning Brick Sorter and the LEGO Turing machine respectively) in their book Building Robots with LEGO MINDSTORMS (Syngress, ISBN: 1-928994-67-9) have suggested, you might want to look at the remote controlled tanks section of your favorite a toy store.

A more radical alternative is to use pre-TECHNIC gears instead of turntables (seen in Figure 6.14), which come in sizes several times larger than the modern ones. LEGO does not manufacture these type of gears any longer, but they are easy enough to find second hand. Look on eBay (Remember that old sets in their original boxes often reach very high prices; loose pre-TECHNIC gears will often go for lower prices). Much harder to find are the LEGO pre-TECHNIC chain links. This chain is bulkier but much stronger than the TECHNIC tread, and plates attach solidly to it. Pre-TECHNIC gears attach to regular LEGO TECHNIC axles and can be driven by regular TECHNIC motors and gears.

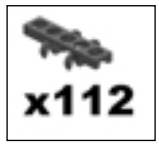
Figure 6.14 Pre-TECHNIC gears



Building the Tread

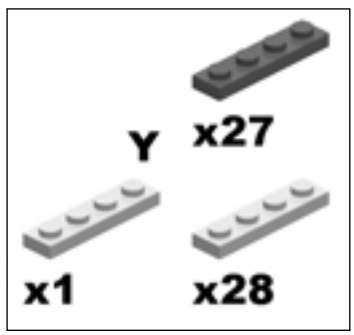
You have to repeat these steps twice to create both tracks.

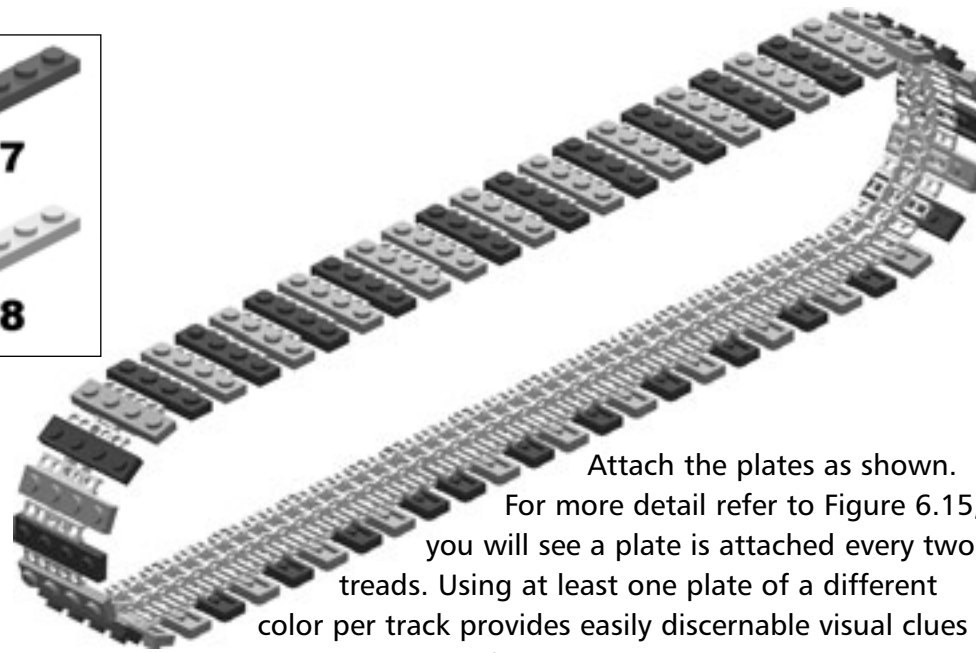
Tread Step 0



Connect the tread links together to create a closed loop. Take into account that 112 links is a *suggested* number. If the tread skips gears or regularly breaks apart, try increasing or decreasing the number of links.

Tread Step 1

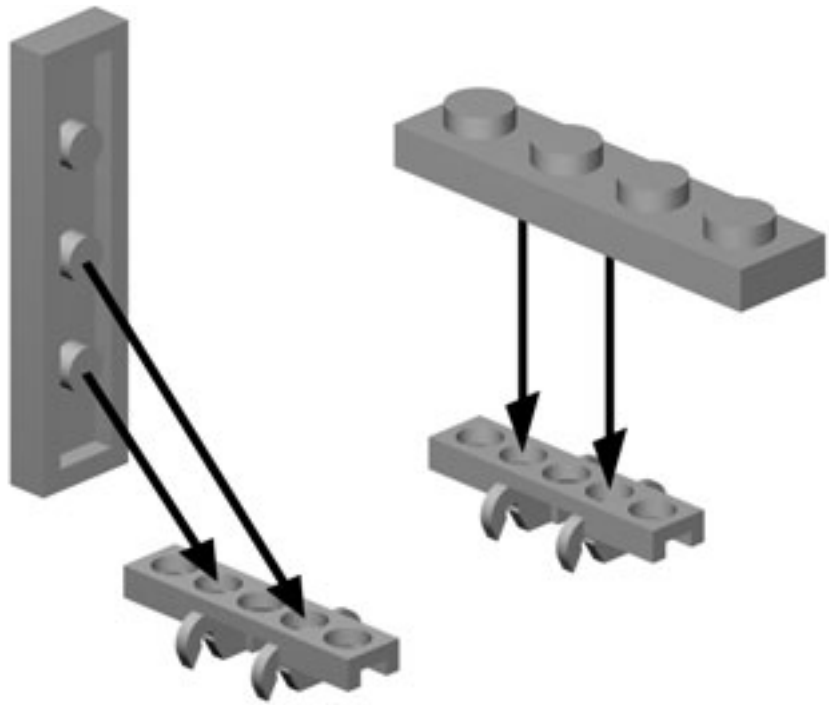




Attach the plates as shown. For more detail refer to Figure 6.15, you will see a plate is attached every two treads. Using at least one plate of a different color per track provides easily discernable visual clues about the movement of the tracks.

Quite literally, there is not much to be said about the tread. Simply attach 112 TECHNIC tread links together (twice), and then attach a plate to one of every two links, as shown in Figure 6.15. Adding the plates to the thread links clears room underneath the robot (in technical terms, this is called improving the vehicle's *ground clearance*), and improves traction: the surface of the tread links is too smooth.

Figure 6.15 How to Attach the Plates to the Links in the Tread Sub-assembly



Controlling the Robot: The Joystick Sub-Assembly

From the beginning, the SSCT was meant to be a remote-controlled vehicle similar to the original VGTV. This provides the Vision Command kit an exciting vehicle to combine with the RCX. However, it is worthwhile to continue experimenting, to try to find ways in which to adapt AI, provided by the RCX into the shape-shifting mechanics, perhaps with the chassis and the RCX—and no camera. Of course, the camera could travel on the vehicle and the RCX stay at the command post attached to the vehicle via cables. Once we start thinking about the control options for an explorer vehicle, the practical possibilities quickly begin to accumulate. While the tethered configuration does impose some limits, it also adds computing power. The Vision Command kit includes a new layer of LEGO MINDSTORMS software to combine with the RCX, which could be used to

drive the vehicle in an autonomous mode, with the added functionality of using the camera as a highly sophisticated sensor.

The SSCT model presented here carries an RCX onboard. This allowed to me to sort out some of the more extreme solutions with regard to the weight and the size of the robot, but also required me to use a remote control mechanism for the robot. If the SSCT is out of sight, it will most likely also be out of sight of the IR tower that would permit us to send commands to the RCX from the command post.

Driver feedback is an obvious (and very important) difference between telepresence and typical remote controlled (RC) vehicles. When operating regular RC vehicles (for instance, the scale cars used in racing competitions), the driver maneuvers by looking directly at the vehicle. When operating a telepresence vehicle, the driver is looking at what the robot is seeing with its camera. The sensation of driving the SSCT around an inaccessible, out of sight place while watching live video is completely different. Driving the SSCT is like playing a video game in two senses.

- The first is that you are awkwardly exploring unusual and unknown geometries.
- The second is that you have the sensation of detachment: of being there, but then again safe and comfortably sitting by the control console.

Of course, the detachment sensation is diminished when you run into trouble and have to physically go and retrieve the vehicle. Start by exploring the familiar unknown land: the space below your bed, behind the couch, in a cluttered attic and so forth. Soon enough, you will be saving snapshots of that newly treaded territory.

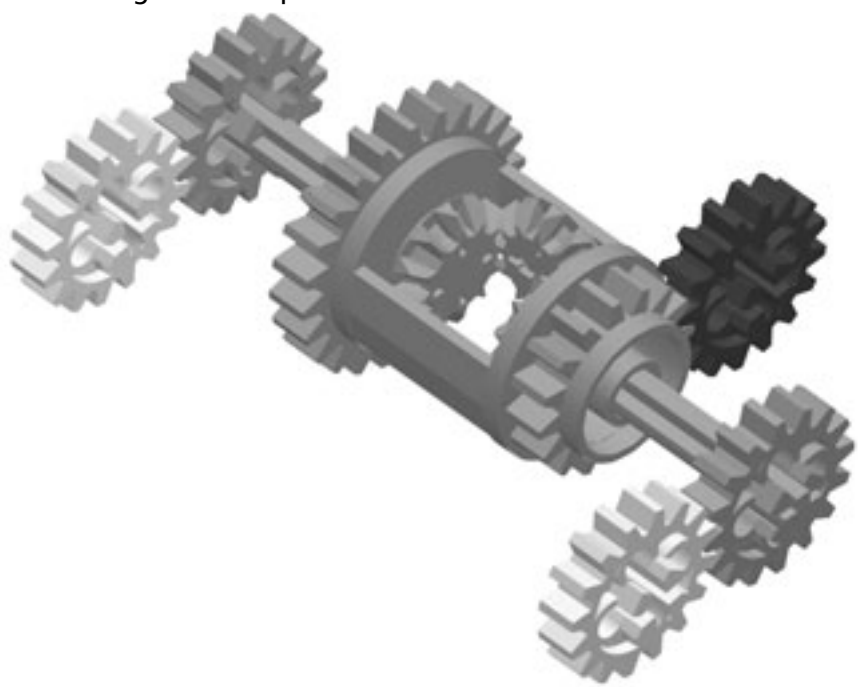
In order to get there, you will need to create a command console. For this you will need a computer to attach the Vision Command camera, and the appropriate software that will allow you to watch (and hear—the camera has a microphone) live video from the camera. Remember, you will drive the SSCT by looking at the computer screen, not by looking at the vehicle. You will also need a way to control the robot, including a cable that is as long as the camera cable. To accomplish this you can either use several sections of regular LEGO cable segments (they come in several size sizes from a few centimeters to over a meter long) or to create a single long cable by cutting a LEGO cable and adding a length of regular electric wire in between the two connectors. Make sure that you reconnect the cables properly after cutting them to add the wire.

One end of this wire connects to the robot and the other to the Joystick sub-assembly. As we will soon see, there might be a need for more than one long connection cable between the joystick and the vehicle. If we need to send more than one control cable between the vehicle and the control center, we can bypass the onboard RCX altogether and drive the SSCT using the LEGO TECHNIC Command center console or even three LEGO TECHNIC electric pole reverses connected to a bank of battery boxes for power. The vehicle has only three motors, and there are a number of ways for you to turn them on and off at will.

The design of the joystick shown here is based on a design for a mechanical LEGO joystick by Eric Brok, best known for his seminal Web site, "LEGO on my Mind," which you can find at the following URL: http://home.zonnet.nl/ericbrok/legomind. Eric is an enthusiastic figure of the Dutch LEGO scene. His site reflects his wide interests and truly plentiful skills in all things LEGO-related. Within the realm of great interest to robot builders, it contains some excellent articles on pure LEGO TECHNIC construction and others equally good on LEGO MINDSTORMS programming.

Eric's design of a mechanical joystick uses a key property of differential gears: they can mechanically add or subtract revolutions from two inputs. What this means is that if we connect two independent gear trains to a differential gear, we can get a single output, which will be the product of the two inputs. For example, in Figure 6.16, if the two white wheels turn at the same time, the speed of the black output wheel will be exactly double of that if only one wheel is turning and the other one remains stationary. Should both white wheels turn in different directions at the same speed, the output wheel will not move (-1+1=0).

Figure 6.16 Adding Gear Outputs with a Differential Gear



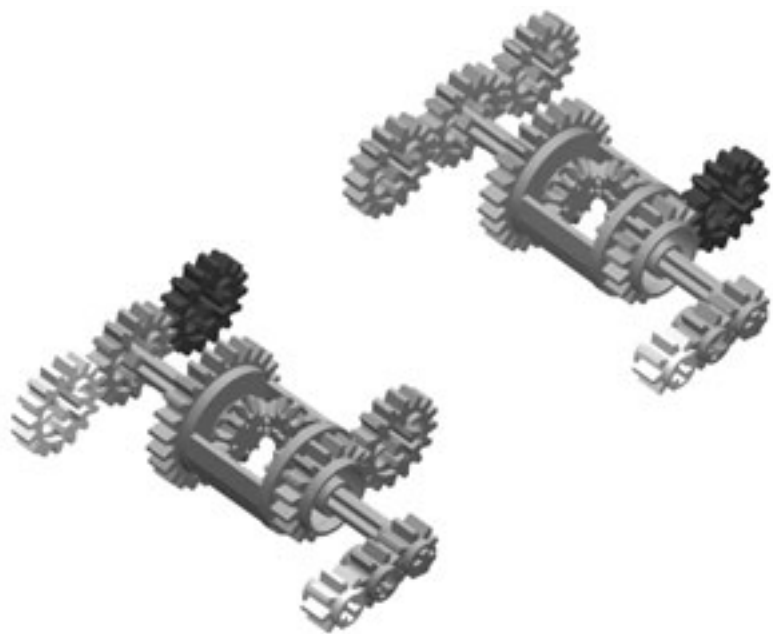
This characteristic of differentials has many practical applications and Eric Brok's joystick employs a very clever gear set-up to read the movements of the two joystick axles:

- One axle controls the left and right movements.
- One axle controls the forwards and backwards movements.

Figure 6.17 shows how this is implemented. You should note that the size of the gears do not matter for this explanation. Assume that the white gears turn at the same speed and in the same direction. If you look at the image in the lower left of Figure 6.17, you will see that the two input gears are moving in the same direction. Since one of the gear trains employs an idle gear, the movement of the input axles is in the opposite directions. As we saw in Figure 6.16, the wheel attached to the differential gear casing will not move (two opposite and equal inputs result in a sum of zero). The black output gear is connected directly to one of the gear trains and moves with it. Now looking at the image in the upper right of Figure 6.17, you will see that only one of the inputs is able to move, while the other input remains stationary. Since 1 + 0 = 1, the differential gear casing will move in the same proportion. When one axle moves and the other is stationary, the movement is transmitted via the differential, but when both axles move, their movement is cancelled at the differential and captured by other means. The mechanism provides an excellent response to any motion on the joystick handle: both outputs can carry some movement at the same time, which means that we can order a vehicle to go forward *and* turn right with the appropriate joystick motion. To implement this in a tracked vehicle, you can use the following rule:

- Control signals forward: both motors turn in the same direction.
- Control signals forward and turn: one track remains stationary; one moves.
- Control signals turn: each track moves in the same direction.

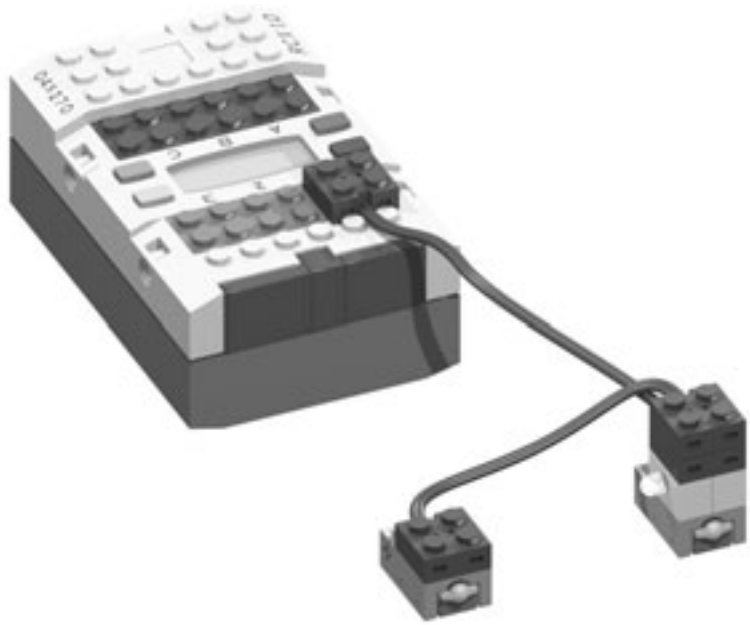
Figure 6.17 Differentiating Axle Movements with a Differential Axle



Eric's design is a *mechanical* device, meaning that it transmits force or speed. We need to turn that movement into electrical signals that can be read by the RCX, and the way to do it is using LEGO touch sensors. Unfortunately, LEGO touch sensors are, in principle, a binary on/off component (although those included in the CyberMaster kit do given different readings if connected in parallel). This means that activating any of them in the joystick sensors will produce the same signal. In order for the RCX to understand our instructions, we need to differentiate between the different touch sensors.

Similar to the method that co-author Kevin Clague suggests in his PneumADDic II model, we can mount the touch sensors *in series* and use resistors in between them to differentiate their outputs. Rob Stehlik suggested a solution in which a pair of LEGO lamp bricks can be used in pairs to act as resistors between sensors mounted in series, similar to the method shown in Figure 6.18.

Figure 6.18 Rob Stehlik's All-LEGO Lamps-as-resistors Set-up



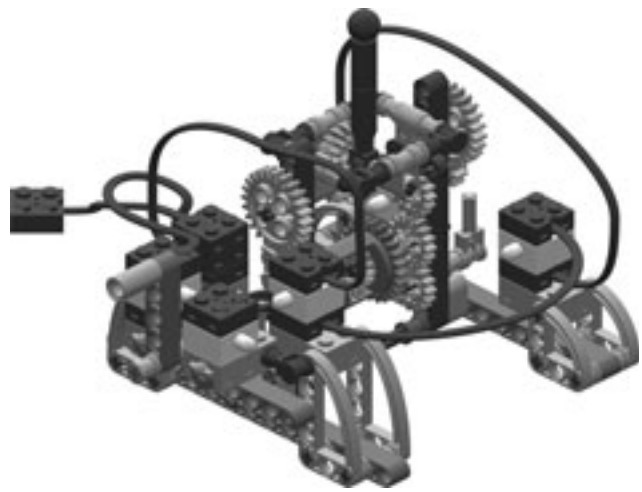
When we set any input port of the RCX to RAW mode and connect the sensors and lamps exactly as shown, we should get three different readings according to which of the sensors, or both, are pressed.

This neat idea is by no means the only one Rob has put forth in the area of LEGO robotics. Check out Rob's Web site at www.ecf.utoronto.ca/~stehlik/ to marvel at his impressive array of robots and gorge on his meaty commentary, photos, images, tips and tricks for the MINDSTORMS builder.

NOTE

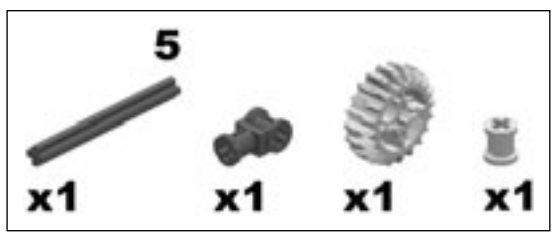
Do not miss Rob's Window Walker robot capable of attaching to and climbing glass surfaces using a vacuum pump made out of LEGO pneumatic parts! Rob's work can also be seen in the book 10 Cool LEGO Mindstorms Robotic Invention System 2.0 Projects (Syngress Publishing, ISBN: 1-931836-61-2).

Building the Joystick



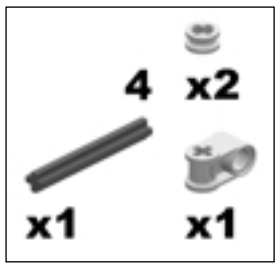
This is how the Joystick sub-assembly will look when it has been assembled. Learn how to program it in the next section and you can use it for any of your other creations.

Joystick Step 0

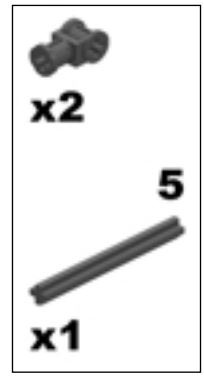


Start by building the mechanism that holds the handle and transmits its movements via two axles.

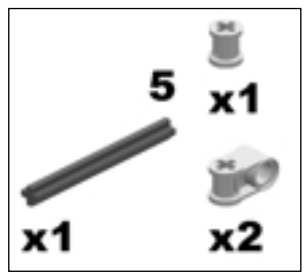
Joystick Step 1



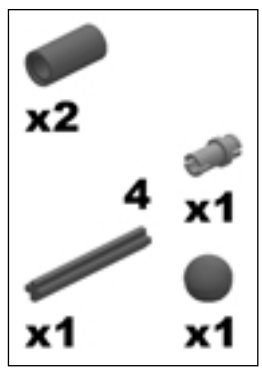
Joystick Step 2

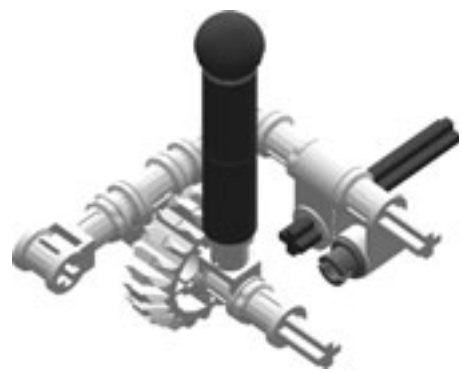


Joystick Step 3



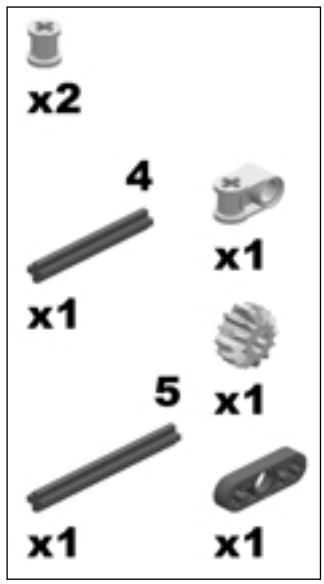
Joystick Step 4

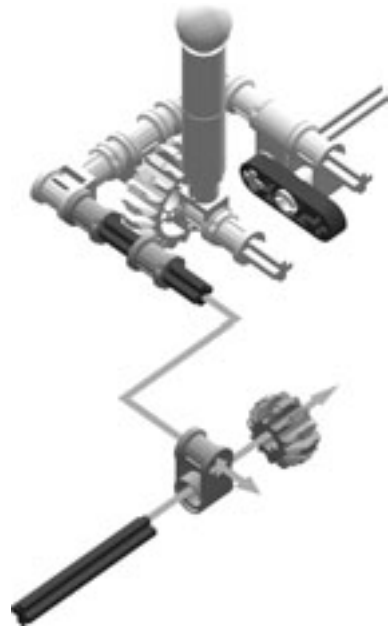




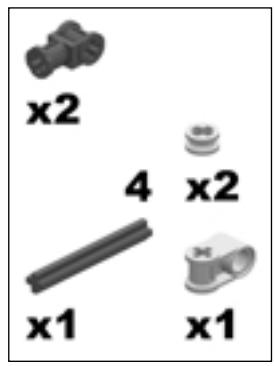
The round pin joiners and axle ball make the joystick handle nice and smooth. Always think of the user when designing user controls.

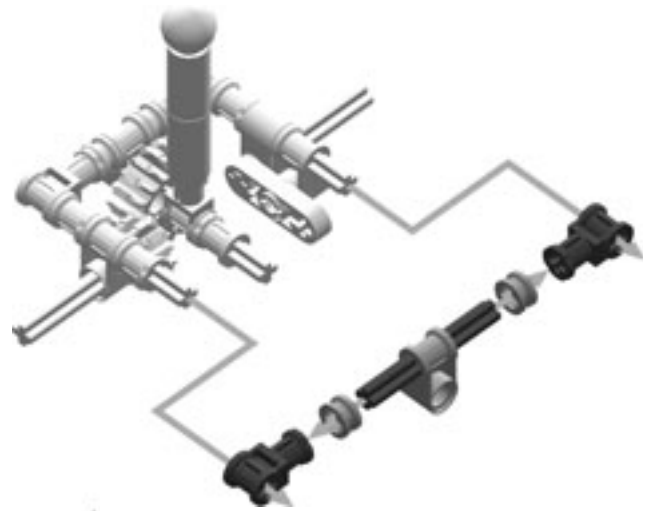
Joystick Step 5

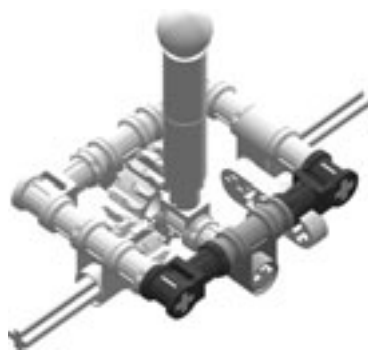




Joystick Step 6

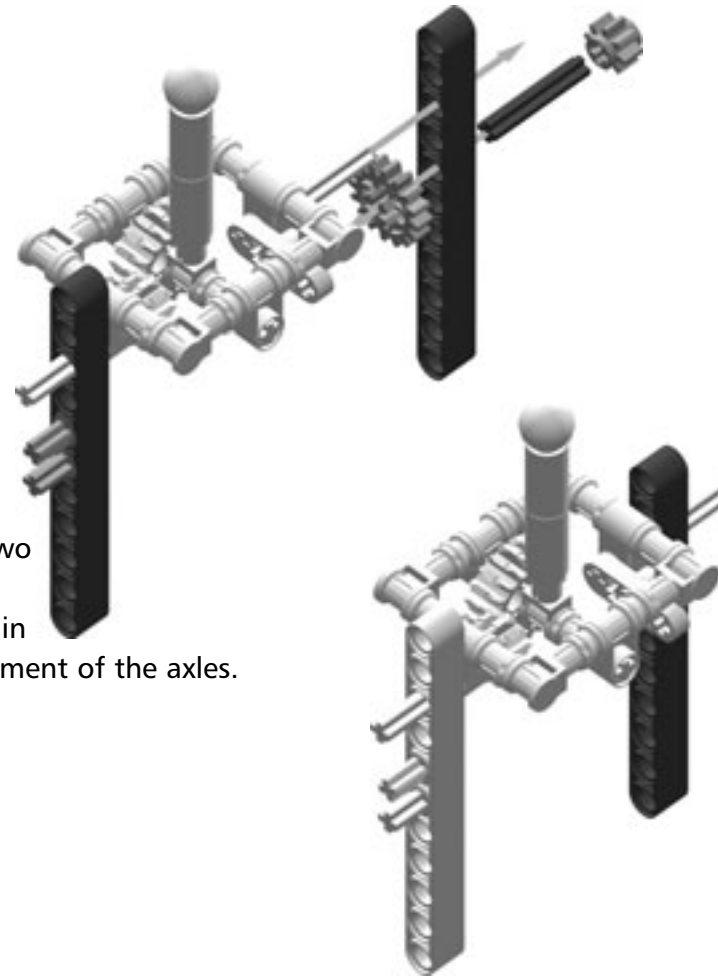






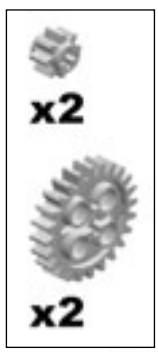
Structures cannot always be built with bricks and liftarms alone. Axles and connectors are often good substitutes, and are especially useful in ultra-compact assemblies.

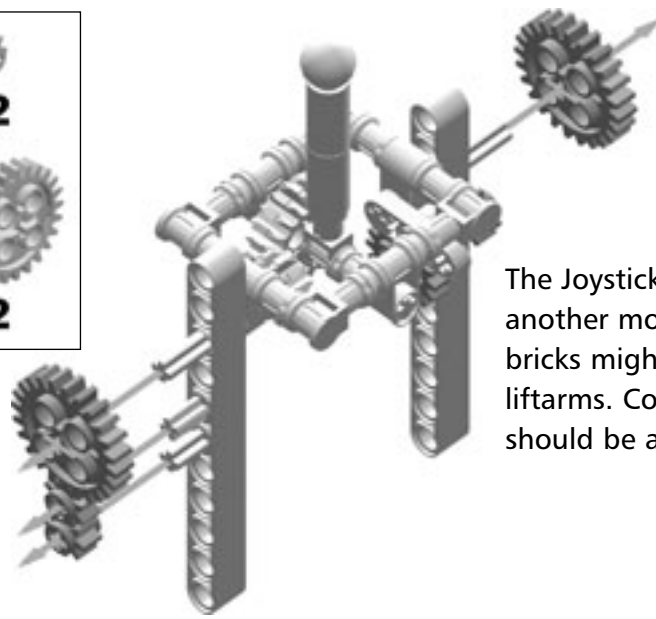
Joystick Step 7



Once the handle with the two output axles is completed, add the differential gear train that breaks down the movement of the axles.

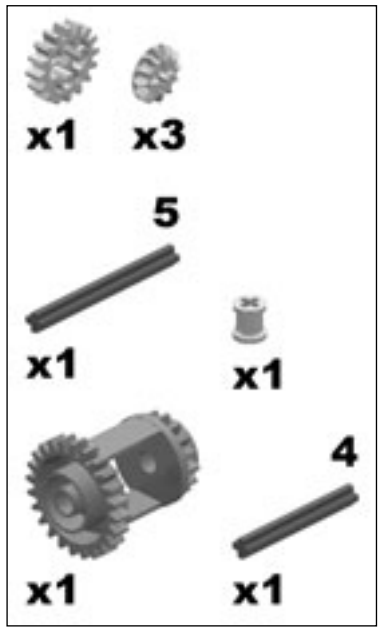
Joystick Step 8

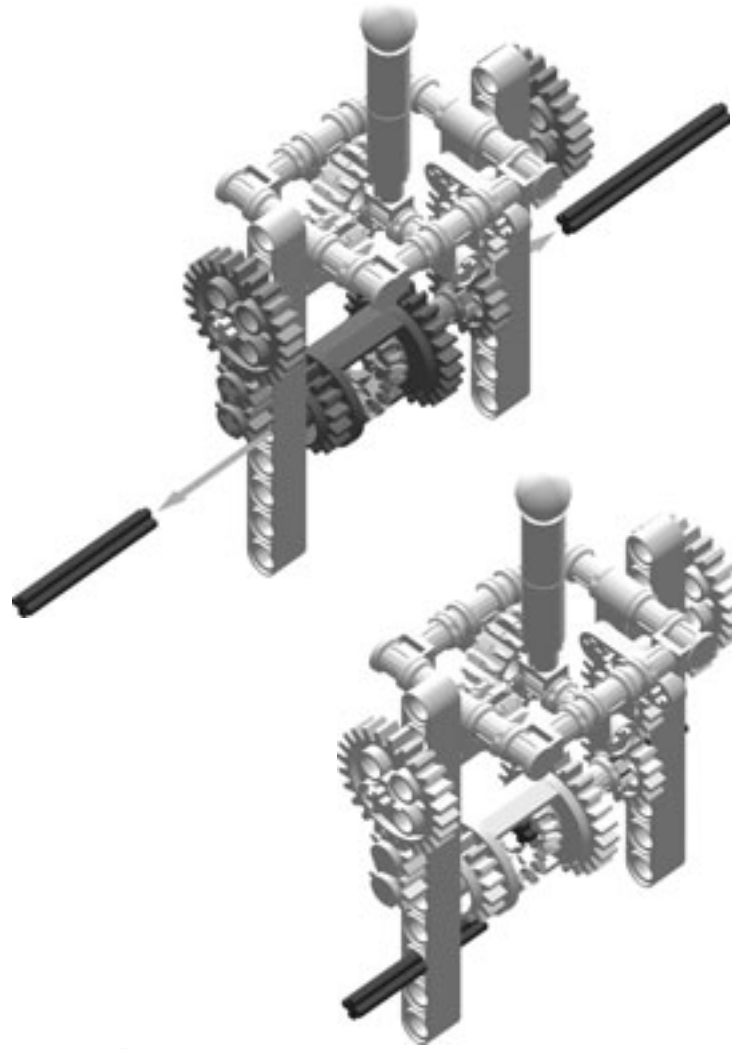




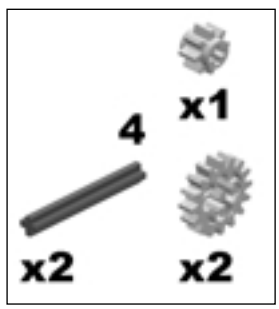
The Joystick sub-assembly is another module where traditional bricks might work better than liftarms. Controls, by definition, should be as firm as possible.

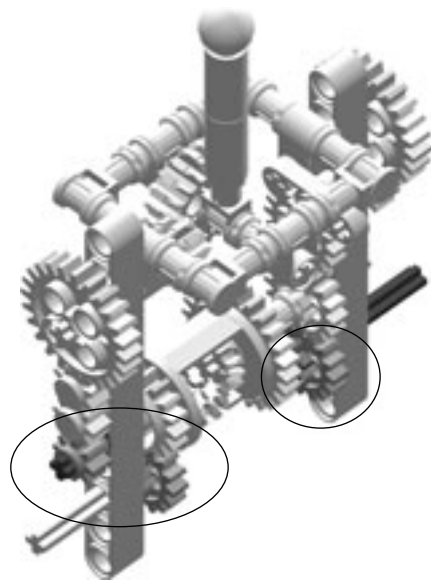
Joystick Step 9



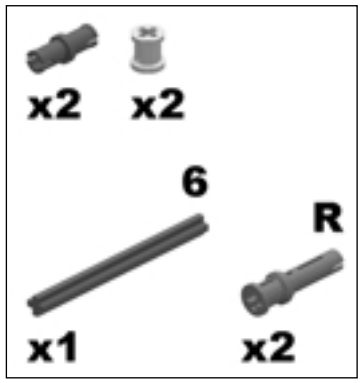


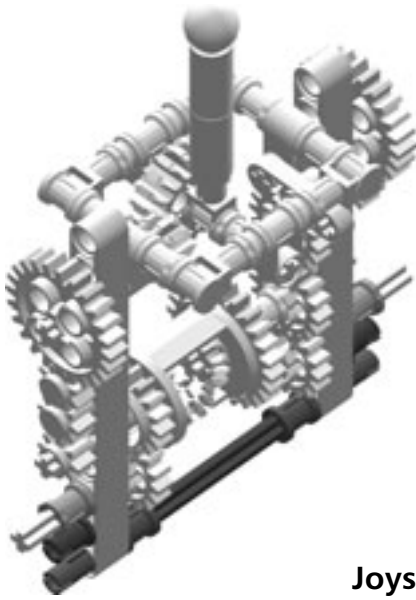
Joystick Step 10



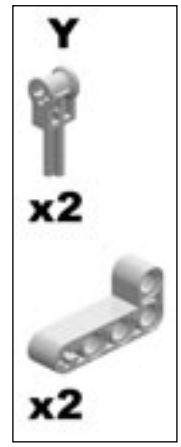


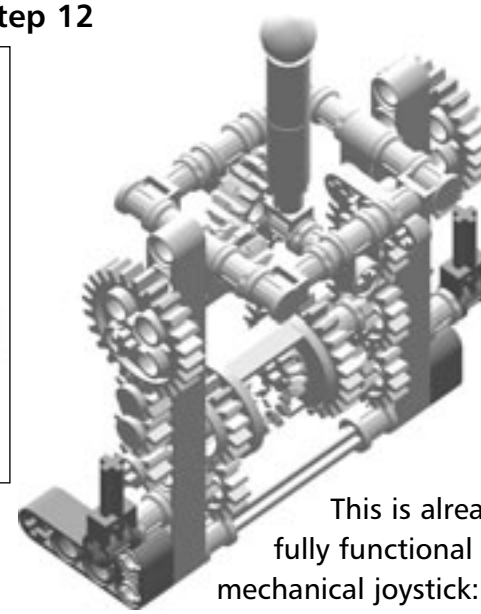
Joystick Step 11





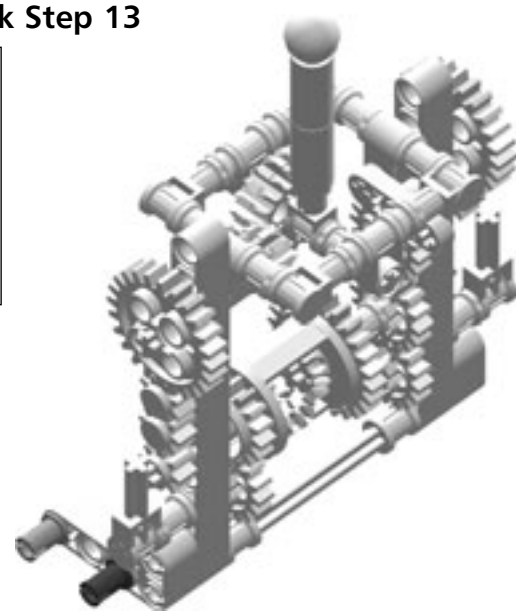
Joystick Step 12



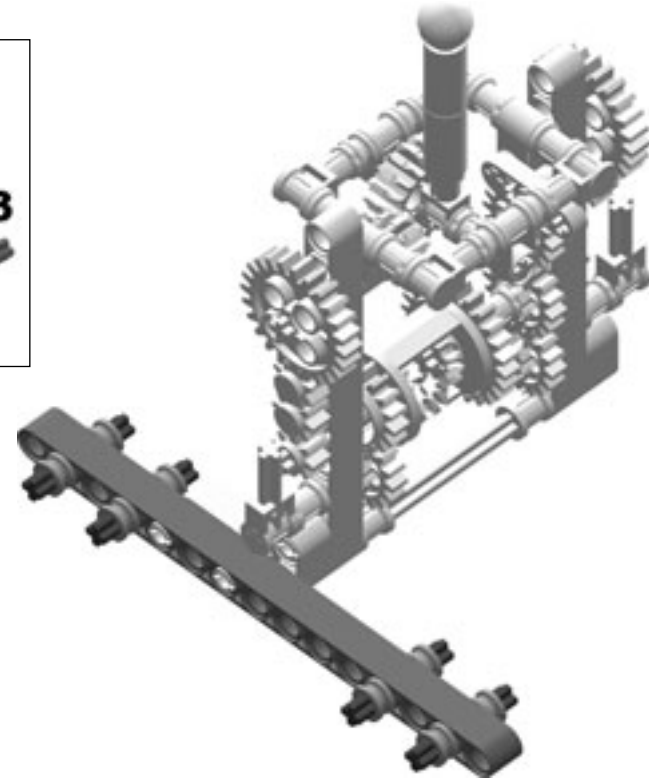


This is already a fully functional mechanical joystick: the movement of the handle is translated into two axle outputs. Now, it is time to connect those axles to touch sensors so their movement can be interpreted by the RCX.

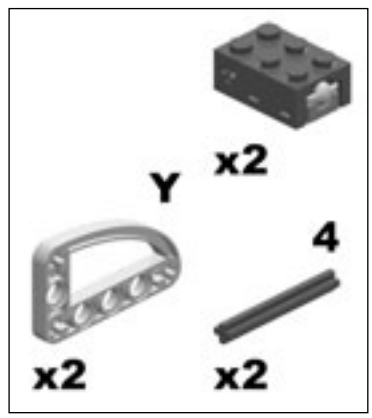
Joystick Step 13

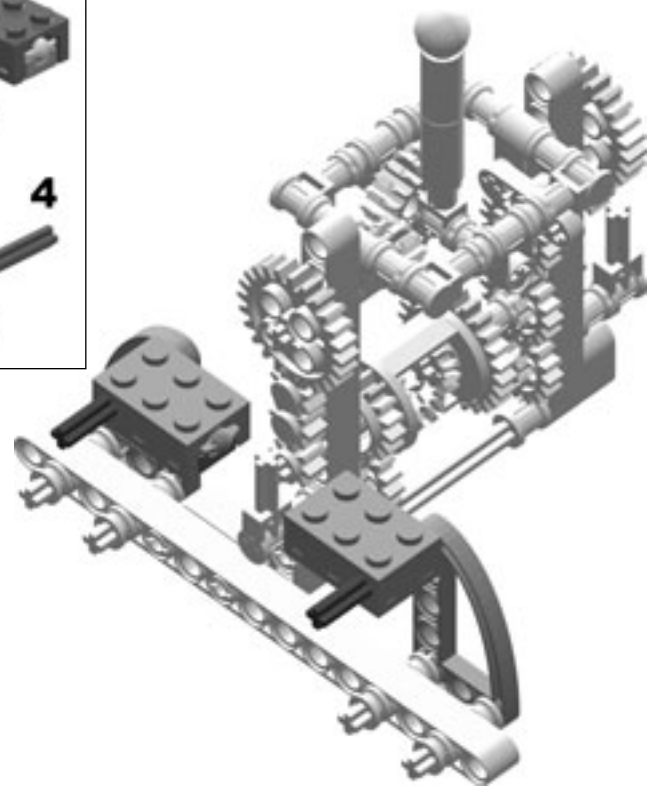


Joystick Step 14

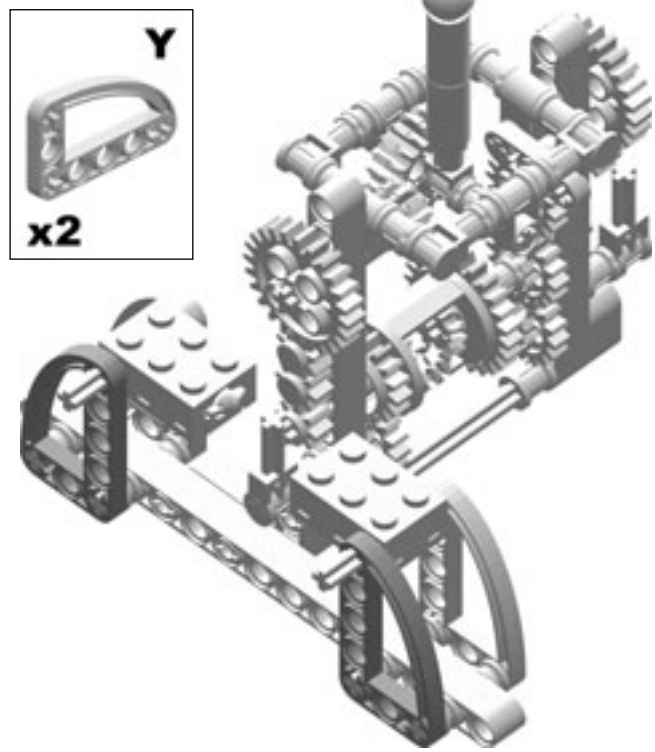


Joystick Step 15

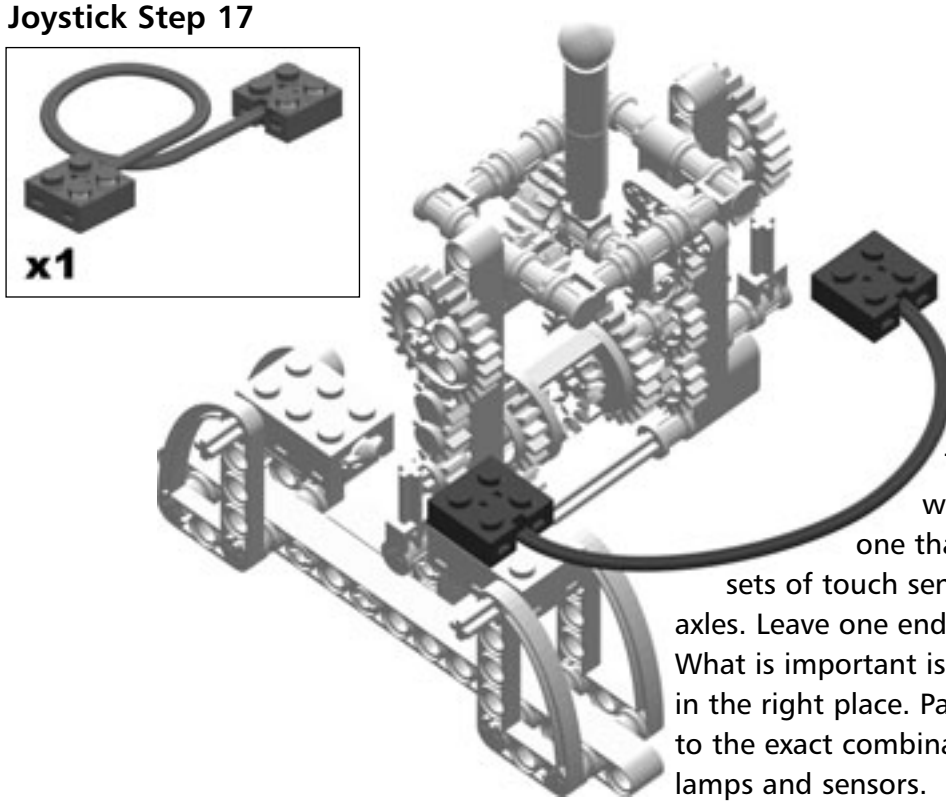




Joystick Step 16

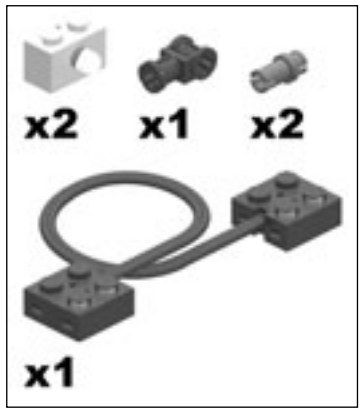


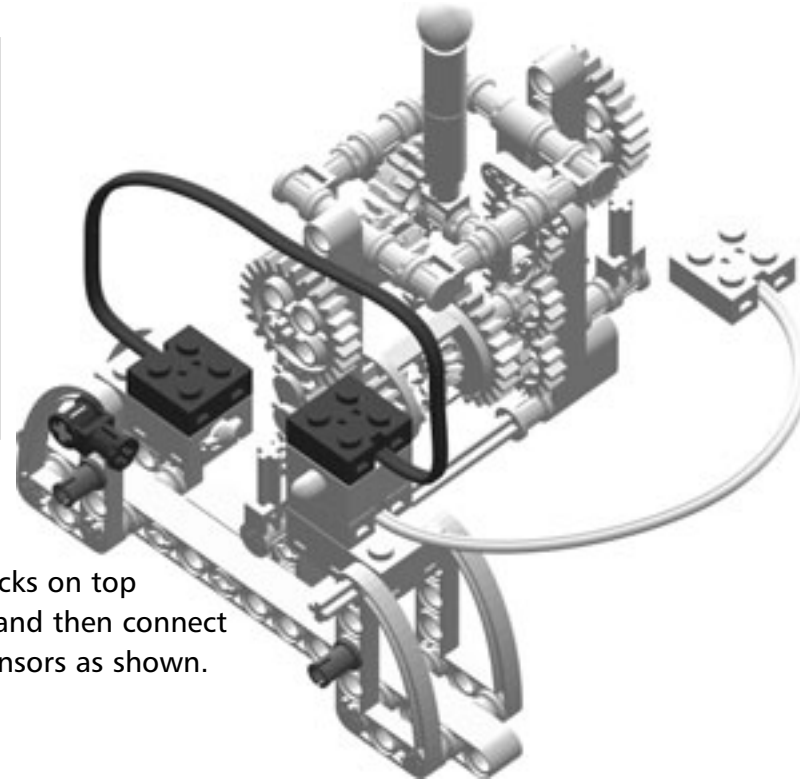
Joystick Step 17



The first connector wire we add is the one that connects both sets of touch sensors on the output axles. Leave one end dangling for now. What is important is to orient the wires in the right place. Pay special attention to the exact combination of wire ends, lamps and sensors.

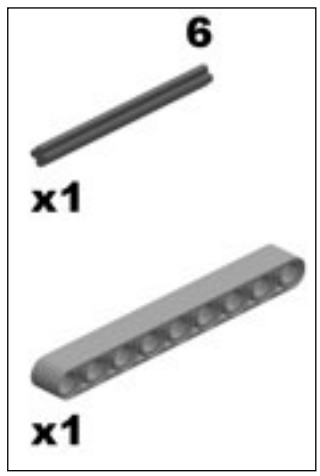
Joystick Step 18

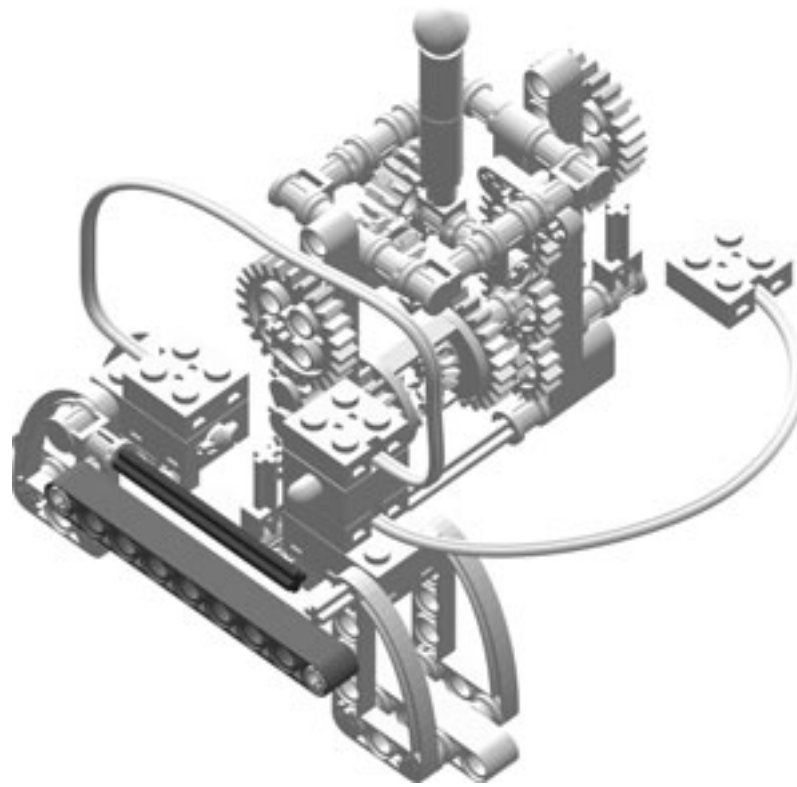




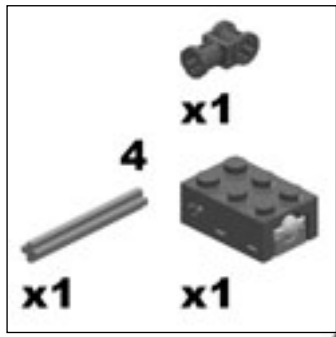
Add the lamp bricks on top of the wire end, and then connect the two touch sensors as shown.

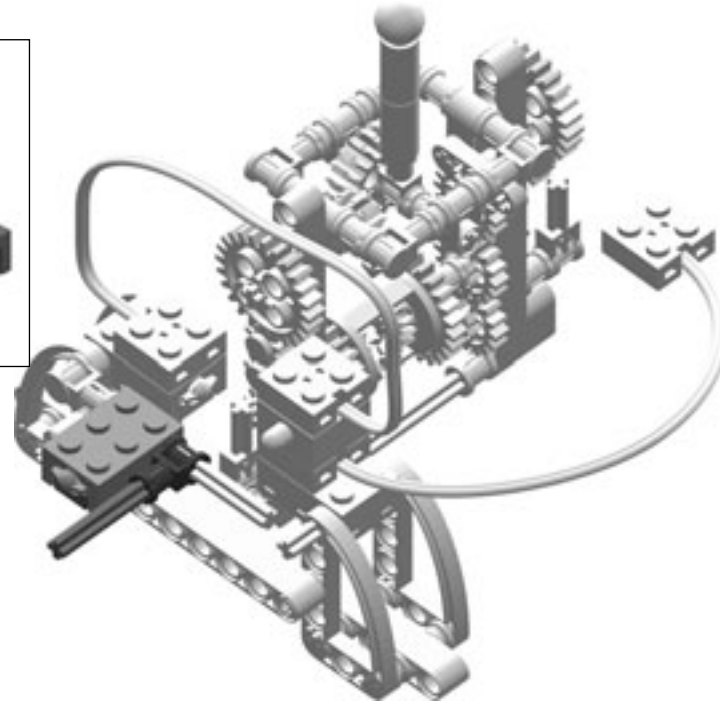
Joystick Step 19



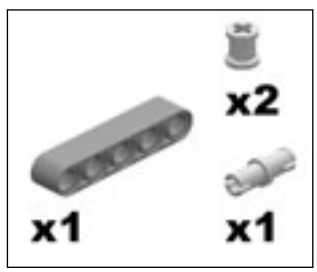


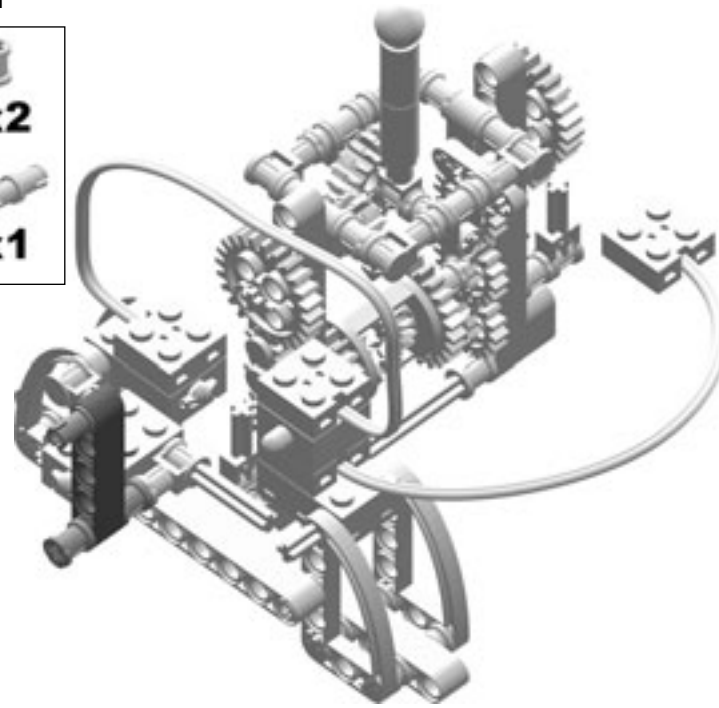
Joystick Step 20



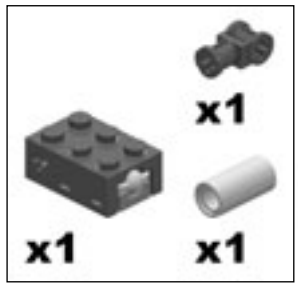


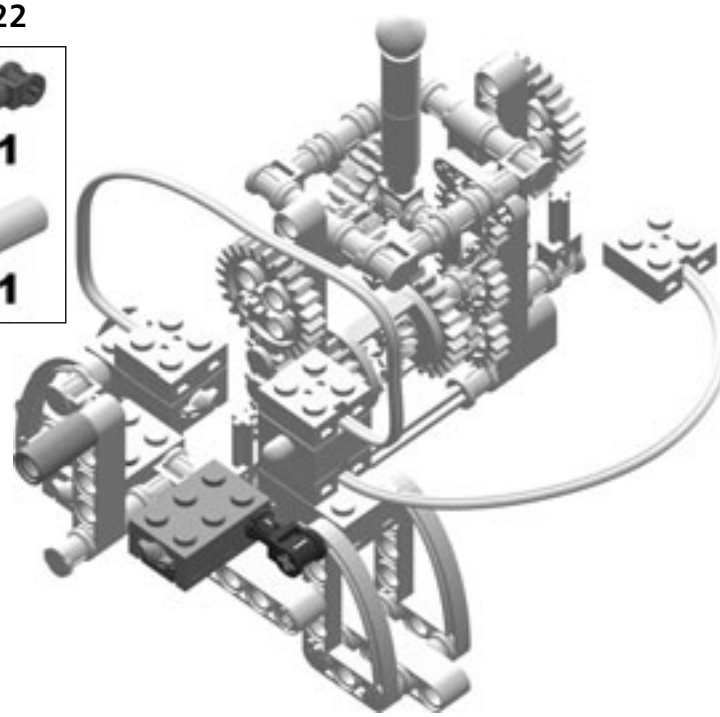
Joystick Step 21



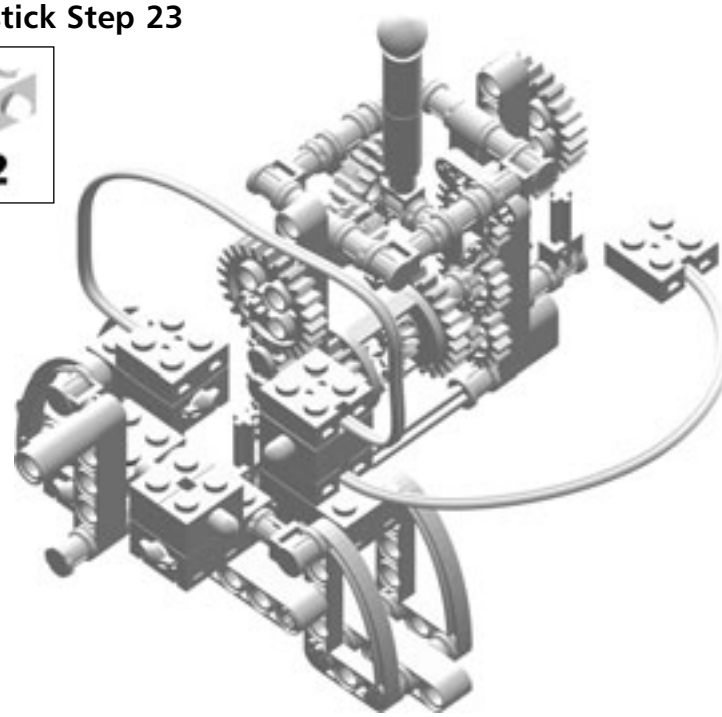


Joystick Step 22

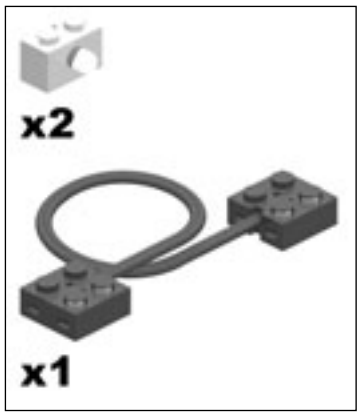


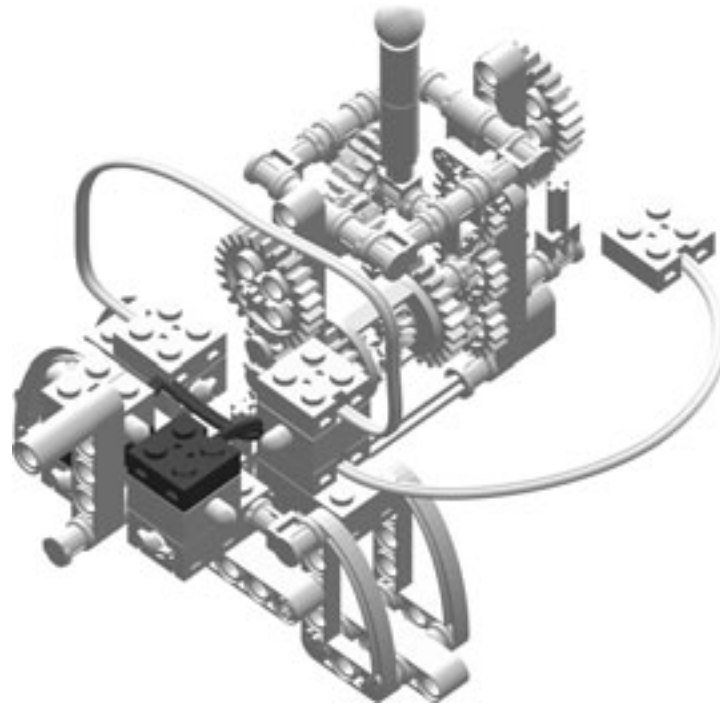


Joystick Step 23

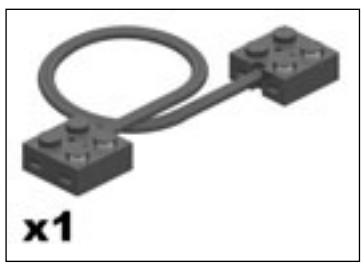


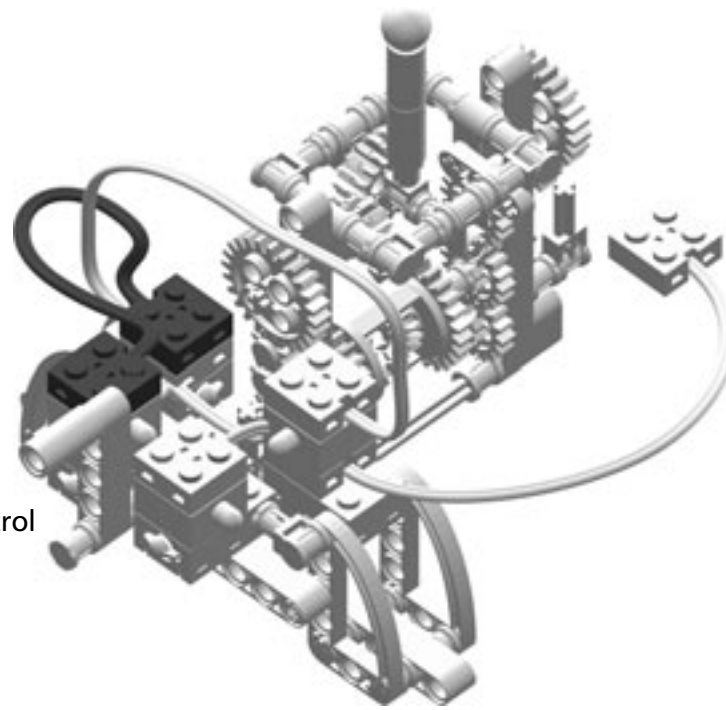
Joystick Step 24





Joystick Step 25

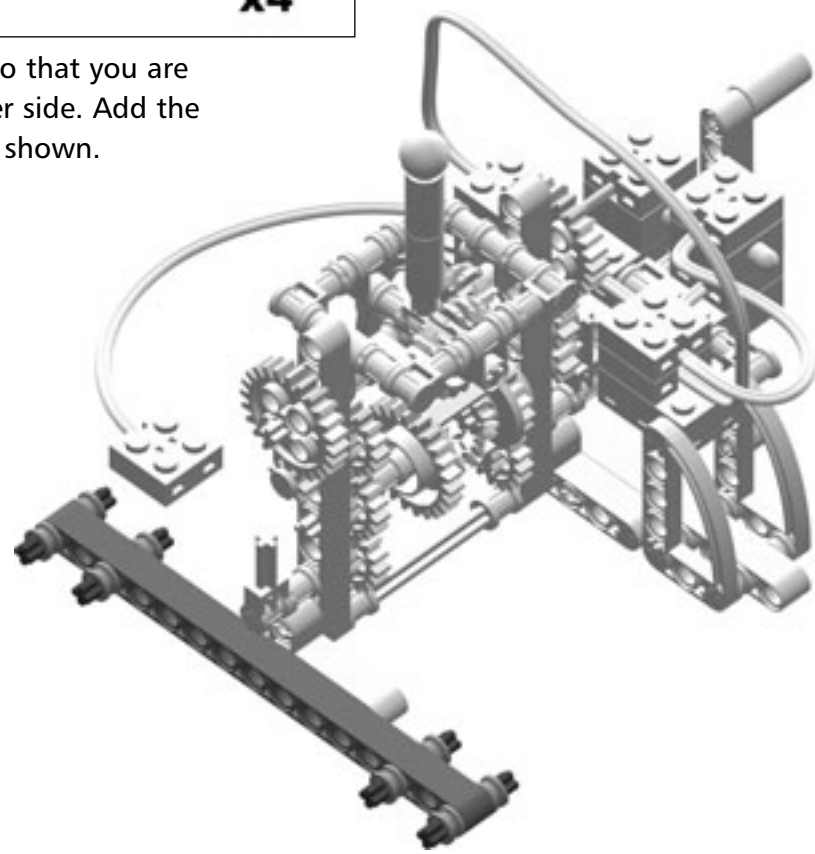




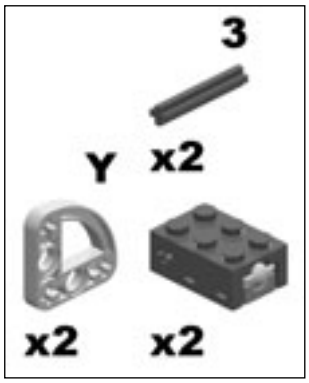
This extra set of sensors control the Folding Muscle Unit sub-assembly of the vehicle.

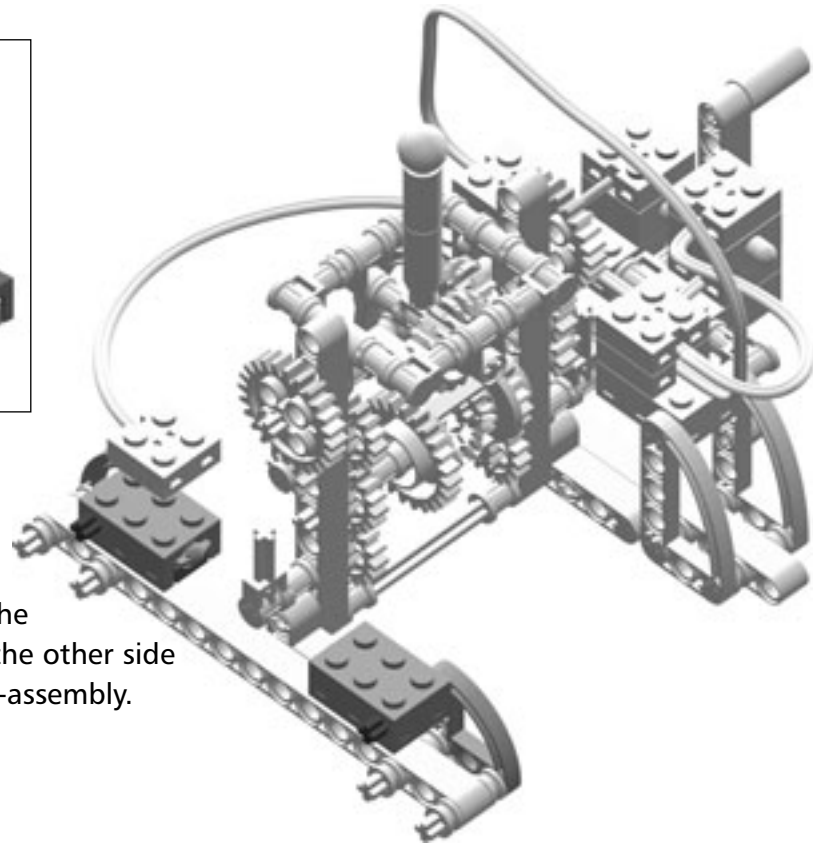
Joystick Step 26

Rotate the model so that you are looking at the other side. Add the liftarm and pins as shown.



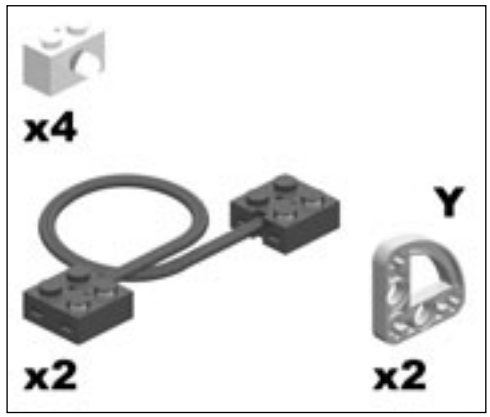
Joystick Step 27

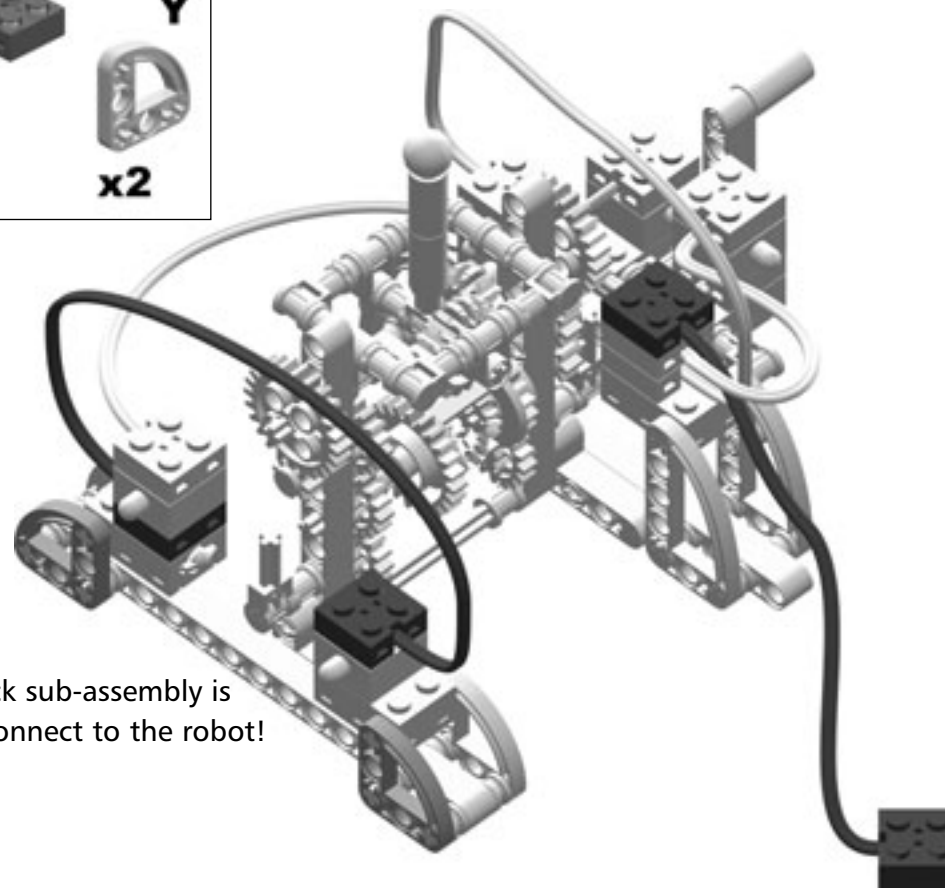




We will now add the touch sensors for the other side of the Joystick sub-assembly.

Joystick Step 28





The Joystick sub-assembly is ready to connect to the robot!

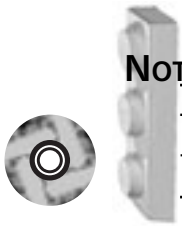
Programming the RCX to Receive the Joystick Input

Programming the RCX to accept the input of the joystick is a straightforward operation. Simply identify the touch sensor signal and activate the motors accordingly. To differentiate the signals from different sensors, we have to set the sensor type for the port to RAW. This way, the port will give a value of approximately 1020 when the touch sensor is not pressed, and about 65 when fully pressed. Using the lamps as resistors between sensors, we should be able to get a different reading for each sensor.

In reality, the system described above is less than perfect, as not all lamps offer the same resistance and often there is no practical way to differentiate between three or four different sensors. Fortunately, there are ways around it. Some sensors will give slightly different readings, and some lamps alter the readings quite a bit. Note the readings that dif-

ferent sensors and lamp combinations send to the RCX and try to differentiate between as many as possible.

Other alternatives include using resistors such as those used in the PneumADDic II model or using more than one sensor port of the RCX to input part of the functions (the flexing of the muscle mechanism, for instance). Below is the program that will work with my sensors and lamps at home. Notice that you need to introduce a small delay between sensor readings (simply leave the motor on) or the robot will not be operating properly.

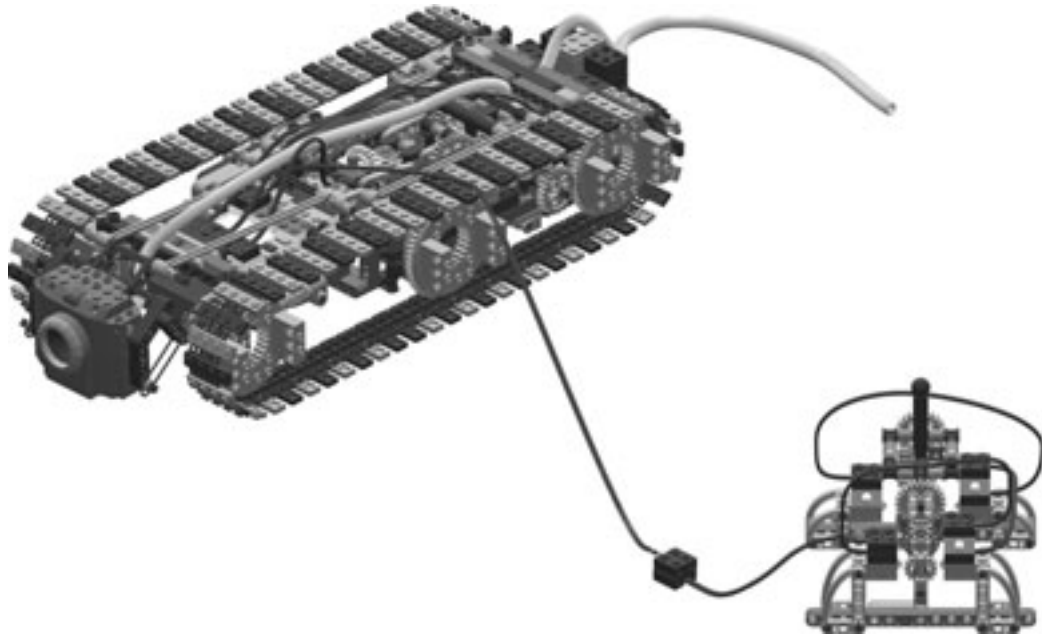


NOTE

The program for the SSCT is available on the CD-ROM that accompanies this book.

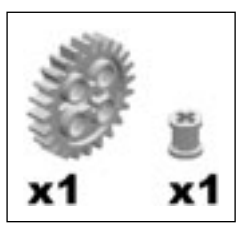
```
task main()
{
  SetSensor(SENSOR_1,SENSOR_MODE_RAW);
  start check_sensors;
}
task check_sensors()
{
  while (true)
  {
    if (SENSOR_1 < 55 ) {OnFwd(OUT_A); OnRev(OUT_B); Wait(10);}
    if (SENSOR_1 > 57  && SENSOR_1 < 70 ){OnFwd(OUT_B); OnRev (OUT_A); wait(10);}
    if (SENSOR_1 > 75  && SENSOR_1 < 85 ){OnRev(OUT_C); Wait(10);}
    if (SENSOR_1 > 90  && SENSOR_1 < 105 ){OnFwd(OUT_A+OUT_B); Wait(10);}
    if (SENSOR_1 > 107 && SENSOR_1 < 130){OnFwd(OUT_C); Wait(10);}
    if (SENSOR_1 > 150 && SENSOR_1 < 180){OnRev(OUT_A+OUT_B); Wait(10);}
    else     {Off(OUT_A+OUT_B+OUT_C);}
  }
}
```

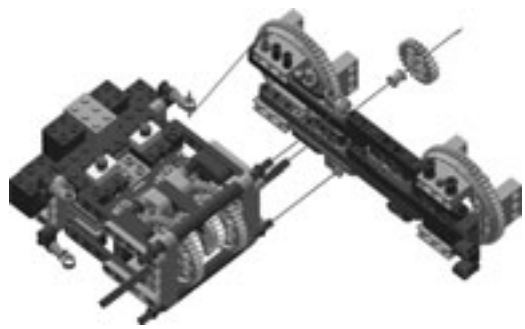
Putting it All Together

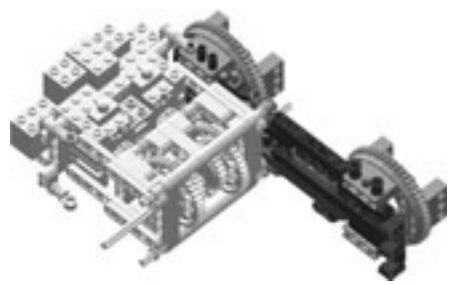


In this last assembly, we bring the complete model together. You will use all the previous subassemblies plus a few extra parts to secure everything together. Here you can fully appreciate the advantages of modular building. It works backwards too: if you want to extract a sub-assembly for redesign, simply follow the steps backwards and you will de-assemble the model back into its main subassemblies.

Final Step 0

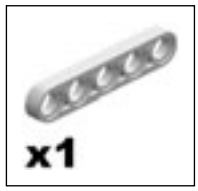


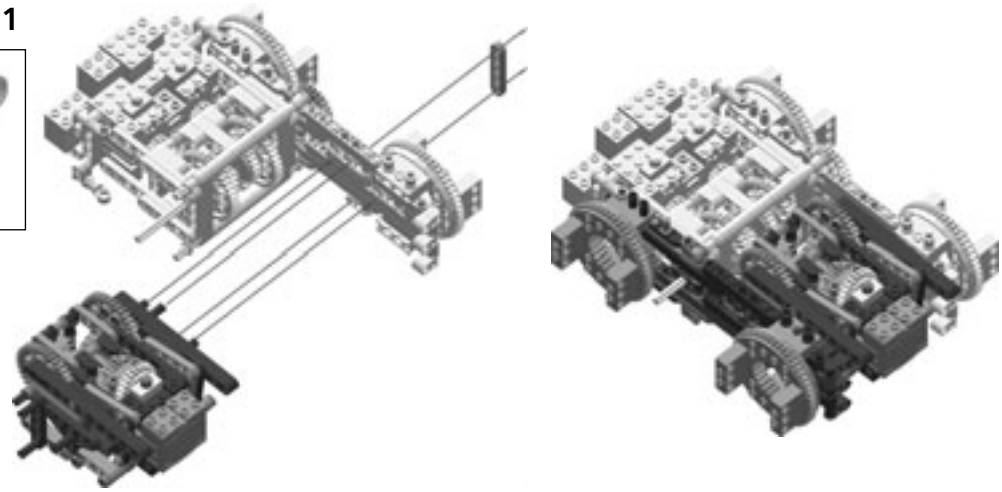




Start by attaching the Propulsion Unit sub-assembly to the Left Side sub-assembly of the chassis. Add a bushing and a 24T gear to transfer the motor output to the turntable that will turn the track. Notice that the Propulsion Unit sub-assembly attaches to the chassis side with two short axles at the front and two latches at the rear. These latches connect to three-quarter length pins directly attached to the turntable top and bottom of the turntable.

Final Step 1

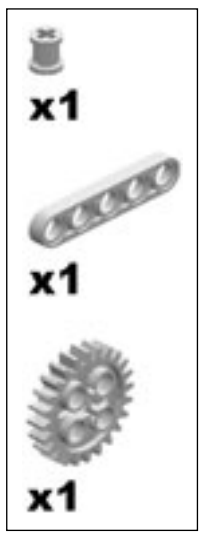


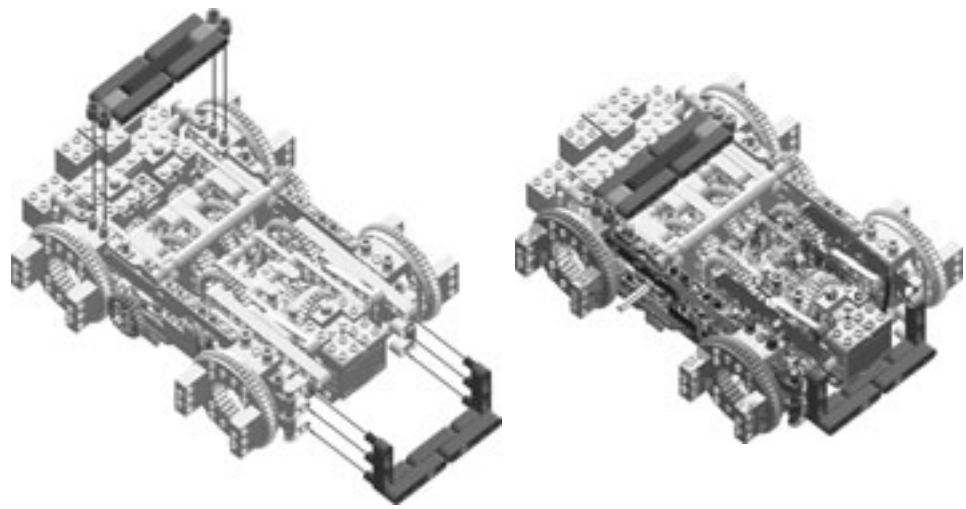


Now, add the Folding Muscle Unit sub-assembly. This sub-assembly attaches to the Left Side sub-assembly (chassis) using long friction pins, two of which are locked in place by a liftarm that acts as a taper of sorts.

With the Folding Muscle Unit sub-assembly in place, attach the Right Side sub-assembly (chassis). Everything is symmetrical and should attach exactly the same way as above.

Final Step 2

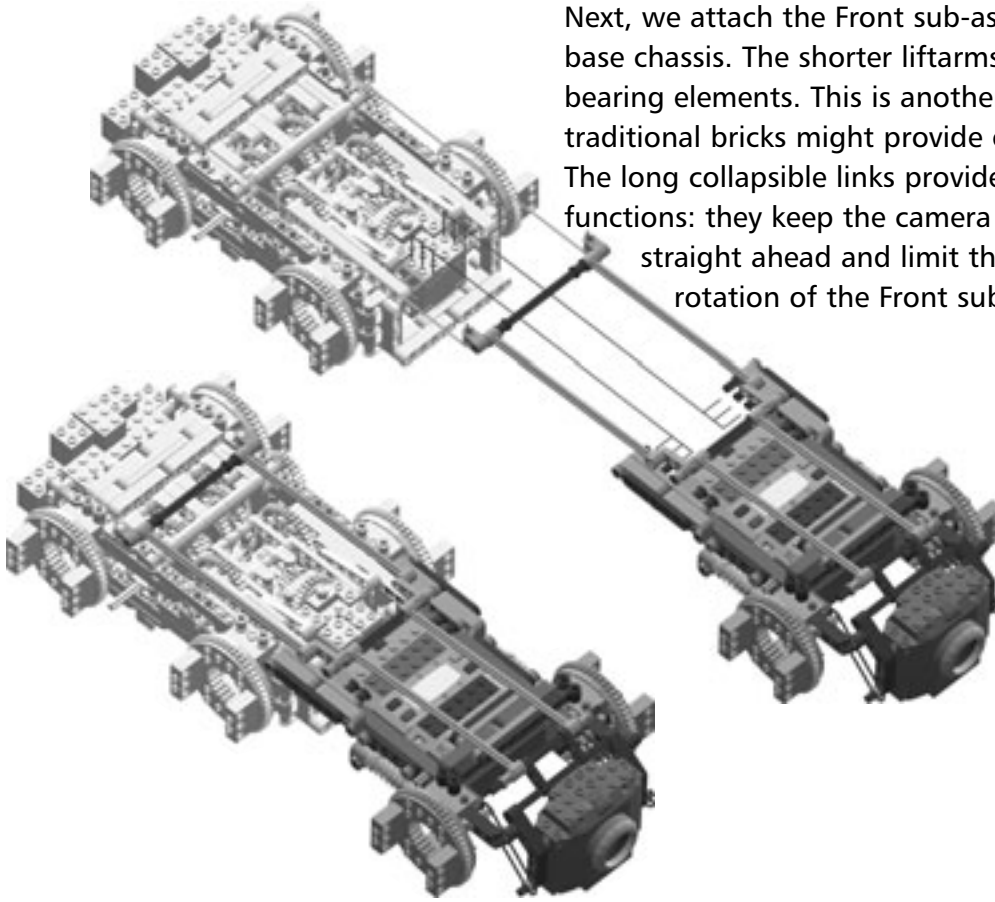


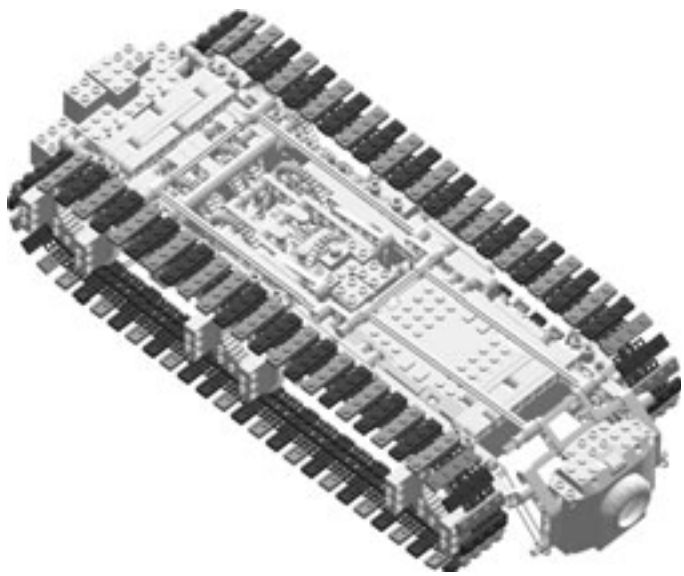


Add the bushing and gear to power the turntable of the Right Side sub-assembly, and the tapering liftarm for the Folding Muscle Unit sub-assembly. The sub-assemblies are now sandwiched together between the sides of the chassis. We then proceed to add the Front and Back Lock sub-assemblies that add plenty of strength to the robot.

Final Step 3

Next, we attach the Front sub-assembly to the base chassis. The shorter liftarms are the weight bearing elements. This is another area where traditional bricks might provide extra strength. The long collapsible links provide a variety of functions: they keep the camera pointed straight ahead and limit the downward rotation of the Front sub-assembly.

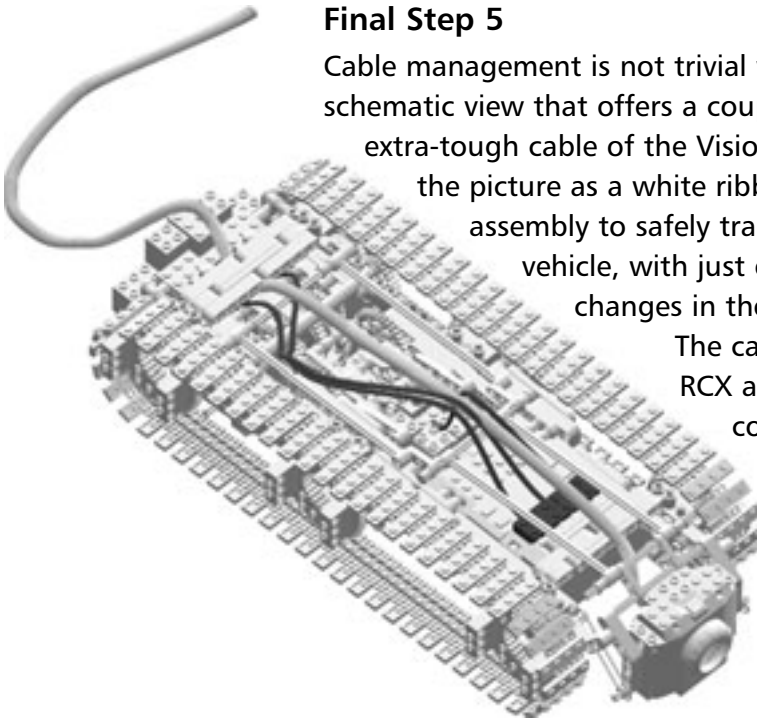




Final Step 4

To attach the two Tread sub-assemblies to the model, open one of the links, circle the turntables with the track and re-attach the links, closing the loop.

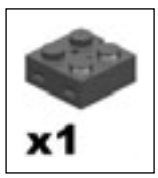
Final Step 5

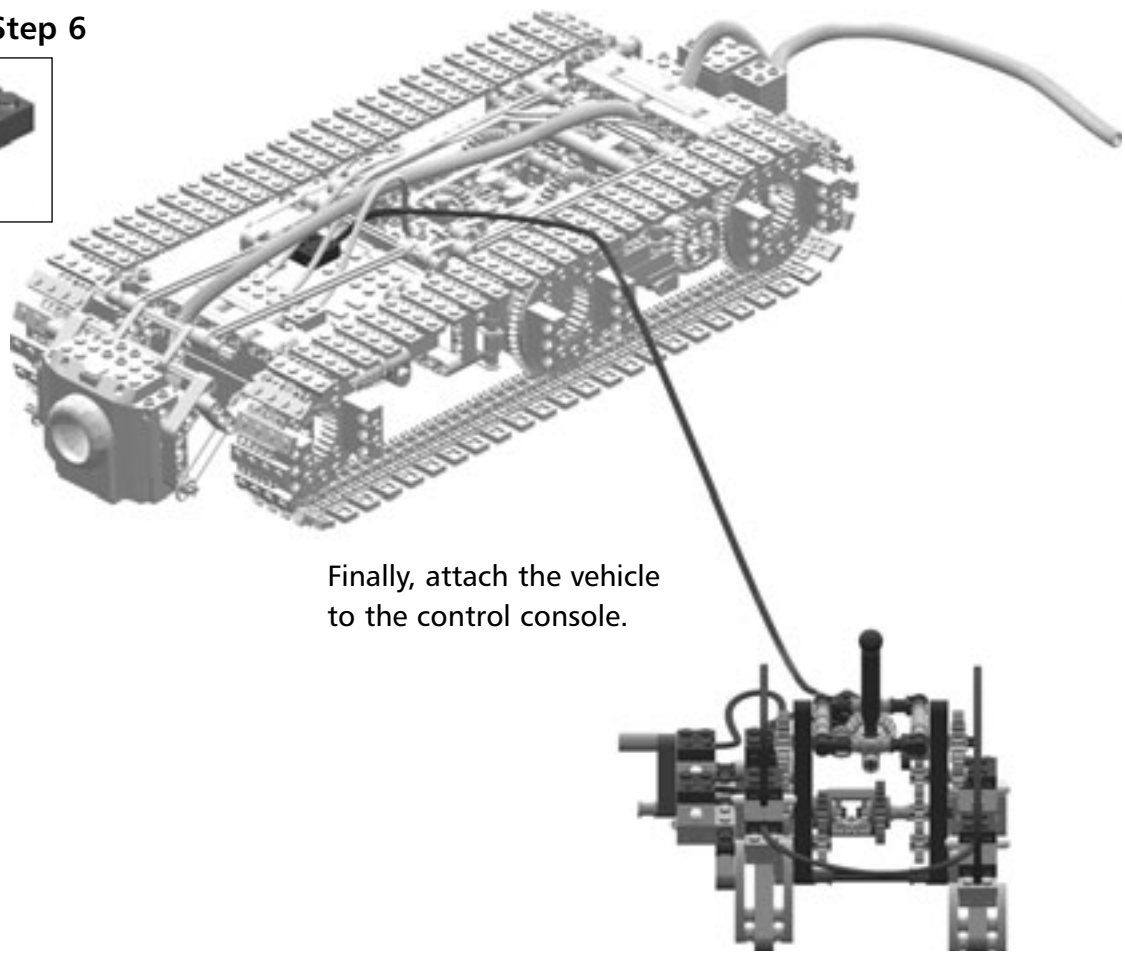


Cable management is not trivial with the SSCT. This is a very schematic view that offers a couple of tips. The main concern is the extra-tough cable of the Vision Command camera (rendered in the picture as a white ribbed hose). Use the Back Lock sub-assembly to safely trap it in place at the end of the vehicle, with just enough slack to allow for shape changes in the vehicle.

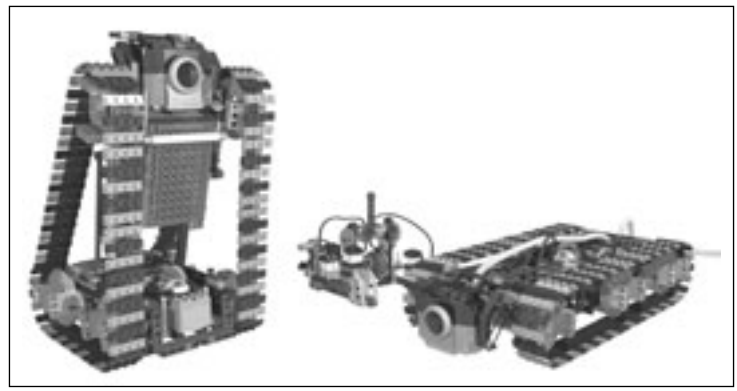
The cables going from the motors to the RCX are less problematic, but you can consider attaching the ones going to the propulsion unit motors to one of the long links. In Figure 6.1, the shape of these cables marks a suggested place to make this attachment.

Final Step 6





Finally, attach the vehicle to the control console.



Summary & A Few Customization Options

As I said at the beginning of the chapter, the large number of alternatives hidden inside the design of the SSCT will keep you occupied for quite a while. In the previous pages, I have included very specific implementations, but more important are the general principles behind them that will allow you to experiment further. Modifications come in all sizes, large and small. In the next few paragraphs, I mention some further options to continue working on the SSCT varied in breadth and scope.

First, the larger a custom model is, the harder it is to finalize. One of the all-time favorite activities of many Lego fans is to "touch up" the official kits, and the larger kits obviously offer a lot more customization options. When you are building a large model, it is really hard to call it quits, because there will always be two or three details at the back of your mind bugging you about it. When you finally get down to solving them, rest assured that a different set of issues creeps quietly into their place.

For instance, one minor issue not resolved in the above model is how to provide light for the camera to see. If our vehicle is going to explore unknown dark territory, it needs to bring its own light. The LEGO electric 1x2 stud brick with side light part used as a resistor in the joystick has a reflector accessory that makes for a nifty-looking lamp. Attaching several such lamps to the camera is very easy. However, feeding power to the lamps is not as straightforward. For once thing, you need an independent supply of it, since you probably want the light on even if the motors aren't moving. You can either add a battery box to the SSCT (I suggest using the accessory tray at the back) or provide it via a long cable that goes all the way to the command post. Each solution has its pros and cons and uses to its advantage in the elements already present in the SSCT: the accessory tray and a thick, tough cable (to which we can attach finer ones).

We can also add sensors to measure the environment in which the SSCT is operating. A temperature sensor within the view angle of the camera allows us to not only see, but to *feel* an object. Not only we can measure the temperature, but also use the sensor as a delicate ram. Since we are watching the action, we can perhaps judge the firmness of an object in base to its response to a bit of prodding by the sensor tip.

Because tracked vehicles have a lot of slippage, attaching rotation sensors directly to their propulsion devices does not translate into a very accurate translation into the distance traveled by the vehicle. In the SSCT, we can connect a rotation sensor to a regular LEGO wheel and drag this contraption behind the vehicle, attached to the accessory tray. This should provide a better reading of the distance traveled.

Of course, every time we attach something to the accessory tray, we make the vehicle larger and give up some maneuverability (the longer a vehicle is, the more space it needs to turn). If we take the RCX out of the vehicle (and connect its motors directly to control devices at the command console), the weight distribution of the SSCT changes quite a bit, but plenty of free space suddenly becomes available. We could move Folding Muscle Unit sub-assembly to the Front sub-assembly and incorporate the wheel-and-rotation-sensor inside the chassis, in the newly vacated space.

Similarly, we could take the Vision Command camera out of the equation. This way, we could program the SSCT as an autonomous robot capable of differentiating between short obstacles and a tall wall. The front assembly would carry a RCX pointed forwards at all times and would be constantly emitting Infrared light, while a light sensor also pointing forward would detect any Infrared light bounced back by nearby objects. This is a relatively standard way of detecting obstacles without actually touching them with LEGO MINDSTORMS robots, and the SSCT adds to it the possibility of detecting the height of the obstacle as well.

Finally, don't forget that the camera can be thought of as a sophisticated sensor. With the appropriate software, not only can it differentiate different light intensities, it also recognizes shapes and colors. This allows us to experiment with artificial intelligence options related to search and rescue. For example, if we can somehow mark known territory, we can also make the robot automatically recognize these marks. Let's say we drop brightly painted ping-pong balls into a crevice once it has been explored and deemed empty. Once the camera recognizes one of those balls, it could be programmed to let us know (by beeping a sound signal, for instance).

This simplistic marked territory example can nevertheless be stretched quite a bit and in fact become the basis of further developments into the inexpensive, deep-burrowing, life-sniffing robotic creatures mentioned at the beginning of the chapter. Indeed, when exploring a large chunk of unknown territory, numbers count. If we employ several robots at the same time and each robot can leave its own signal (a trail of a specific type of ping pong balls, for instance) and recognize and differentiate between the signals left by other robots, we have tools to shape the behavior of each robot and the group dynamically. Moreover, if the robots send that information back to us, we can create a clearer and more consistent composite picture of the unknown territory as it is being explored by our robotic vehicles.

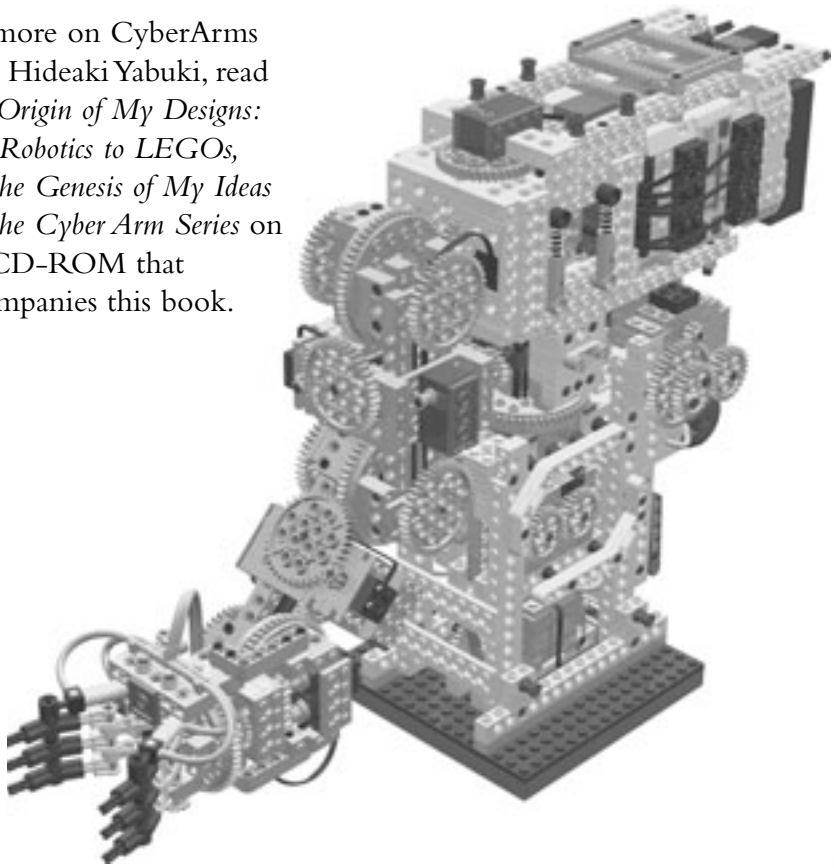
Play—and explore—well!

Masterpiece 7

CyberArm IV: Robotic Arm With Feedback Sensors

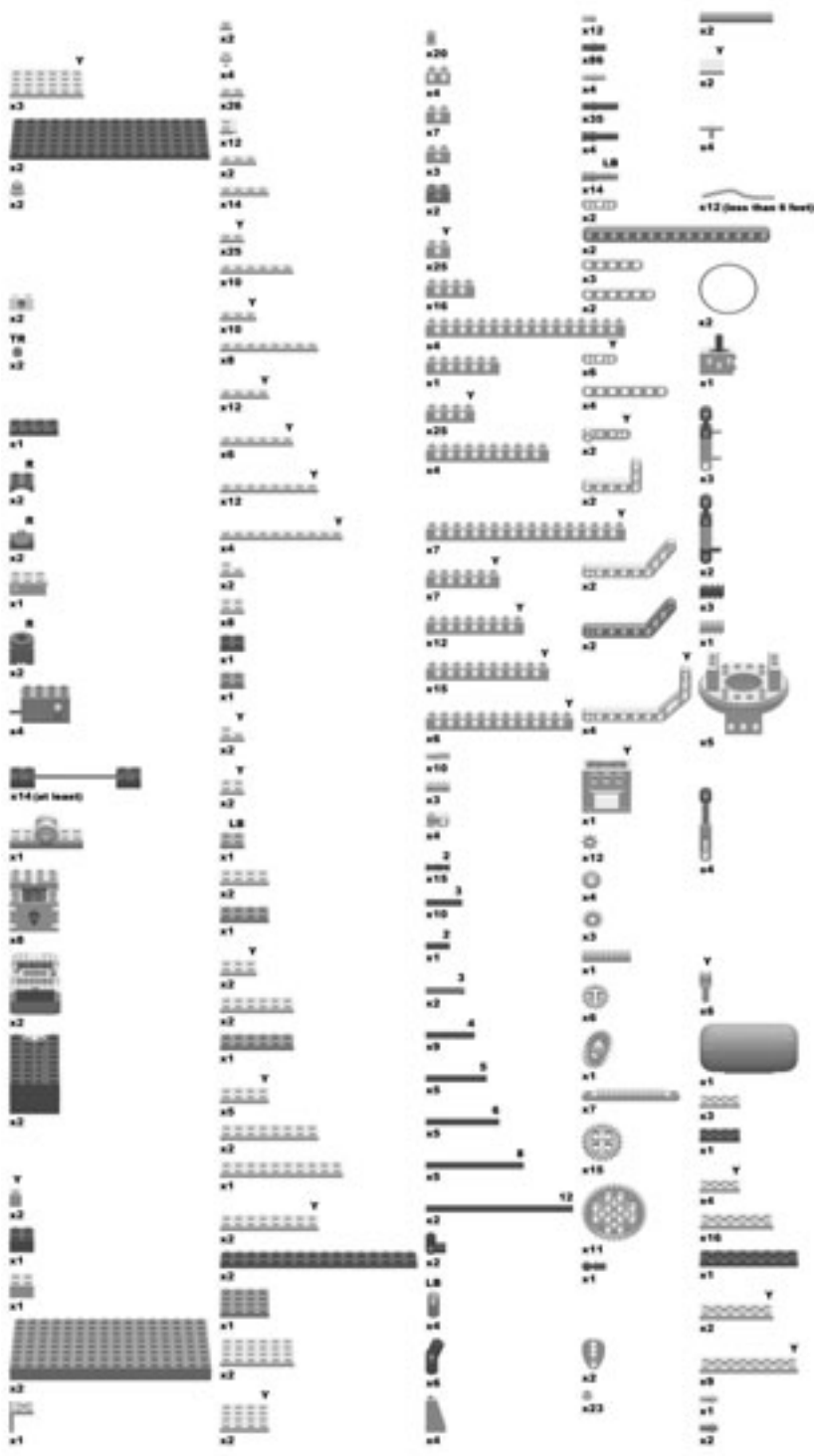
Hideaki Yabuki

For more on CyberArms from Hideaki Yabuki, read *The Origin of My Designs: from Robotics to LEGOs, and the Genesis of My Ideas and the Cyber Arm Series* on the CD-ROM that accompanies this book.



Bill of Materials

These are the parts you will need to build the CyberArm IV as shown:



Introduction

Robotic arms are my passion, and my original reason for acquiring the LEGO Robotic Invention System (RIS) 1.0. Though I enjoy building small robots, over the past four years, I have spent most of my building time building robotic arms. In building robotic arms, however, I realized the overwhelming superiority of the human arm over my creations. Without the support of the highly active LEGO community (which includes builders, programmers, designers and enthusiasts), my creations would have surely stopped long ago at a rudimentary level. With each iteration of my CyberArm series, I have been striving to improve my designs with the purpose of getting closer to the functionality and capabilities of a real human arm. Over the years, I have invented many small modular mechanisms for use in robotic arms, like the Power Glove sub-assembly which is used to control the CyberArm IV, which you will see at the end of this chapter.

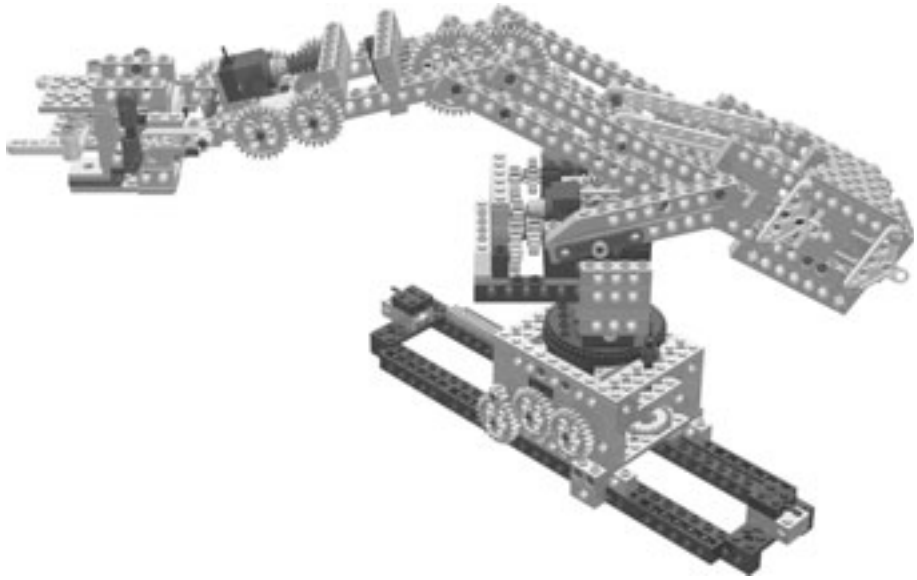
My CyberArm series differs from other robotic arms in object manipulation factors. Inventors must take into account the balance between weight, torque and the complexity and precision of the design. I love the organic workings of robotic arms. CyberArm IV combines many of these mechanisms, making this my greatest robotic arm yet.

History of the CyberArm Series

I have built five versions of the CyberArm series: I to V. The first version of my robotic arm, CyberArm I (1998), was only a prototype. For this arm, I attempted to create a power drive mechanism. For instance, I used wires to move various parts of the arm. As I was building this prototype, I began to collect LEGO TECHNIC parts to build a more complex creation.

My second attempt, CyberArm II (1999), can be seen in Figure 7.1. For this arm, the arm was driven by six motors and was constructed using over 600 pieces. This was my first experience using two RCX bricks simultaneously. CyberArm II moved horizontally along a track, while three micromotors controlled the grabber on/off, wrist up/down, and arm's angle adjustments. I invented a mechanism that could create motion along two axes using only a single motor—or so I thought. Shortly after I had built CyberArm II, I came upon a LEGO TECHNIC instruction booklet from 1989 (TECHNIC Set #8094–Control Center), which contained building instructions for a model that also used the concept of a robot being able to move the upper arm, the forearm, and even the grabber simultaneously by using only a single motor. I worked with this mechanism for several months but in the end, I could not enhance on the design beyond the specifications provided with Set #8485-1, and went back to the drawing board. I still love this mechanism and the interesting movements it produces, and am very proud that I was able to design an arm similar to the instructions for Set #8094 in my own way.

Figure 7.1 CyberArm II with Track



For my next creation, I decided to design and build a whole new arm based on the photographs on the RIS set box of the "Grabber Arm Version 1.0" created by master builder Anthony Fudd. CyberArm III (2000) was the result. It was designed to be driven by nine motors and was constructed with over 1,200 pieces. You can see CyberArm III in Figure 7.2. My goal for CyberArm III was to build an arm that was capable of picking up and moving an egg. CyberArm III was the champion of the LEGO Hall of Fame world contest on the LEGO MINDSTORMS official Web site in March 2000.

NOTE

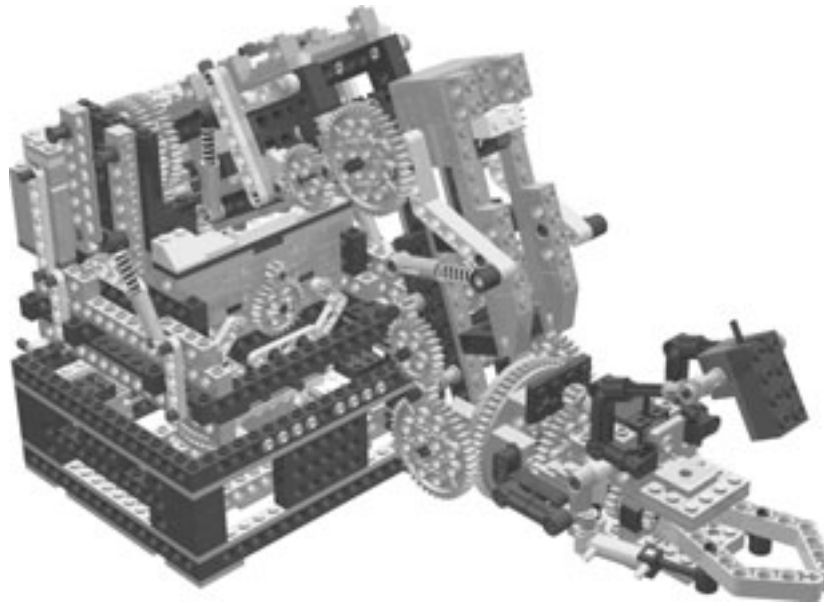
After winning the Hall of Fame contest, LEGO posted an interview with me. This interview is located on the MINDSTORMS Web site: http://mindstorms.lego.com/community/pioneers/joda.

However, the operative range was restricted for some of the joints in CyberArm III, and the large-scaled arm had a serious problem: due to its excessive weight, it was difficult to control the arm with timing alone.

NOTE

DAT files of the CyberArm II and CyberArm III are available in the Forum section on the official LEGO MINDSTORMS Web site: http://mindstorms.lego.com.

Figure 7.2 CyberArm III



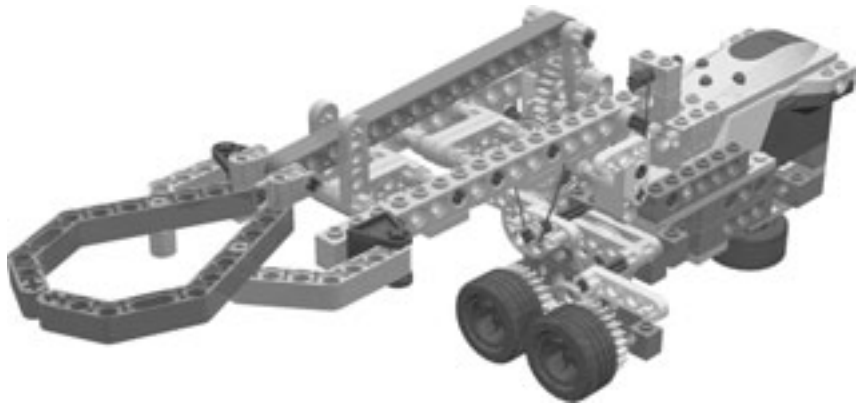
In a sense, CyberArm III was a mere mishmash of mechanisms and lacked sophistication. For my next project, I resolved to redesign the arm in an attempt to create a more precise, controllable design. CyberArm IV was the next robot in the CyberArm series. I built the first iteration of the CyberArm IV to correct the operating issues that I encountered in CyberArm III's powerful gear mechanisms. I decided to attach rotation sensors to each joint so that the arm could be controlled better using NQC programming. I was convinced that if I wrote the programs based upon sensor feedback, rather than calculation time, it would result in a more precise operating interface. After much trial and error and with great encouragement from my friend JP Brown, a master builder and creator of the famous "CubeSolver," I realized my vision: CyberArm IV (2001). By building a copy, JP Brown also helped verify the abilities of CyberArm IV when he succeeded in lifting a Rubik's Cube using four pneumatic cylinders.

NOTE

JP's CubeSolver was a monumental landmark in LEGO robotics. I encourage you to check out this MINDSTORMS Masterpiece on his site at: http://jpbrown.i8.com/cubesolver.html

After completing this first version of the CyberArm IV, I grew weary of using large parts and wanted an excuse to use Joe Nagata's gear changer with four 16T gears. I decided it was time for yet another addition to the CyberArm series. This time CyberArm V (2002) was the result. CyberArm V is controlled by moving your hand over the light sensor on a Micro Scout, and is powered by a clever mechanism consisting of a single Micro Scout motor, four pulleys, and two rubber bands. I first saw this mechanism used in the MINDSTORM Droid Development Kit (Set #9748), and the Exploration Mars Expansion Kit (Set #9736). To work this arm, only the power of concentration and hand gestures are needed, much like how Yoda can move object in the *Star Wars* movies. This is why this CyberArm V received the nickname of Force Trainer. I honestly believe the CyberArm V is one of the smallest MINDSTORMS grabbers in the world.

Figure 7.3 CyberArm V: Force Trainer



CyberArm Design and Construction

In the summer of 2002, I rebuilt the CyberArm IV to reinforce some of its structures, color coordinate the bricks, and fine-tune the positioning of the pneumatic tubing and electric wires. I attempted to optimize the combination of parts to see if I could adjust the design or any of the sub-assemblies for individuals who were interested in building their own CyberArm IV. Additionally, I completed the Power Glove control mechanism that enables real-time operation of the arm. The following is a list of a few exceptional features of this second version of CyberArm IV:

- There are five degrees of freedom (axis) when using a hybrid system of motors and pneumatics.
- The movement is smooth and accurate despite the large and heavy body.
- The arm is controlled by a sensor feedback system rather than time-based programming.

- Duplicated actions are possible because the arm returns to home position exactly.
- Modular construction makes replacing batteries or adding on components easy.
- Real-time operation is available with the Power Glove (optional).

Table 7.1 lists some of the vital statistics for CyberArm IV, including weight, height, and degrees of freedom.

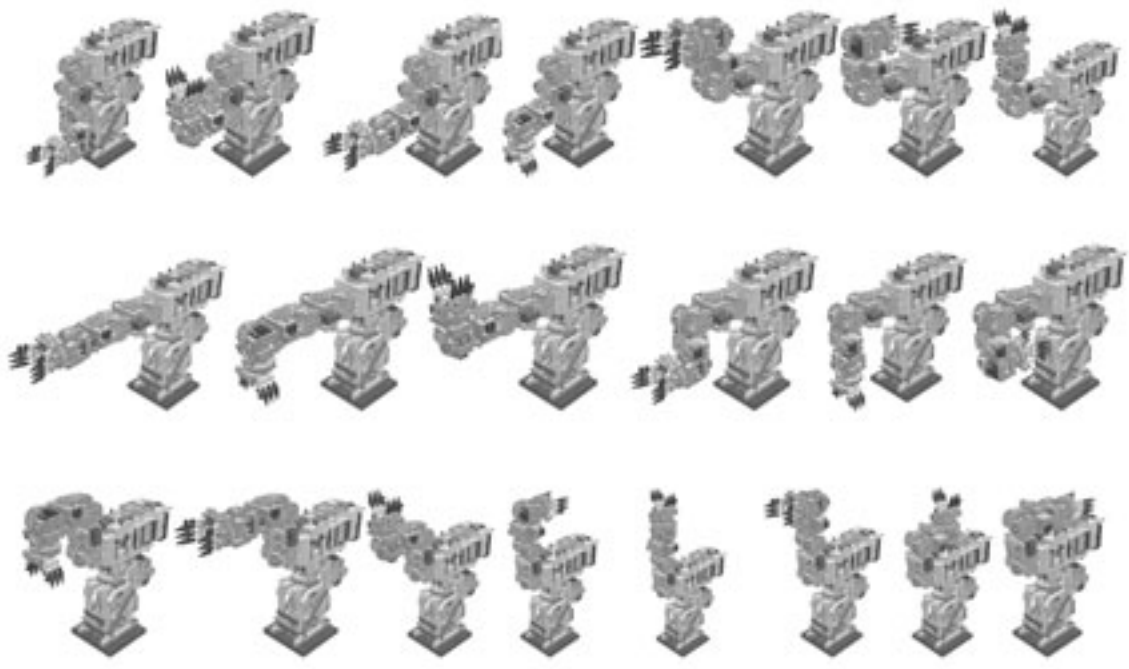
Table 7.1 Vital Statistics for CyberArm IV

Axis (Degrees of Freedom)	5
Pieces	Approximately 950
Number of RCXs needed	2
Number of motors needed	10 (eight 5225 geared motor two 5119 micromotor)
Number of gear needed	45+ (ignore ornamental)
Number of TECHNIC large turntable needed	5
Number of sensors needed	5 (four rotation sensor 1 touch sensor)
Weight	2.1+ kg (4.6 lbs) (arm: 700g back of arm with RCX: 750g tower: 650g)
Height (arm down)	Approximately 300mm
Maximum length of arms	Approximately 400mm
Vertical travel	Approximately 0 - 560mm
Horizontal travel	Approximately 70 - 300mm
Shoulder angle (with cables)	180 degrees (+/- 90)
Elbow angle (with cables)	180 degrees (+/- 90)
Wrist angle (with cables)	180 degrees (+/- 90)
Wrist turning (with cable and tubes)	270 degrees (+/- 135)
Turntable angle (with cables)	360 degrees (+/- 180)
Maximum lifting capacity	Approximately 150g

There are many differences between CyberArm III and the new CyberArm IV: the frame, gear mechanisms, and even the style. I reconsidered the design of the basic structure to allow better-articulated joints, better operation ranges, more precision, greater power and so on. In the case of my previous arms, the centerline of each arm was positioned on the beams holding the motors and the gears. Abandoning my previous symmetric design, I used asymmetrical ideas to develop an arm that had an ideal centerline location—through the arm itself. This eliminated the greatest obstacle in my previous arms: the arm itself! As a result, the new arm has an increased and exceptional range of movement. Every joint rotates using a LEGO TECHNIC turntable and has 180 degree

(+/- 90 degrees) of freedom even with all cables and pneumatic tubing connected. You can see the sequential movements of CyberArm IV in Figure 7.4.

Figure 7.4 Degrees of Freedom: Sequential Actions



The mounting of CyberArm IV rests on a tower and hangs down like a human arm. In CyberArm II and CyberArm III, the RCXs were kept outside of the tower, but embedding the RCXs into CyberArm IV turned out to be a successful and better design. Using two RCXs (about 500g) and two small 9V battery boxes (about 150g) for counterweight, the arm is balanced by itself. The tower is robust and can support the arm with two RCXs attached (weighing over 1,450g/3lbs) and the twin gear motors turn without any problems. Bi-directional (horizontal and vertical) building with LEGO parts proved to be more difficult than ever in this project, but I enjoyed the challenge. I even used some plastic washers to control sensitivity in the drive mechanism.

The Pneumatic System

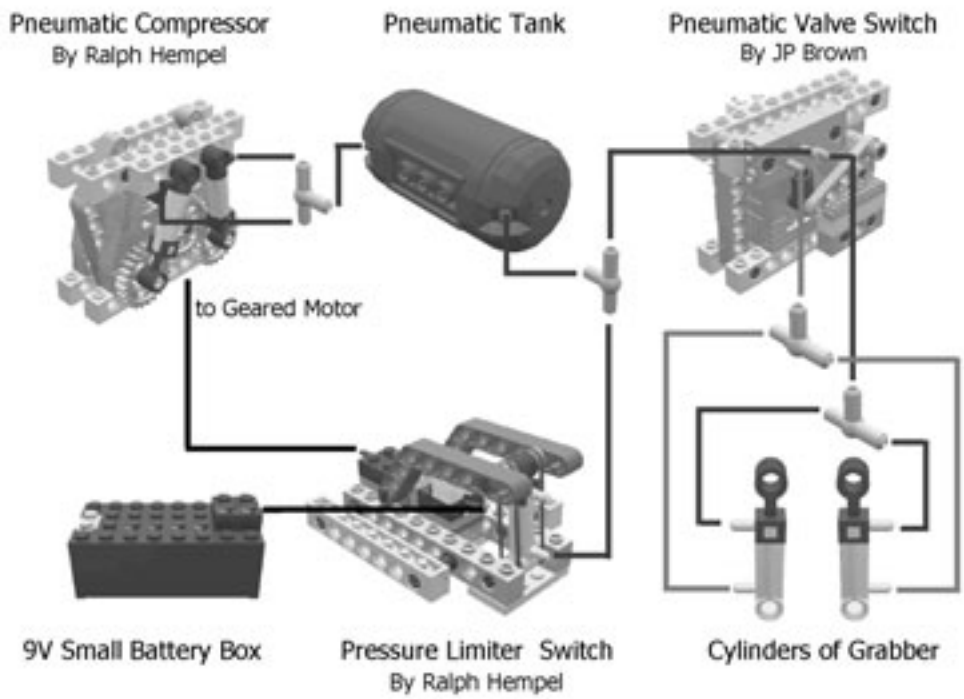
The tower contains many famous pneumatics components. The double-acting pneumatic pump and the pressure limiter switch were designed by Ralph Hempel (creator of the pbForth programming language). The quick pneumatic valve switch was designed by JP Brown: this is a mechanical (analog) system is useful to avoid accidentally powering down the RCX if the motor's current powering each joint and the compressor simultaneously ever gets too high. When the air pressure is too low, a cylinder on the pressure limiting switch contracts, causing the liftarms turn the pneumatic valve switch (polarity or pole

reverser) switch on. The double-acting pneumatic pump would then quickly and automatically replenish the pneumatic tank with air! Another advantage to this quick pneumatic valve switch is that it is compact and works quickly. Please refer to Figure 7.5, which illustrates the pneumatic schematics.

NOTE

You might want to flag this page, as the building instructions will refer back to Figure 7.5 when explaining the connections (both electrical and pneumatic) for the pneumatic system.

Figure 7.5 A Diagram of CyberArm IV's Pneumatic System

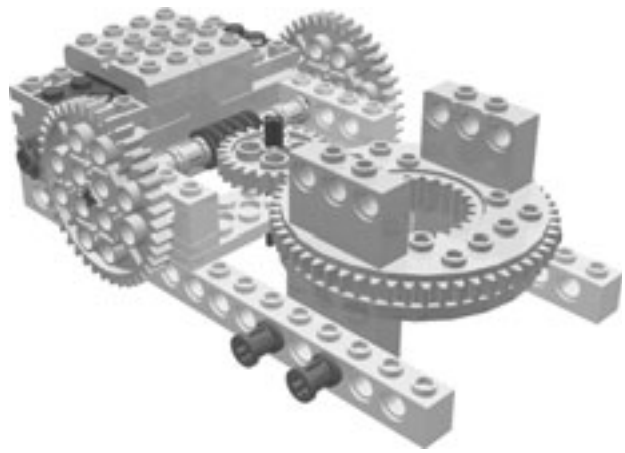


The Gears and Motors

Similar to my previous creations, CyberArm IV was built with modular components. I was convinced that my new arm would only be successful if the design incorporated the largest available TECHNIC gears: the TECHNIC turntables. The length of each arm segment is limited by the TECHNIC maximum beam length; in this case, 1x16 TECHNIC beams were used for their strength. Starting with the shoulder joint, I began working

with the new designs. The key to the success of CyberArm IV turned out to be the new drive mechanism for the shoulder joint that enabled it to be able to deal with the weight of the rest of the arm. Incidentally, CyberArm III used 22 gears in the shoulder joint, a design necessity and an act of desperation; so, for CyberArm IV, I wanted to use as few gears as possible. The result is the drive mechanism shown in Figure 7.6. As you'll see later, this shoulder drive mechanism is reincarnated (with a few modifications) in the elbow joint of the upper arm.

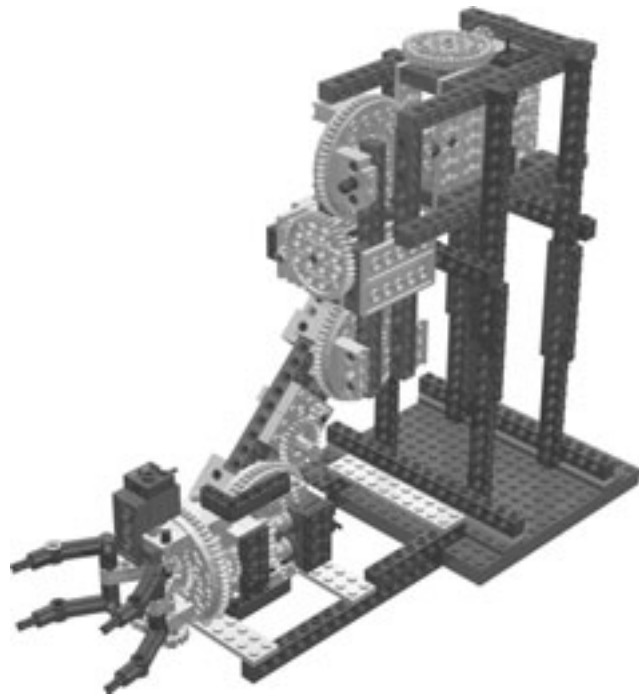
Figure 7.6 The Shoulder Joint Power Drive Mechanism with Twin Motors



I tried to limit the remaining components to only five motors for the grabber, wrist, and forearm sub-assemblies, in an effort to reduce the number of part, and the total weight or the arm; however, I was ultimately unsuccessful. The dual micromotors in the forearm sub-assembly often stalled due to weight overload, and I could not figure out how to build the powerful grabber sub-assembly with only a single micromotor. The biggest problem I found with grabber design was the groups of cables and pneumatic tubing needed to operate the grabber; they acted as a spring, and ultimately affected the joint. More torque was needed to overcome the natural forces created by the cables and tubing.

All great models require a prototype. The prototype for CyberArm IV is shown in Figure 7.7. At this point, I had decided upon the ultimate structure and had even figured out that I would be inserting two RCXs into the back of the arm as a counterweight for balancing purposes. In the end, I actually built two prototypes, so I could try different joint designs for comparison purposes before transferring final decisions to the main body of CyberArm IV.

Figure 7.7 An Early Prototype of CyberArm IV



NOTE

For a real industrial robotic arm, End Of Arm Tooling (EOAT) is optional. In the real world, EOATs include welding, airbrushing, and screwdriver tools. The grabber on CyberArm IV was designed to be similar to a real industrial robotic arm, by pulling out two L-shaped liftarms. When you want more gripping power, you can change the power grabber to use four pneumatic cylinders or a worm gear drive mechanism. Object manipulation factors include power, operation range, precision control, and various EOAT.

Getting Down to Business: Things to Keep in Mind While Building CyberArm IV

The power of the drive mechanism for the shoulder joint exceeded my expectations, so I decided to embed four of these drive mechanisms in the arm (the wrist is still powered by micromotors). The movement of each segment of the arm is smooth and deliberate but not too slow. Even a completely straight arm is no problem for these joints. I found that if two arm segments worked at the same time (for example, the forearm sub-assembly and wrist sub-assembly), the combination would act like a parallel linkage

mechanism. As a result, the grabber can maintain a level surface and, for example, not spill water from a cup.

Things to keep in mind when building the CyberArm IV:

- The CyberArm IV consists of two major components: the tower and the whole arm.
- The tower consists of the seven sub-assemblies: the Base, the Pneumatic Compressor, the Pneumatic Valve Switch, the Tower frame, the Turntable Base, the Pressure Limiter (Polarity) switch and the Turntable Drive sub-assemblies.
- The whole arm consists of the following six sub-assemblies: the Grabber, the Wrist, the Forearm, the Upper Arm, the Shoulder, and the RCXs.
- The tower frame is designed so that the RCXs can be easily separated from the tower when the batteries need to be changed. Another element of the tower design is that the arm can be separated from the tower by removing only a few TECHNIC pins.

Of course, I also included a drive mechanism for the turntable on the tower. The pneumatic systems must be set in the tower, so I had to abandon my original idea of setting the driving component of the turntable just under the arm (the armpit). After much deliberation, trial and error, and trying to strike a compromise between weight balance and strength, I decided that the turntable would attach to the arm in reverse. Although I wanted an all-in-one unit, I went ahead and separated the TECHNIC turntable from the gear mechanism. I believe this detachable design would allow for easy maintenance: sliding the component into the tower horizontally and meshing it with a TECHNIC turntable, while secured on a robust ceiling board.

Finally, I finished my design of the CyberArm IV using 10 motors. Six ports on two RCXs operate the six moveable parts. The moveable parts are:

- The turntables of the tower
- The shoulder
- The elbow
- The wrist
- The wrist rotation
- The grabber (opening/closing motion with a pneumatic valve switch controlled by a geared motor)

The turntables of the tower, shoulder, and wrist rotation use two motors each because of the large amount of torque required. Another additional motor is used independent of the RCX to run the compressor. I thought about using fairing for decoration, but gave up to avoid an even greater load on the motors. In fact, I hear the faint sound of motors

humming when the batteries are low, so please be careful and take care of the motors: the design draws as much power from the motors as possible.

The most difficult part of the reconstruction was arranging the cables. I have attempted to thread all the power cables and pneumatic tubing into the arm through the tower because I think a professional work needs to have beautiful cabling. As a result, over 20 cables and tubes previously strewn about outside have now disappeared. This leads us to the next list of things to remember before attaching the wires and cables to the CyberArm IV:

- Before you begin, you will need to remove all of the decoration from the 40T gears, several sub-assemblies, and all of the electric wires and pneumatic tubes that pass through the inside of the arm.
- You might have to cut tubes for specific components depending on the situation. Conversely, some users might have to splice pneumatic tubes or electric wires during the building process.
- You have the option of using the TECHNIC Flex System hoses (flex tubes) as a substitute for the couplers (joints) used to join separate pneumatic tubes.

I did not think about color coordination when I first built CyberArm IV. However, during the rebuilding process, this was something I took very seriously. The answer, in the end, was very simple: Since the colors of the motors and gears are light gray, and the RCX is yellow, the color scheme was easily decided:

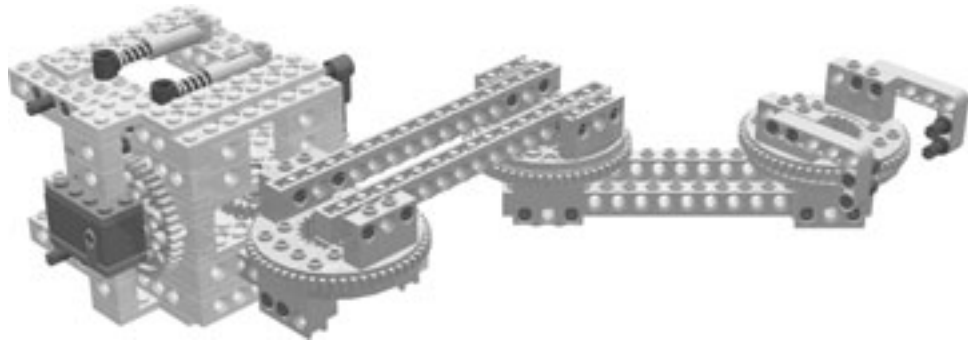
- The driving mechanisms with gears are light gray.
- The structural frame, including the tower, is yellow.
- Blue is used for a bit of charm and the remaining parts were black and dark gray.

Adjusting the color of the LEGO parts made CyberArm IV even more impressive looking than before. Having so many black parts replaced by yellow parts made the entire creation brighter.

WARNING!

My creation is a large model with huge sub-assemblies and consisting of many pieces. Therefore, I cannot make non-stop instructions as the parts would look really small in the directions. For example, the arm itself is composed of the shoulder, upper arm, forearm, and wrist, each with a LEGO TECHNIC large turntable. If you created the components separately, you would not be able to connect them firmly with black pins. Therefore, you must build the framework (with turntables and beams as seen in Figure 7.8) first, and then add each component's part to the beams.

Figure 7.8 The Framework and All Joints of the Whole Arm



The arm itself is an asymmetrical design. You must pay attention to the direction (up or down) of the beams and the turntables that are alternately arranged. You should also pay attention when handling the Power Drive Mechanism. If a finger is caught in the driving gears, you could end up hurting yourself. Another word of caution: be careful not to drive the arm outside its operation range. If you do, then you may end up breaking some parts.

My suggestion is to not follow the instructions brick by brick. Use these instructions as a guide to for the basic structure of each component and mechanism. I think you may be able to create your own version of the CyberArm IV using the bricks and sets you have at your disposal. For the most part, many of the pieces used in CyberArm IV are readily available. The exceptions that are less readily available are included in the following list:

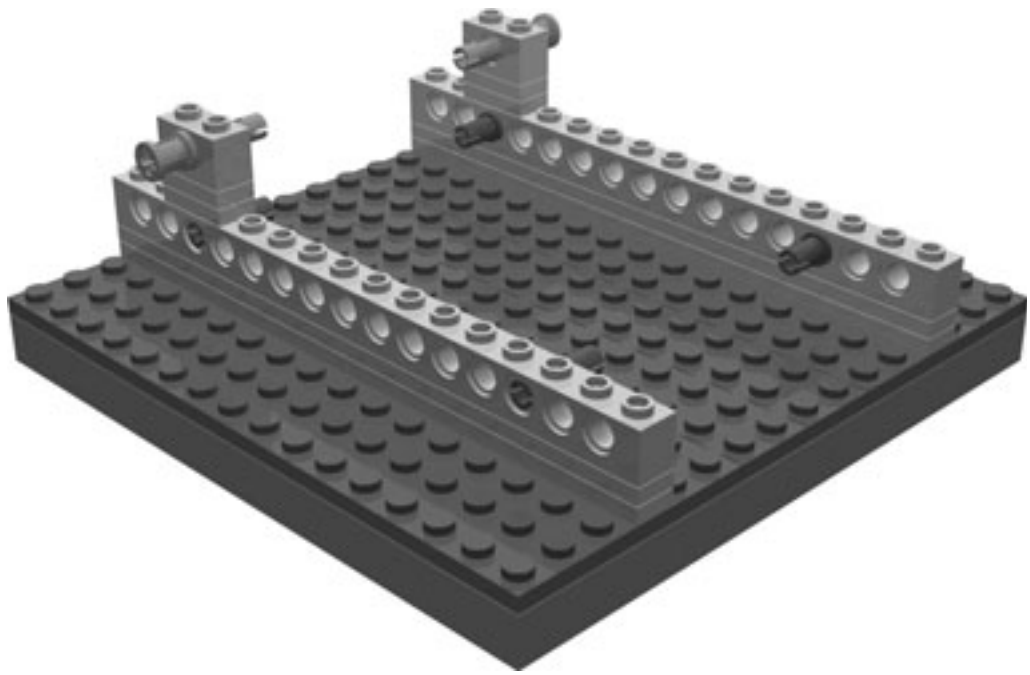
- **One conducting electric plate** used to connect two micromotors each other. Substitution is two electric wires.
- **Two small 9V battery boxes** used to supply power to the pneumatic compressor and balance weight. You can purchase this (Item#: 5391) from LEGO Shop At Home (http://shop.lego.com), or you may need to reposition the RCXs further back and use large 9V battery box (Item#: 5115) separately.
- **One pole reverser (Polarity) switch** used in the pressure-limiting switch. Please refer to the following section "Two Alternate Designs for the Pole Reverser Switch".
- You may find **micro- and geared-motors, specific gears, TECHNIC turntables, and rotation sensors** at Pitsco LEGO Educational Division (www.pitsco-legodacta.com).
- **Several pneumatic parts** are handled at LEGO Shop At Home: TECHNIC Pneumatic Pack with Air tank (Item#: 5218) and Pneumatic Tubing (Item#: 5109).

- **LEGO Remote Control** (Item#: 9738) is available separately or by Ultimate Accessories Kit (Item#: 3801) at LEGO Shop At Home (http://shop.lego.com).
- **Two 3x5 L Shape TECHNIC liftarms** are used on the wrist. These parts are included within the expansion set of Vision Command (Item#: 9731, two black), Ultimate Builders Set (Item#: 3800, two yellow and two black), or BIONICLE (Tarakava: Item#: 8549, four light gray).
- **TECHNIC Gearbox 2x4 x 3&1/3** in the turntable drive. These parts are included within the expansion set of Vision Command (Item#: 9731, two clear) and Ultimate Builders Set (Item#: 3800, one clear) or Exploration Mars (Item#: 9736, two yellow).

NOTE

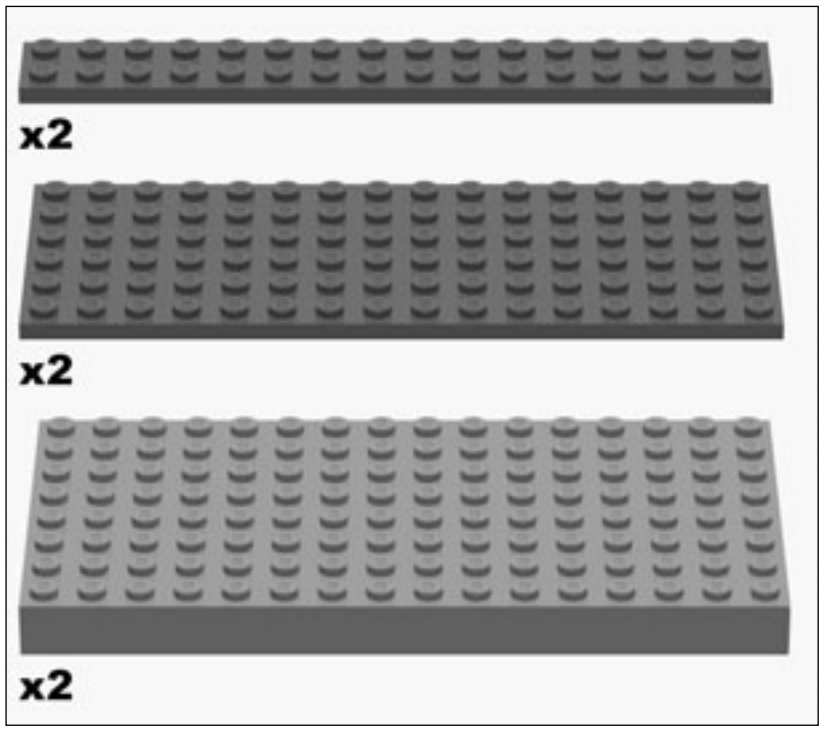
Fear not—many of these many of these elements can be readily exchanged for other solutions you may find through your own creativity and perseverance.

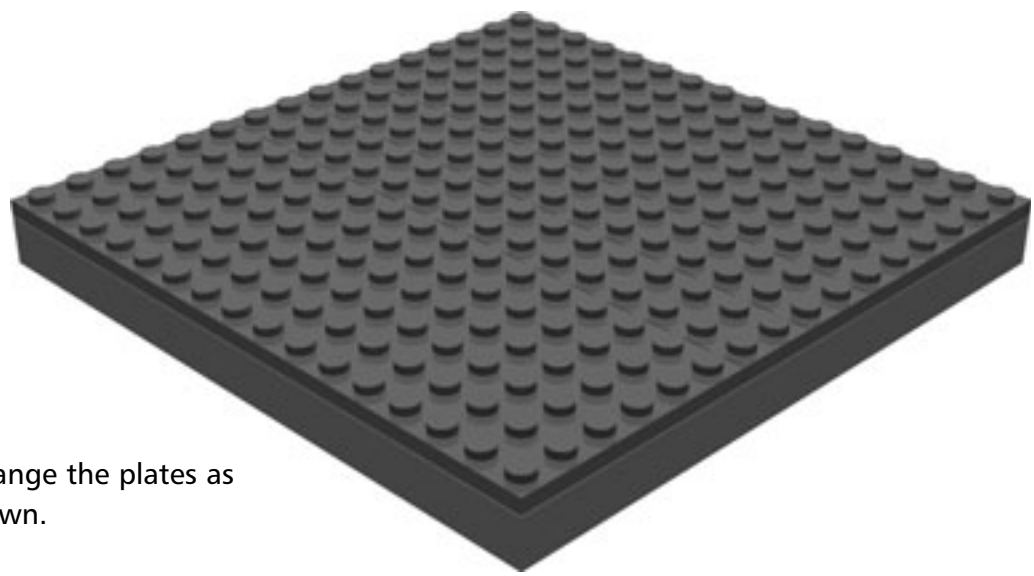
The Base



The Base is used as a plain surface to firmly hold the arm. The steadiness of the arm is secured with minimum space.

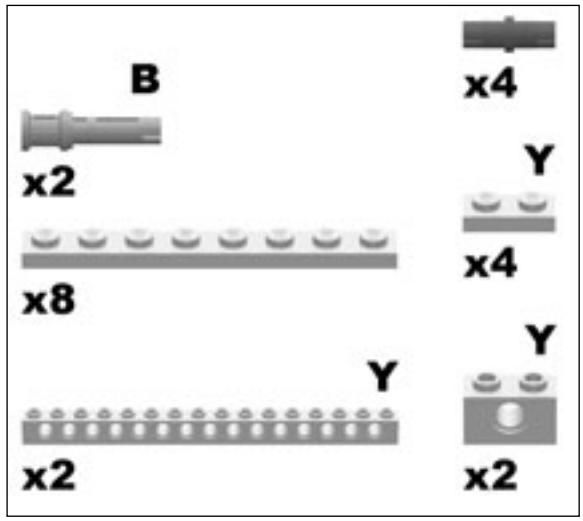
Base Step 0

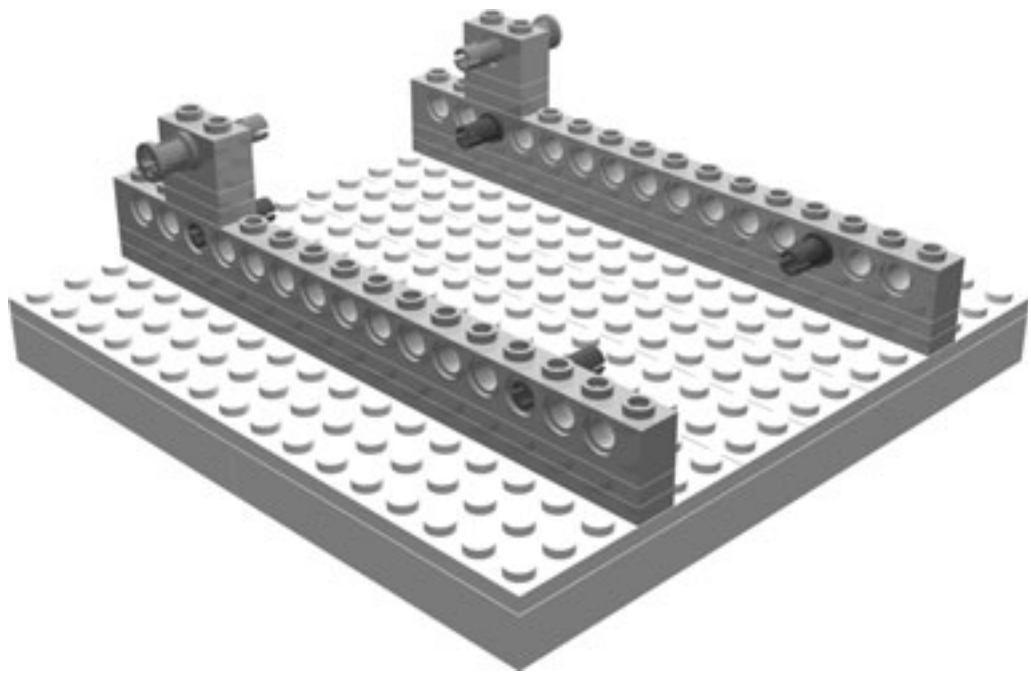




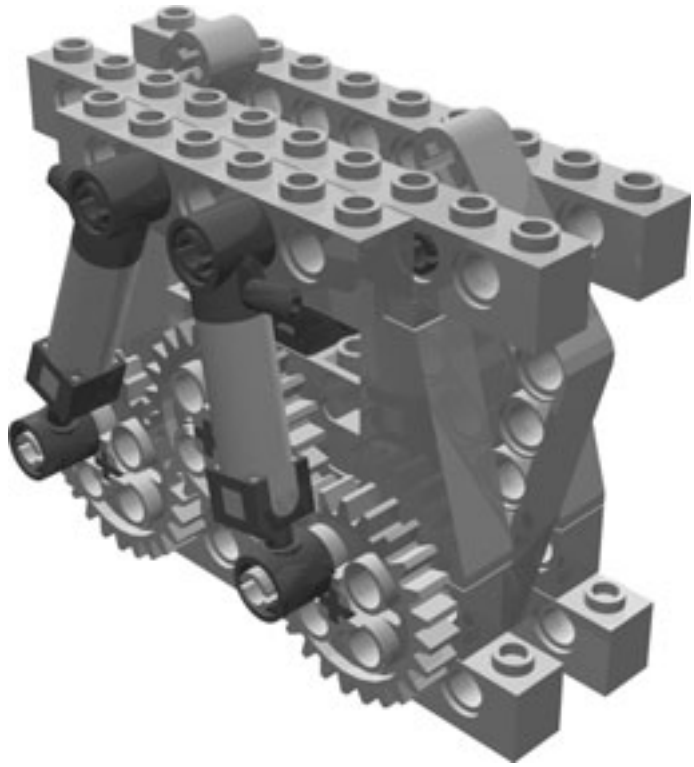
Arrange the plates as shown.

Base Step 1



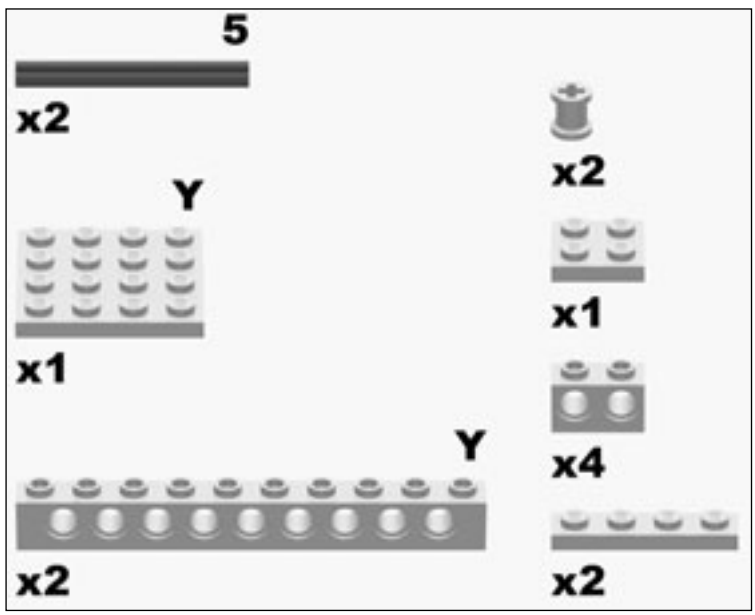


The Pneumatic Compressor

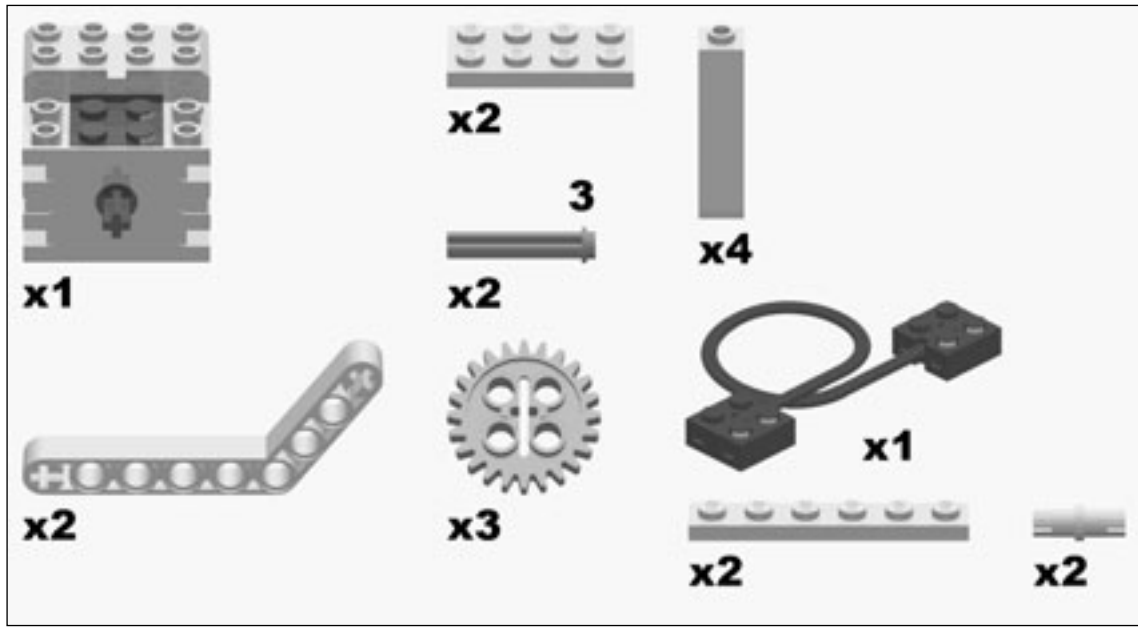


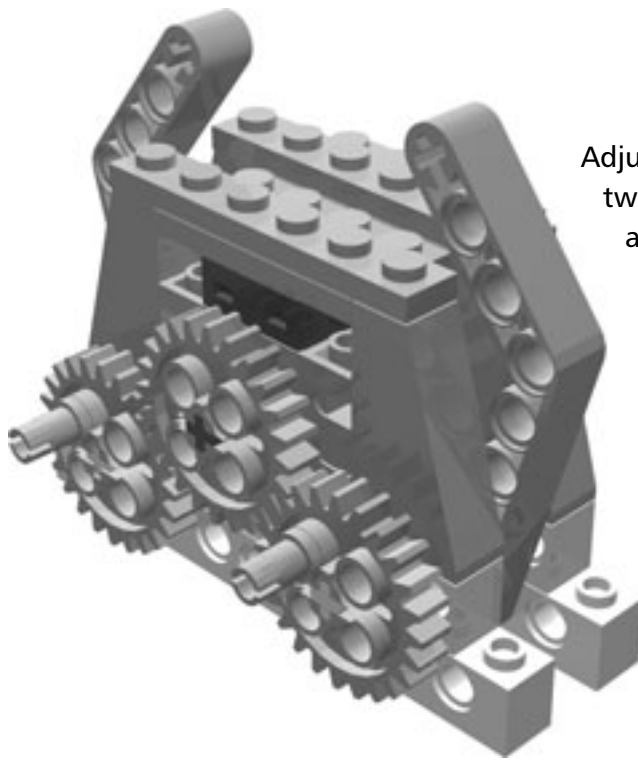
Ralph Hempel is the original designer of the Pneumatic Compressor sub-assembly. This Pneumatic Compressor sub-assembly is both small and powerful.

Compressor Step 0



Compressor Step 1

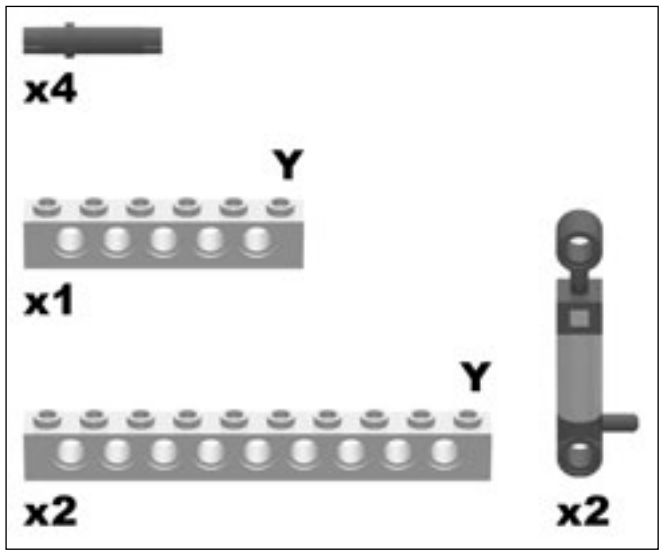


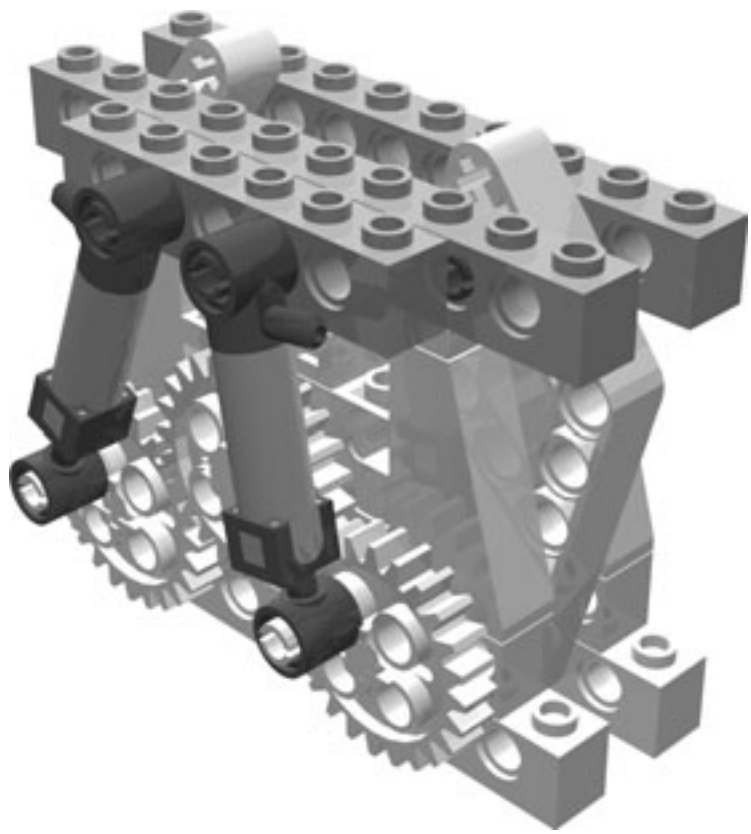


Adjust the phases of the three gears and the two TECHNIC pins. When the four pulleys are attached to the 24T gears, it will be more powerful because of the expanded working ranges of the cylinders.

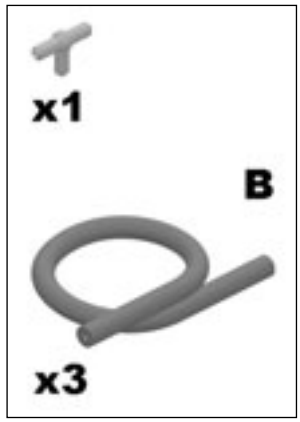
The electric wire used in this step will be connected to the pressure limiter switch in **Limiter Switch Step 0**. For a visual reference, you should refer back to Figure 7.5.

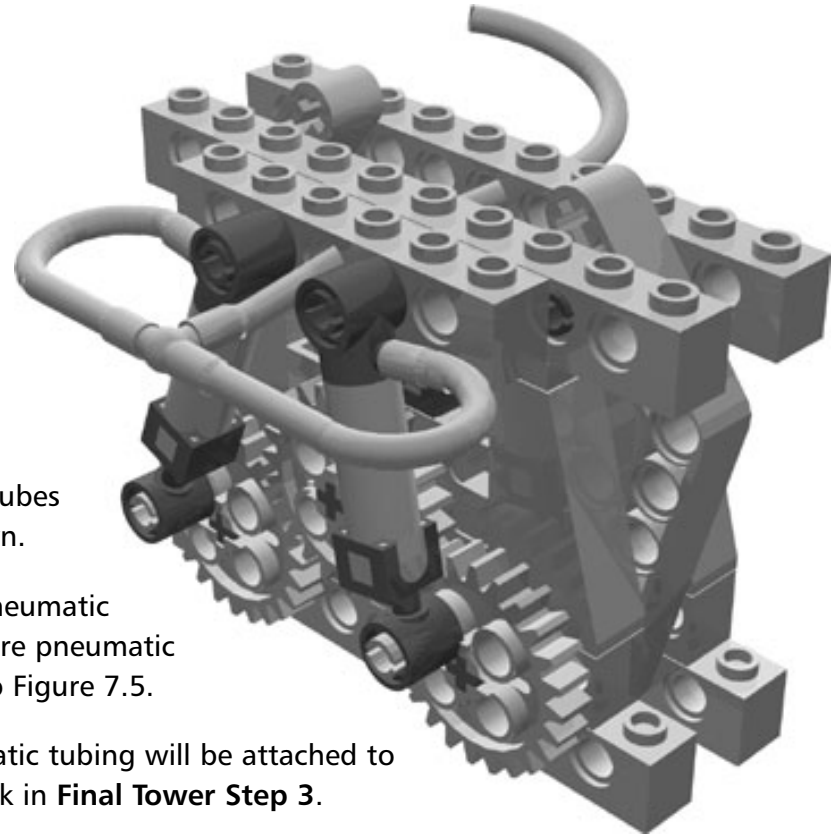
Compressor Step 2





Compressor Step 3



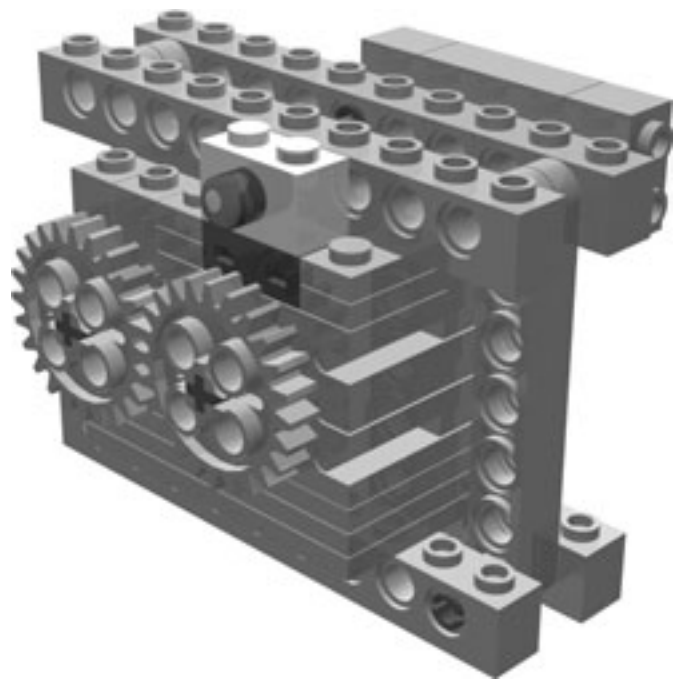


Please connect pneumatic tubes with a T-connector as shown.

For an overview of these pneumatic tubes in relation to the entire pneumatic system, please refer back to Figure 7.5.

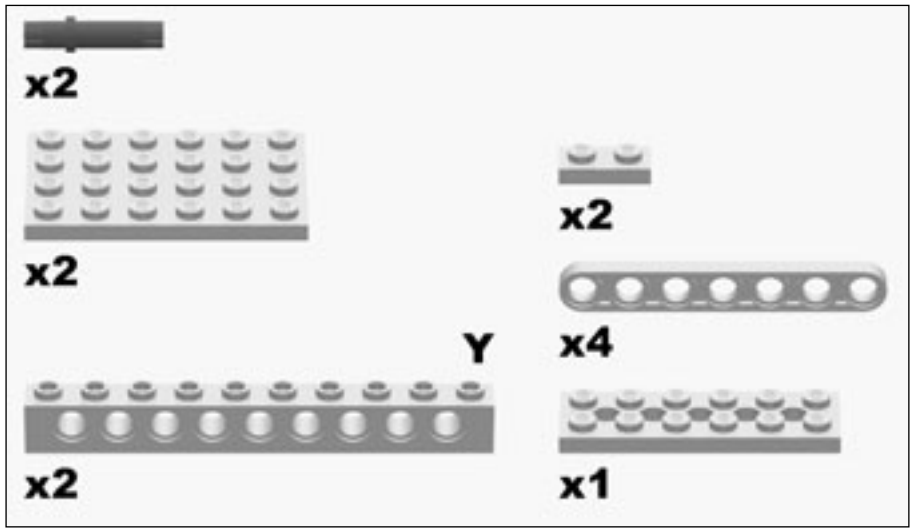
The free end of the pneumatic tubing will be attached to the pneumatic pressure tank in **Final Tower Step 3**.

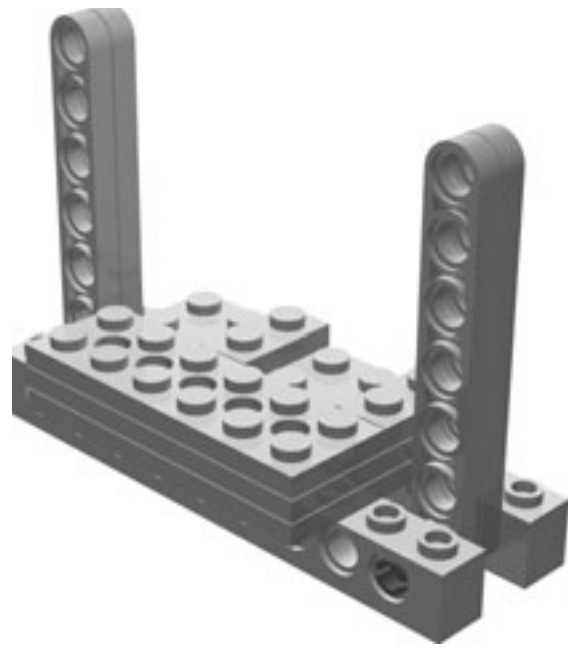
The Pneumatic Valve Switch



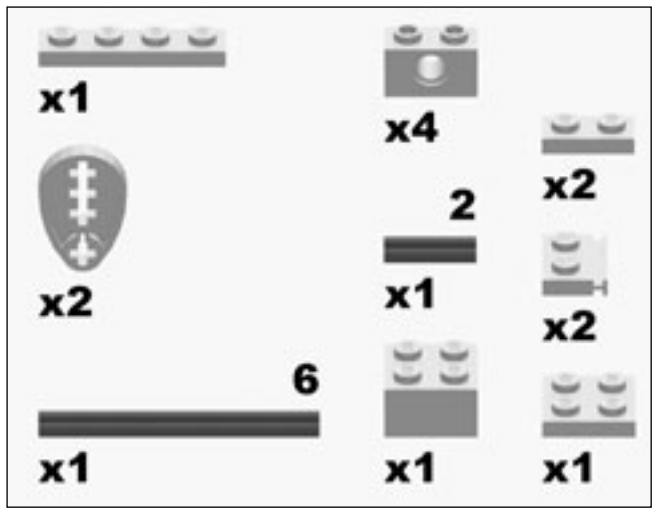
JP Brown originally designed this sub-assembly. This component is used in CyberArm IV because of its small size and its ability to quickly change the position of the valve.

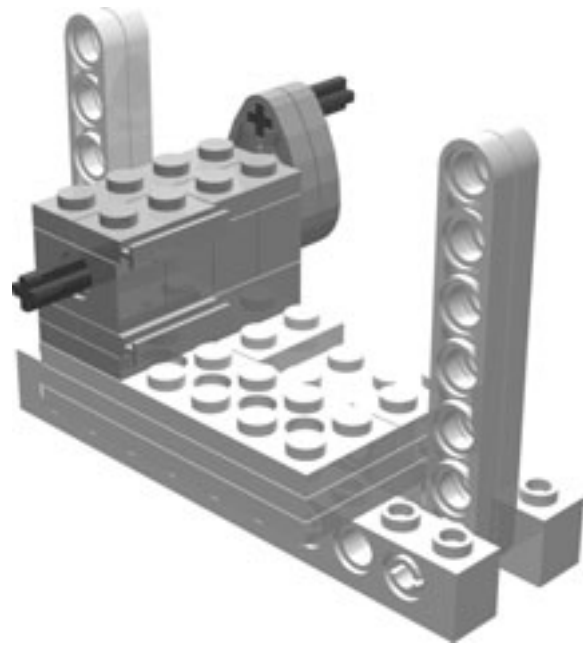
Valve Switch Step 0



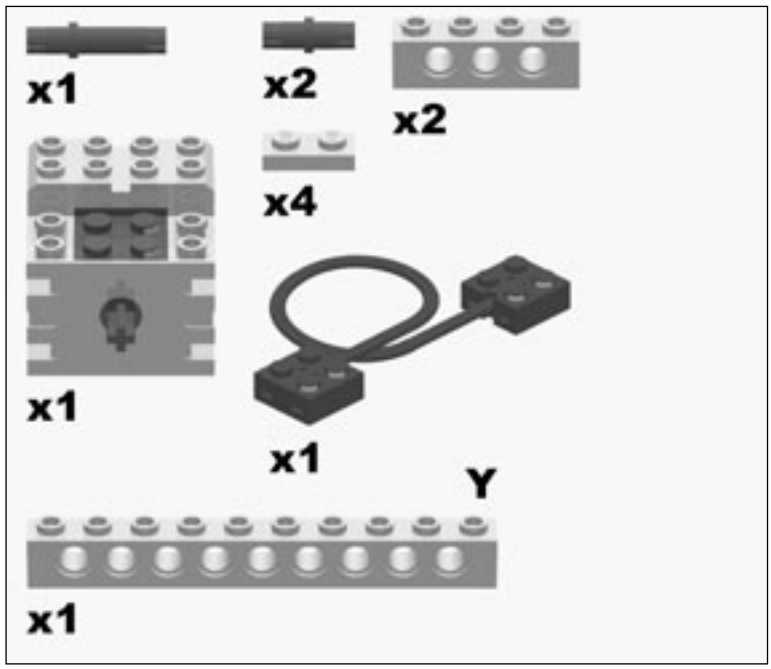


Valve Switch Step 1

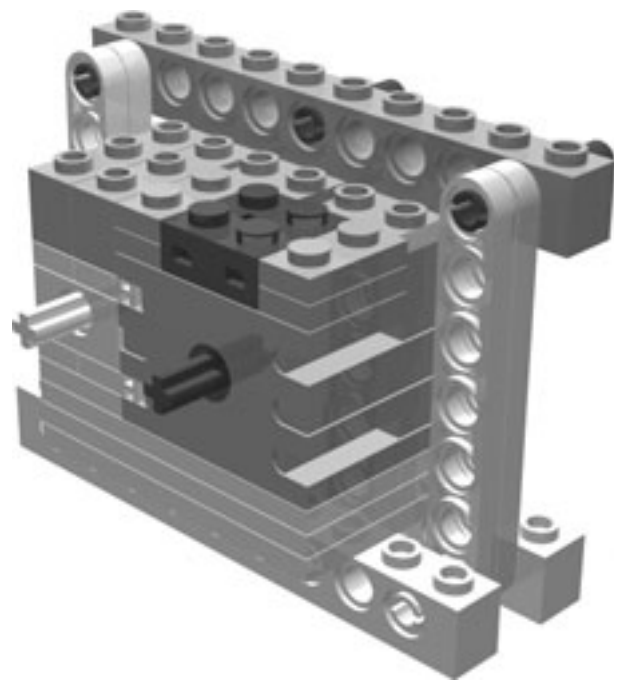




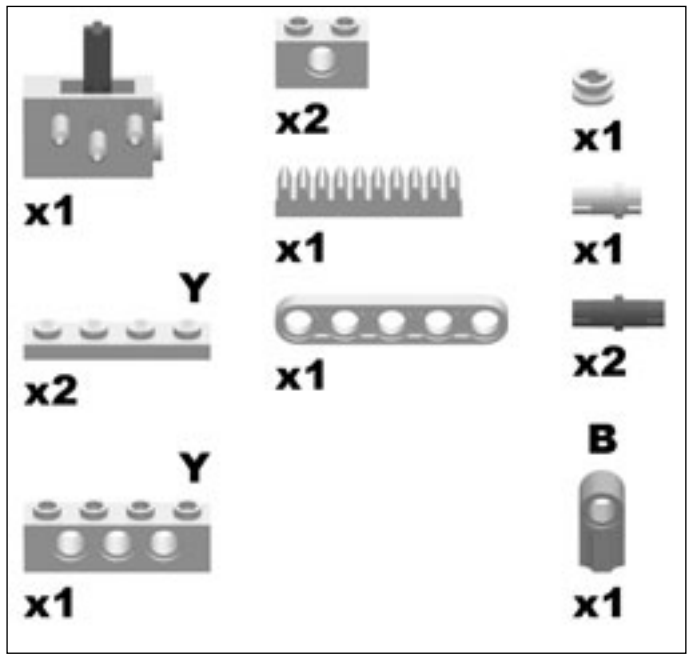
Valve Switch Step 2



The electrical connector in this step is attached to Output Port A on RCX 2 in **Final Arm Step 4**. For a visual reference, you should refer to wiring schematic diagram in Figure 7.13, found later in the chapter.



Valve Switch Step 3

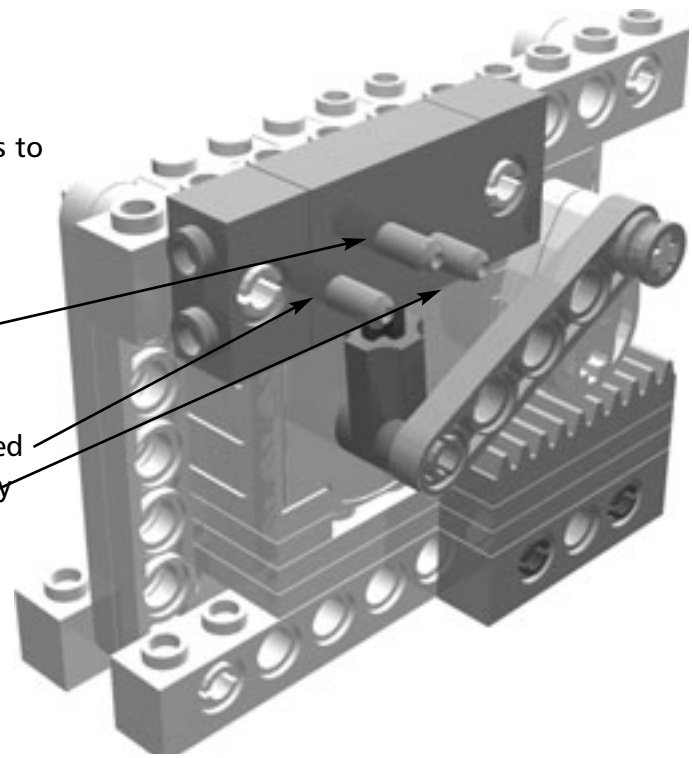


In future steps you will be connecting pneumatic tubes to these ports.

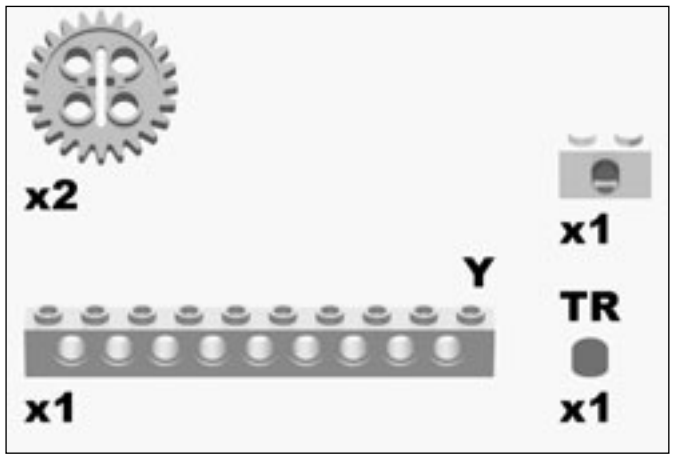
This port will connect a T-Connector for the pneumatic tank in **Final Tower Step 3**.

These ports will be connected to the Grabber sub-assembly in **Grabber Step 4**.

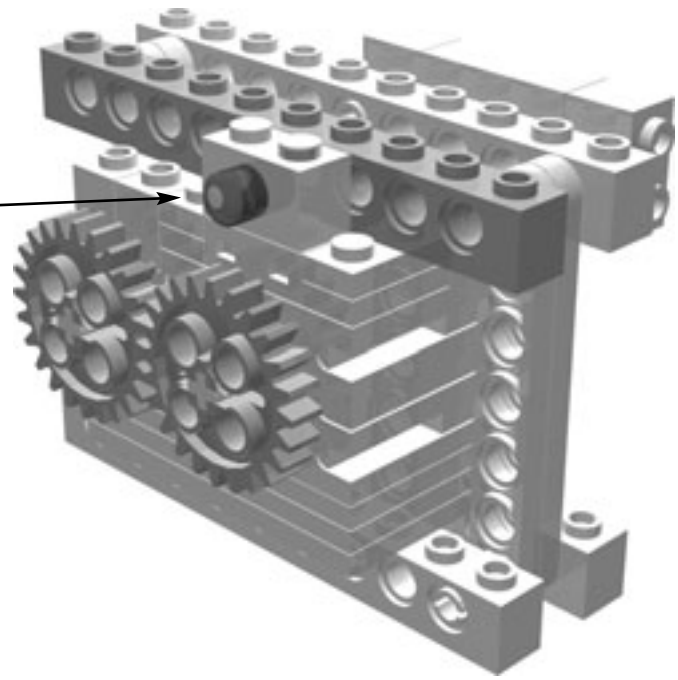
You should refer back to Figure 7.5 for assistance on tubing placement, and for a detailed schematic of the pneumatic system.



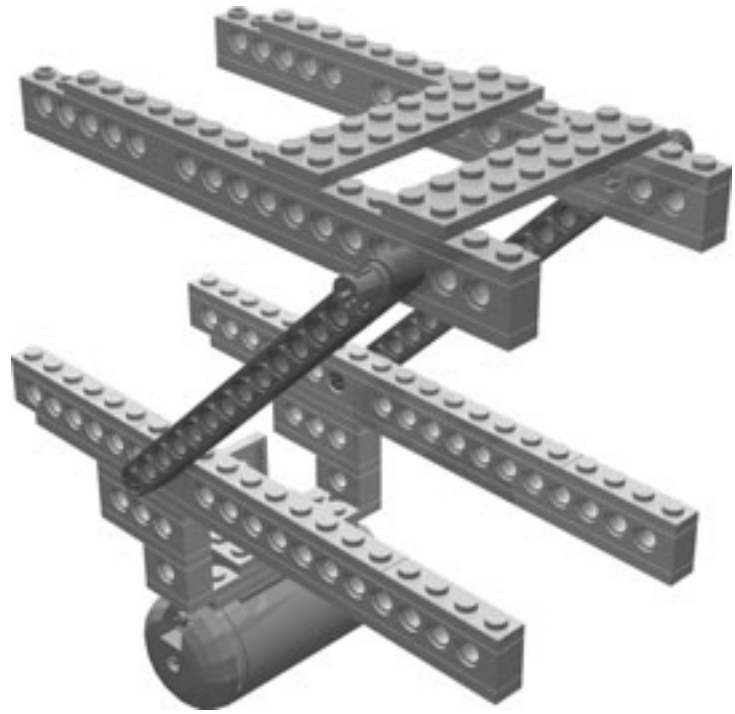
Valve Switch Step 4



This lamp brick is used as an indicator to confirm that the mechanism is working properly.

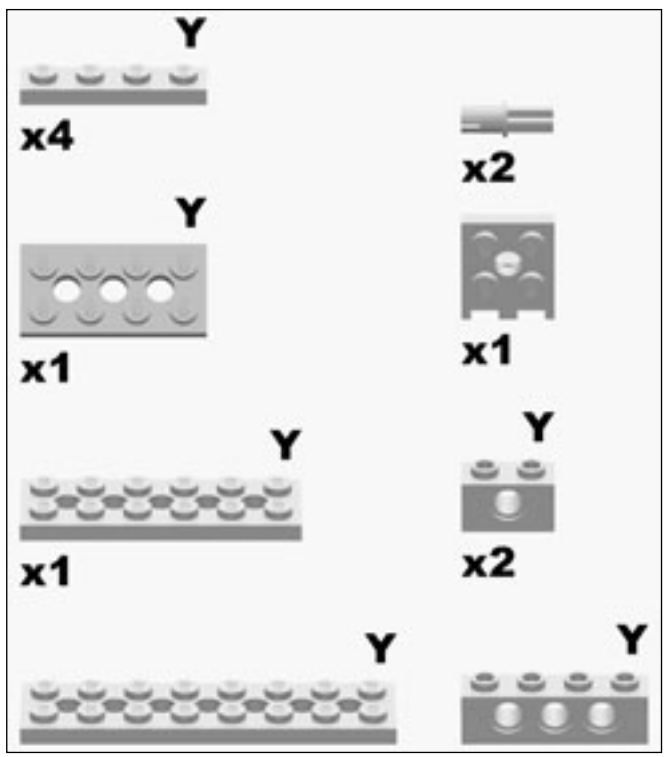


The Tower Frame

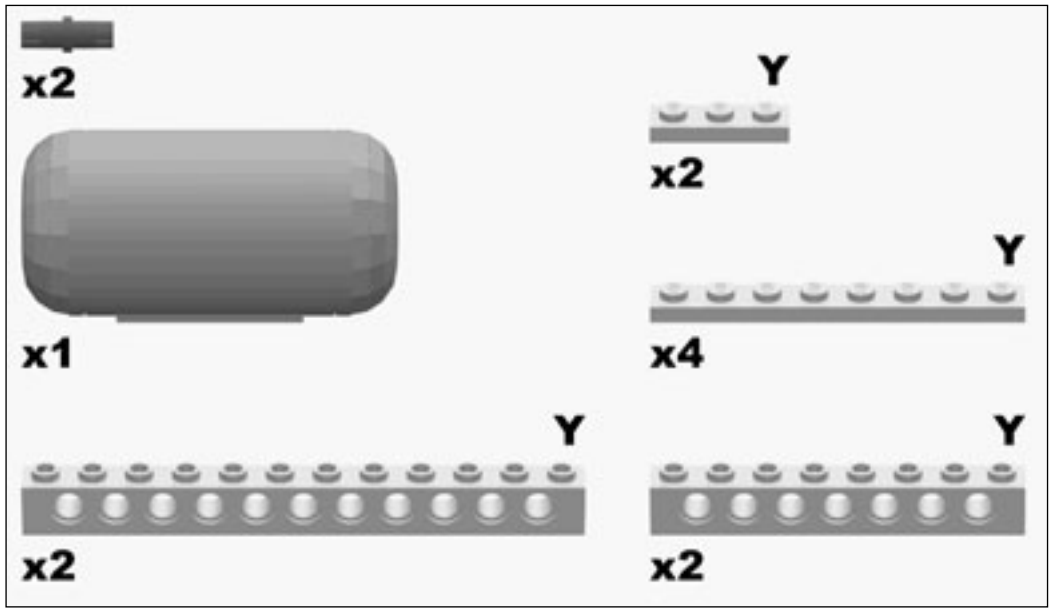


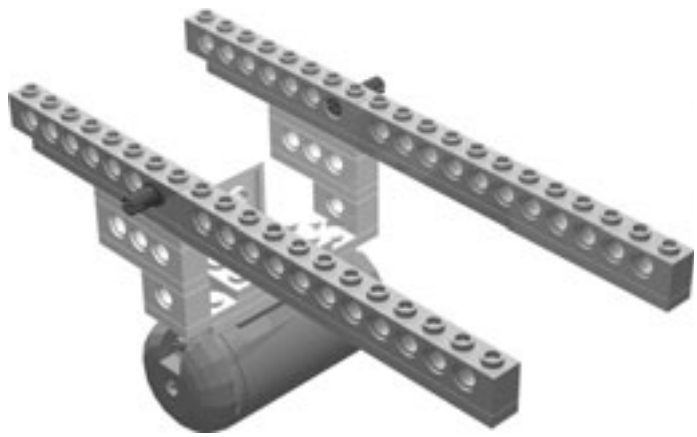
The Tower Frame sub-assembly provides the connection points for the components of the completed Tower sub-assembly. In addition to providing a boxy framework for the elements of the Tower sub-assembly, the Tower Frame sub-assembly contains the air pressure tank for CyberArm IV.

Tower Frame Step 0

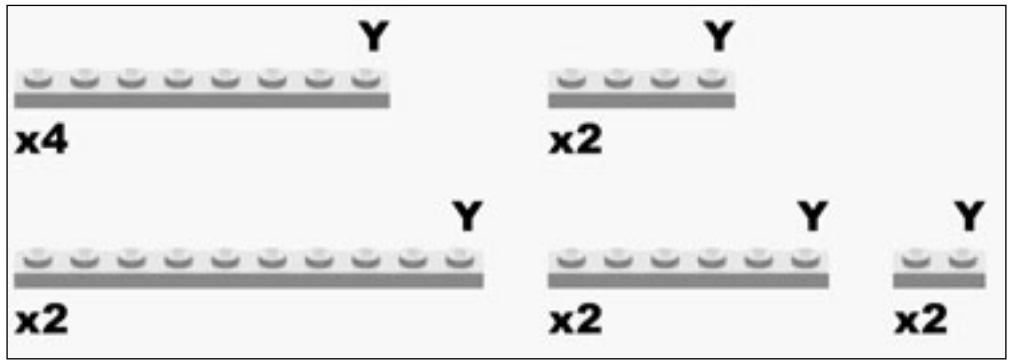


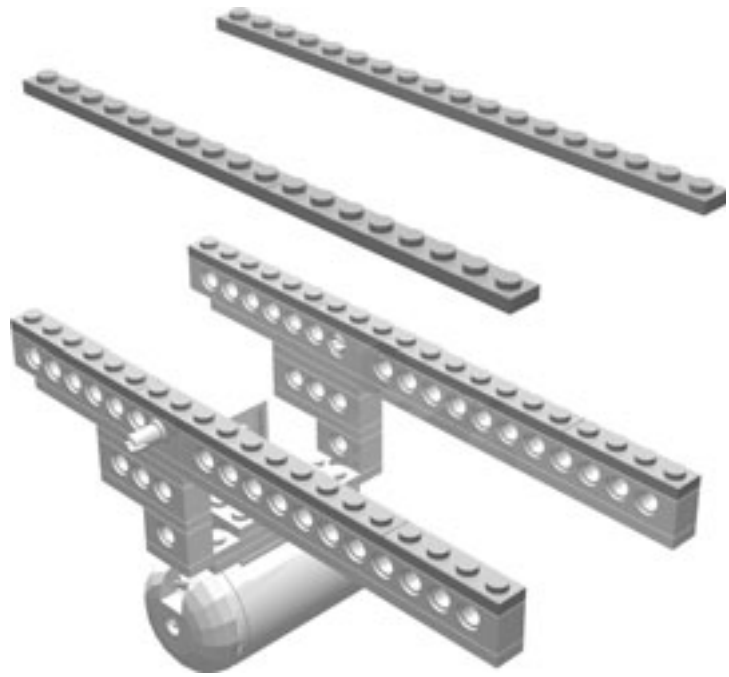
Tower Frame Step 0



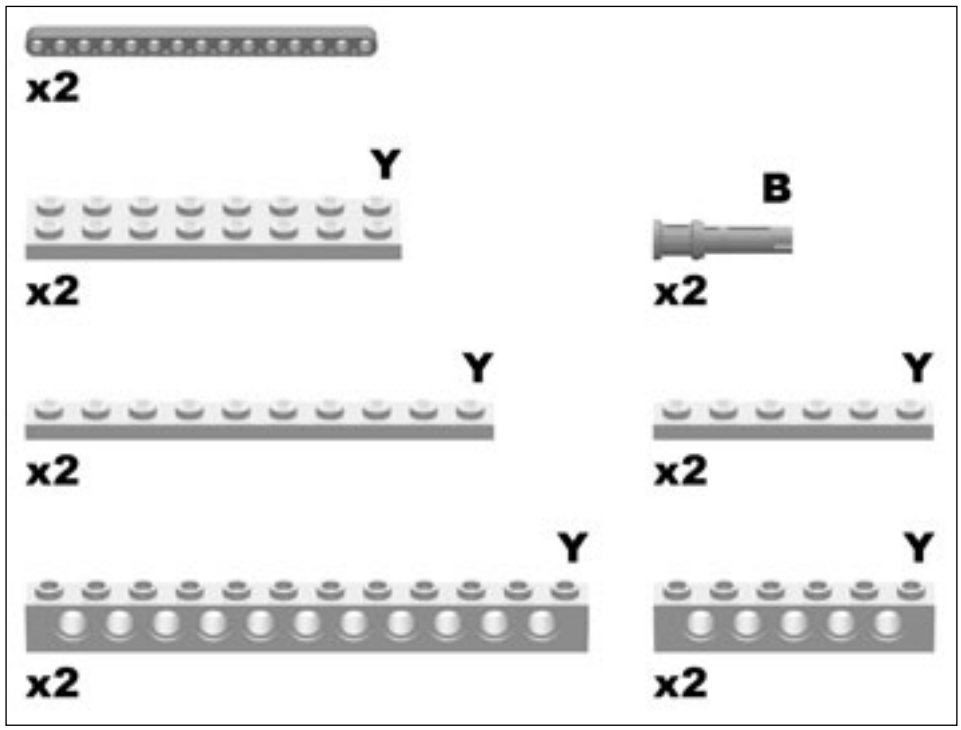


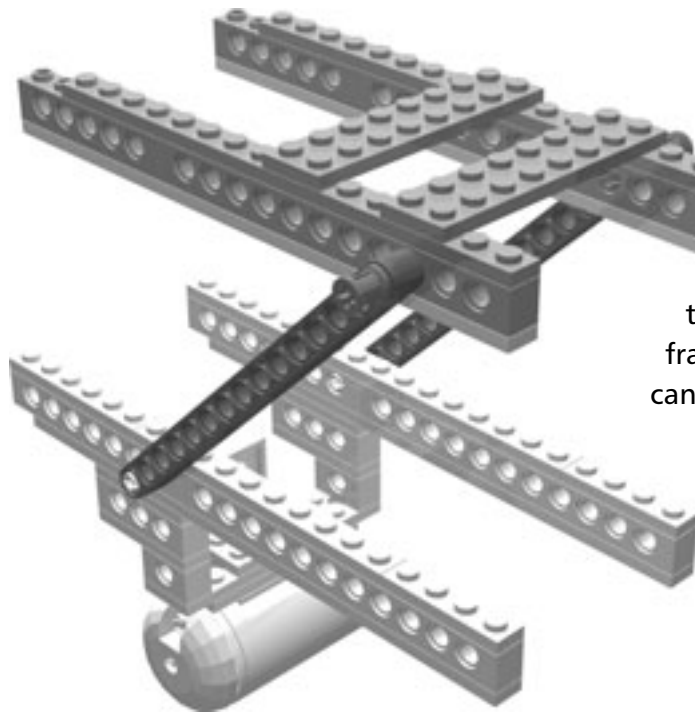
Tower Frame Step 2





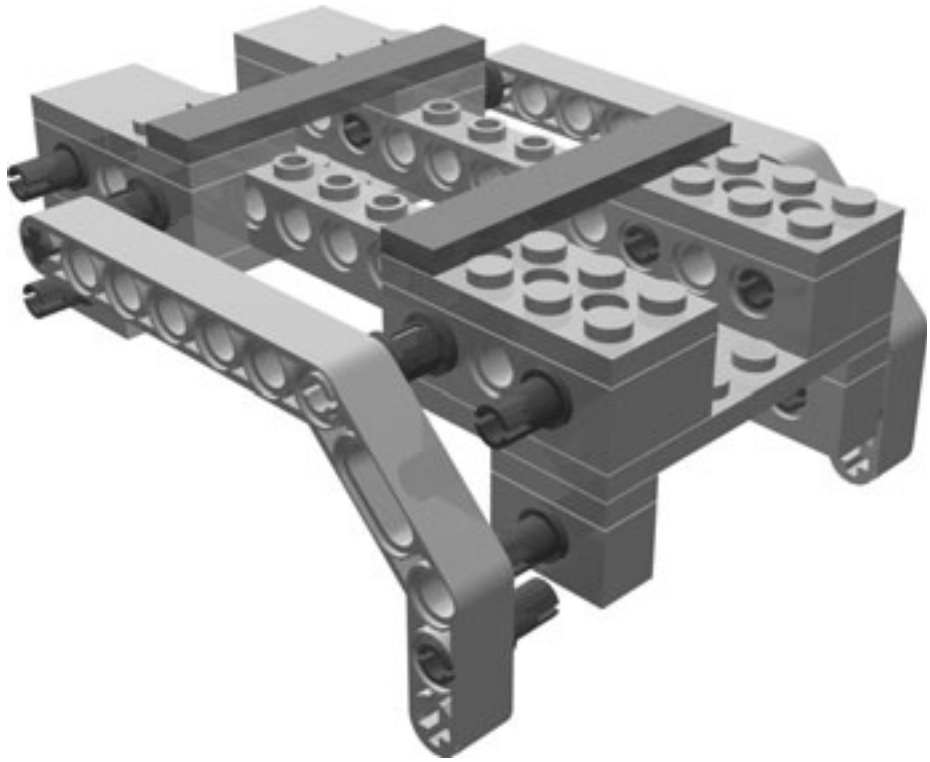
Tower Frame Step 3





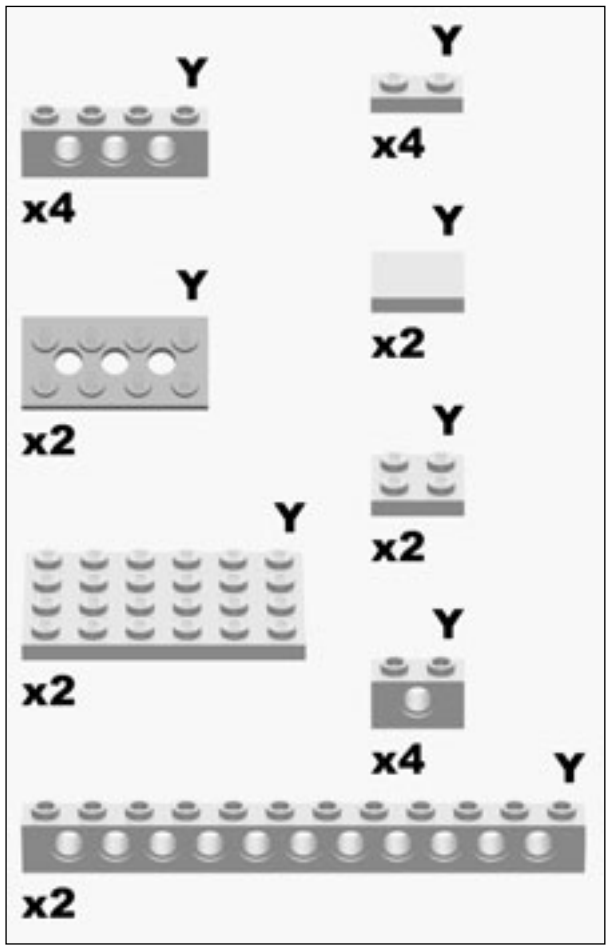
Long braces are useful to increase the robustness of the frame (triangulation). You can use a double-bent liftarm as an alternative to this design.

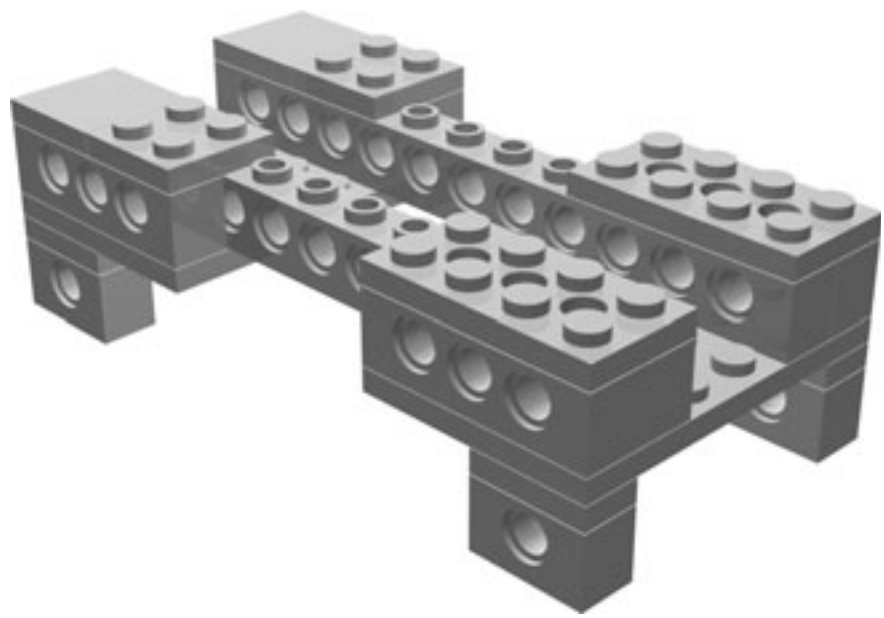
The Turntable Base



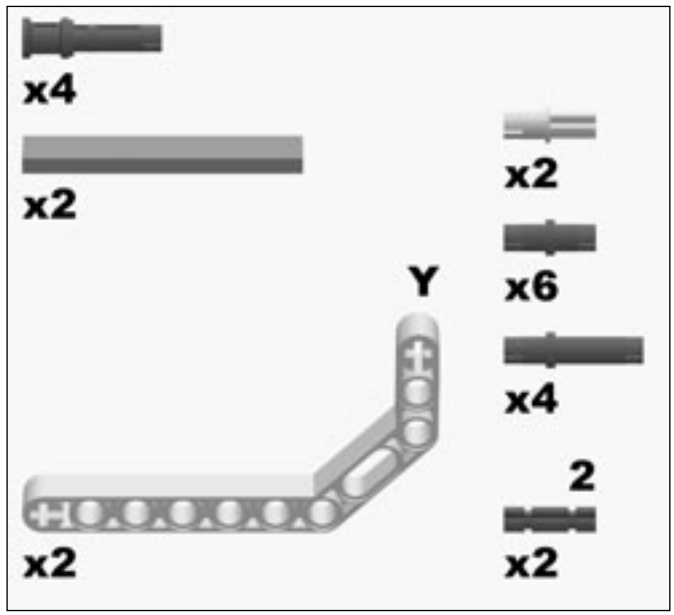
This Tower Base and the Tower Frame sub-assemblies must support the weight of the whole arm (over 3lbs).

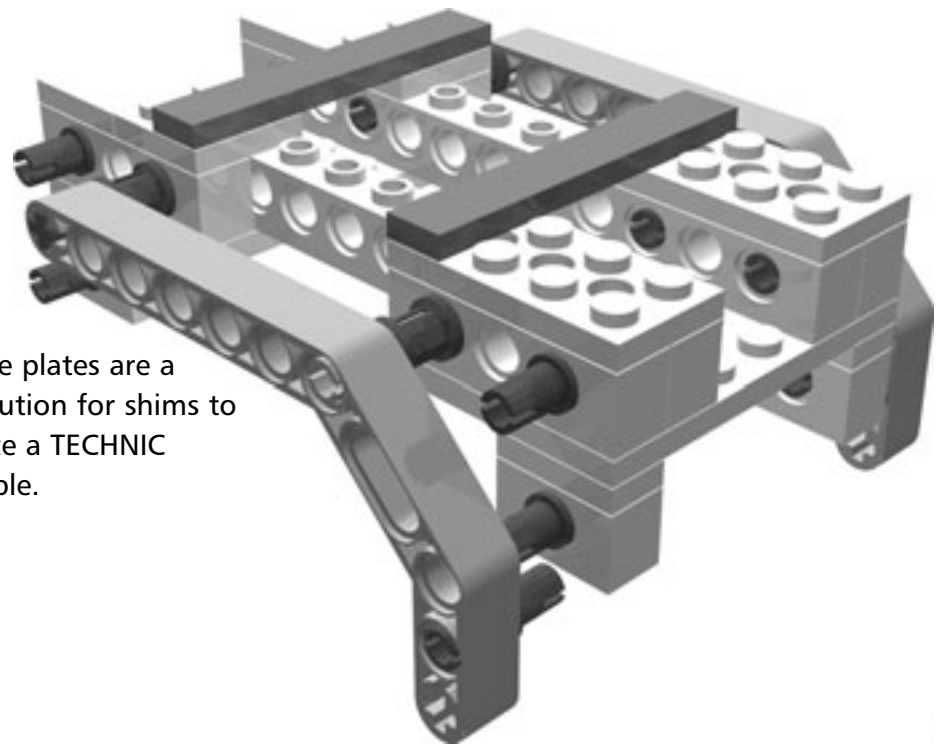
Turntable Base Step 0





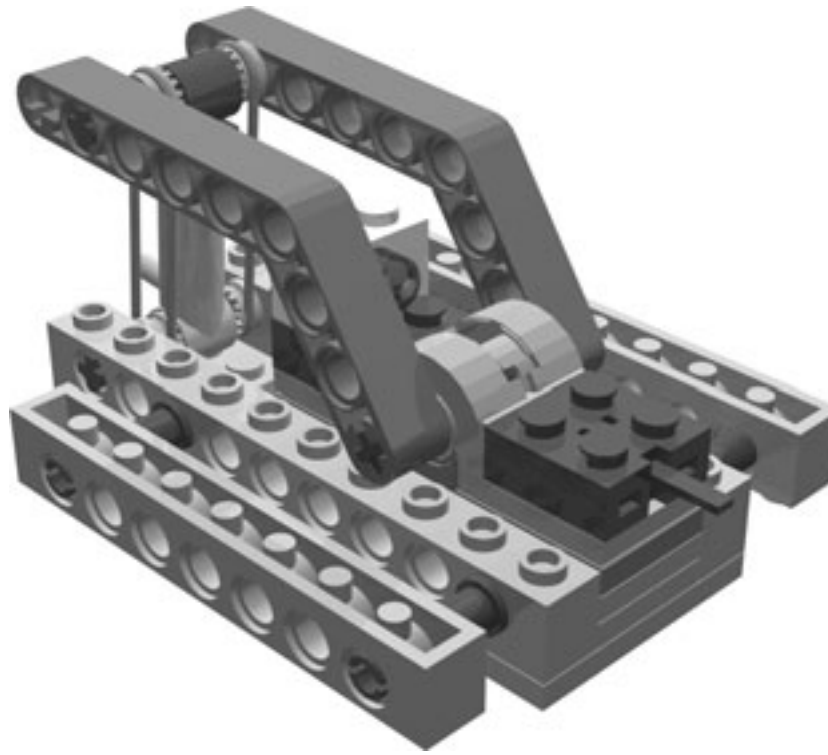
Turntable Base Step 1





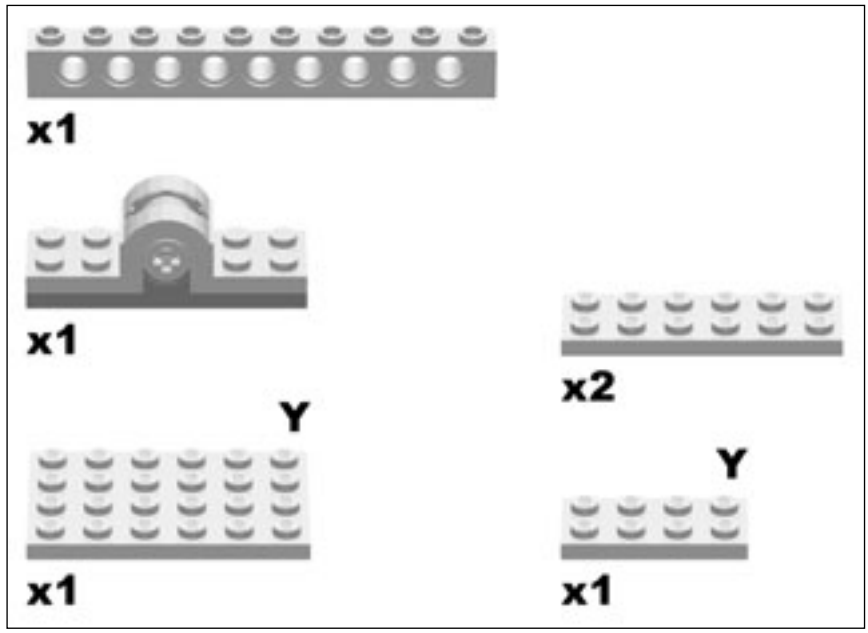
Two tile plates are a substitution for shims to stabilize a TECHNIC turntable.

The Pressure Limiter Switch



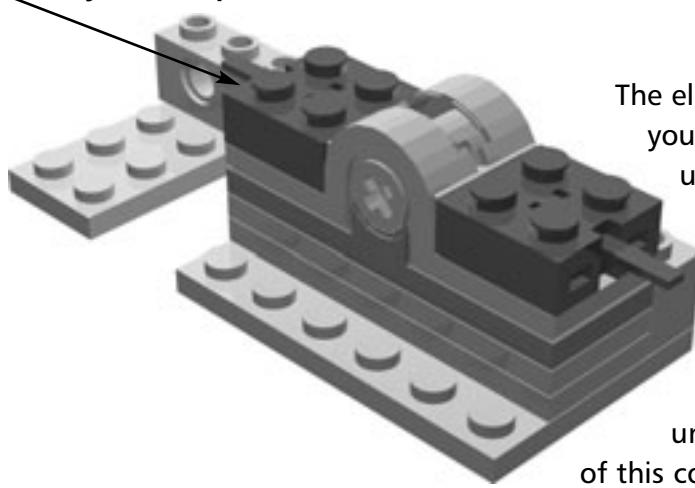
Ralph Hempel conceived the original design of the Pressure Limiter Switch subassembly. This is an ideal component for the CyberArm IV because it is a very small component. Another great feature of this component is that when the air pressure that is present in the switch is low, the power automatically remains on.

Limiter Switch Step 0



The pole reverser switch used in this step may be difficult to find in general circulation. However, I have provided you with some replacement solutions in the section, "Two Alternate Designs for the Pole Reverser Switch," later in the chapter.

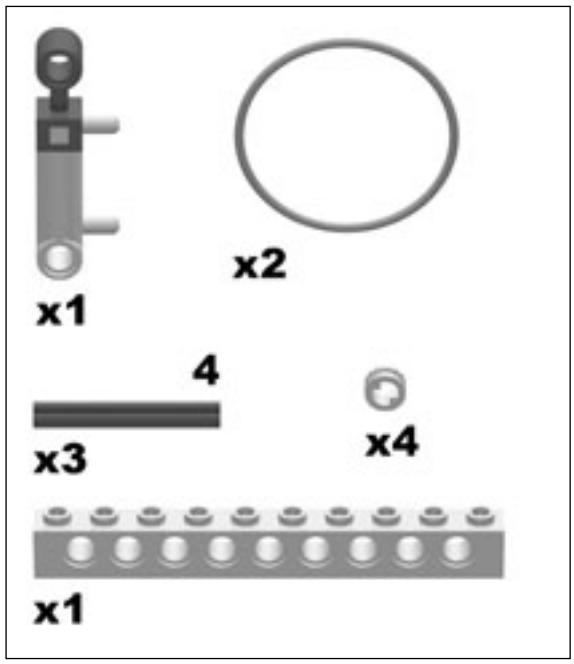
This electrical connector will be added when you add the Battery Box sub-assembly to the robot in **Battery Box Step 2**.



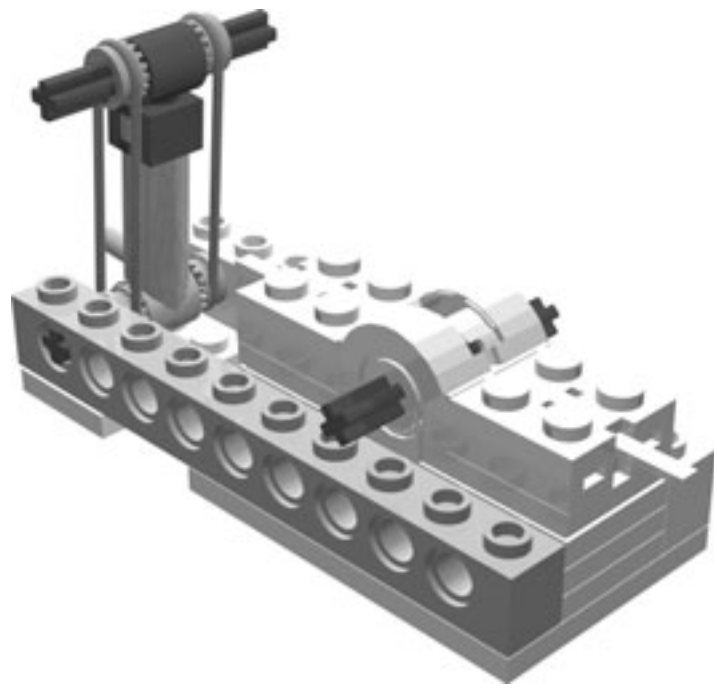
The electrical connector that you see in this step is the wire used in **Compressor Step 1**.

You might want to refer back to Figure 7.5 for guidance, and for a better understanding of what the role of this connection is in the pneumatic system for CyberArm IV.

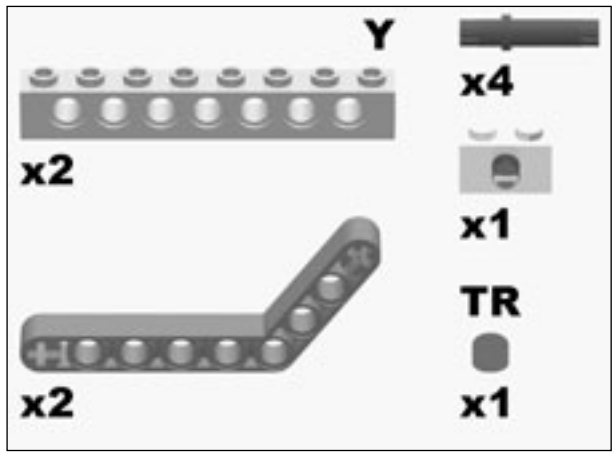
Limiter Switch Step 1

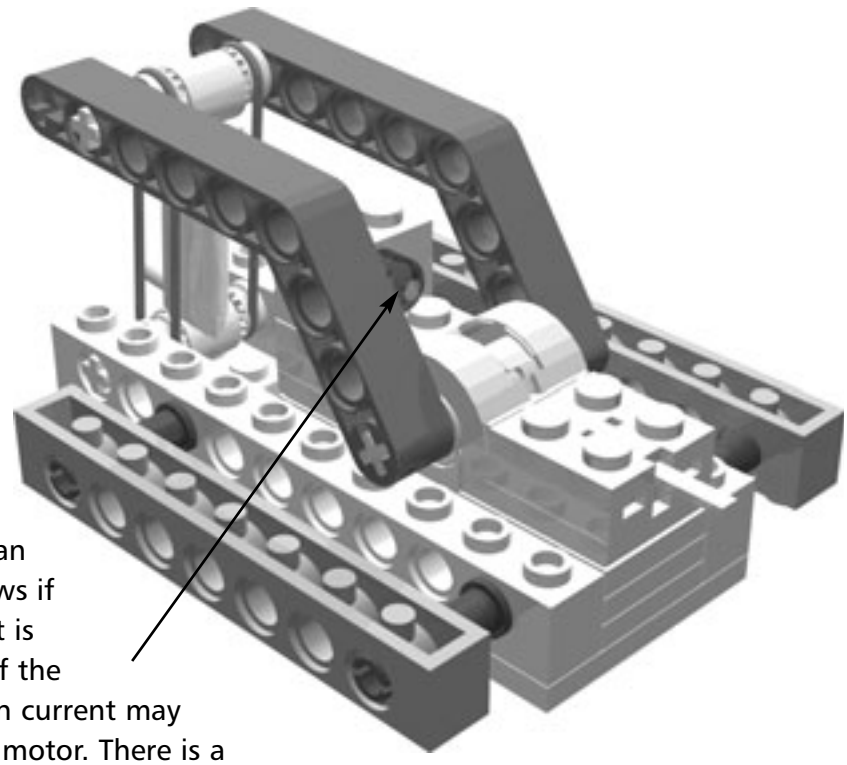


Of course, you can select different rubber bands to adjust air pressure.



Limiter Switch Step 2





This lamp brick is an indicator that shows if the electric current is flowing properly. If the battery is low, then current may flow to a stopped motor. There is a possibility that the motor will burn out. A red lamp warns of this danger.

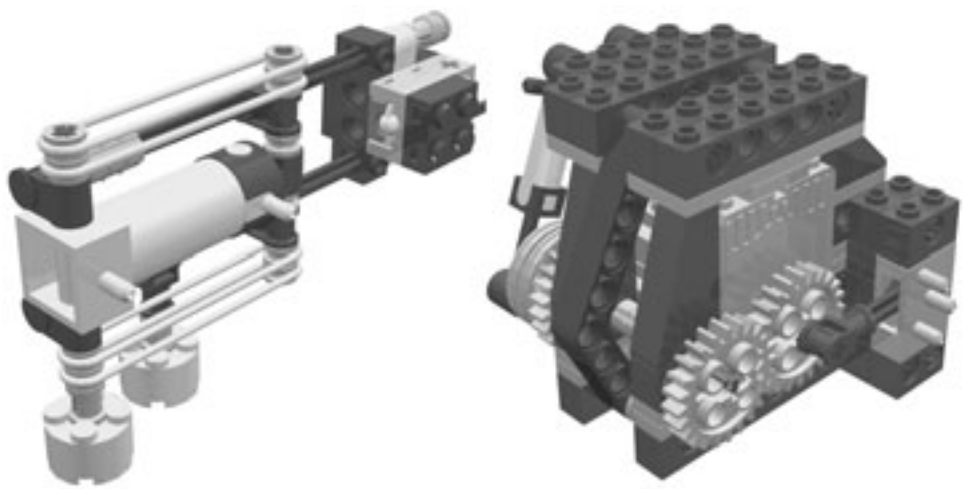
Two Alternate Designs for the Pole Reverser Switch

In doing research for this project, I determined that the current pole reverser (polarity) switch for the pressure-limiting switch is not easy to obtain for a majority of builders. With this in mind, I began to look for an alternate system for builders who are unable to obtain this part. Fortunately, Edmund Nussbaum and JP Brown were most helpful and were able to provide some creative solutions to this problem. The whole pneumatic system of CyberArm IV can be replaced by one of the two components described in the following sections. If you choose to use these alternate solutions, the two small 9V battery boxes will be unnecessary, but you will have to reposition the location of the two RCXs for balance.

Alternate Design One: The way of the HanoiSolver

JP Brown invented this pneumatic component that can be used in place of the pole reverser switch (Figure 7.9). Initially created for his HanoiSolver creation, JP built a pneumatic component with one touch sensor combined with the valve switch created by Ralph Hempel by way of a 24T gear. When I saw this, I felt as if the blindfold had fallen from my eyes. JP made me realize how inflexible I had been. Gears aren't the only parts that turn! By employing this design rather than the pole reverser switch, builders are able to conserve one motor. Since there is room for an extra sensor on RCX 2, you can use this design without any problem. The downside is that you must write a short piece of code to be able to control the pneumatic pressure. Building instructions for this pneumatic component are available on JP's homepage at: http://jpbrown.i8.com.

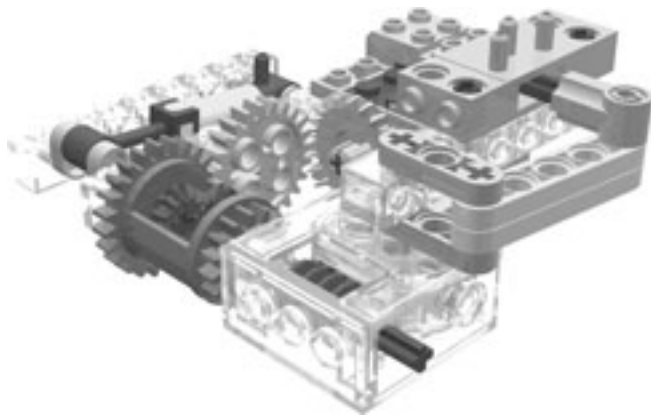
Figure 7.9 Alternate Pneumatic Component by JP Brown
(A Gear on the Valve Switch is Never Turned)



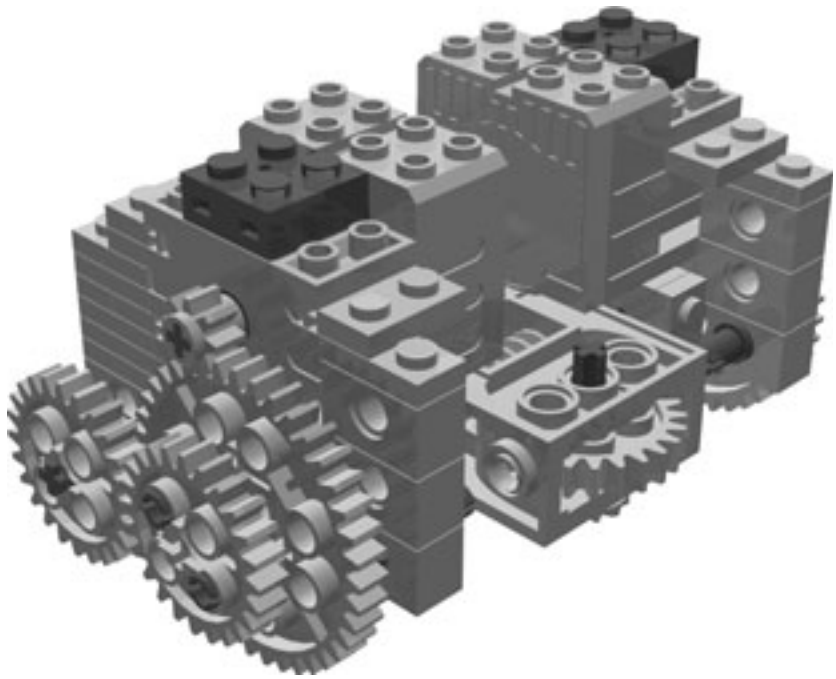
Alternate Design Two: The Way of Eddy's Component

Edmund Nussbaum has invented a clever component (Figure 7.10) that utilizes the properties of a differential gear (the half of the sum of revolutions from both side's input). When one side of the gear stopped, the other side is turned twice as fast. Unfortunately, Eddy's component does not control pneumatic pressure, though you can use it without any program code. Using a single motor, this component activates the compressor when the pressure is too low. Conversely, if pressure is too high, then the component acts as a valve switch. Though you may experience a slight bit of time delay, this mechanism is quite smooth and safe. Another appealing feature of this component is its small size, which allows it to easily fit into the tower of the CyberArm IV. Building instructions for this pneumatic component are available on Eddy's homepage at: http://home.t-online.de/home/sibylle.eddy/eddy.html.

Figure 7.10 Alternate Pneumatic Component by Edmund Nussbaum

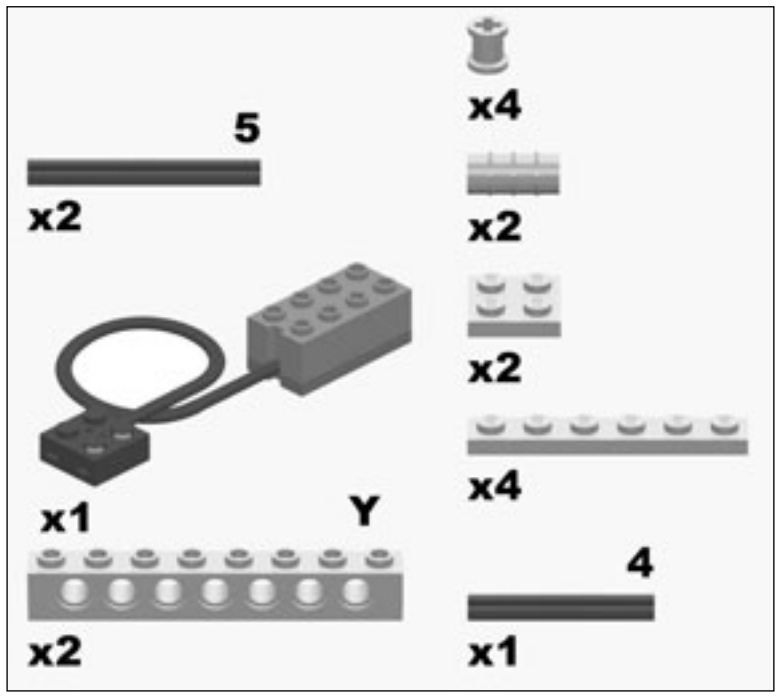


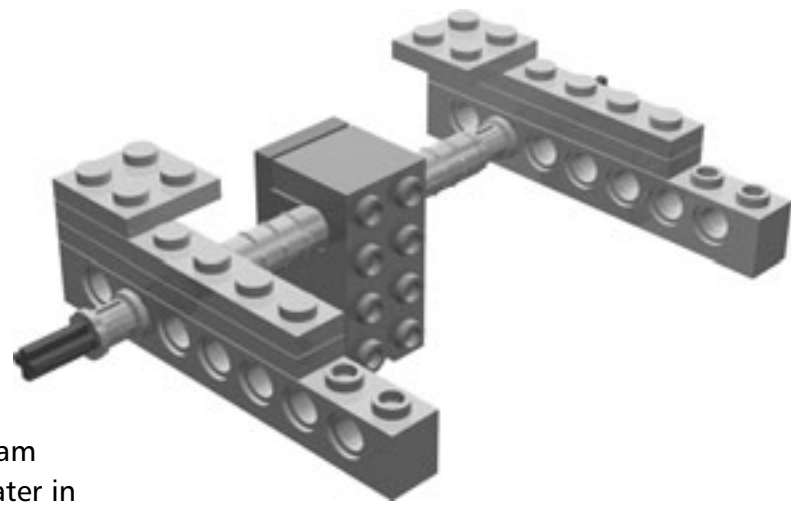
The Turntable Drive



This drive mechanism was invented as the power drive of the whole arm assembly.

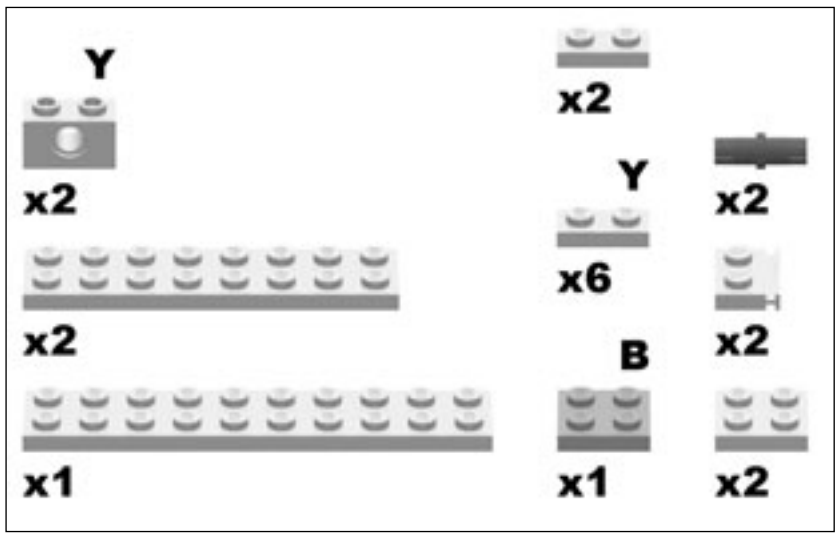
Turntable Drive Step 0

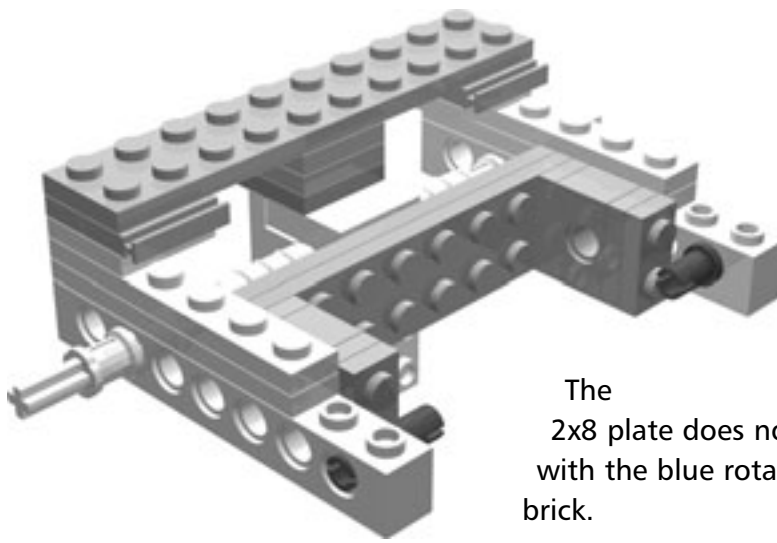




The rotation sensor in this step is attached to Input Port 3 on RCX 2 in **Final Arm Step 4**. For a visual reference, you should refer to wiring schematic diagram in Figure 7.13, found later in the chapter.

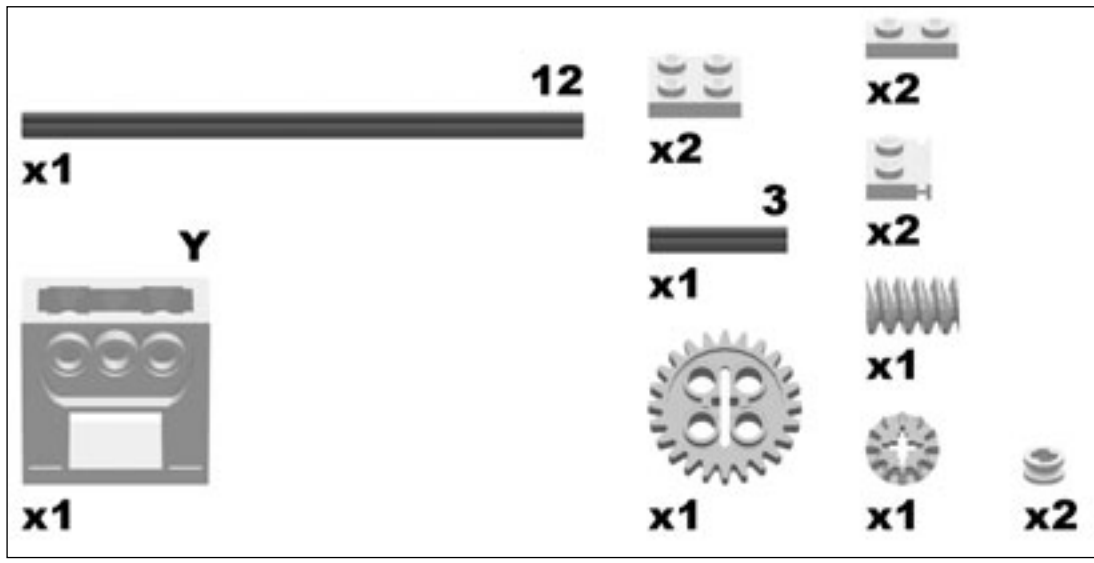
Turntable Drive Step 1

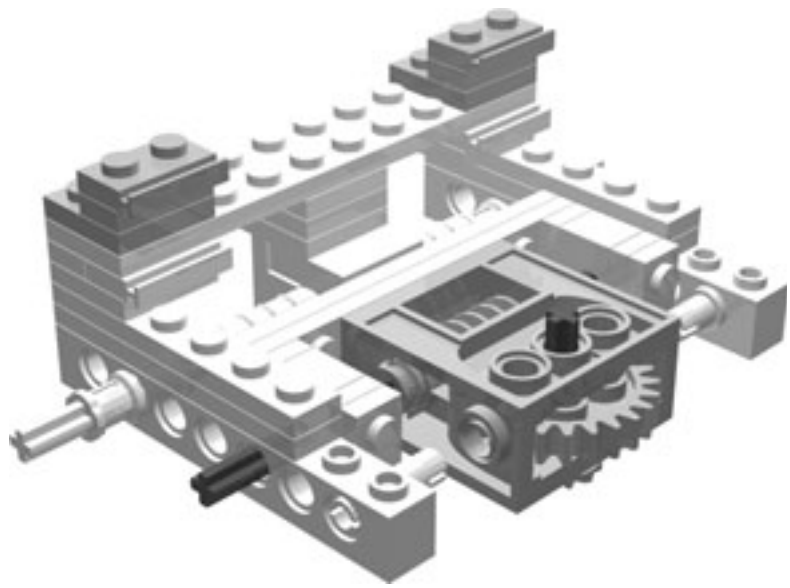




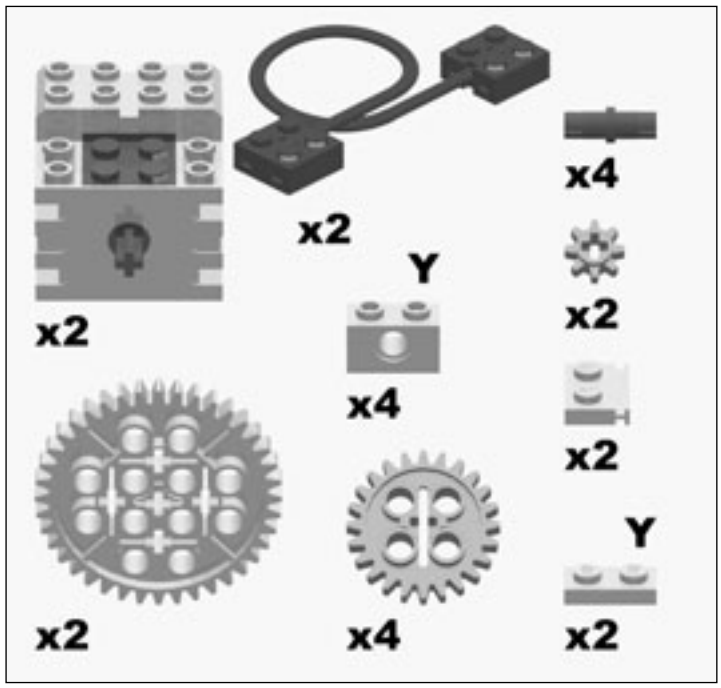
The 2x8 plate does not connect with the blue rotation sensor brick.

Turntable Drive Step 2

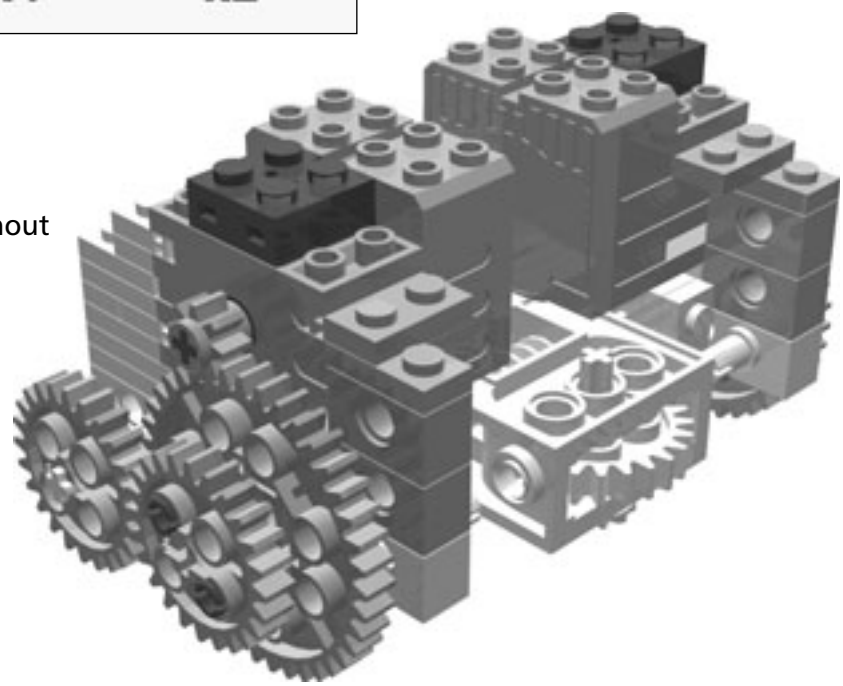




Turntable Drive Step 3



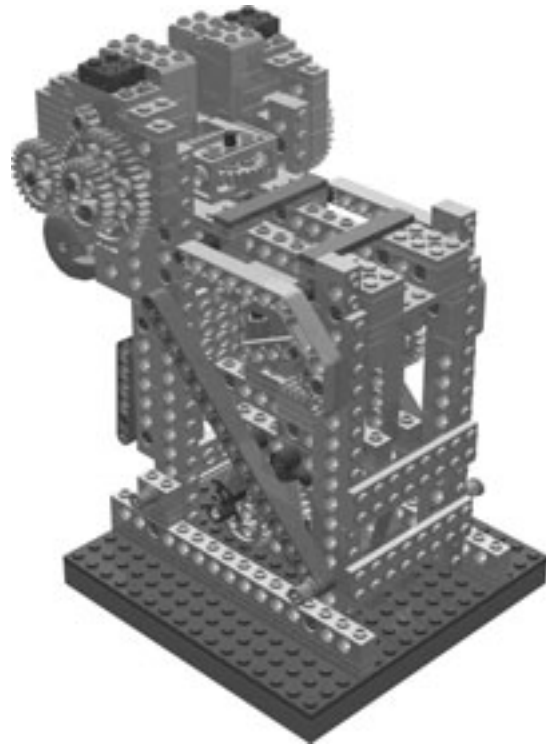
This is a way to connect gears without using an axle.



NOTE

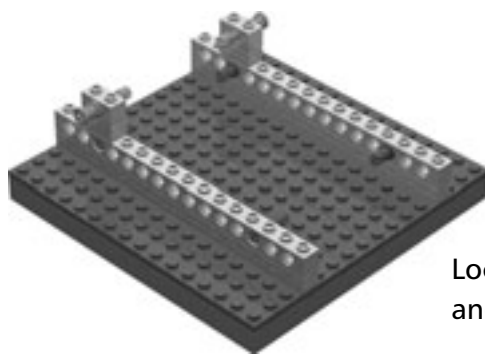
Although this parts list indicates that you will need two electrical connectors, in fact what you will need to do is to splice two electrical connectors together creating an electrical connector with three connection points. This is necessary as both motor outputs will need to be connected to Output Port C of RCX 2 in **Final Arm Step 4**. You should refer to Figure 7.13 for a visual aid for how the electrical connector should look.

Completing the Tower



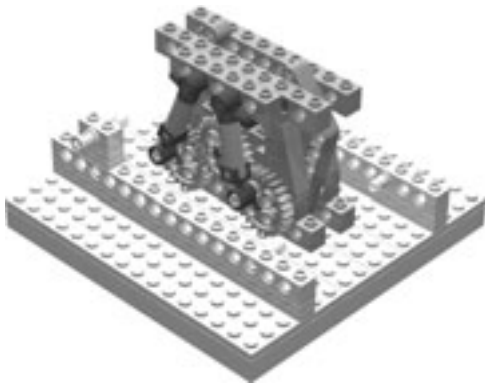
In this next series of steps, you will complete the first module of CyberArm IV: the tower. For this series of steps, you will connect the all of the sub-assemblies that you have built thus far.

Final Tower Step 0



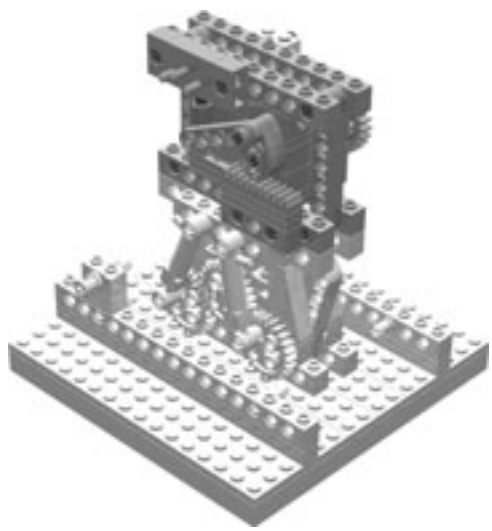
Locate the Base sub-assembly built previously, and orient it as shown.

Final Tower Step 1



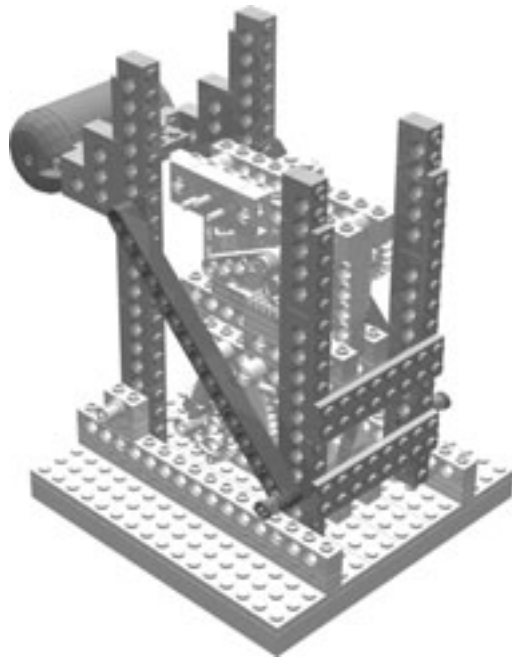
Next, attach the Pneumatic Compressor sub-assembly to the Base sub-assembly as shown.

Final Tower Step 2



In this step, the Pneumatic Valve Switch sub-assembly is added to the Pneumatic Compressor sub-assembly. Be sure to align these components as shown.

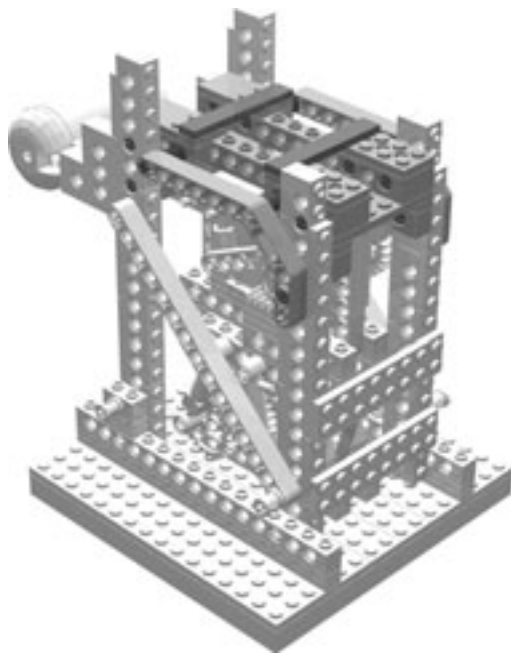
Final Tower Step 3



Locate the Tower Frame sub-assembly and mount this component onto the Base sub-assembly as shown.

Before attaching the Tower Frame sub-assembly to the Final Tower sub-assembly you should run the pneumatic tubing from the Pneumatic Compressor sub-assembly and the Pneumatic Valve Switch sub-assembly to the large pressure tank as shown.

Final Tower Step 4

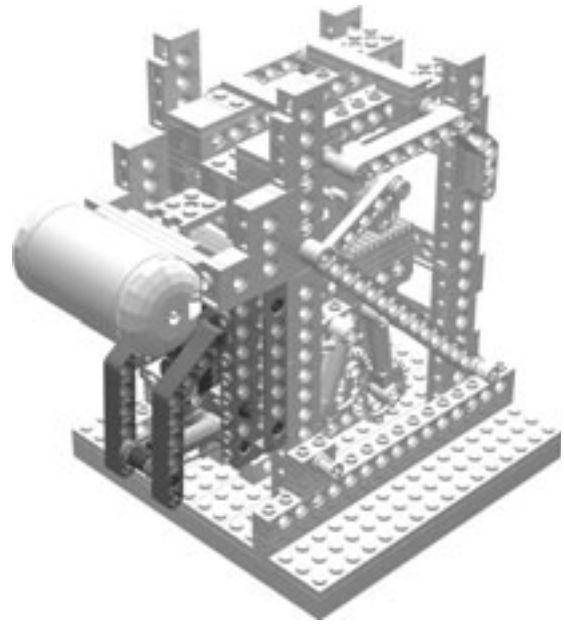


In this step, the Turntable Base sub-assembly is added.

To attach the Turntable Base sub-assembly, you should first remove the double bent liftarms and insert the component into the tower. Then, reattach the liftarms as shown to secure the sub-assembly.

Final Tower Step 5

As you can see, the pressure limiter switch is added to the rear of the Final Tower sub-assembly and is connected to the Base sub-assembly underneath the air pressure tank of the Tower Frame sub-assembly.

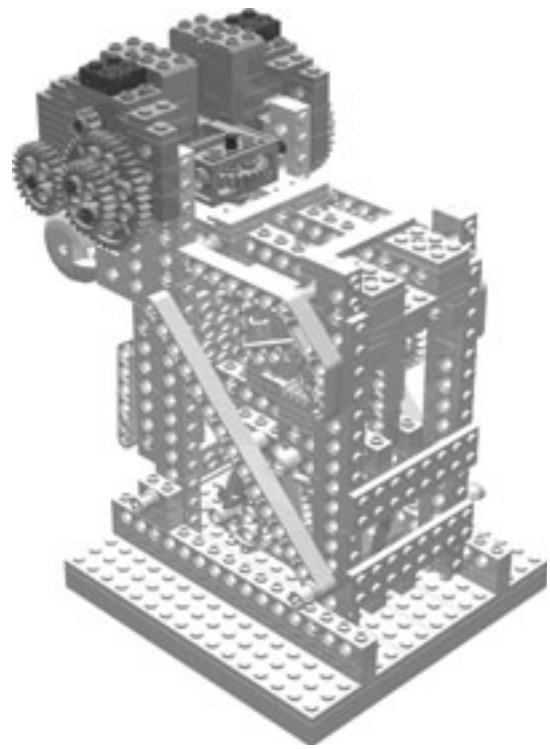


Final Tower Step 6

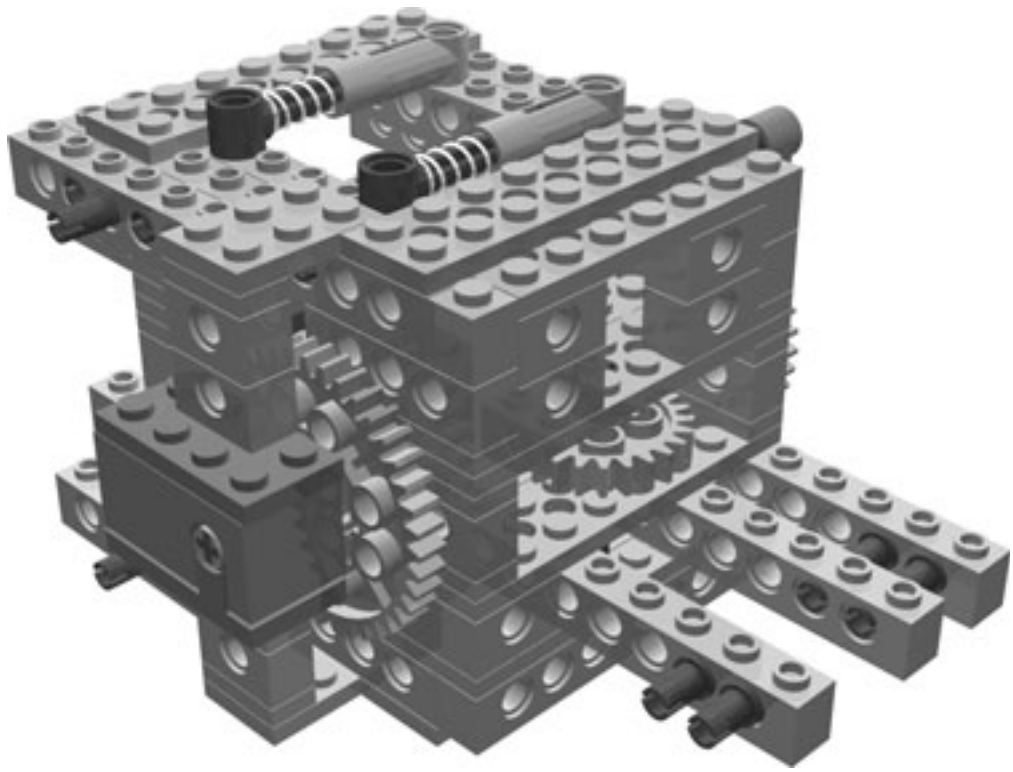
The final component of the Final Tower sub-assembly is the addition of the Turntable Drive sub-assembly.

Attach the Turntable Drive sub-assembly to the structure as shown. This component rests on the Turntable Base sub-assembly and partially hangs over the air pressure tank of the Final Tower Frame sub-assembly.

This large and complicated structure is now complete!

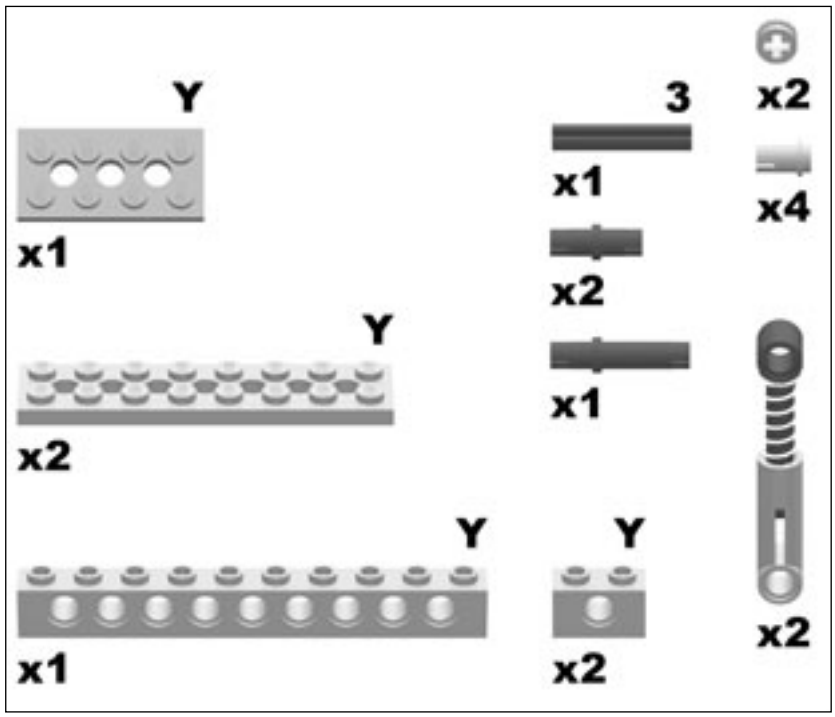


The Shoulder



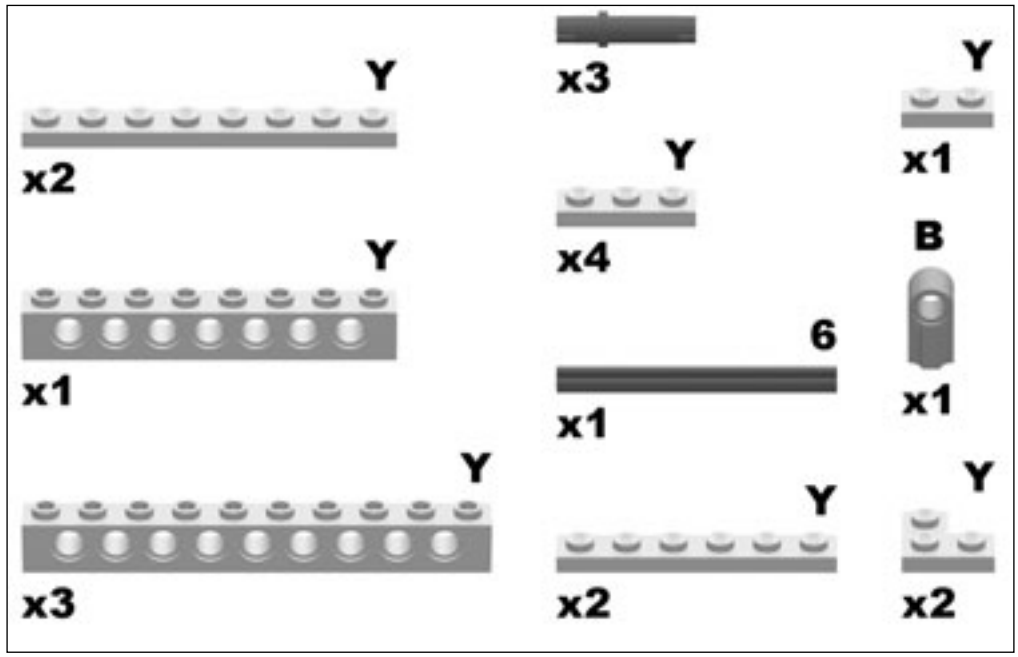
Now that we have finished with the tower, we will move on to building the components of the arm. The first sub-assembly we will build is the Shoulder sub-assembly. This component and drive mechanism was originally invented as the power drive of the heavy arm.

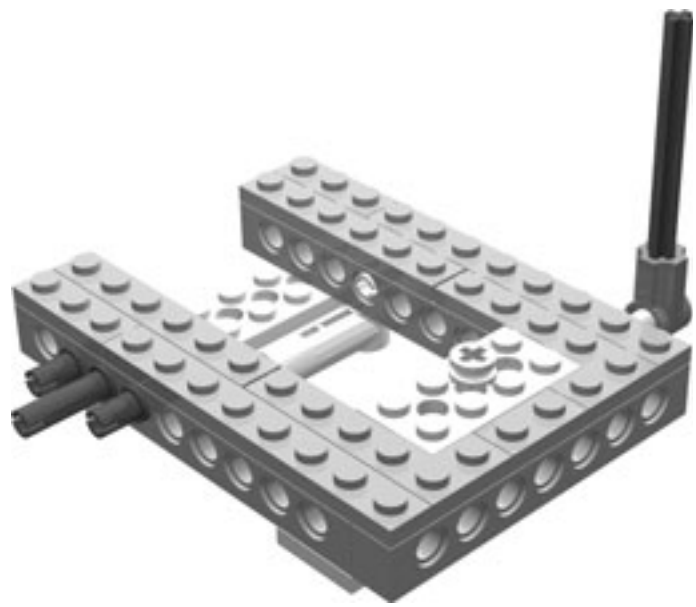
Shoulder Step 0



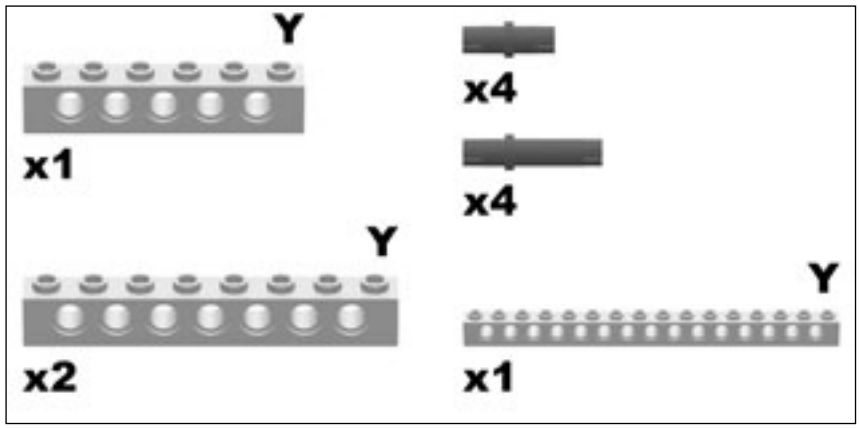
These shock absorbers are for decoration only.

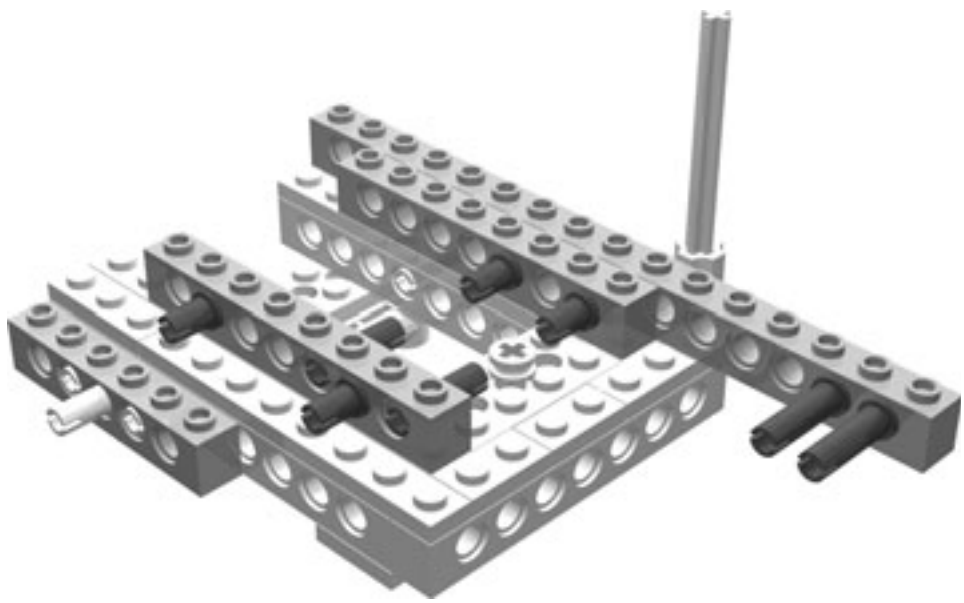
Shoulder Step 1



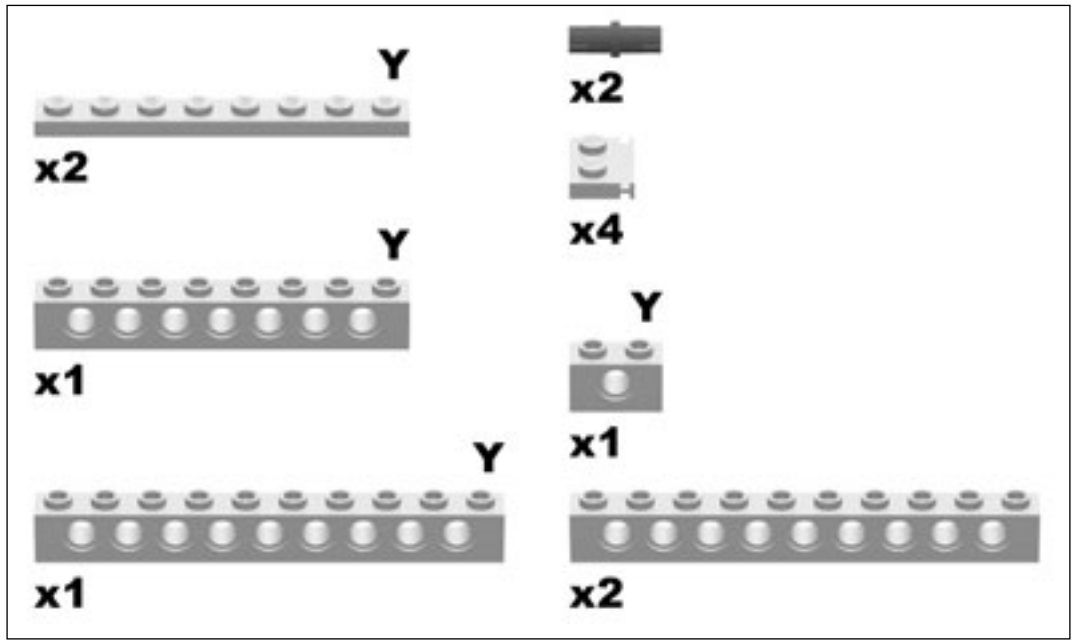


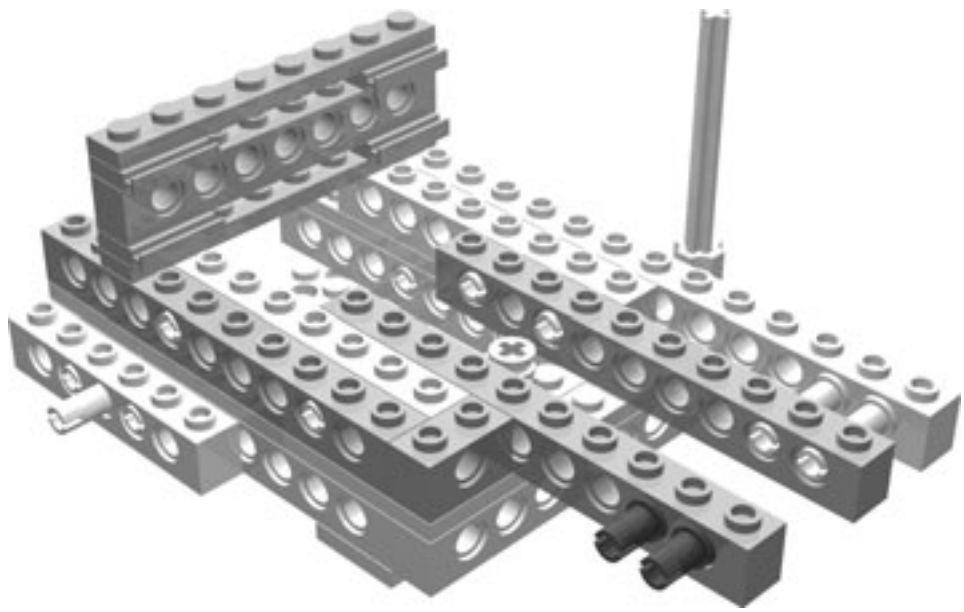
Shoulder Step 2



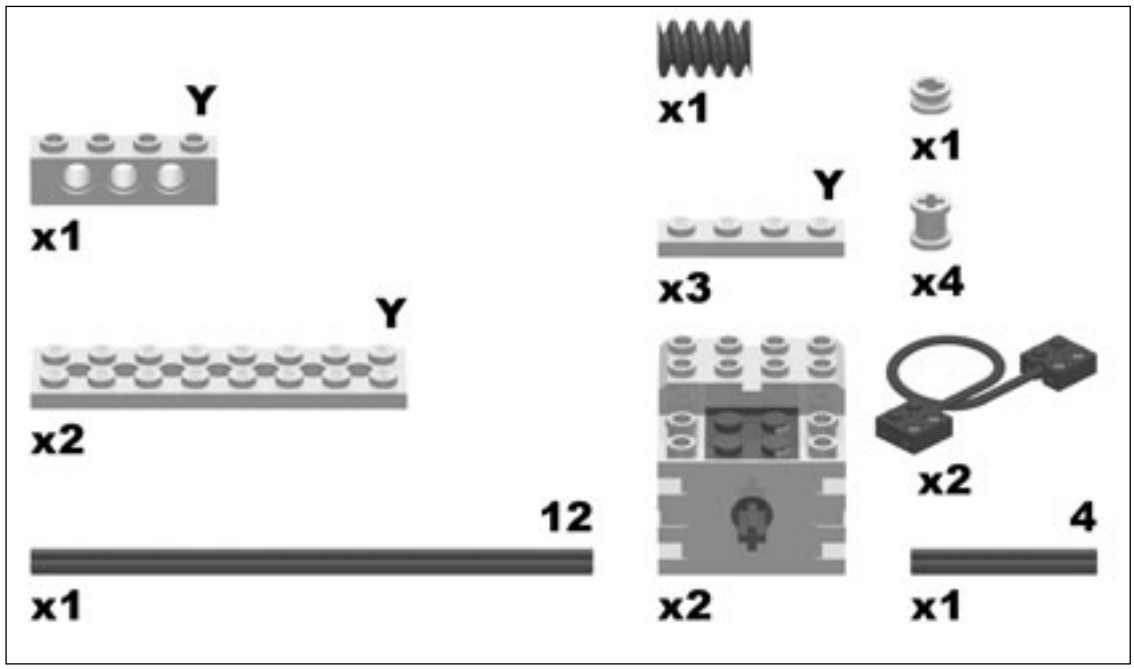


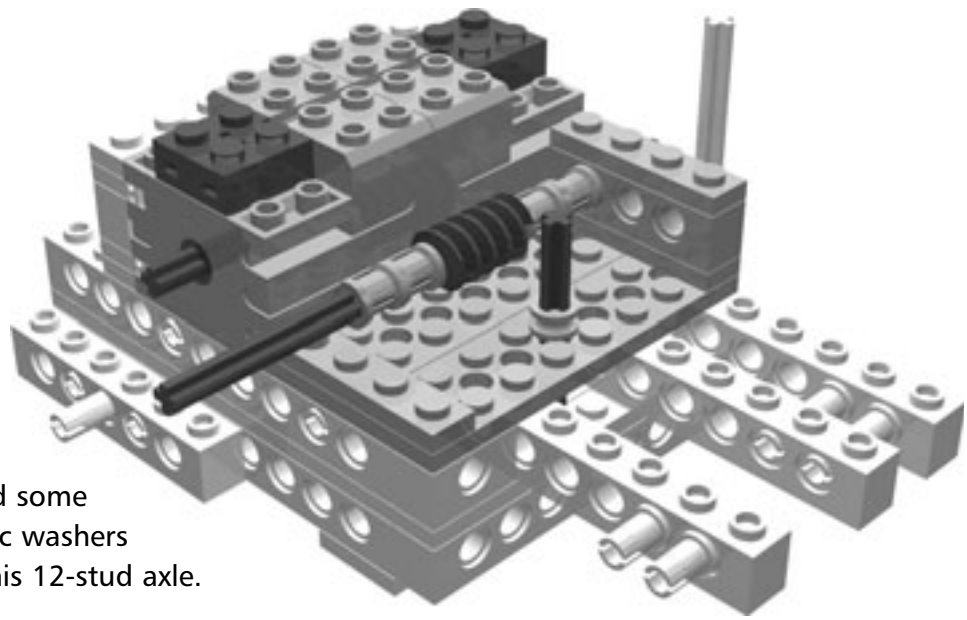
Shoulder Step 3





Shoulder Step 4

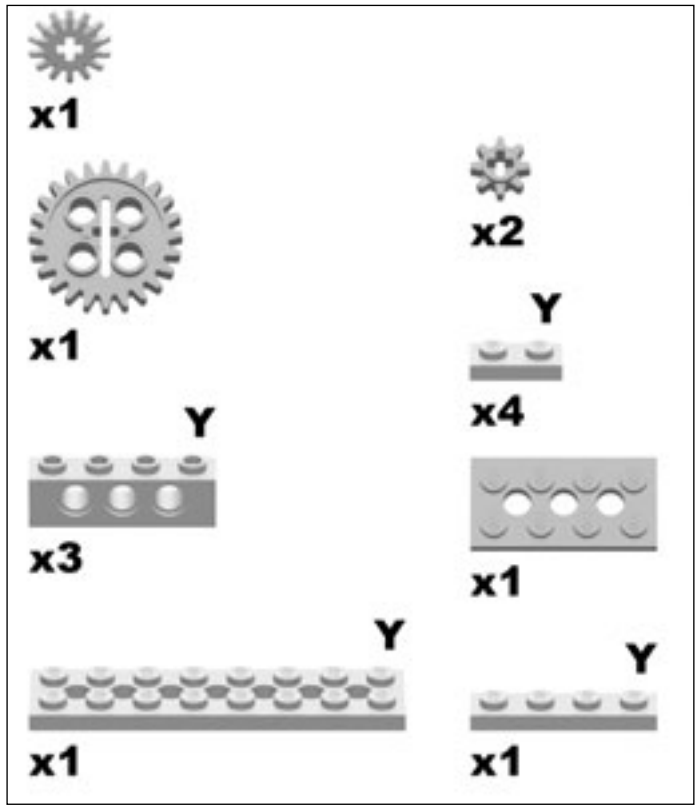




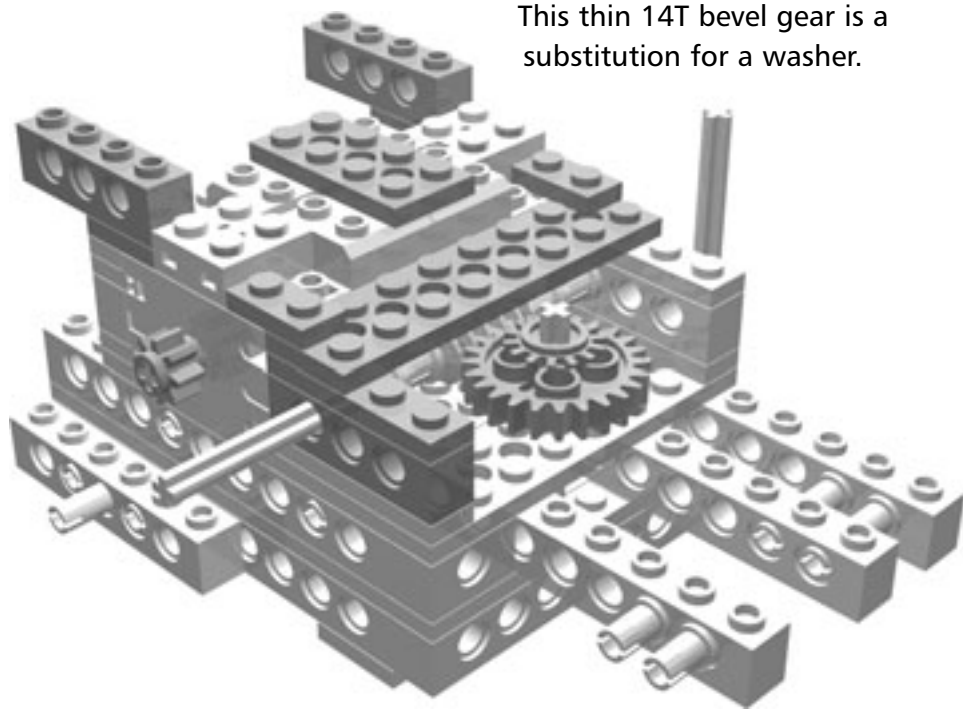
I used some plastic washers for this 12-stud axle.

You will need to splice two electrical connectors together for this step similar to the connection you required to make in **Turntable Drive Step 3**. This connector will be used to connect the Shoulder to Output Port C of RCX 1 in **Final Arm Step** 4, for additional assistance refer to Figure 7.12.

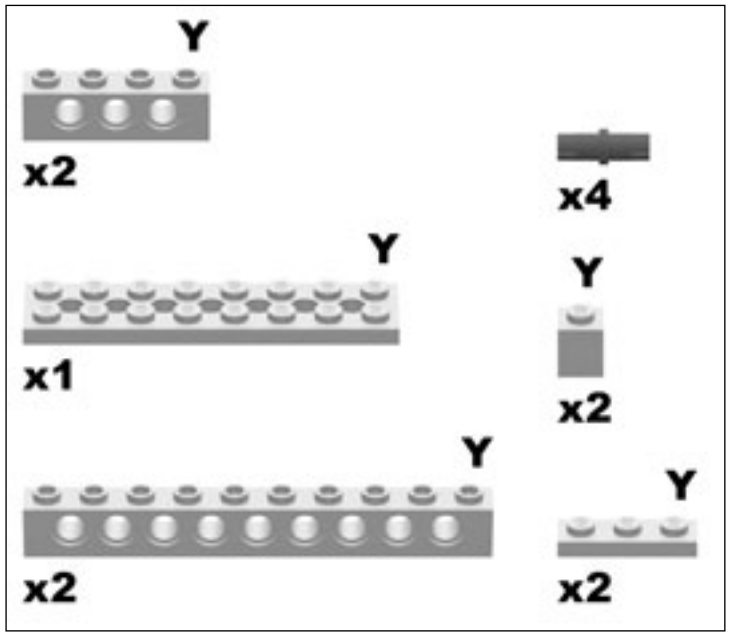
Shoulder Step 5

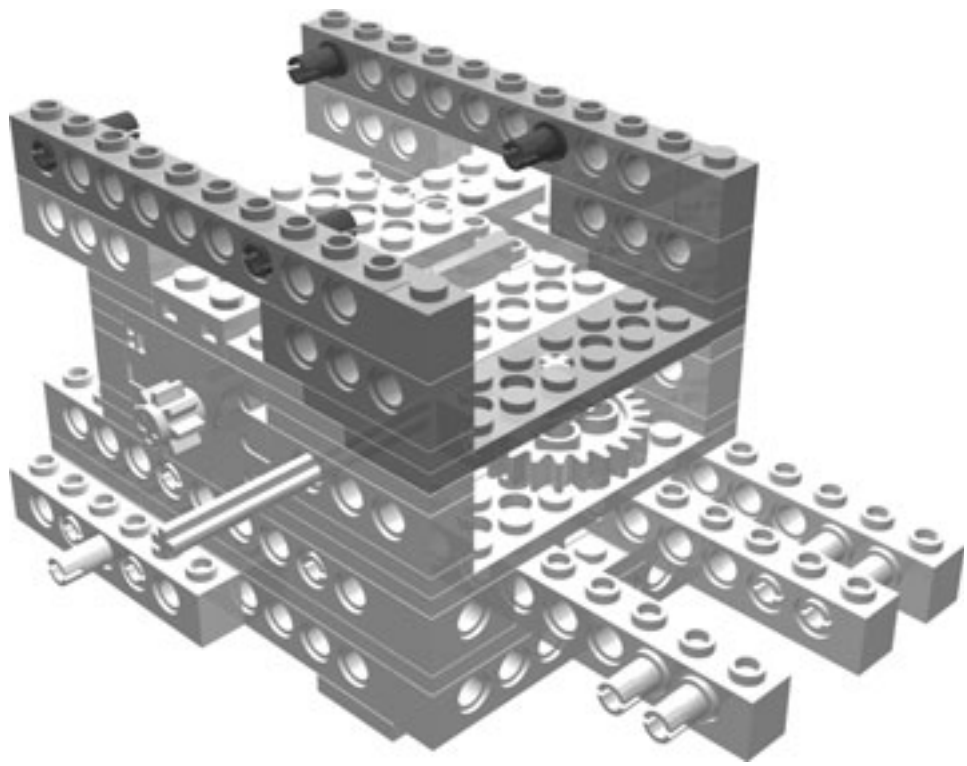


This thin 14T bevel gear is a substitution for a washer.

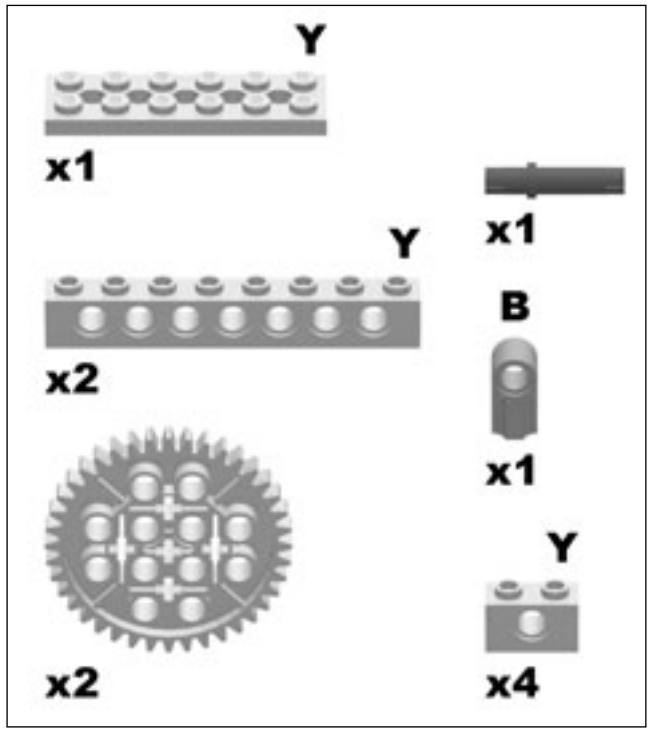


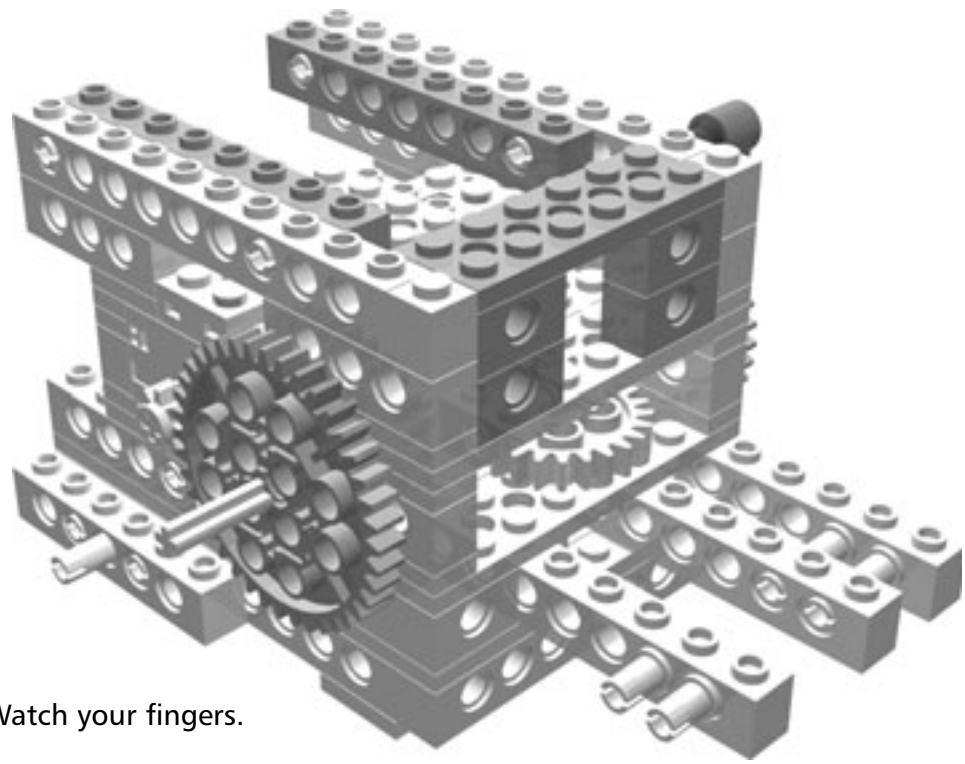
Shoulder Step 6





Shoulder Step 7



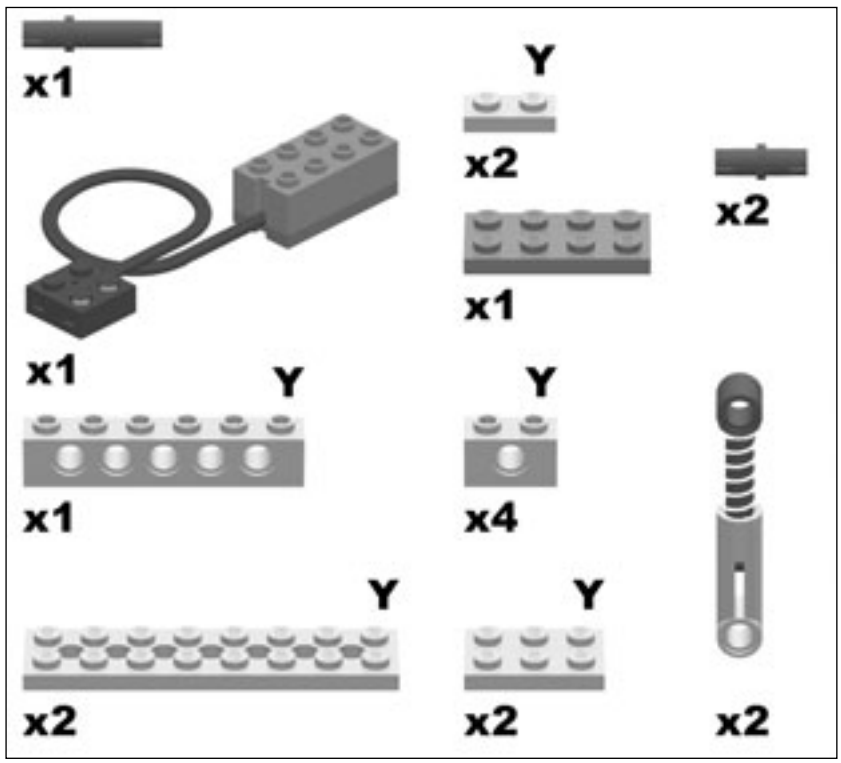


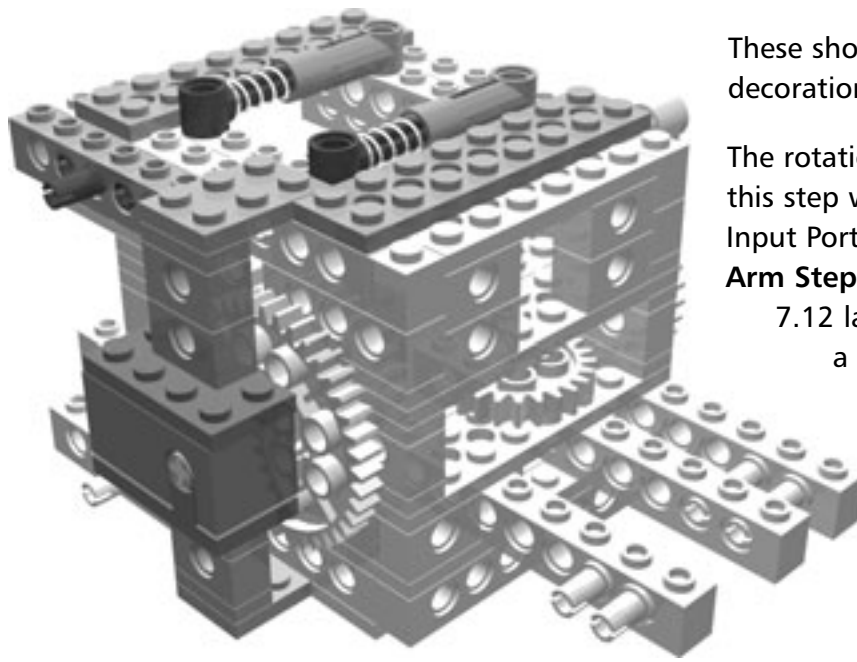
Caution! Watch your fingers.

NOTE

This power drive mechanism portion of the shoulder is capable of tremendous power. Be careful not to catch your finger in the gears, or you could hurt yourself. As well, during operation, keep an eye on the operating gears. If you drive the arm out of the operation range, parts may break.

Shoulder Step 8

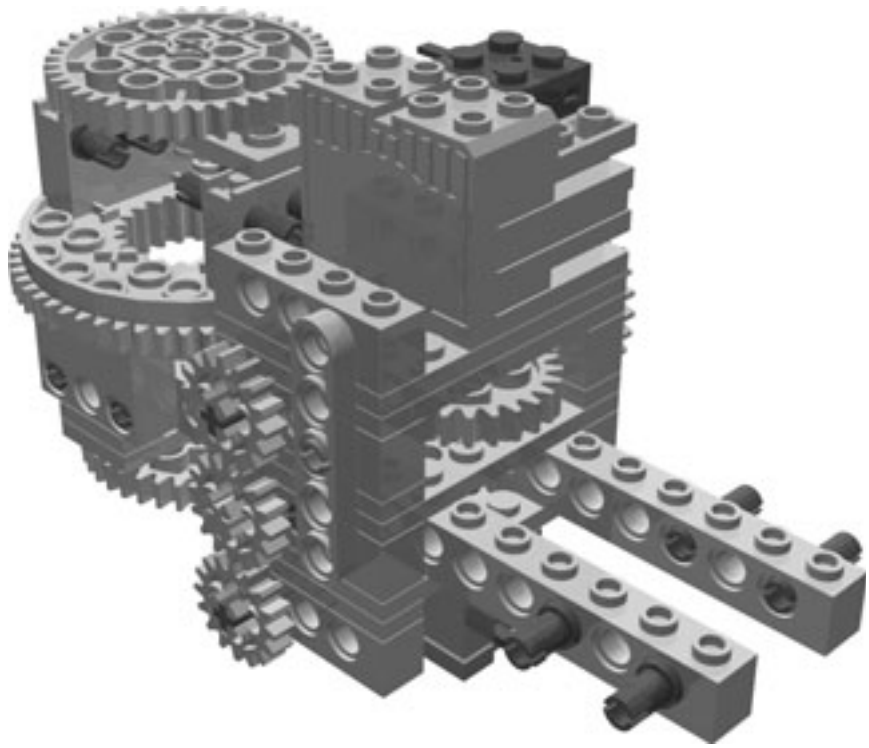




These shock absorbers are for decoration only.

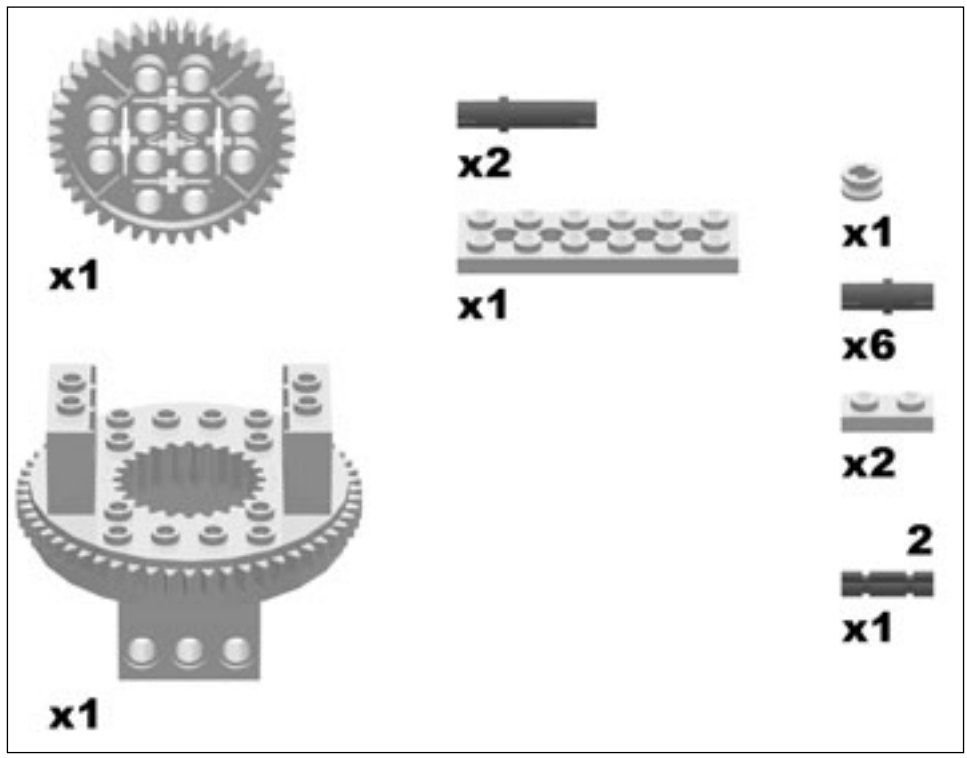
The rotation sensor added in this step will be attached to Input Port 3 on RCX 1 in **Final Arm Step 4**. Refer to Figure 7.12 later in the chapter for a detailed wiring schematic for the sensor.

The Upper Arm



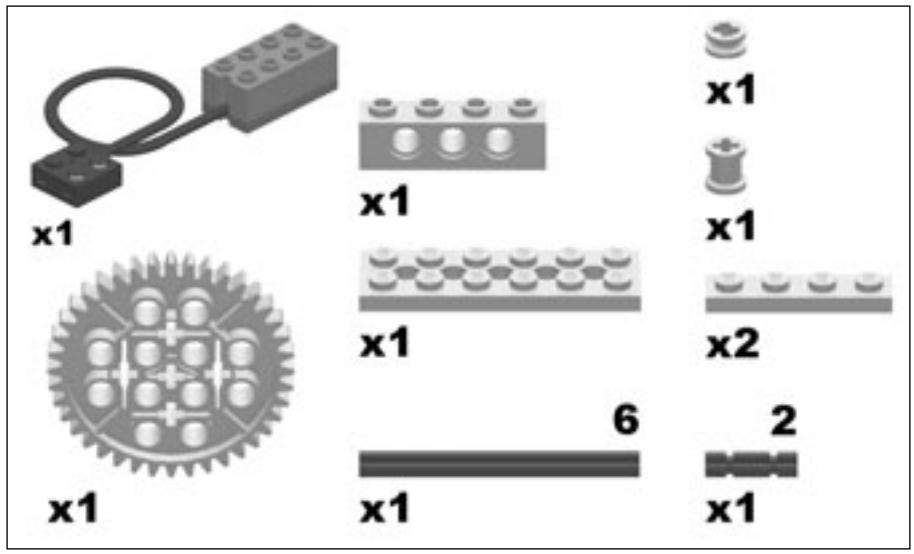
You'll notice that the turntable mechanism used here in the Upper Arm sub-assembly originated from the power drive mechanism and is used multiple times throughout CyberArm IV.

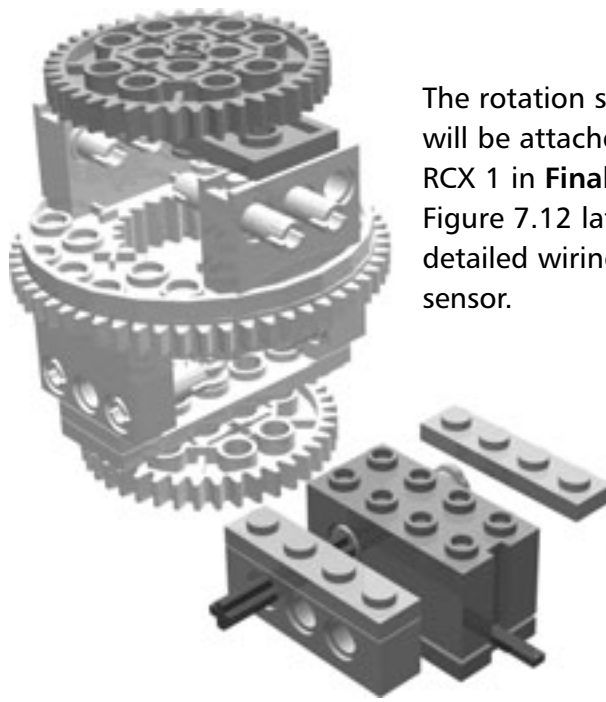
Upper Arm Step 0



This turntable is the joint of the shoulder. The 40T gear is for decoration.

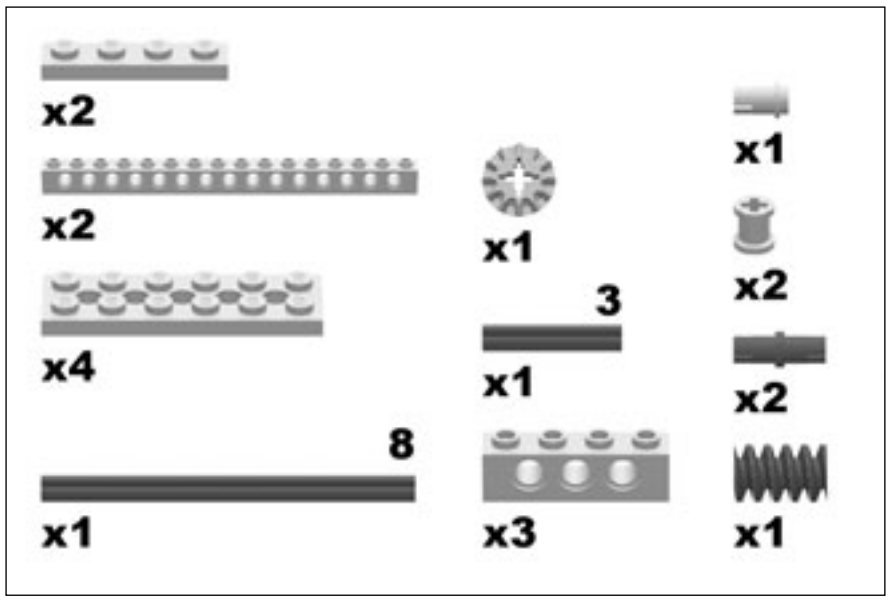
Upper Arm Step 1

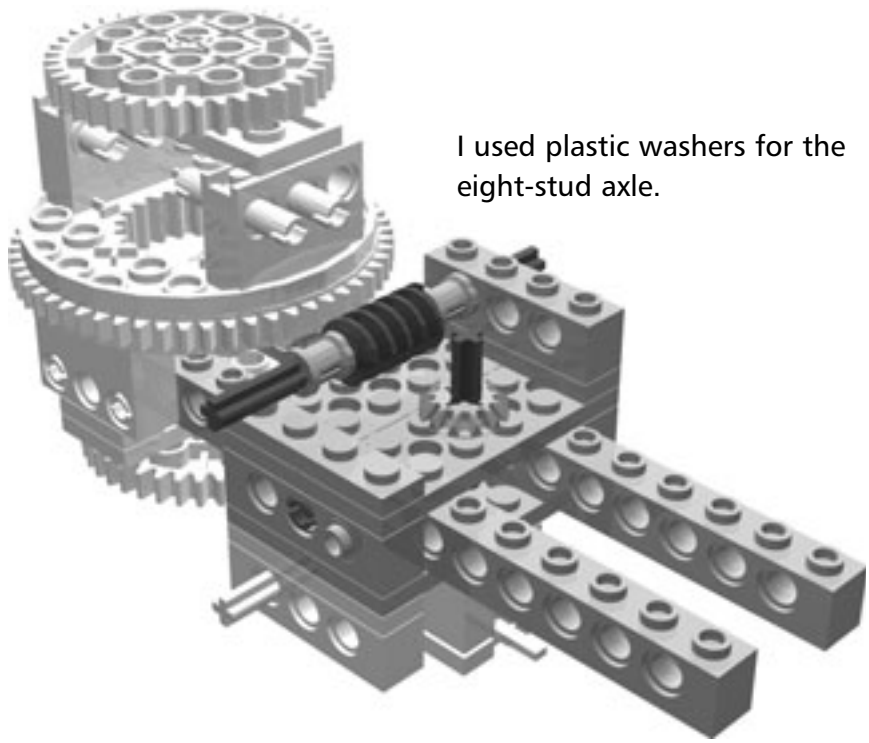




The rotation sensor added in this step will be attached to Input Port 2 of RCX 1 in **Final Arm Step 4**. Refer to Figure 7.12 later in the chapter for a detailed wiring schematic for the sensor.

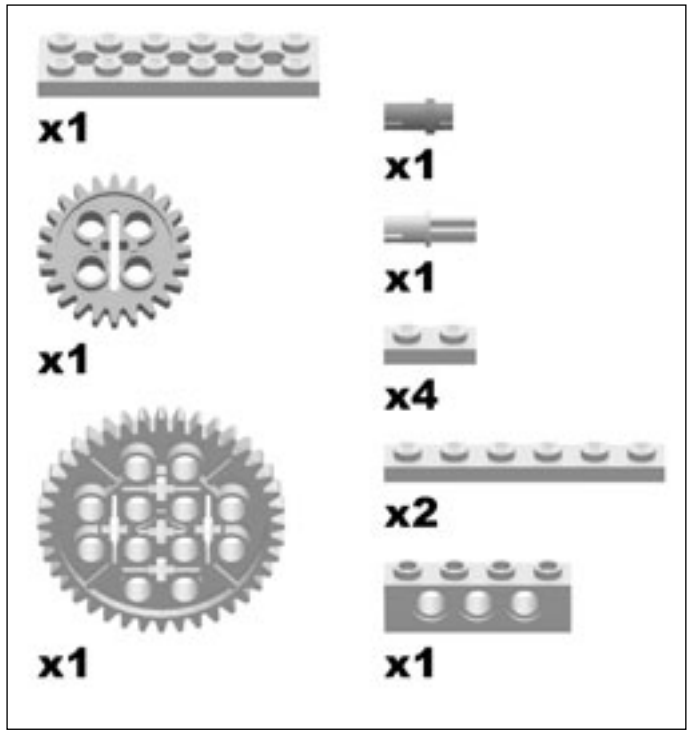
Upper Arm Step 2

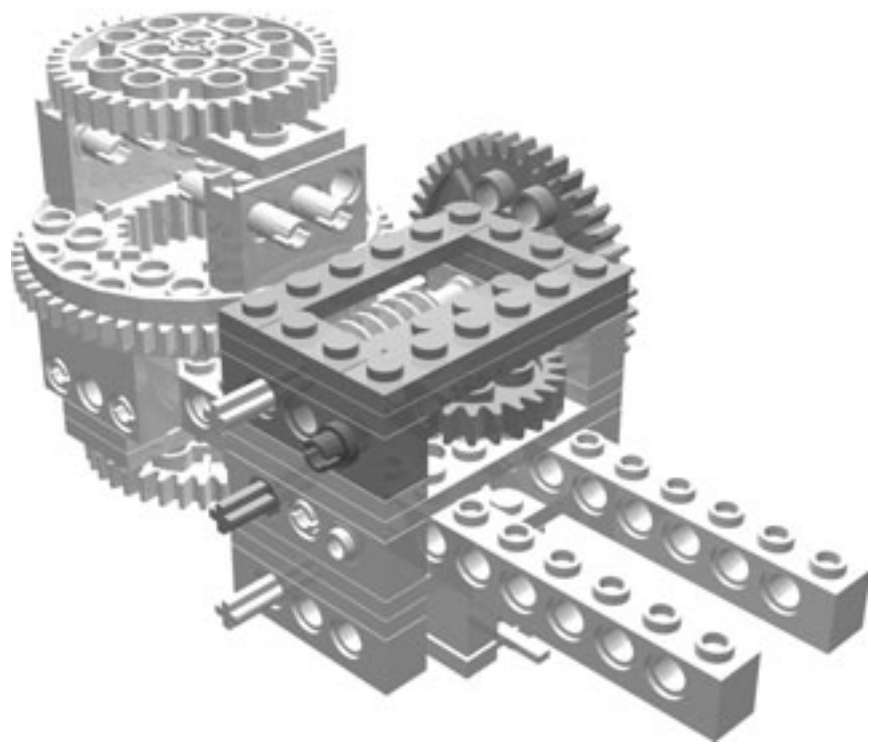




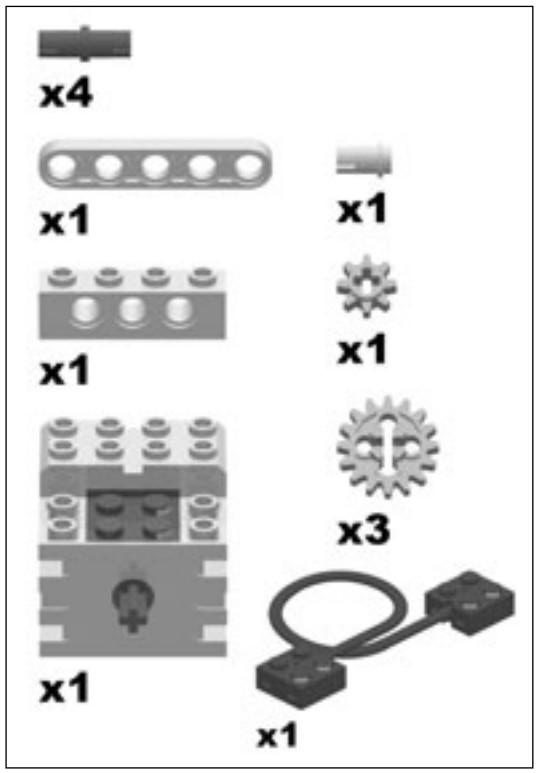
I used plastic washers for the eight-stud axle.

Upper Arm Step 3



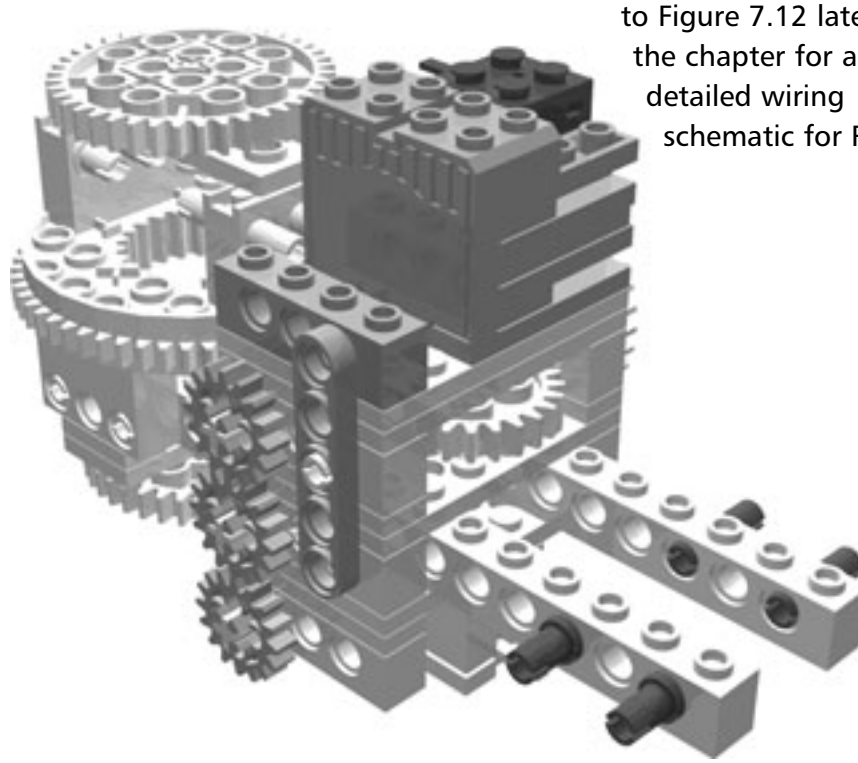


Upper Arm Step 4

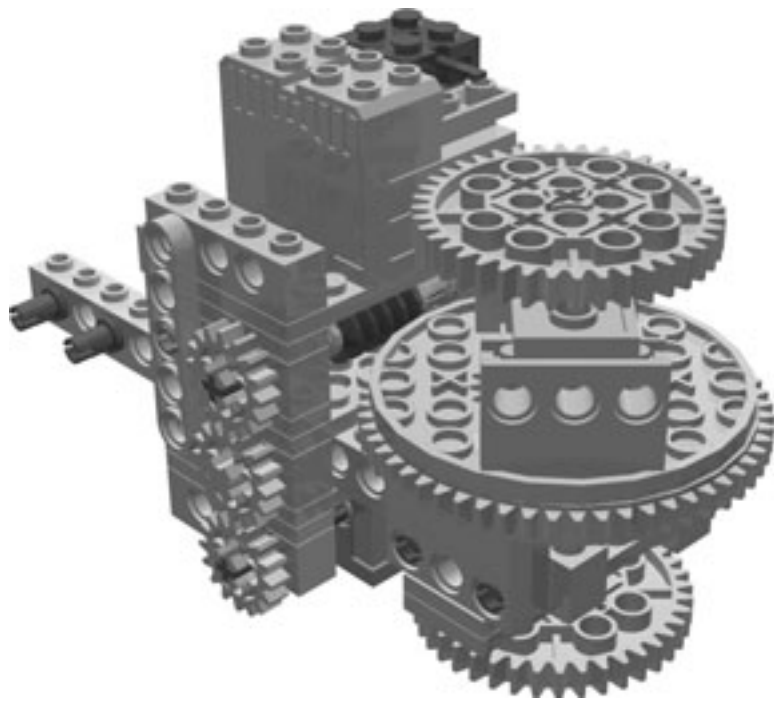


For more power, you may need to reinforce the mount of this motor.

The electric connector is attached to Output Port B of RCX 1 in **Final Arm Step 4**. Refer to Figure 7.12 later in the chapter for a detailed wiring schematic for RCX 1.

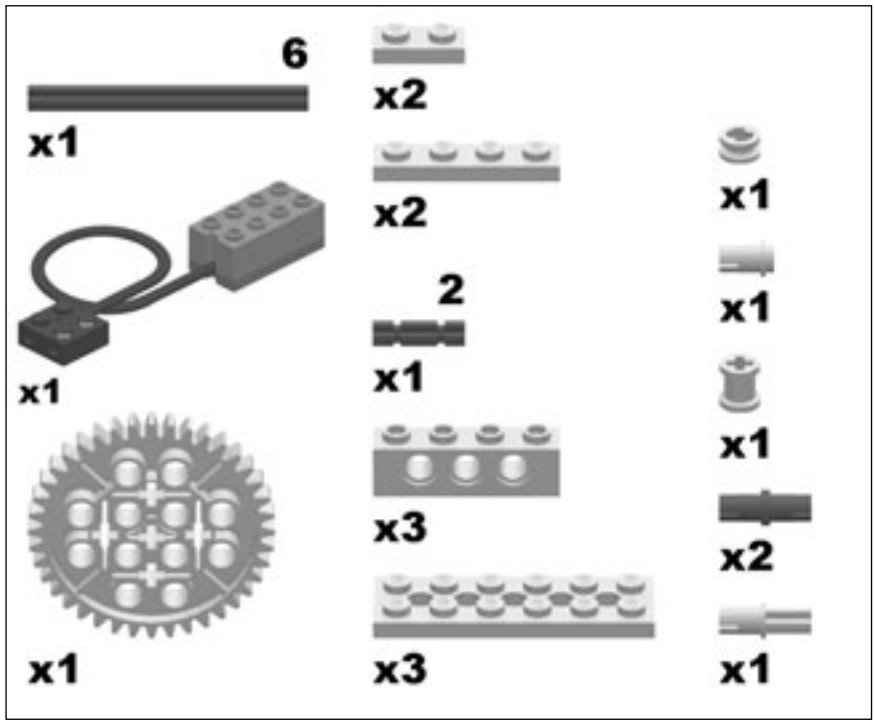


The Forearm

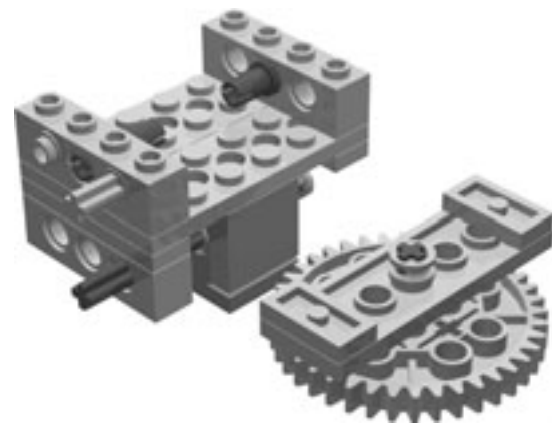


Now, we will build the power drive mechanism for the Forearm sub-assembly. While building this sub-assembly, you should take note of the similarities between this construction and the Upper Arm sub-assembly.

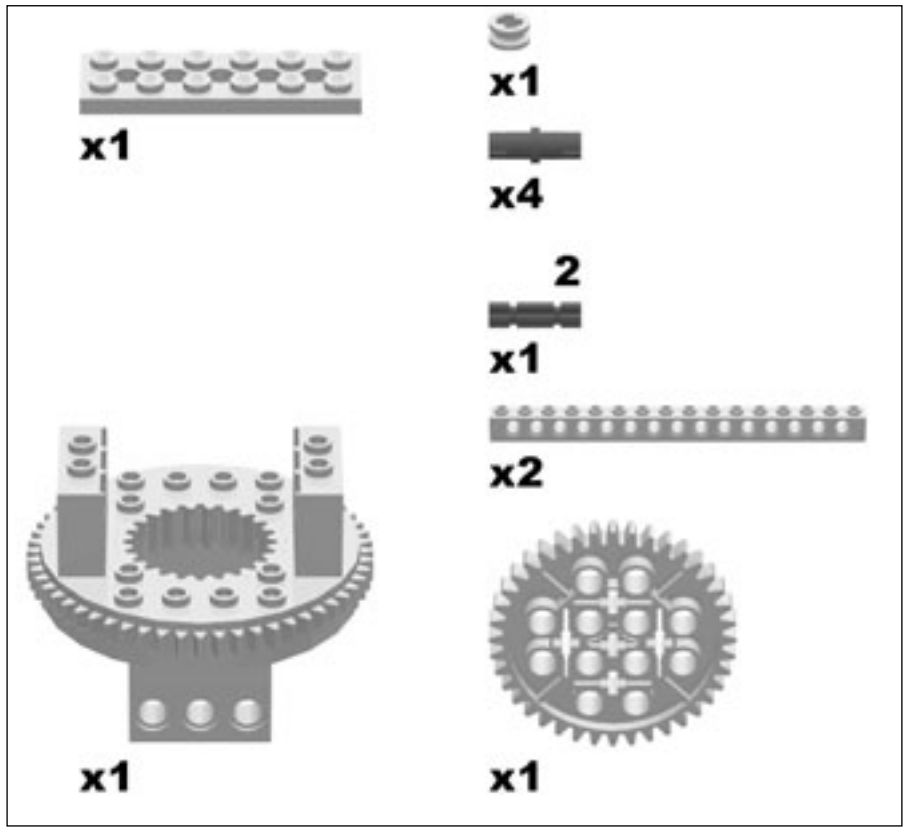
Forearm Step 0



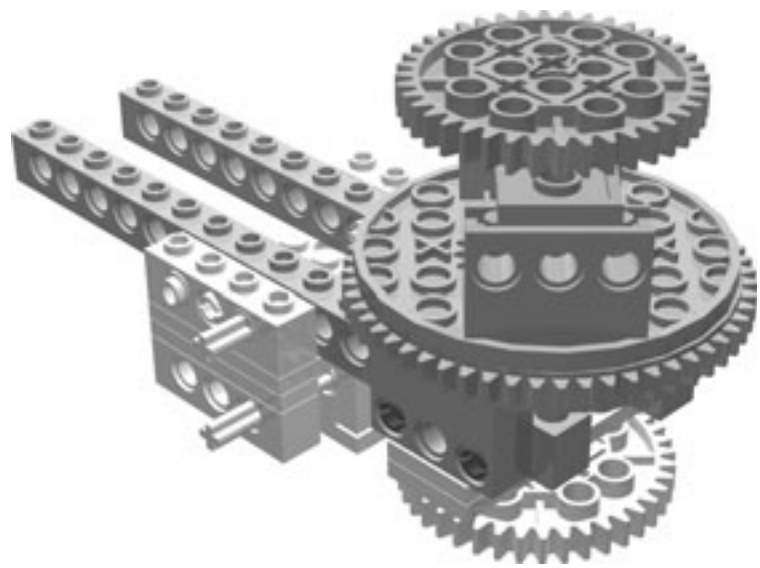
The rotation sensor added in this step will be attached to Input Port 1 of RCX 1 in **Final Arm Step 4**. Refer to Figure 7.12 later in the chapter for a detailed wiring schematic for the sensor.



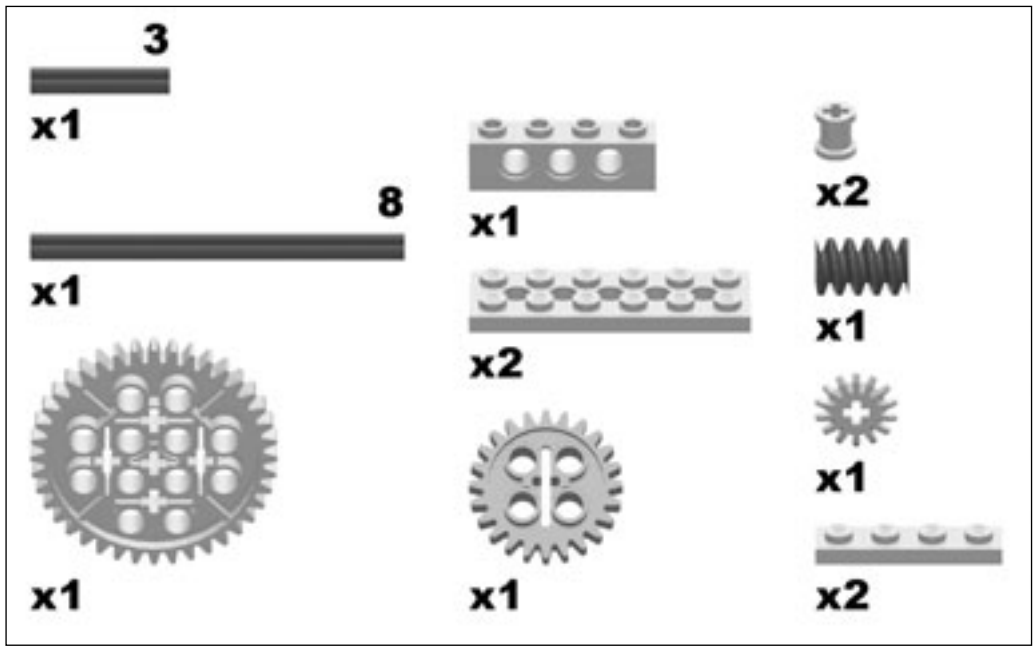
Forearm Step 1



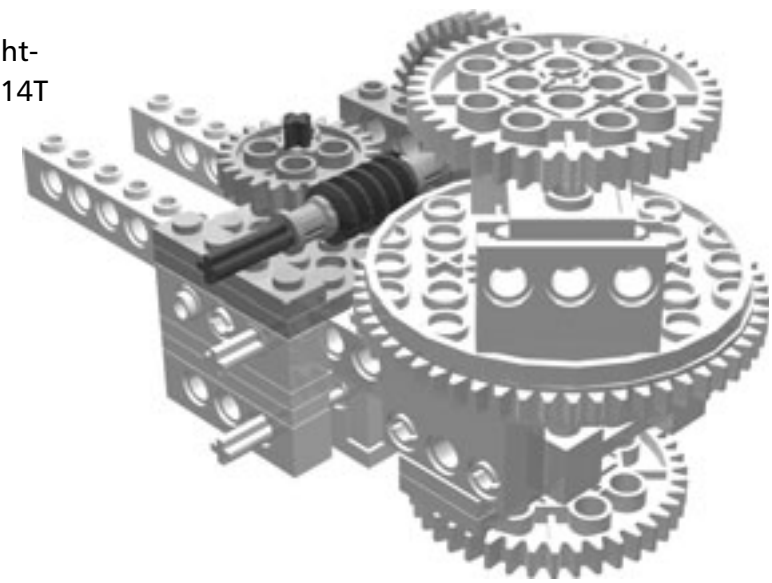
This turntable is the elbow joint.



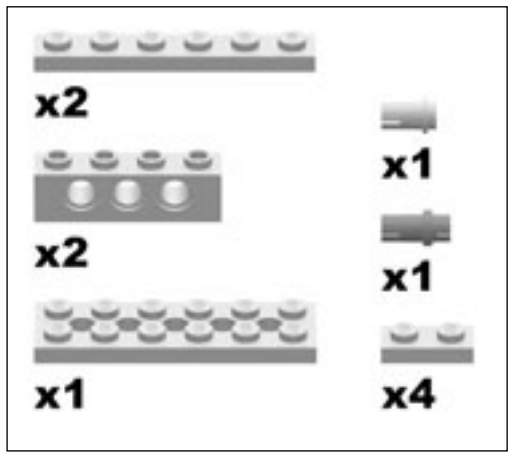
Forearm Step 2

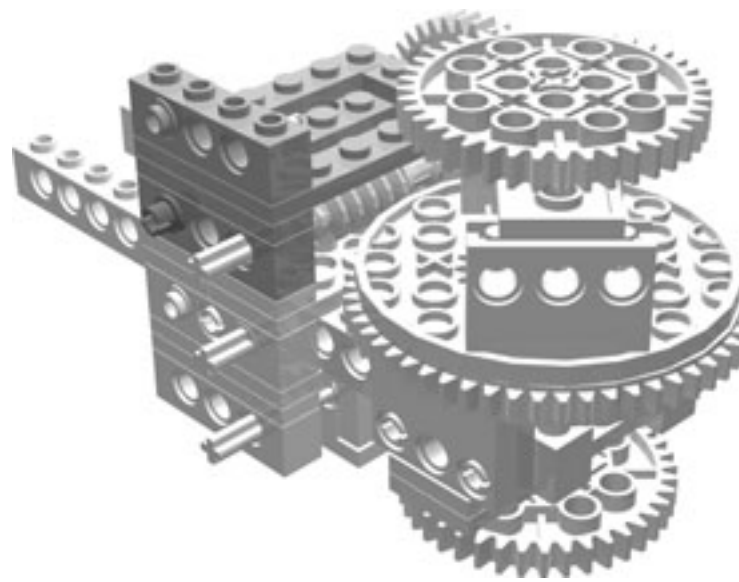


I used some plastic washers for the eight-stud axle. The thin 14T bevel gear is a substitution for a washer.

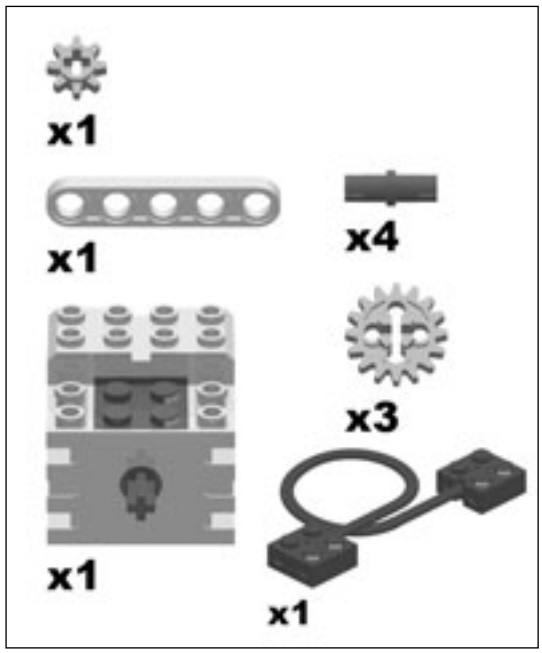


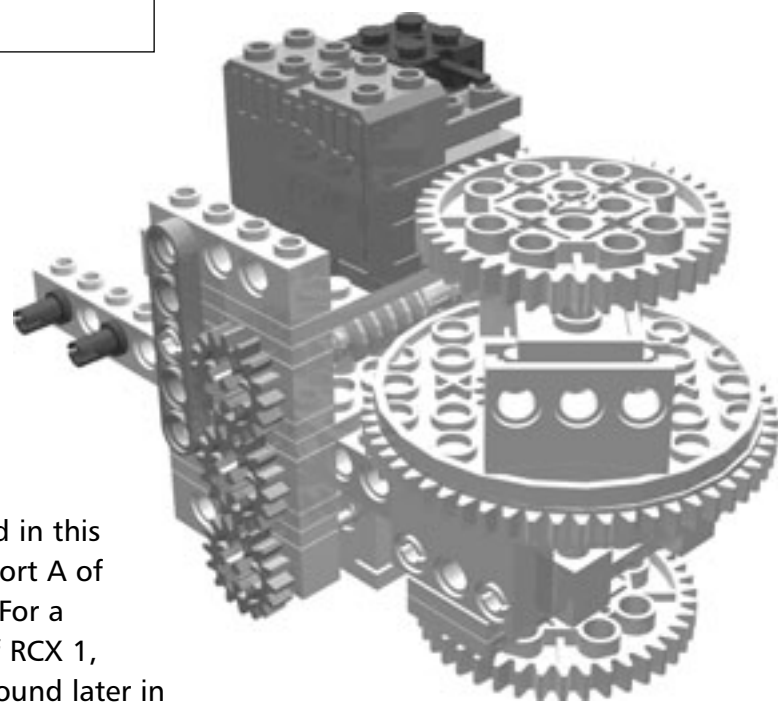
Forearm Step 3





Forearm Step 4

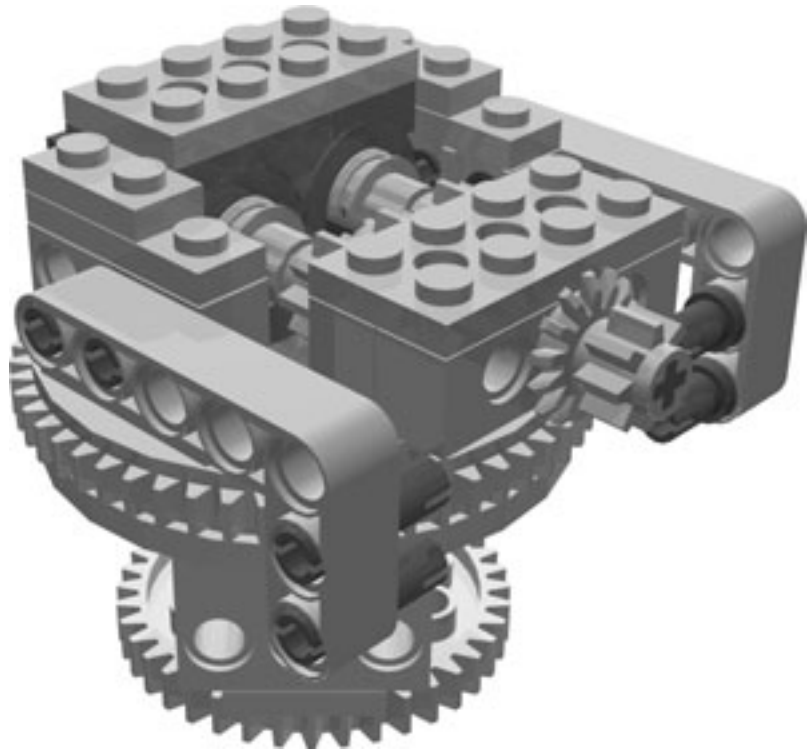




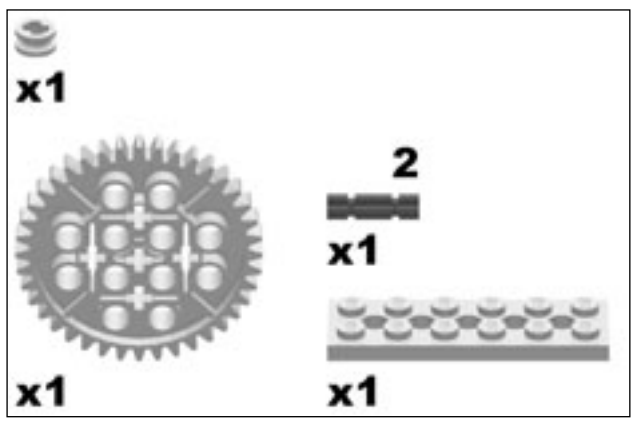
For more power, you may need to reinforce the motor mount.

The electrical connector used in this step is attached to Output Port A of RCX 2 in **Final Arm Step 4**. For a detailed wiring schematic of RCX 1, please refer to Figure 7.12 found later in this chapter.

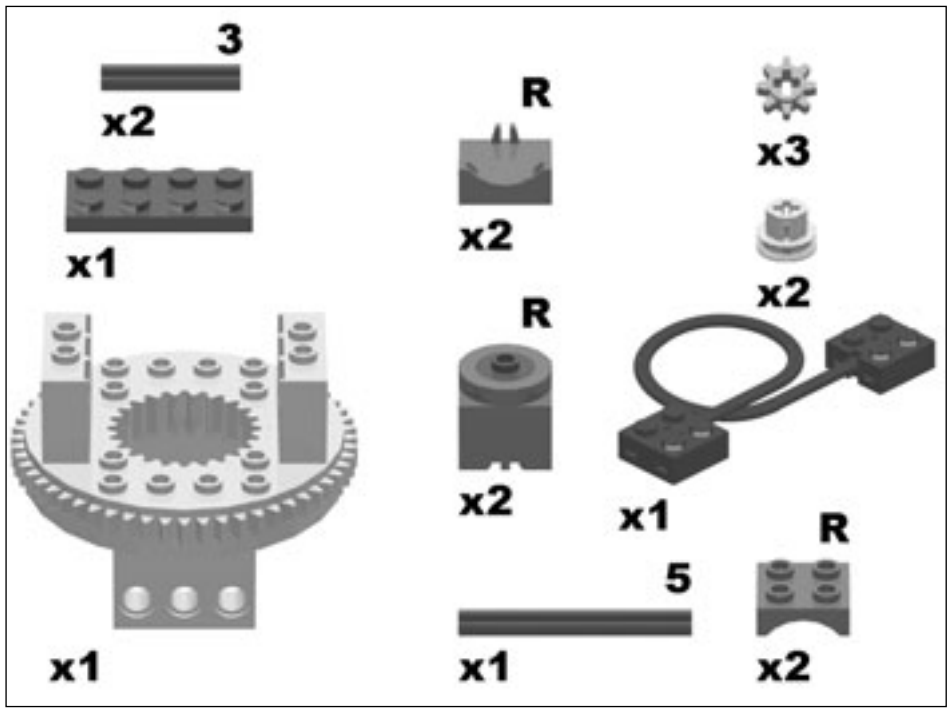
The Wrist



Wrist Step 0

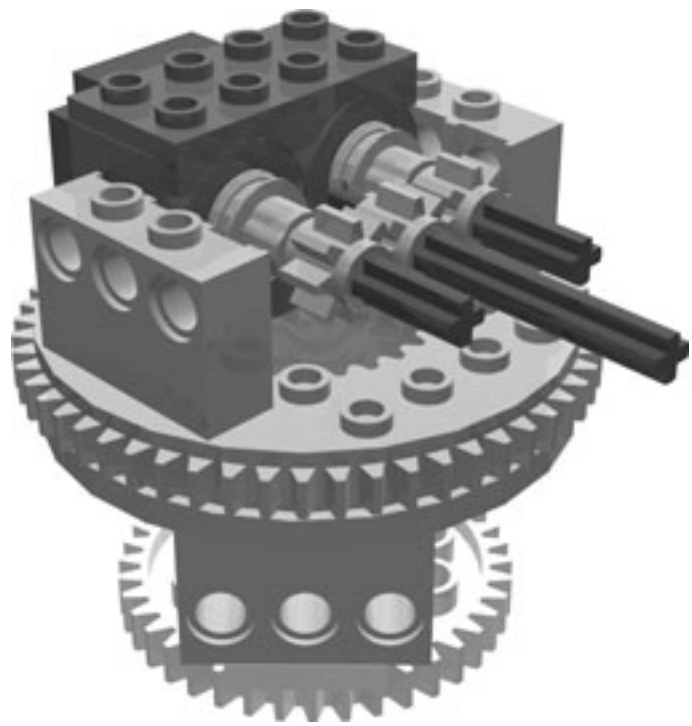


Wrist Step 1

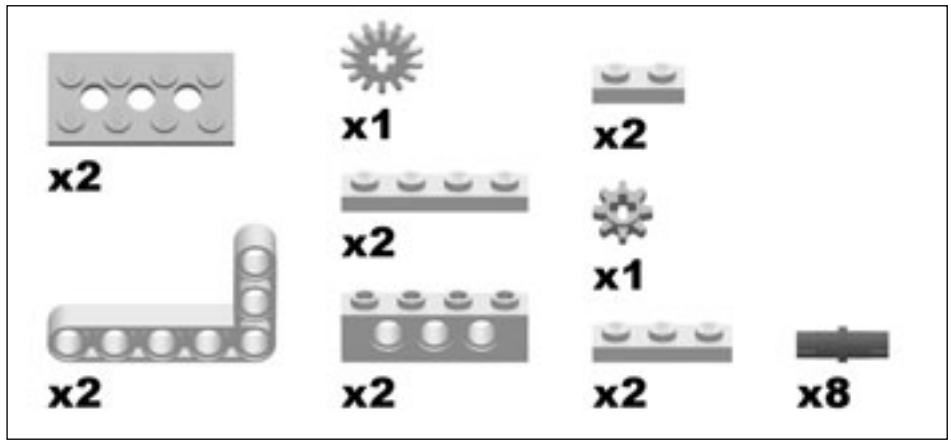


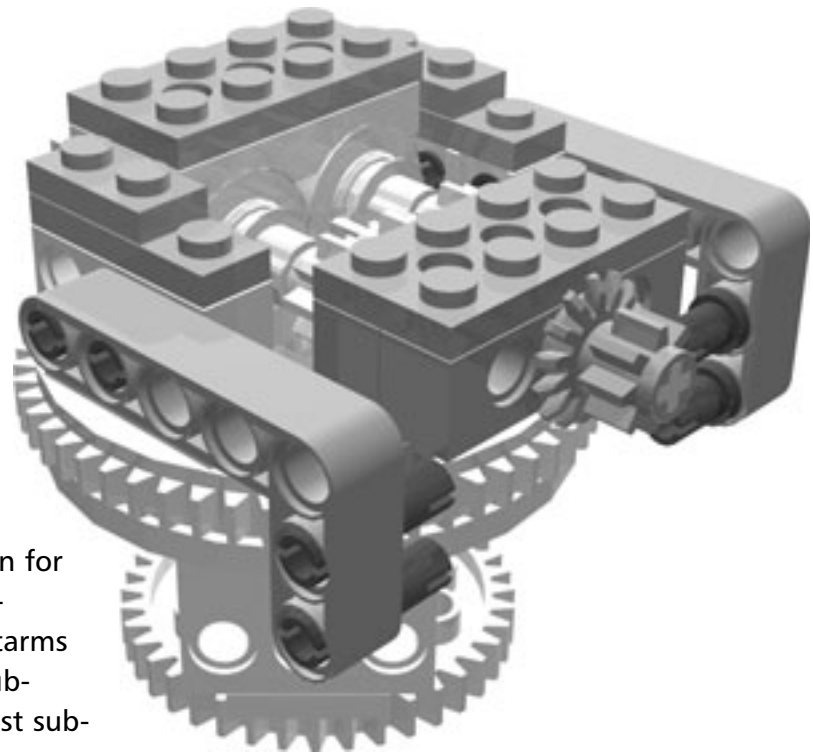
When you use a conducting electric plate, please check the polarity of the connected micromotors.

This electrical connector is attached to Output Port B of RCX 2 in **Final Arm Step 4**. For a detailed wiring schematic of RCX 2 please refer to Figure 7.13 found later in this chapter.



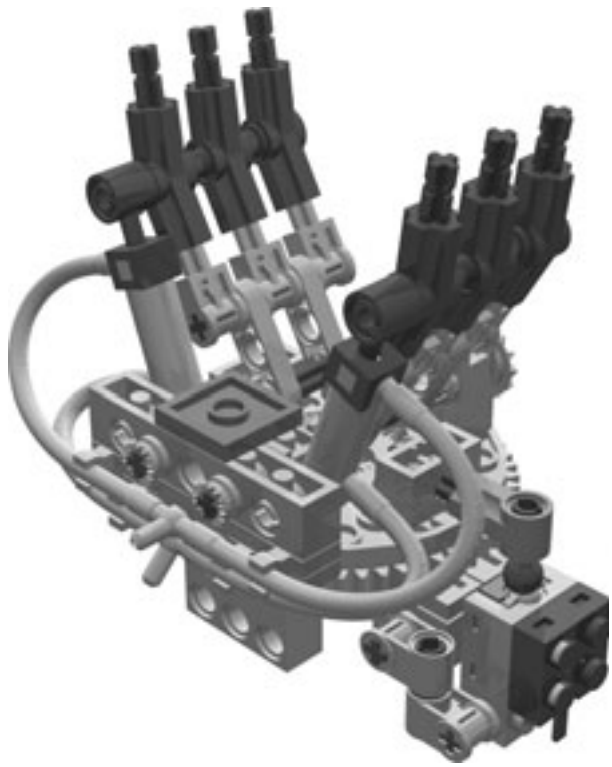
Wrist Step 2





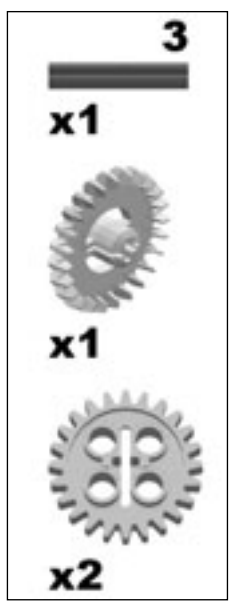
This thin 14T bevel gear is a substitution for a washer. The 3x5 L-Shaped TECHNIC liftarms hold the Grabber sub-assembly to the Wrist sub-assembly.

The Grabber



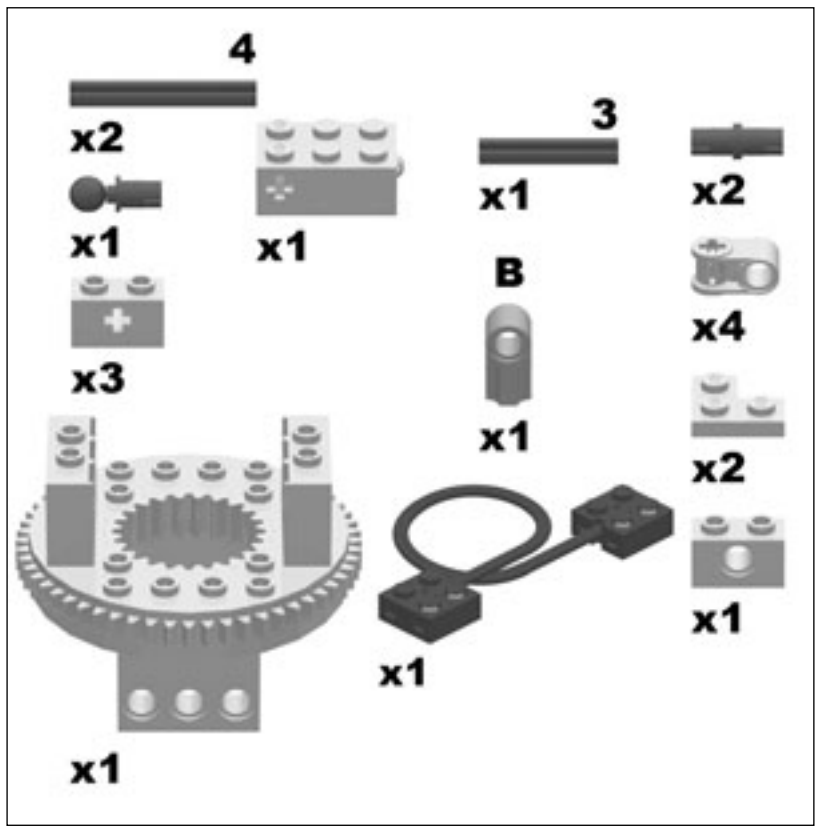
The Grabber sub-assembly is the hand of CyberArm IV. As you can see, this attachment lends itself to multiple modifications, so feel free to experiment with your grabber mechanism. The 3x5 L-shaped liftarms in the wrist sub-assembly allow the Grabber Sub-Assembly to be easily detached from the Wrist sub-assembly.

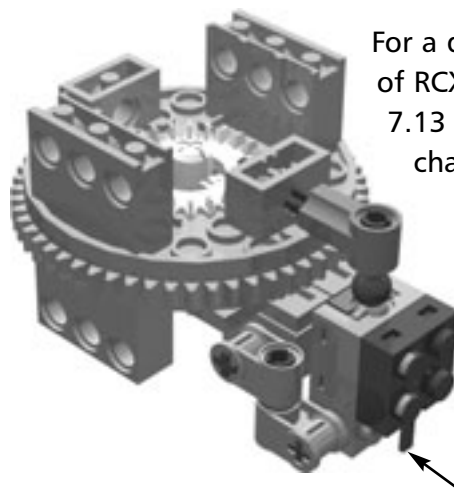
Grabber Step 0



This detachable gear mechanism is central to the grabber sub-assembly. The bottom 24T gear is driven by the 8T pinion gear in the Wrist sub-assembly.

Grabber Step 1

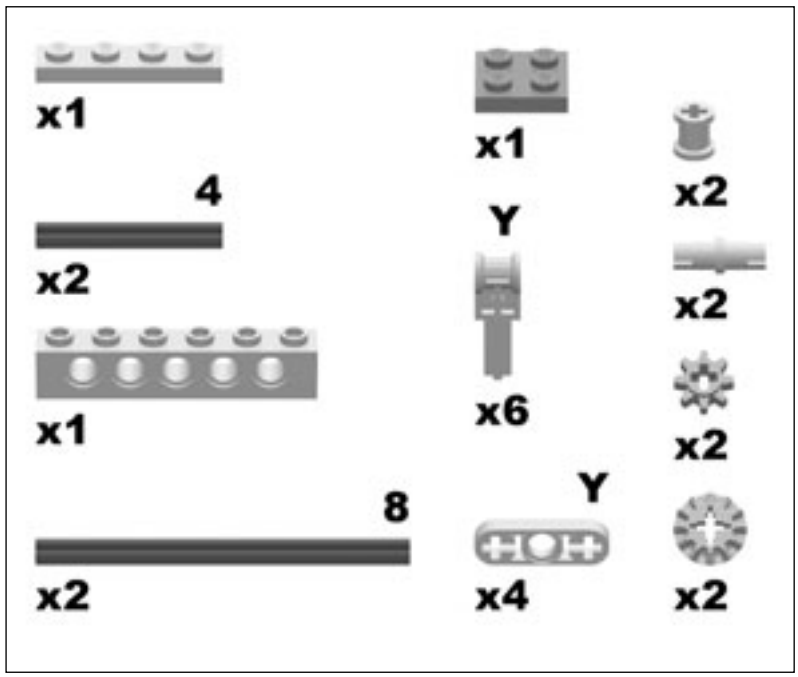




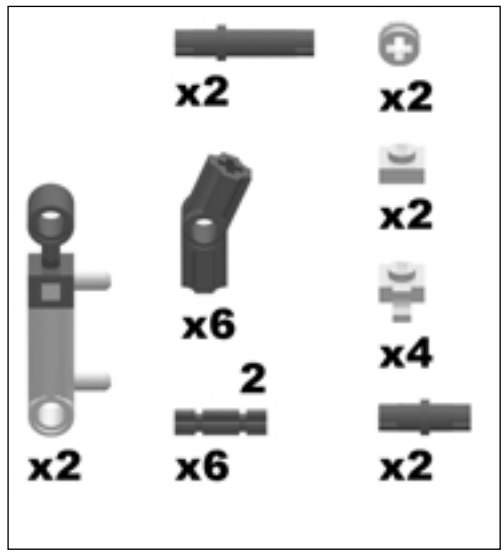
For a detailed wring schematic of RCX 2, please refer to Figure 7.13 found later in this chapter.

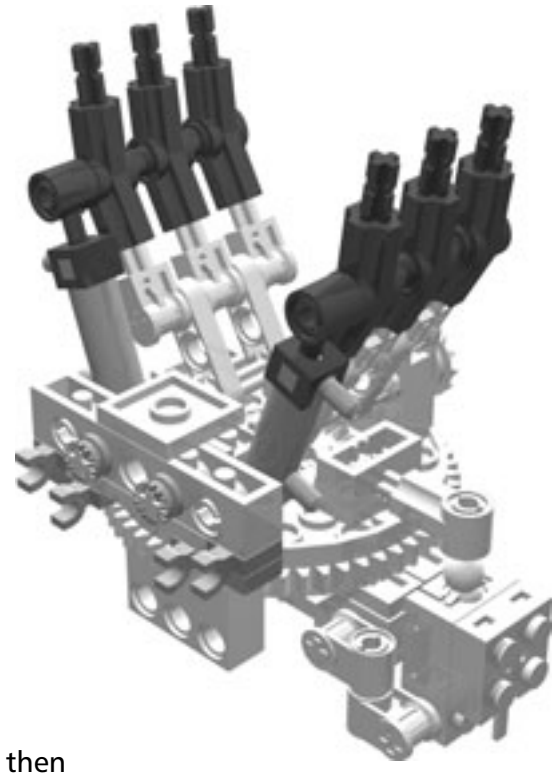
The electrical connector in this step is attached to Input Port 2 of RCX 2 in **Final Arm Step 4**.

Grabber Step 2



Grabber Step 3

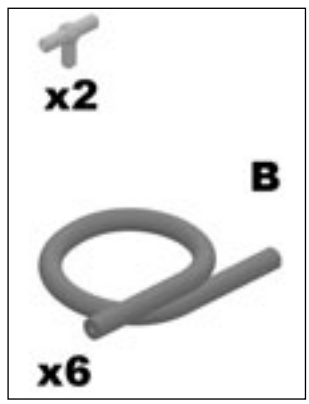




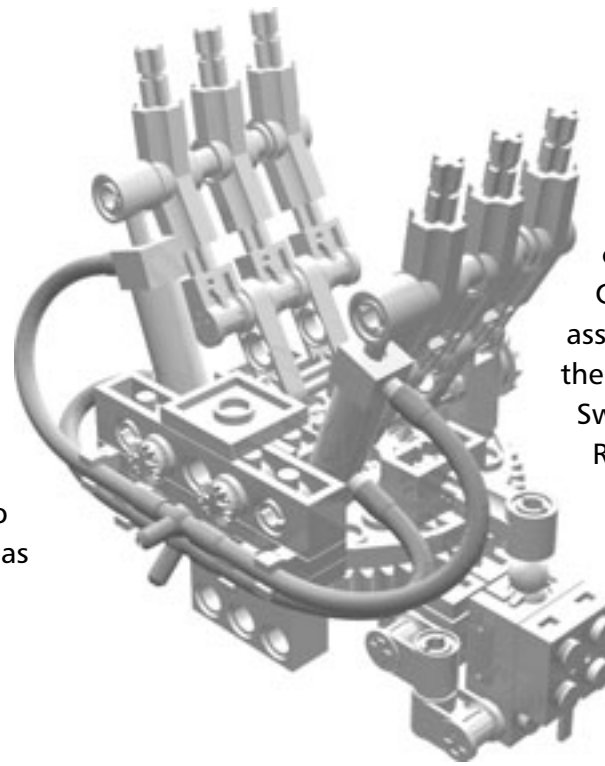
The TECHNIC angled-axle joiners used in this step have a #3 stamped on the side of the part.

If you require additional grabbing power, then use four pneumatic cylinders instead of two.

Grabber Step 4

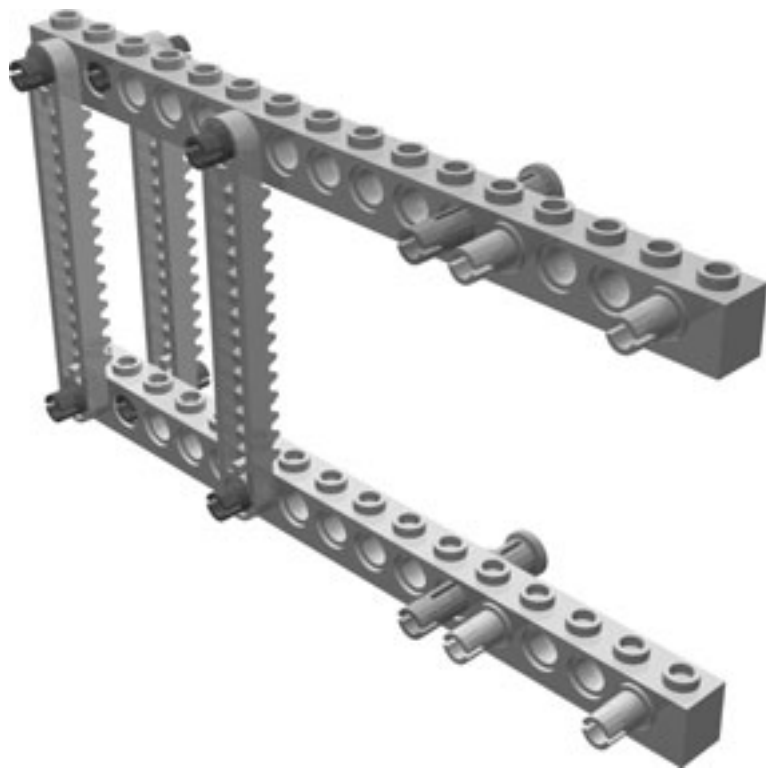


Connect the pneumatic tubing and T-connectors to the Grabber sub-assembly as shown.



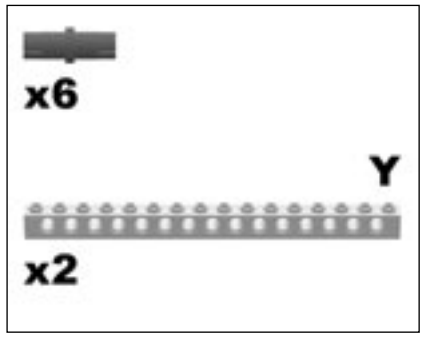
Also not shown, you will need to attach two additional pieces of pneumatic tubing to the open T-Connectors to connect the Grabber sub-assembly and the and the Pneumatic Valve Switch sub-assembly. Refer back to Figure 7.5 for a visual aide for these connection points.

The Lower RCX Frame

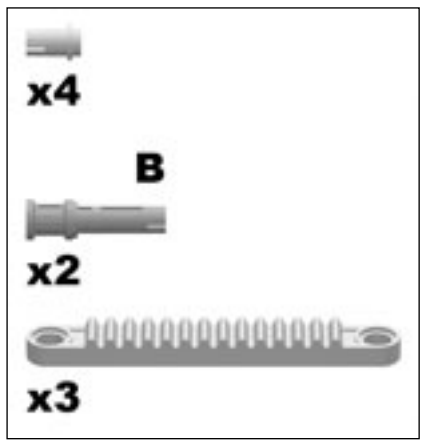


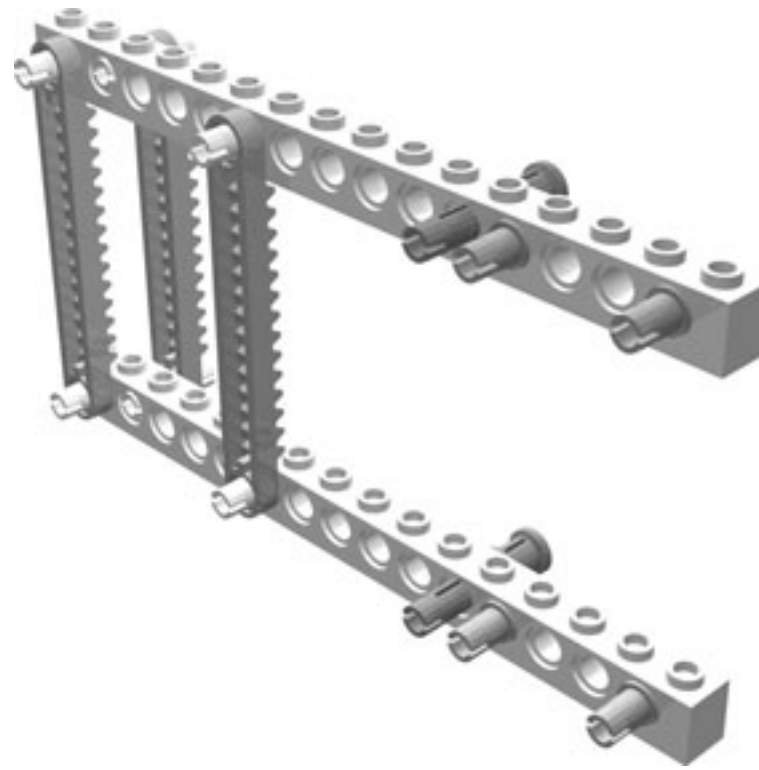
This is the arm's Lower RCX Frame sub-assembly to which you will attach the previously built elements. You will note that **Lower RCX Frame Step 1** uses 1x8 gear racks with holes. If you do not have accessibility to this part, you can modify this design using two 1x6 TECHNIC lif-tarms and two 1x8 TECHNIC beams.

Lower RCX Frame Step 0

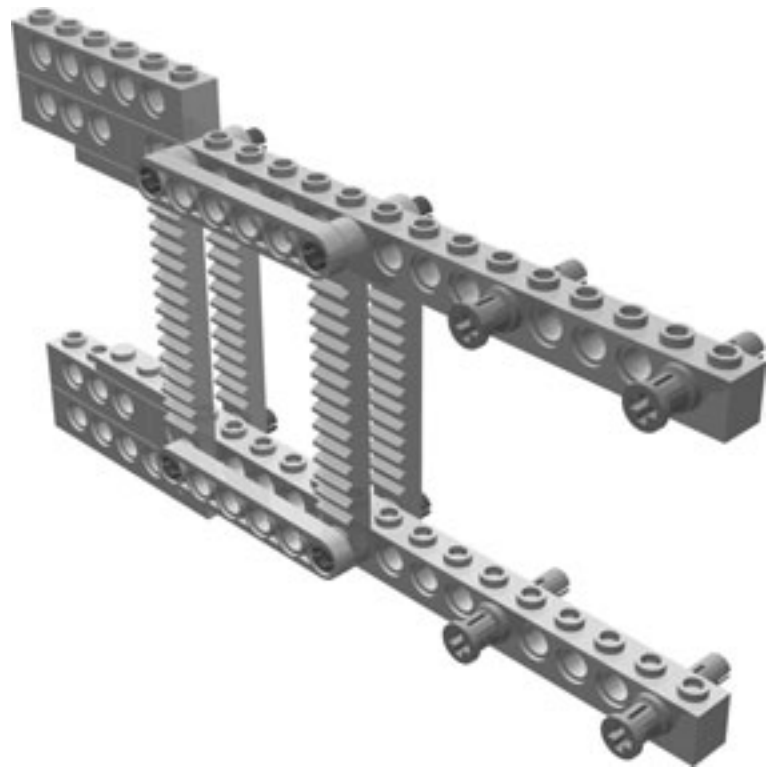


Lower RCX Frame Step 1



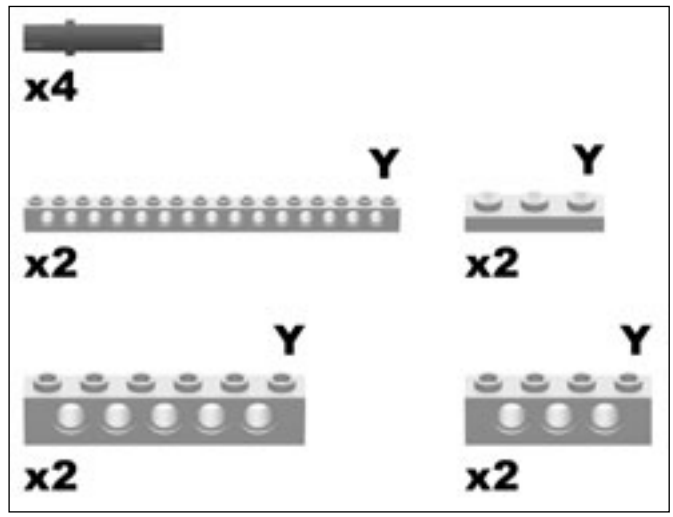


The Upper RCX Frame

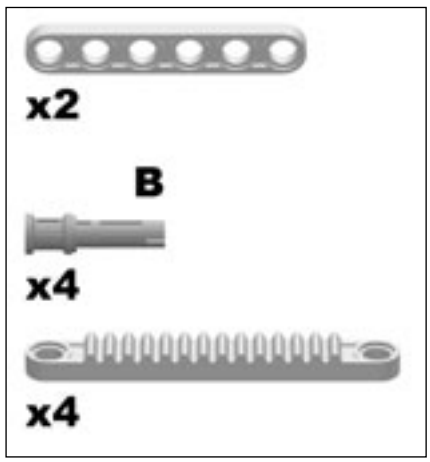


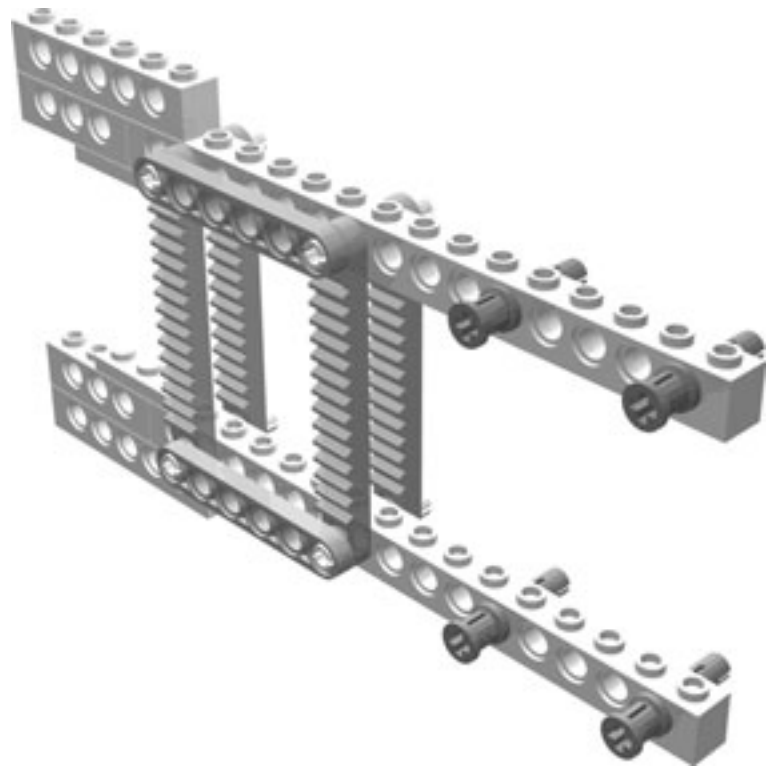
The Upper RCX Frame sub-assembly is the second part of the arm. It serves as the connection point for the power drive mechanisms of the other arm elements.

Upper RCX Frame Step 0



Upper RCX Frame Step 1

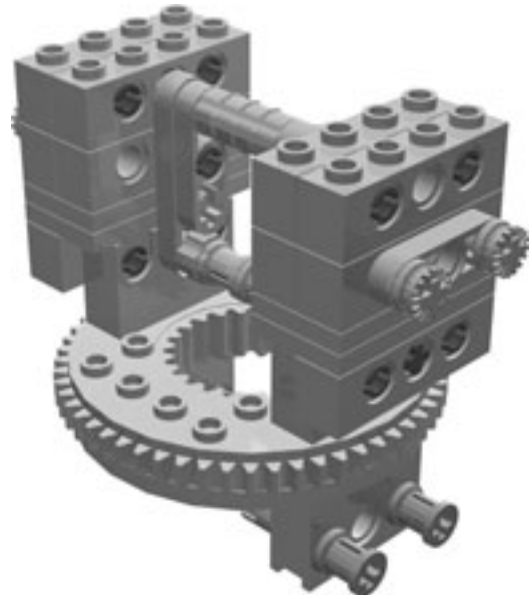




NOTE

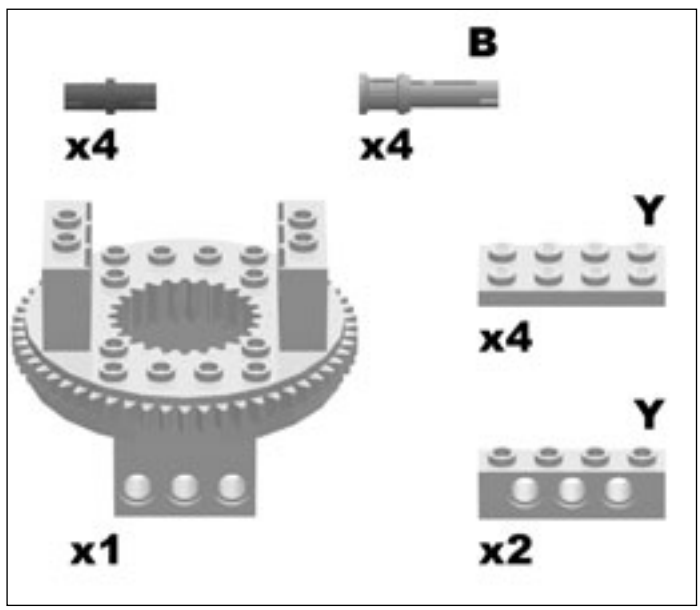
Again, if you do not have access to 1x8 gear racks with holes you can modify this design and use two 1x6 TECHNIC liftarms and two 1x8 TECHNIC beams in the same way that you modified the Lower Frame sub-assembly.

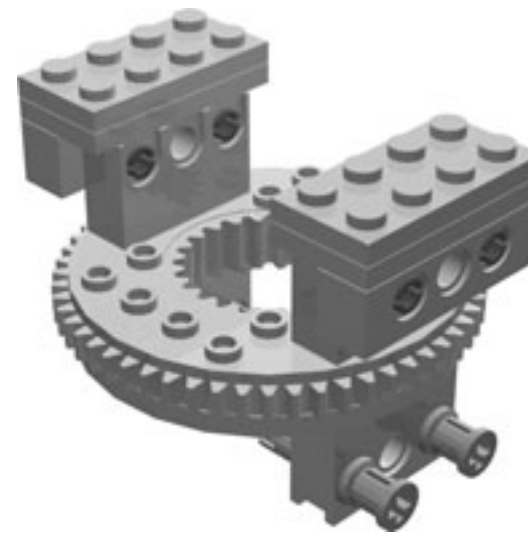
The Turntable



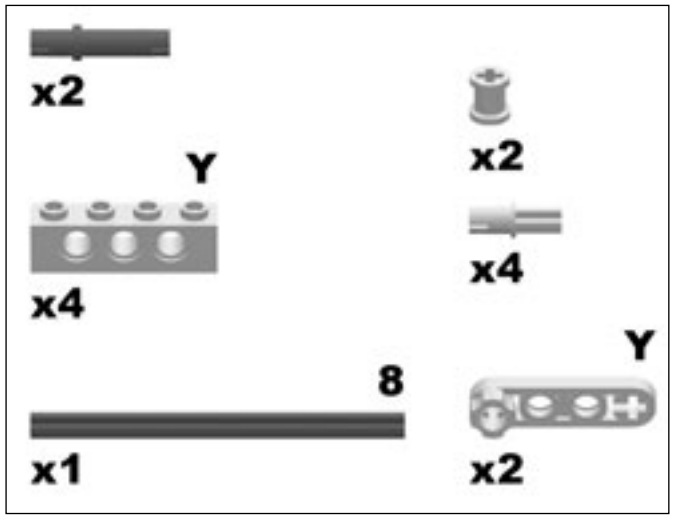
The Turntable sub-assembly is a functional part of the entire Final Arm sub-assembly. You will use this component in **RCX Step 1**.

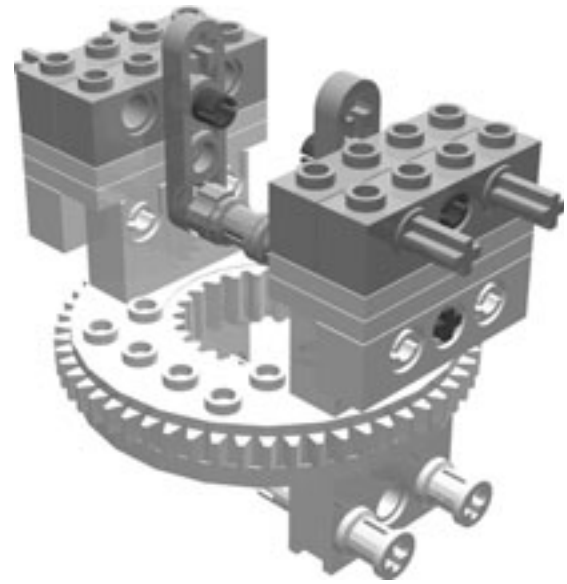
Turntable Step 0



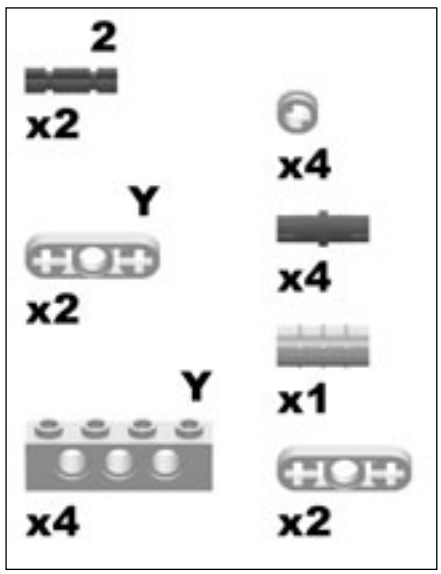


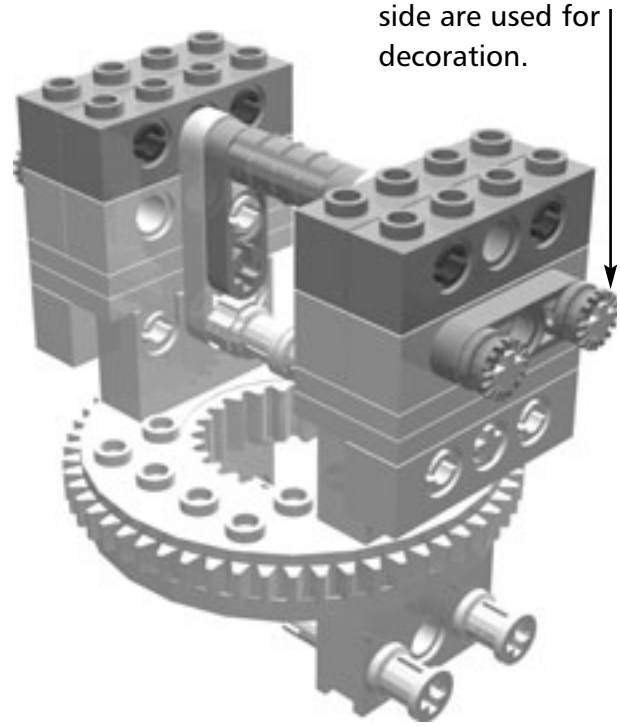
Turntable Step 1



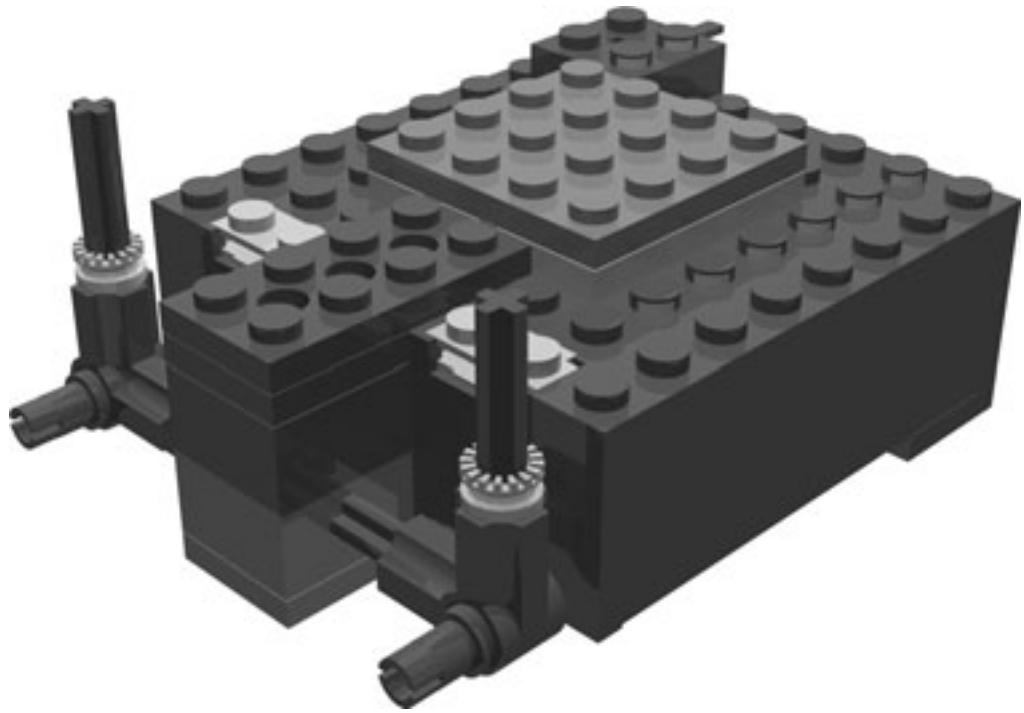


Turntable Step 2



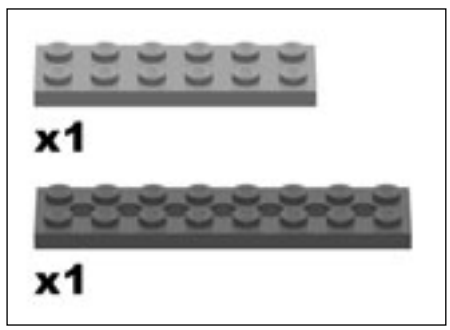


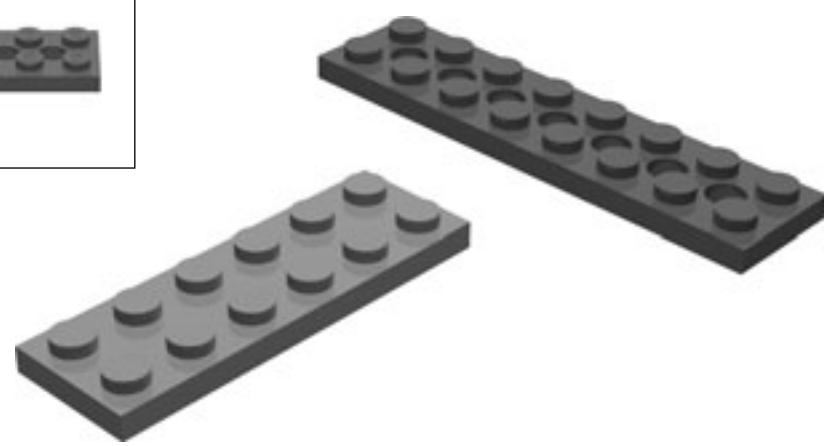
The 9Volt Battery Box



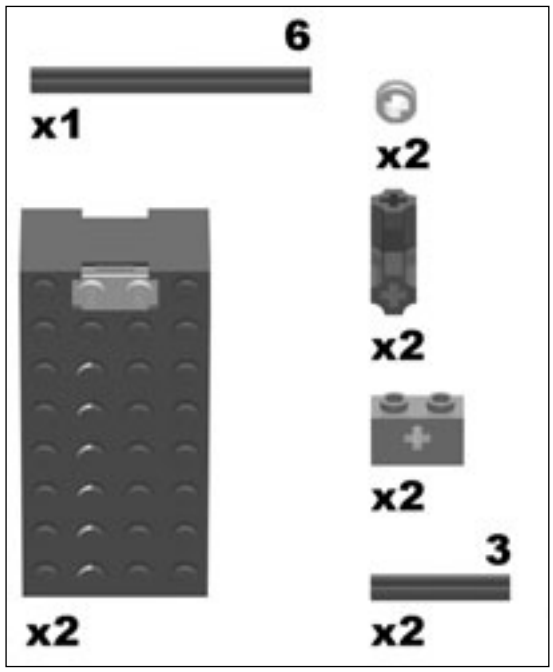
This component supplies power to the pneumatic compressor independently, and provides weight balance. If you decide to use a large external 9V (or 6AA) battery box rather than the battery box shown here, you will have to shift the location of the RCX to maintain the balance of the arm.

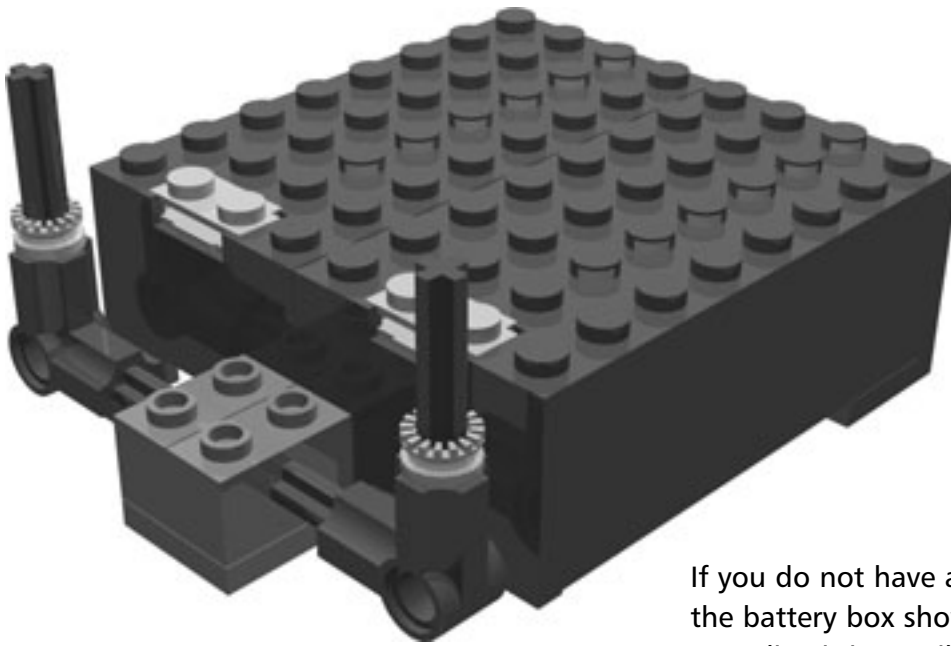
Battery Box Step 0





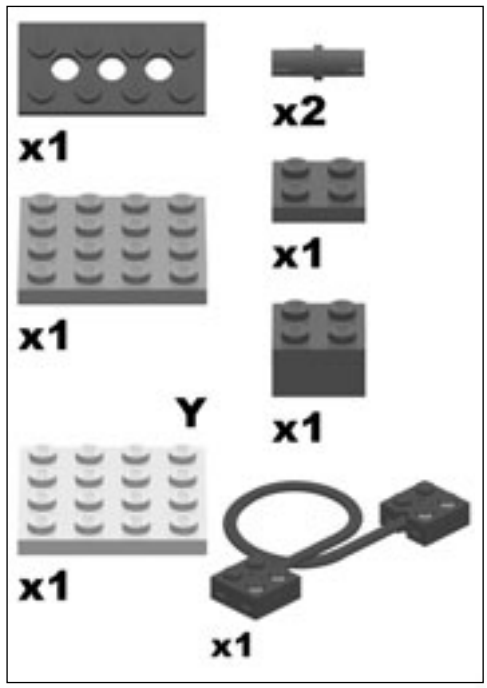
Battery Box Step 1

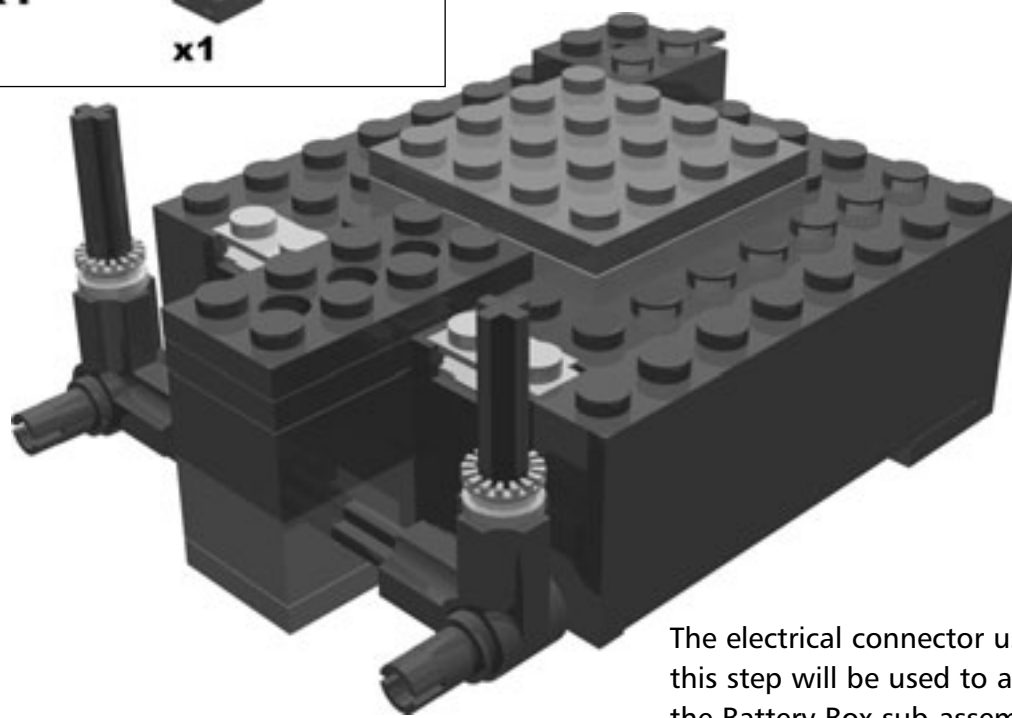




If you do not have access to the battery box shown in the parts list, it is possible to substitute this with the battery box shown here.

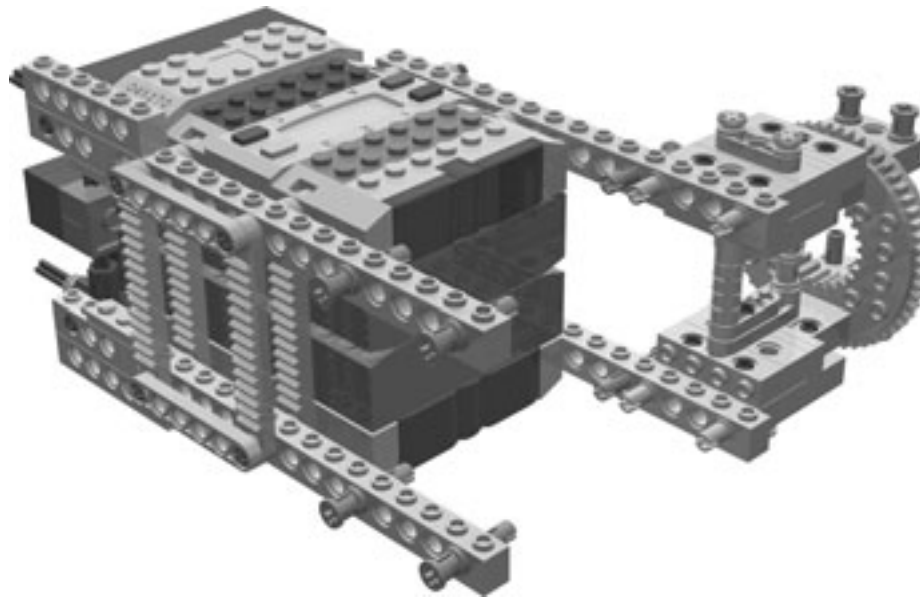
Battery Box Step 2





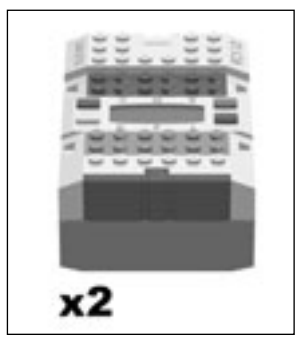
The electrical connector used in this step will be used to attach the Battery Box sub-assembly and the Limiter Switch sub-assembly. You should refer back to Figure 7.5 for an illustration of this connection as it applies to the pneumatic system for CyberArm IV.

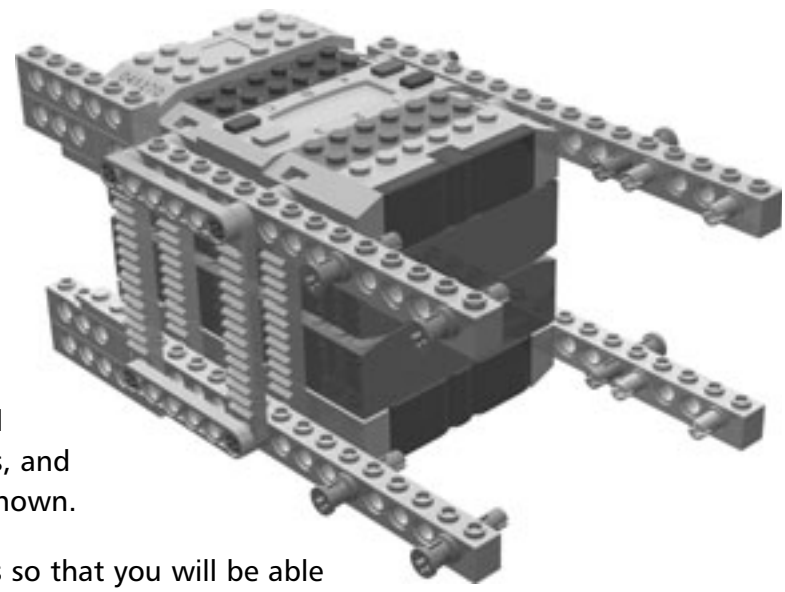
RCX 1 and RCX 2



The Upper and Lower RCX frame sub-assemblies are able to separate easily (for changing batteries) by pulling out the blue TECHNIC snap connectors.

RCX Step 0

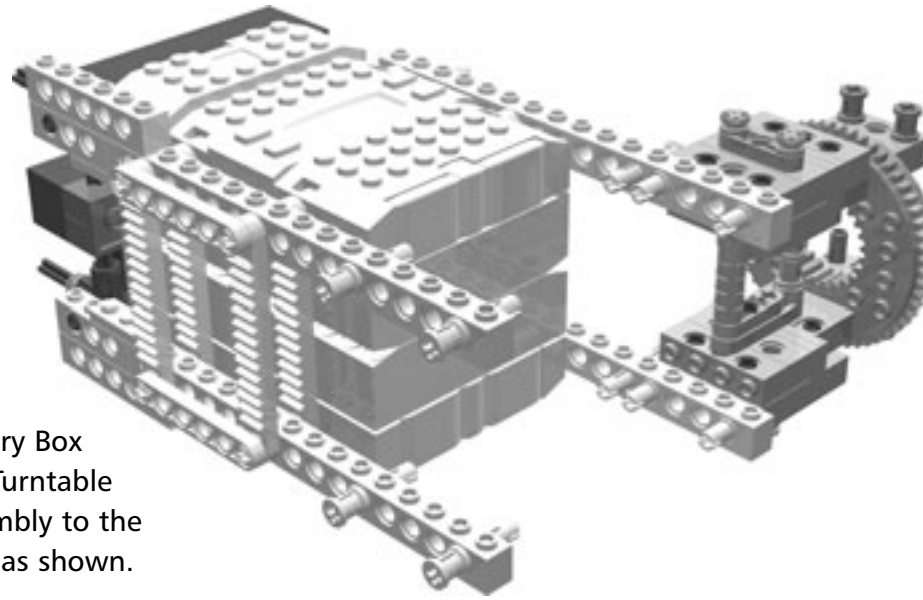




Locate the Upper Frame and Lower Frame sub-assemblies, and attach to the two RCXs as shown.

It is useful to mark the RCXs so that you will be able to differentiate between which RCX is designated RCX 1 and which RCX is designated RCX 2 when making the electrical connections in **Final Arm Step 4.**

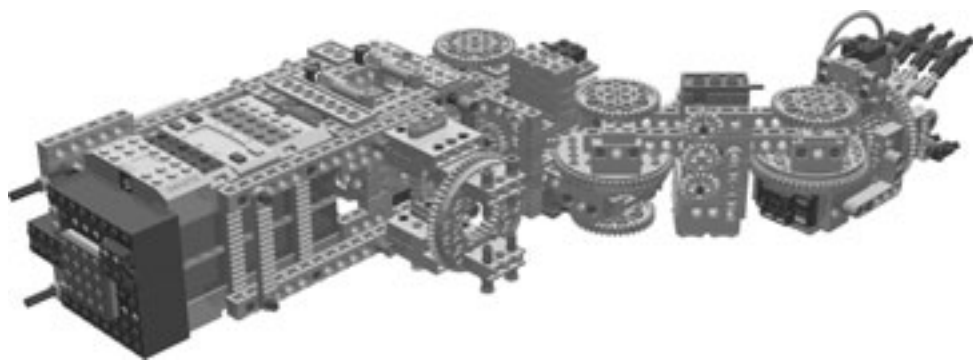
RCX Step 1



Attach the Battery Box and the Turntable sub-assembly to the structure as shown.

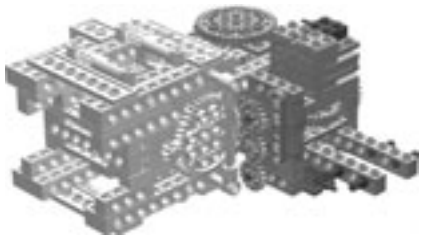
When you attach the Turntable sub-assembly, you will be connecting the studs of bricks stuck into holes of bricks. This is a tighter connection than the normal brick-on-brick connection.

Completing the Arm



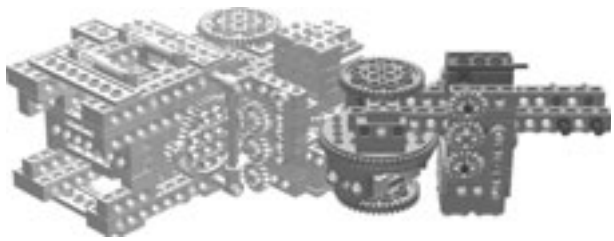
We will complete the Arm sub-assembly in this next series of steps. I recommend looking through all the steps in this section before trying to attach everything together.

Final Arm Step 0



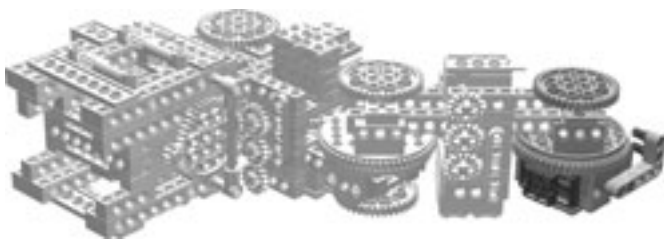
Attach the Shoulder sub-assembly to the Upper Arm sub-assembly as shown.

Final Arm Step 1



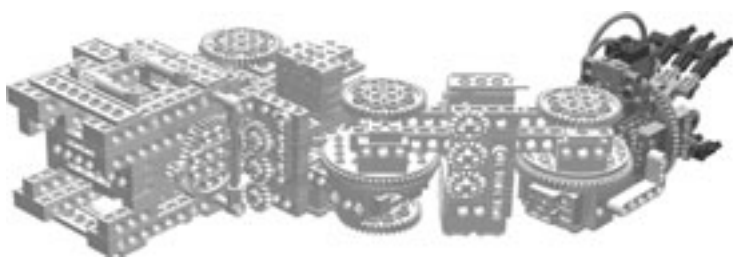
Next, attach the Forearm sub-assembly to the structure as shown.

Final Arm Step 2



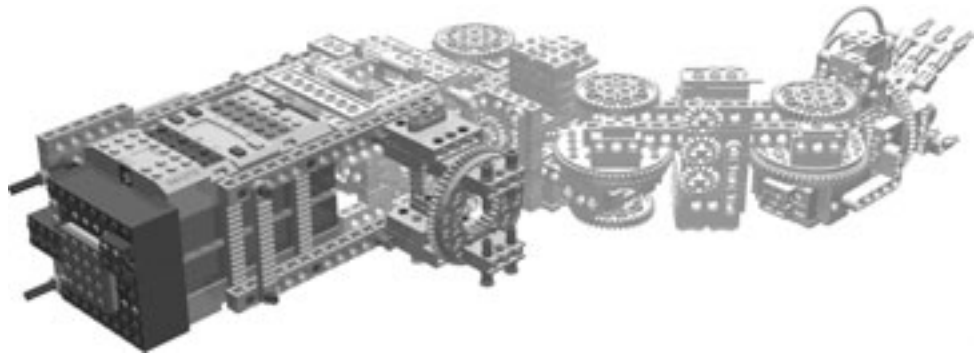
The Wrist sub-assembly is attached next.

Final Arm Step 3



Attach the Grabber sub-assembly to the liftarms of the Wrist sub-assembly.

Final Arm Step 4



Finally, attach the RCX Frame to the structure. I still find it amazing that this LEGO structure contains enough strength to support this large, heavy complicated structure.

This is the step where you should attach all of the electrical connectors that are hanging free from the various components of the Final Arm sub-assembly. Please refer to Figures 7.12 and 7.13 for detailed schematics of the proper port placements.

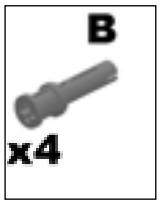
Also as an aide you might find it useful to review Table 7.2, which lists the connections as well.

Figure 7.2 Wiring Connections for Final Arm Step 4

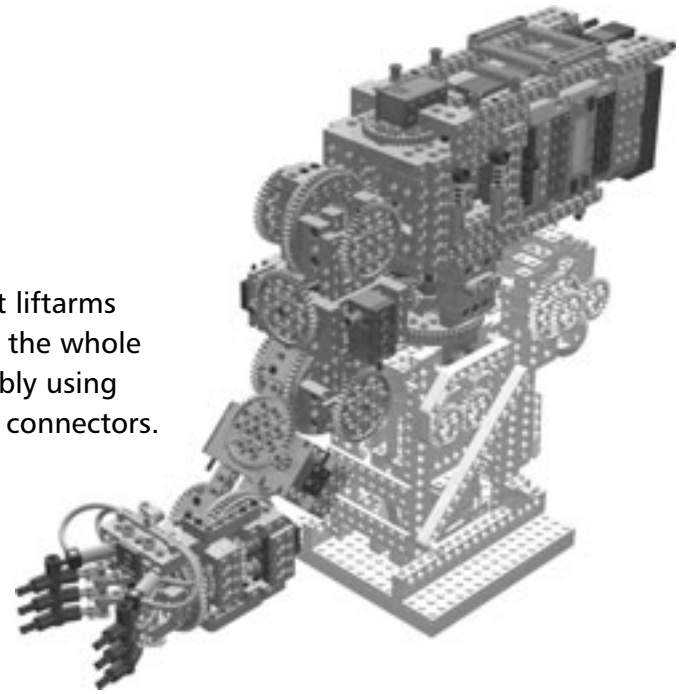
Component	Port
Forearm Rotation Sensor	RCX 1 Input Port 1
Forearm Motor	RCX 1 Output Port A
Upper Arm Motor	RCX 1 Output Port B
Upper Arm Rotation Sensor	RCX 1 Input Port 2
Shoulder Motors	RCX 1 Output Port C
Shoulder Rotation Sensor	RCX 1 Input Port 3
Grabber Touch Sensor	RCX 2 Input Port 2
Wrist Motors	RCX 2 Output Port B
Turntable Drive Rotation Sensor	RCX 2 Input Port 3
Turntable Drive Motors	RCX 2 Output Port C
Pneumatic Valve Switch Motor	RCX 2 Output Port A

Putting It All Together

Final Step 0



Remove the double bent liftarms parts first, then connect the whole arm to the tower assembly using four blue TECHNIC snap connectors.



Programming the CyberArm IV

The CyberArm IV uses two RCX bricks to move the arm. The bricks communicate with each other using the *SendMessage()* command. By using *SendMessage()* in the subroutine, the arm is cable of performing two simultaneous actions. For example, the arm can be raised/lowered while being rotated on the turntable. Using the repeat statement *while(true)*, commands in between the brackets are repeated forever until you push the Message Button #2 on the LEGO remote control. In the case of the CyberArm IV, a series of *SendMessage()* commands fly back and forth the between the two RCXs.

Figure 7.11 Bricx Command Center 3.3

This robotic arm was created by combining independent components. With the exception of the pneumatic system, all the moveable components have their own sensors. This type of modular construction provides an ideal feedback system for the arm as a whole. A specific NQC program can control each individual component. Furthermore, when fully assembled, the CyberArm IV has one free sensor port. If you have an extra light sensor, you could add it to the robot to the color of a sponge ball, allowing the arm to sort balls according to their colors (similar to the duties that could be accomplished with CyberArm II). One thing to keep in mind while creating the program for the additional light sensor is that you would have to be able to check the threshold values for the light sensor each time you wish to use the CyberArm IV in a new environment. Additionally, you could use LEGO Vision Command (with the Logitech QuickCam SDK) to turn the CyberArm into an advanced robotic arm.

After analyzing the arm movement, I was able to define the minimum amount of motion and code this into the movement subroutines. Using variables to keep track of the motions of the arm, you can know where the arm is in the air at all times.

NOTE

The RCX allows for up to eight subroutines and ten tasks.

The subroutines used in the program for CyberArm IV are as follows:

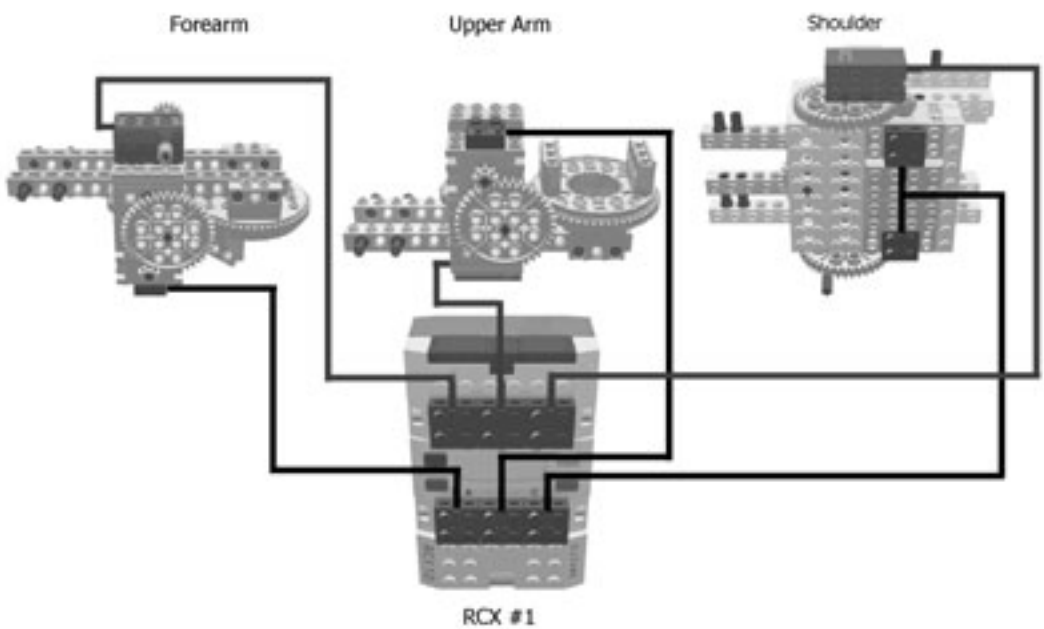
- **Upper Arm** This subroutine raises and lowers the upper arm.
- **Forearm** This subroutine raises and lowers the forearm.
- **Grabber** These subroutines are responsible for raising and lowering the arm, as well as rotating the grabber clockwise and counterclockwise, and opening and closing the fingers of the grabber.

- **Turntable** This subroutine rotates the turntable in either a clockwise or counterclockwise direction.

Previously, the motion of the Grabber sub-assembly (wrist rotation and open/close of the grabber) was controlled by using direct timing. In CyberArm IV, a touch sensor controls the wrist rotation in order to return to its home position. Besides the wrist rotation, basic movements of the arm are controlled by the feedback information from the rotation sensors, coupled with the joints. This solved the problem of being able to control the arm with precision, which I had encountered with my CyberArm III. As the weight and length of the arm increases, the motions become increasingly imprecise. For example, the time to raise the arm a certain distance was long but the time it took to lower the arm the same distance was much shorter. I really struggled with the timing on my older arms since battery power, payload, and even gravity changed every situation from movement to movement.

Figures 7.12 and 7.13 show the system diagrams for the RCXs.

Figure 7.12 System Schematic for RCX 1

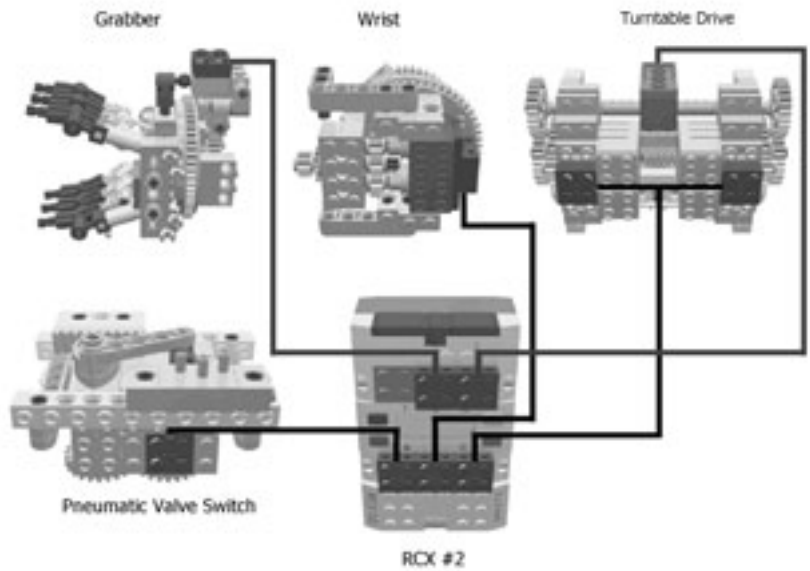


RCX 1 (controlled by master.nqc) controls the shoulder, the upper arm and the forearm. More specifically, the parameters for controlling the arm are as follows:

- **Moving the Wrist up and down by Forearm** *OUT_A* (one motor) and *IN_1* (rotation sensor)
- **Moving the Forearm up and down by Upper arm** *OUT_B* (one motor) and *IN_2* (rotation sensor)
- **Moving the Upper arm up and down by Shoulder** *OUT_C* (two geared motor) and *IN_3* (rotation sensor)

Watch for these constructs and syntax in the code that follows later in this section.

Figure 7.13 System Schematic for RCX 2



RCX 2 (controlled by slave.nqc) controls the turntable, turning the wrist and opening and closing the grabber. More specifically, the parameters for controlling the arm are as follows:

- **Grabber (open/close)** OUT_A (one motor on the pneumatic valve switch)
- **Grabber (turn)** OUT_B (two micro-motor on wrist) and IN_2 (touch sensor)
- **Turntable OUT_C** (two geared motor on turntable drive) and IN_3 (rotation sensor)

To minimize unstable factors, I used AC adaptors to stabilize motor power; however, this problem remained unresolved in previous CyberArms. With rotation sensors providing feedback information, even if the batteries are low and it takes longer for the arm to get into position, the position will be correct. A sample program for testing the arm

performs the following numbered list of actions. This is a simple, useful way to check that all the movements and sensors work correctly. The Message #1 Button on the LEGO remote control starts the program. If the polarity is reversed, then you must change the command *OnFwd()* with *OnRev()*.

1. Raise the grabber up.
2. Raise the forearm up.
3. Turn the arm counterclockwise.
4. Lower the forearm.
5. Lower the grabber.
6. Grasp something light like a sponge ball.
7. Raise the grabber up.
8. Raise the forearm up.
9. Raise the upper arm up.
10. Turn the arm clockwise.
11. Turn the wrist one way and back in the air.
12. Turn the arm counterclockwise to the center.
13. Lower the upper arm.
14. Lower the forearm.
15. Lower the grabber.
16. Take off the ball.

Without further ado, let us take a look at the code for CyberArm IV. This program can also be found on the CD-ROM that accompanies this book.

NOTE

Before running the program, please be sure to set the CyberArm IV into the position that is shown on the first page of this chapter. This is the home position for the arm.

```
/*
   CyberArm IV (Code for Master RCX - RCX #1)
   NQC 2.4r3
   master.nqc
   Hideaki Joda Yabuki, 2002, Japan
*/
```

```
#define UA_RCOUNT  30       // Upper arm Rotation Sensor Value
#define FA_RCOUNT  20       // Forearm Rotation Sensor Value
#define WT_RCOUNT  40       // Wrist Rotation Sensor Value
sub upperarm_up()
{
  SetPower(OUT_A,7);
  OnFwd(OUT_A);

  while (true)
  {
    if (SENSOR_1 == UA_RCOUNT)
    {
      Off(OUT_A);
//    SendMessage(10);
      break;
    }
  }
}
sub upperarm_down()
{
  SetPower(OUT_A,7);
  OnRev(OUT_A);
  while (true)
  {
    if (SENSOR_1 == 0)
    {
      Off(OUT_A);
      break;
    }
  }
}
sub forearm_up()
{
  SetPower(OUT_B,7);
  OnFwd(OUT_B);
  while (true)
  {
    if (SENSOR_2 == FA_RCOUNT)
    {
      Off(OUT_B);
```

```
      break;
    }
  }
}
sub forearm_down()
{
  SetPower(OUT_B,7);
  OnRev(OUT_B);
  while (true)
  {
    if (SENSOR_2 == 0)
    {
      Off(OUT_B);
      break;
    }
  }
}
sub wrist_up()
{
  SetPower(OUT_C,7);
  OnFwd(OUT_C);
  while (true)
  {
    if (SENSOR_3 == -WT_RCOUNT)
    {
      Off(OUT_C);
//    SendMessage(10);
      break;
    }
  }
}
sub wrist_down()
{
  SetPower(OUT_C,7);
  OnRev(OUT_C);
  while (true)
  {
    if (SENSOR_3 == 0)
    {
      Off(OUT_C);
```

```
      break;
    }
  }
}
task main()
{
  SetSensor(SENSOR_1,SENSOR_ROTATION);
  SetSensor(SENSOR_2,SENSOR_ROTATION);
  SetSensor(SENSOR_3,SENSOR_ROTATION);
  SetSensorMode(SENSOR_1, SENSOR_MODE_ROTATION);
  SetSensorMode(SENSOR_2, SENSOR_MODE_ROTATION);
  SetSensorMode(SENSOR_3, SENSOR_MODE_ROTATION);
  ClearSensor(SENSOR_1);
  ClearSensor(SENSOR_2);
  ClearSensor(SENSOR_3);
  Wait(100);
  while (true)
  {
    ClearMessage();
    until (Message() != 0);
    if (Message() == 1)
    {
       wrist_up();
       SendMessage(10);
    }
    if (Message() == 11)
    {
       wrist_down();
       SendMessage(12);
    }
    if (Message() == 13)
    {
       wrist_up();
       forearm_up();
       upperarm_up();
       SendMessage(14);
    }
    if (Message() == 15)
    {
       upperarm_down();
```

```
        forearm_down();
        wrist_down();
        SendMessage(16);
     }
     if (Message() == 2)
     {
       StopAllTasks();
     }
   }
}
/*
   CyberArm IV (Code for Slave RCX - RCX #2)
   NQC 2.4r3
   slave.nqc
   Hideaki Joda Yabuki, 2002, Japan
*/
#define TT_RCOUNT  60       // Turntable Rotation Sensor Value
sub turntable_cw()
{
  SetPower(OUT_C,7);
  OnRev(OUT_C);
  while (true)
  {
    if (SENSOR_3 == TT_RCOUNT)
    {
      Off(OUT_C);
      tt_rc =  SENSOR_3;
      Wait(100);
      break;
    }
  }
}
sub turntable_ccw()
{
  SetPower(OUT_C,7);
  OnFwd(OUT_C);
  while (true)
  {
    if (SENSOR_3 == -TT_RCOUNT)
    {
```

```
      Off(OUT_C);
      Wait(100);
//    SendMessage(11);
      break;
    }
  }
}
sub turntable_home()
{
  SetPower(OUT_C,7);
  OnFwd(OUT_C);

  while (true)
  {
    if (SENSOR_3 == 0)
    {
      Off(OUT_C);
      Wait(100);
      break;
    }
  }
}
sub turn_wrist()
{
  OnRev(OUT_B); Wait(200);
  Off(OUT_B); Wait(200);
  OnFwd(OUT_B);
  while (true)
  {
    if (SENSOR_2 == 1)
    {
      Off(OUT_B);
//    SendMessage(11);
      break;
    }
  }
}
sub grab_open()
{
  OnRev(OUT_A); Wait(30);
```

```
  Off(OUT_A); Wait(50);
}
sub grab_close()
{
  OnFwd(OUT_A); Wait(30);
  Off(OUT_A); Wait(50);
}
task main()
{
  SetSensor(SENSOR_2,SENSOR_TOUCH);
  SetSensor(SENSOR_3,SENSOR_ROTATION);
  SetSensorMode(SENSOR_3, SENSOR_MODE_ROTATION);
  SetSensorMode(SENSOR_3, SENSOR_MODE_ROTATION);
  ClearSensor(SENSOR_3);
  while (true)
  {
    ClearMessage();
    until (Message() != 0);
    if (Message() == 1)
    {
      while (true)
      {
        ClearMessage();
        until (Message() != 0);
        if (Message() == 10)
        {
          turntable_ccw();
          SendMessage(11);
        }
        if (Message() == 12)
        {
          grab_close();
          SendMessage(13);
        }
        if (Message() == 14)
        {
          turntable_cw();
          turn_wrist();
          turntable_home();
          SendMessage(15);
```

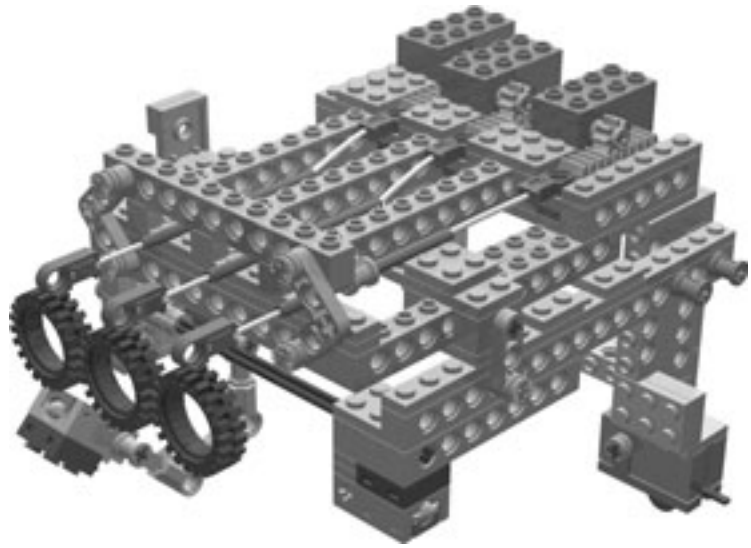
```
        }
        if (Message() == 16)
        {
          grab_open();
        }
      }
    }
    if (Message() == 2)
    {
       StopAllTasks();
    }
  }
}
```

Troubleshooting the Rotation Sensors

When programming CyberArm IV, the largest problem occurs with the rotation sensors. Occasionally while in development, the rotation sensors would misread their value seemingly without reason. As a result, the arm location would slip sometimes, thus rendering the entire sensor feedback system useless. I thought that perhaps the sensors had an effective speed range, outside of which they would not register correctly. To test this theory, I built a simple simulator for investigative purposes. As you know, the rotation sensor has 16 steps for one full rotation for the axle. Consequently, 1 step equals 22.5 degrees. Since I had first put the rotation sensor on the 24T gear, it turned very slowly. I had first written my trial NQC programs with a low rotation threshold value (under 5 or 6). However, like in the case of a turntable, there are many slips. I calculated the speed of this segment of the arm with the accuracy of measurement, ignoring the load. A turntable has 56T and the specification of a geared motor is 350RPM. I thought the gear ratio should be is 1/(5*24*2.3)=1/276.

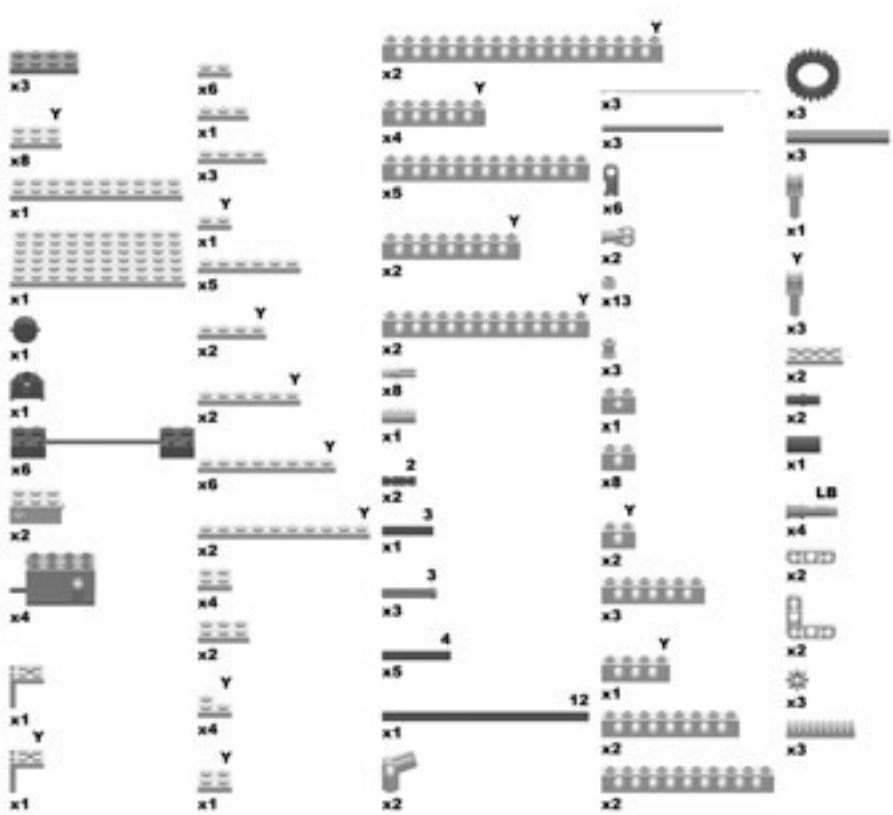
In this case, the length of the arm from the center of the turntable was 300mm, which would make it 50.7mm per step of the rotation sensor. This means that for every step that the rotation sensor registered, the arm would move 50.7 (roughly 2 inches). This was much too imprecise. The reason for the slippage is the rotation sensor turned too slowly. Setting the rotation sensor on the 40T gear, I calculated a distance of 2.1mm (about 3/32 inch) per step. I tested and confirmed this on the simulator. Of course, I could not ignore gear backlash and I had to rebuild the sensor mechanism for the arm. In the end, the improvement in accuracy was remarkable, and I got a large arm that has powerful yet sensitive movement.

Building the Power Glove



Bill of Materials

These are the parts you will need to build the Power Glove as shown:

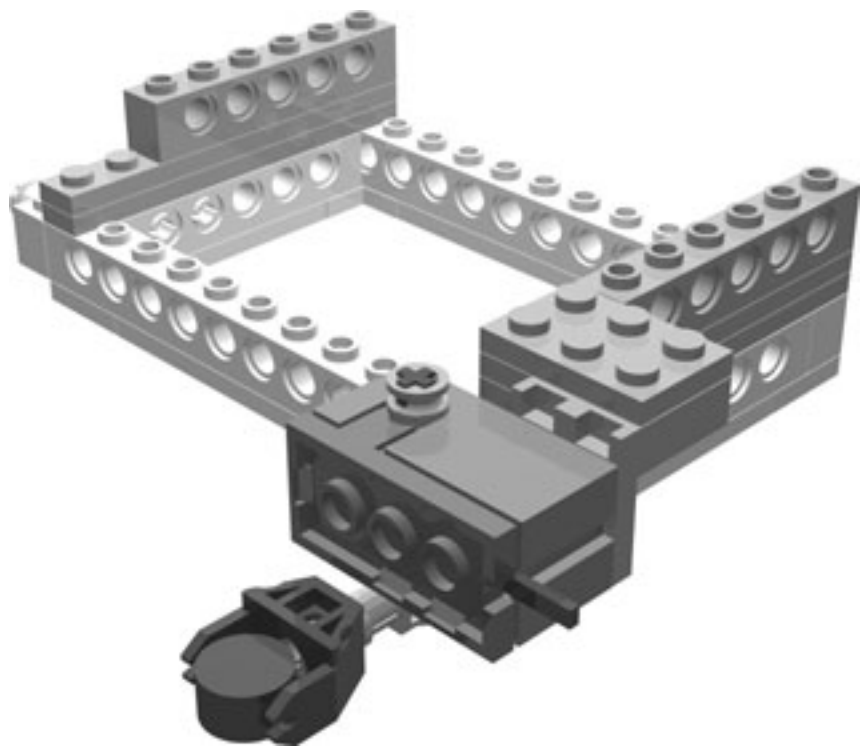


Introduction

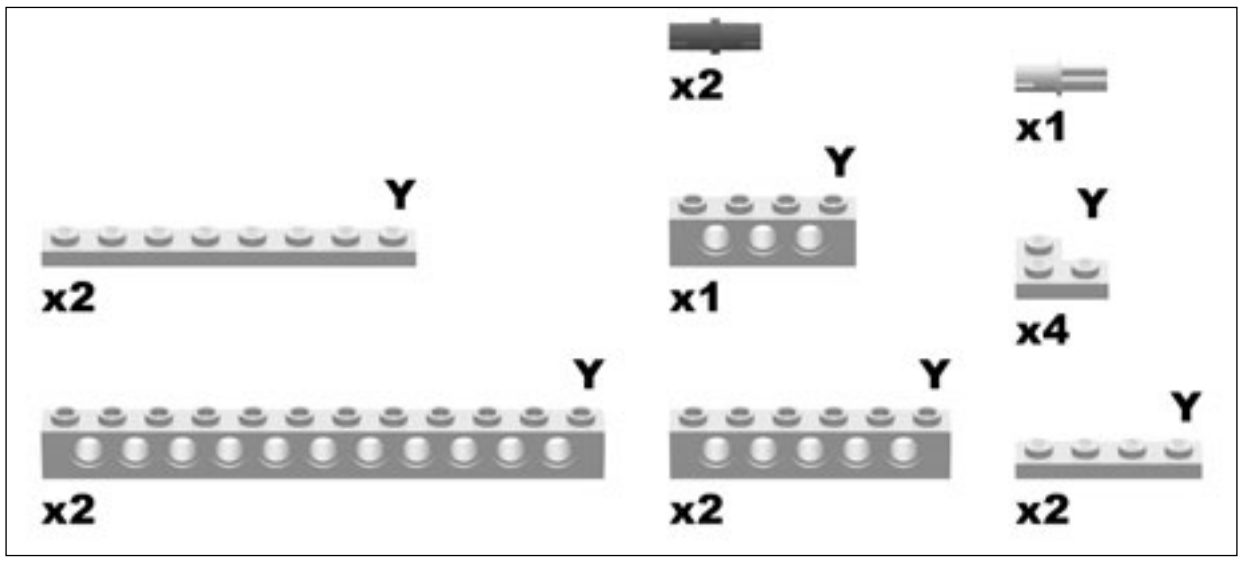
One of the real attractions of the robotic arm is real-time operation. CyberArm IV requires at least six switches to operate correctly. Because the LEGO MINDSTORMS remote controller is not capable of controlling two RCXs at the same time, I have used two 9V Battery Boxes (six AA batteries each) for testing. I added six pole reverser switches to the battery boxes when I was testing the arm and realized that even with two hands, I could not operate the remote control and the six switches. Thus, the Power Glove was born. This Power Glove is a luxury because it requires two RCXs, four rotation sensors, and two touch sensors for a remote control.

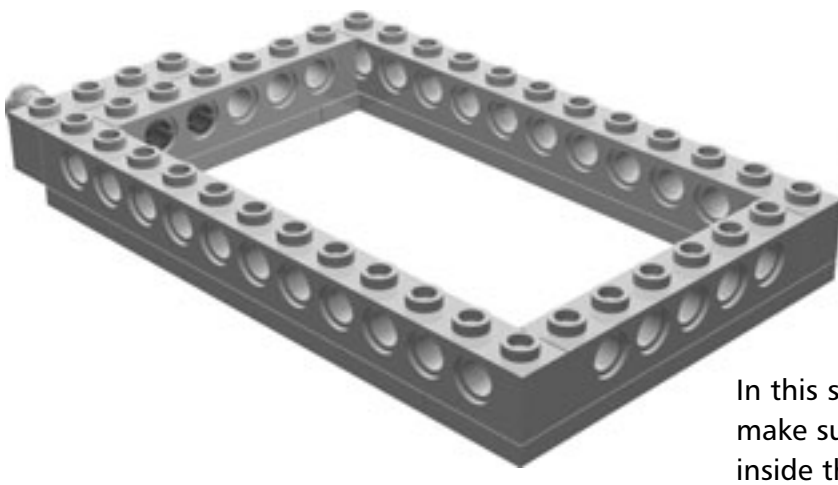
I wanted to design the Power Glove so that the turning of my wrist would activate a rotation sensor, and which, in turn, would be the signal to turn the turntable of the tower. I realized, however, that my entire forearm turns with my wrist. After vacillating this way and that, I found that the easiest way to compensate for this problem was to use a small pendulum (using magnet parts). When I use the Power Glove, I hang my upper arm down and raise my forearm. The sample program has a limiter that controls the operation range at each joint to avoid breaking the arm. Though it takes some practice to get used to the Power Glove, I think that anyone can become quite skillful at it. I wrote a test program based on messaging. I defined many *task()* for parallel tasking. One touch sensor is a toggle switch that opens/closes the grabber and the other makes the wrist turn (the wrist turns a little, waits, and then goes back to home position).

The Wrist



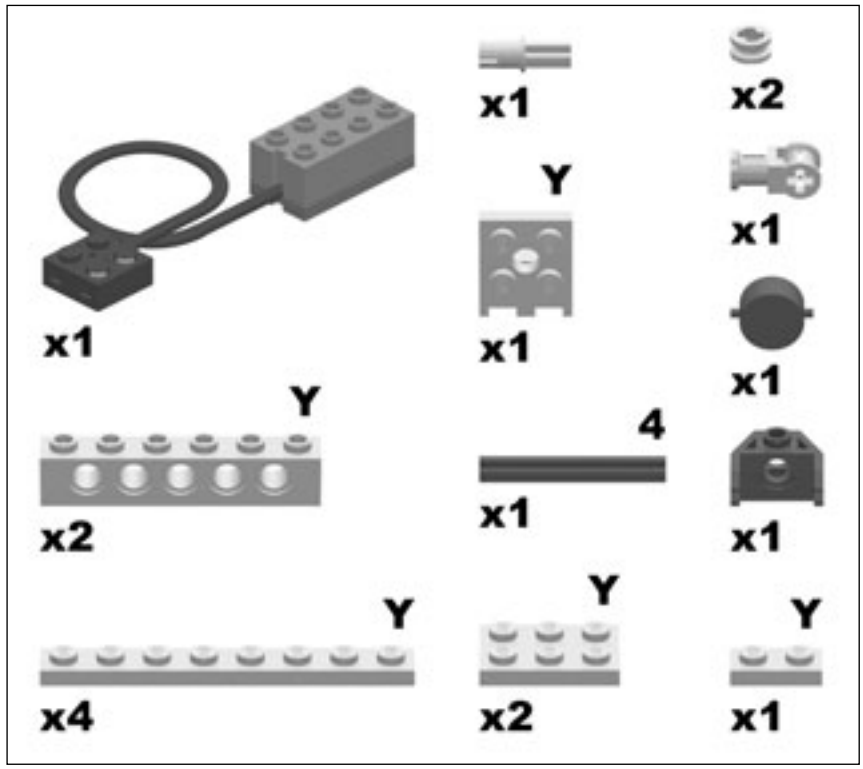
Wrist Step 0

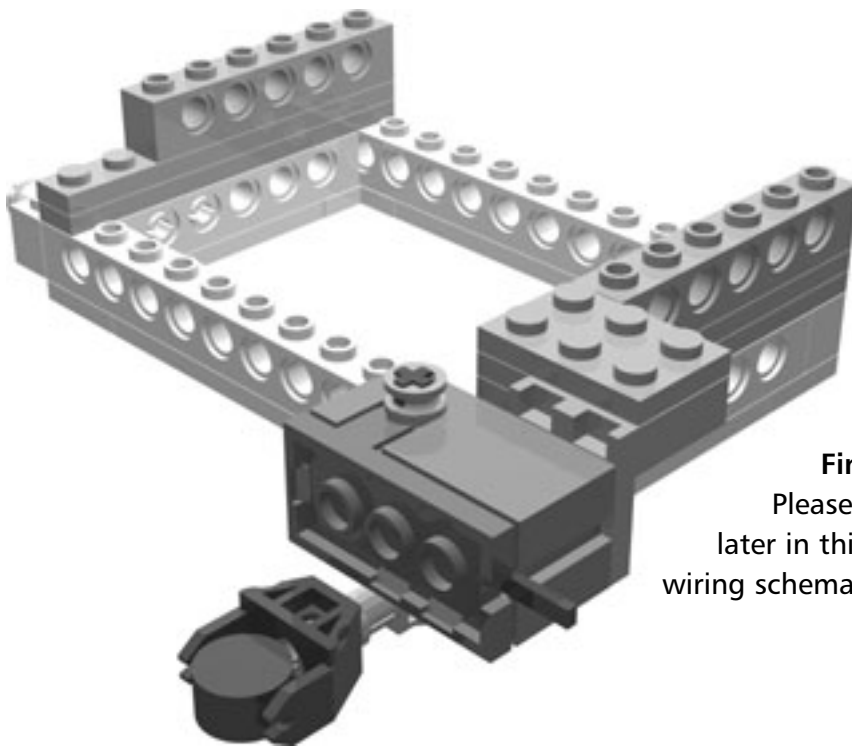




In this step, you should check to make sure that your hand fits inside the frame. If not, you should increase the size of the rectangle.

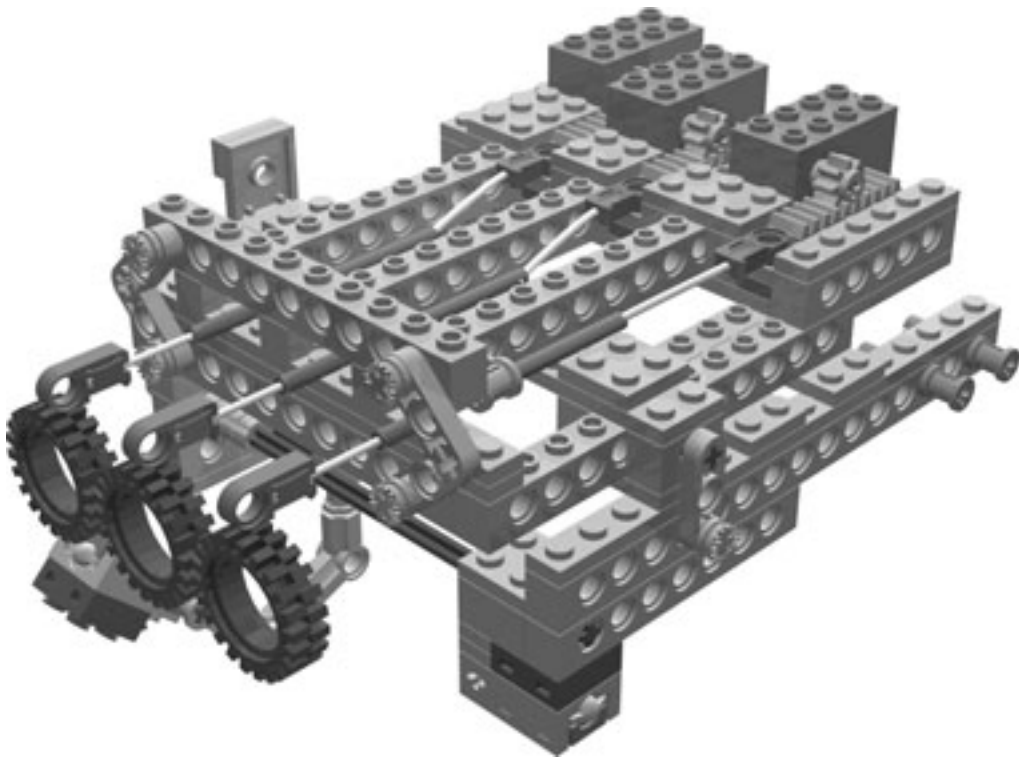
Wrist Step 1





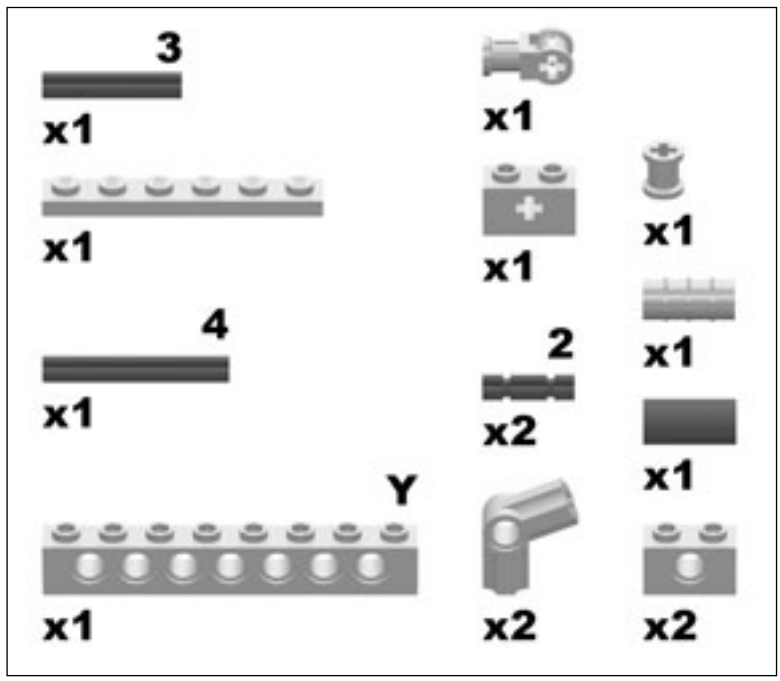
The rotation sensor added in this step will be attached to Input Port 1 of RCX 3 after completing **Final Power Glove Step 0**. Please refer to Figure 7.14 found later in this chapter for a detailed wiring schematic.

The Hand



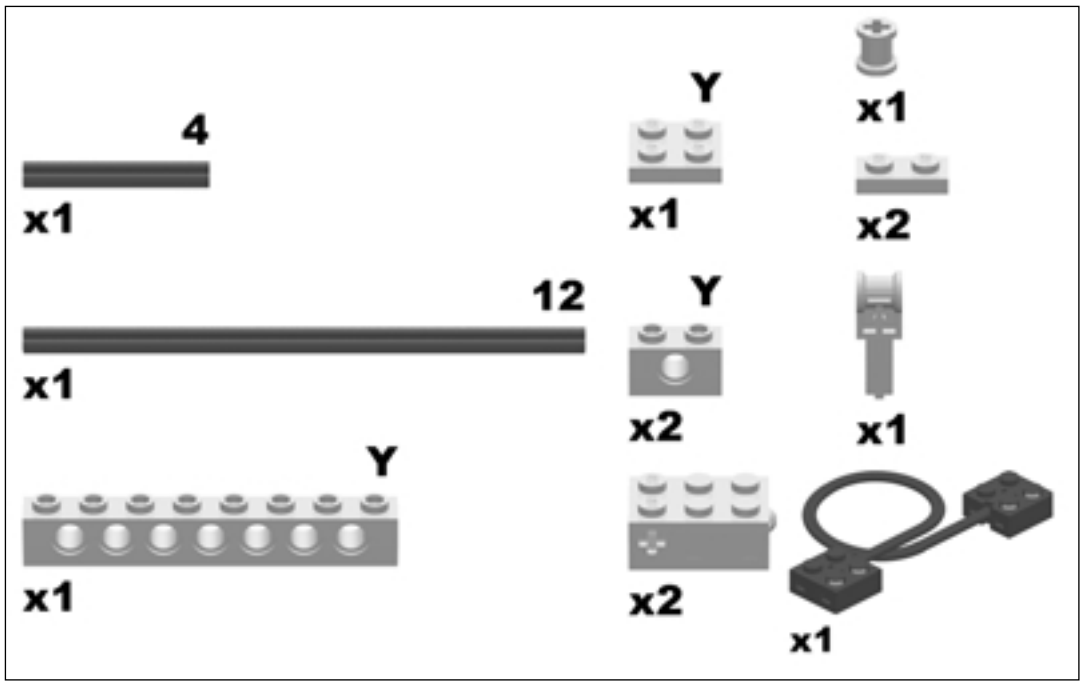
You should be aware the building instructions for the Hand sub-assembly were created for individuals who are going to use the Power Glove with their right hand. If you are left-handed, you should swap the orientation of these elements for your fingers.

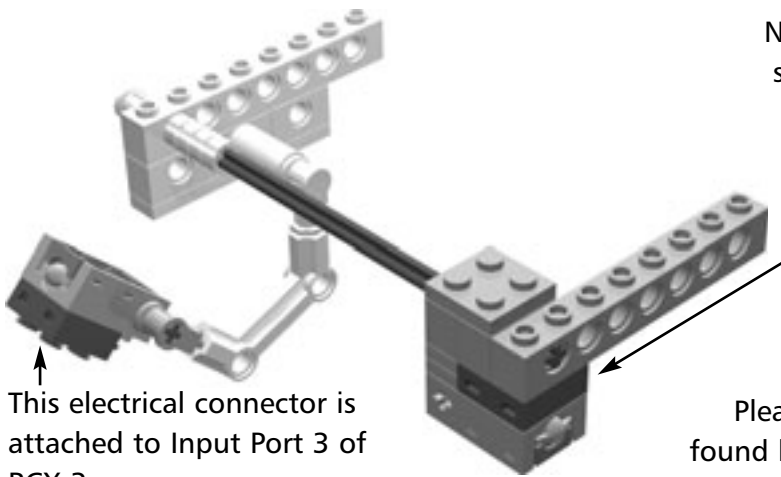
Hand Step 0



This is the support structure for the toggle switch. You will operate this element using your little finger.

Hand Step 1





This electrical connector is attached to Input Port 3 of RCX 3.

Next, create the toggle switch for your thumb.

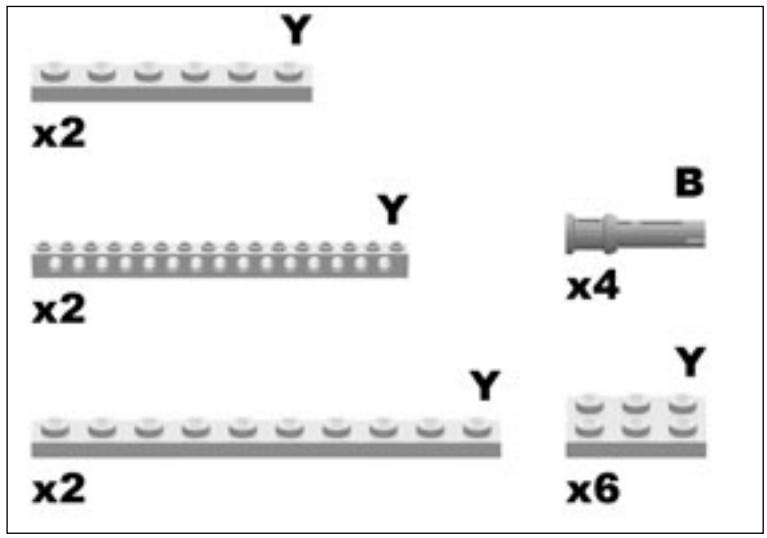
The electrical connector added in this step will be attached to Input Port 2 of RCX 3 after completing **Final Power Glove Step 0**.

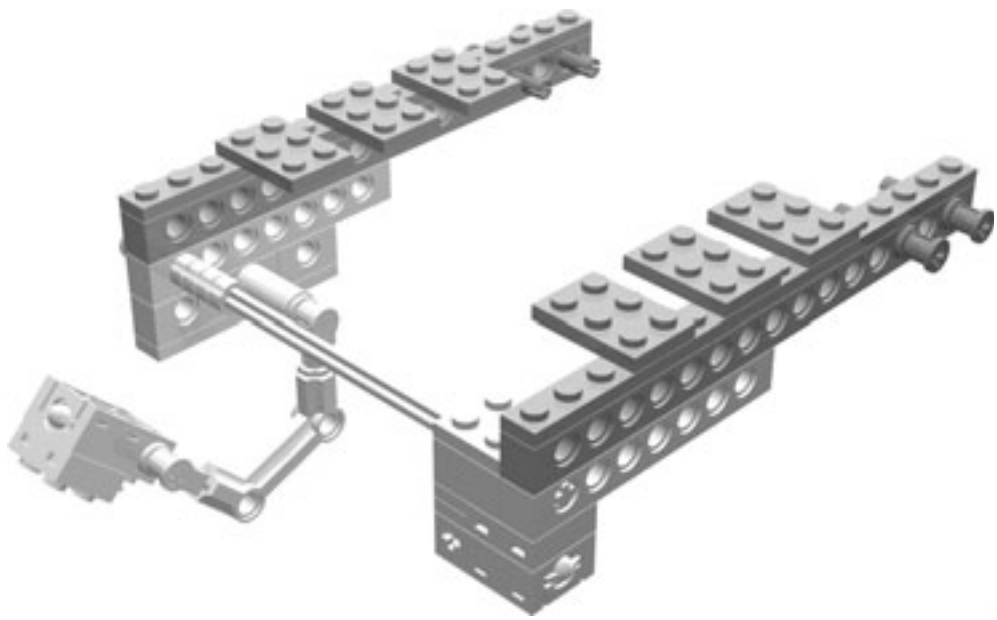
Please refer to Figure 7.14 found later in this chapter for a detailed wiring schematic.

NOTE

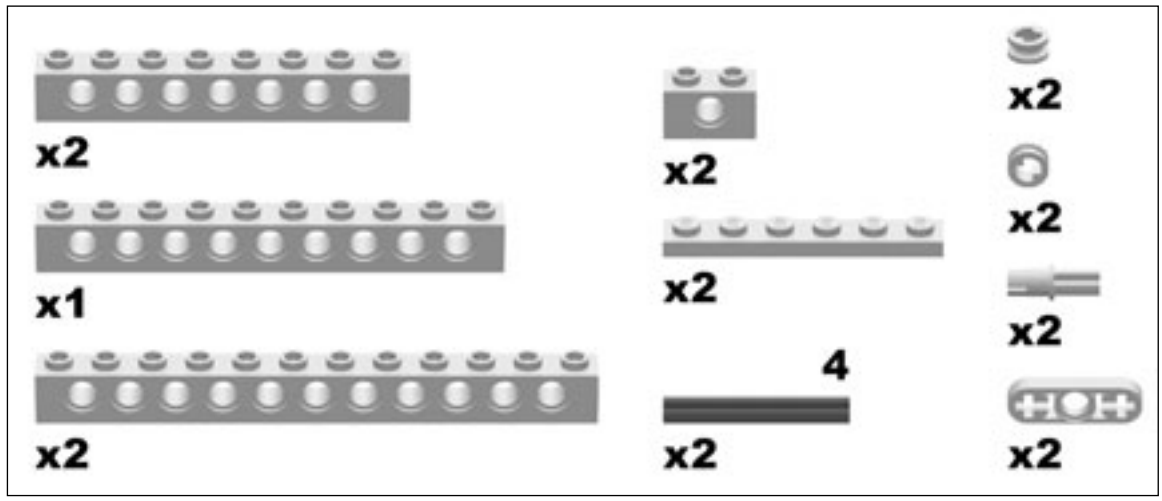
Remember: If you are left-handed, you will want to build Hand Step 1 so that the toggle switch for the thumb is in the proper position.

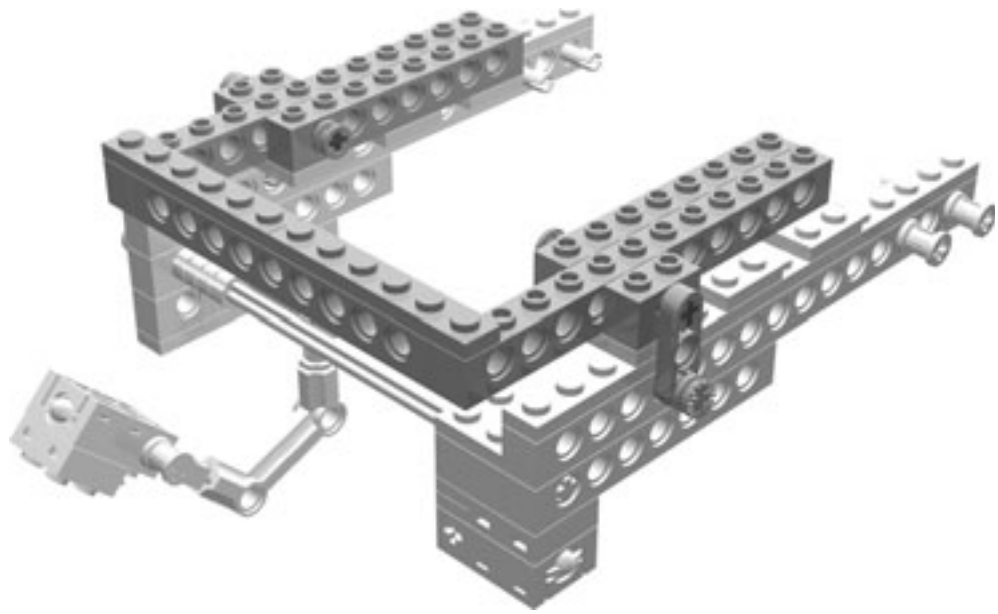
Hand Step 2



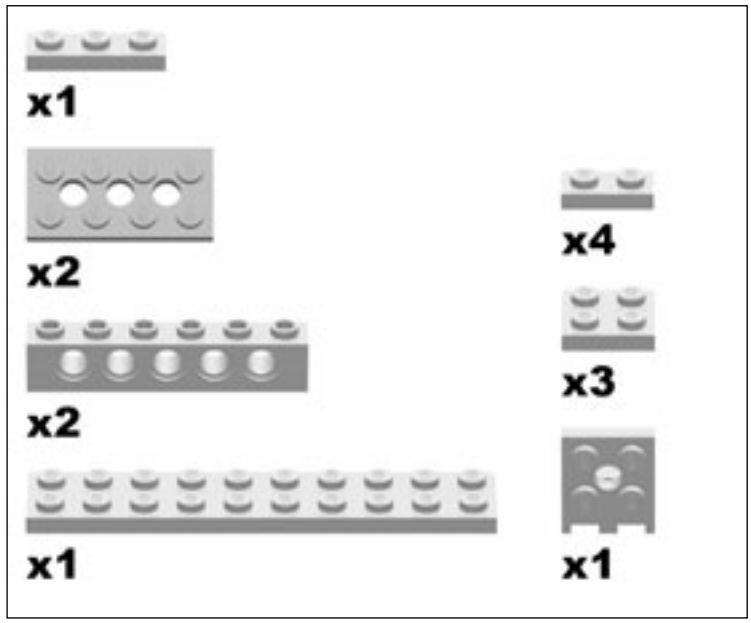


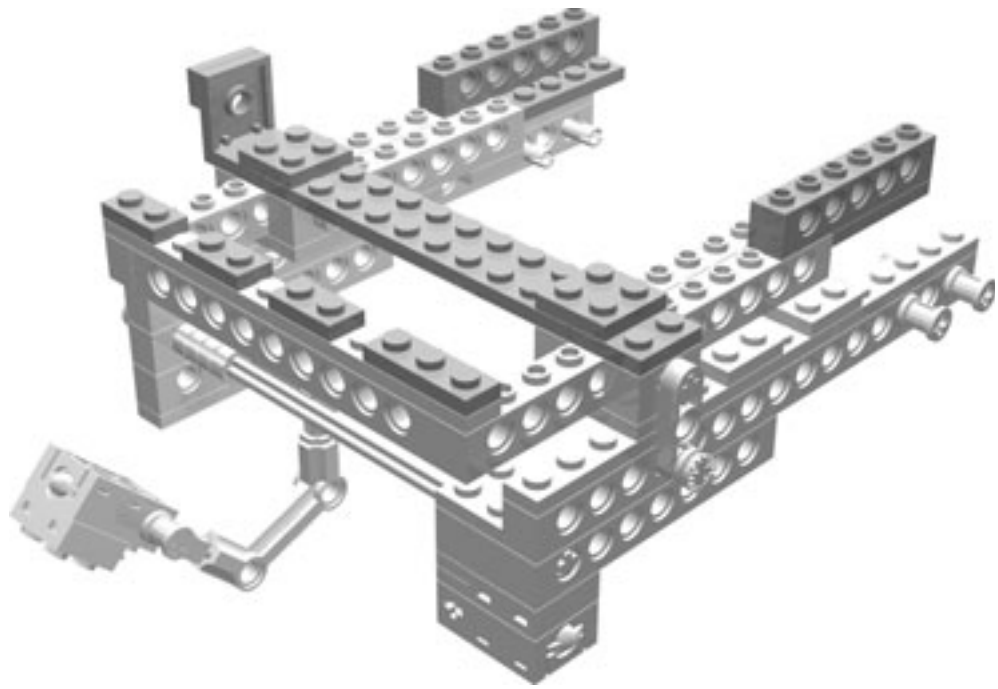
Hand Step 3



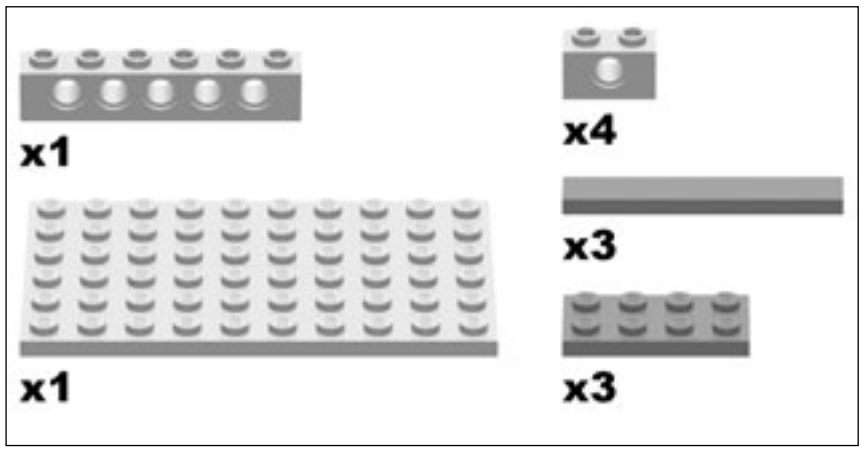


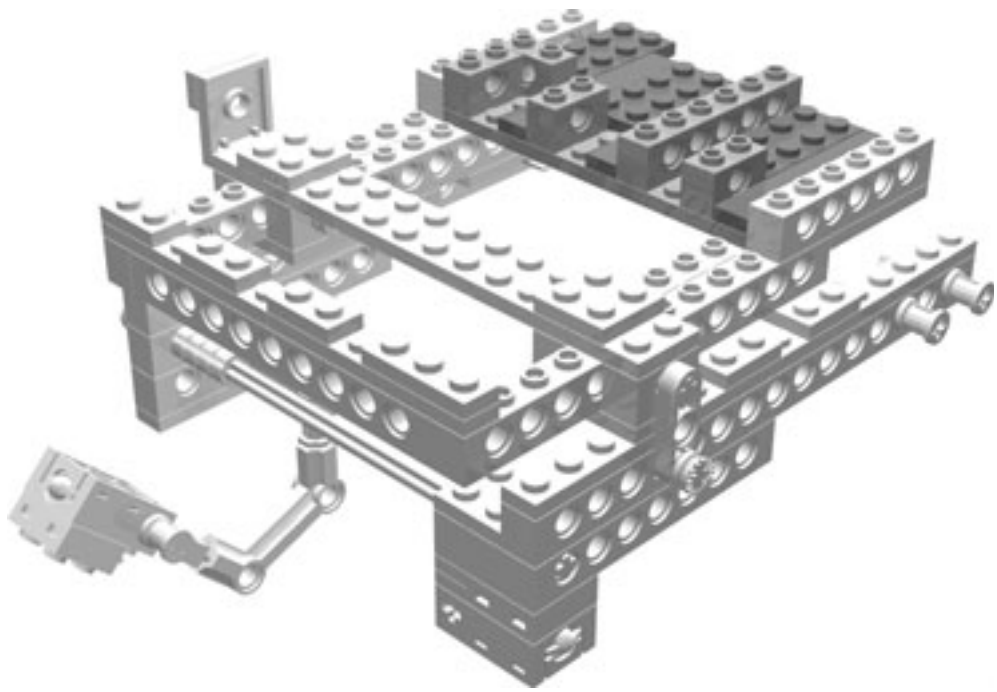
Hand Step 4



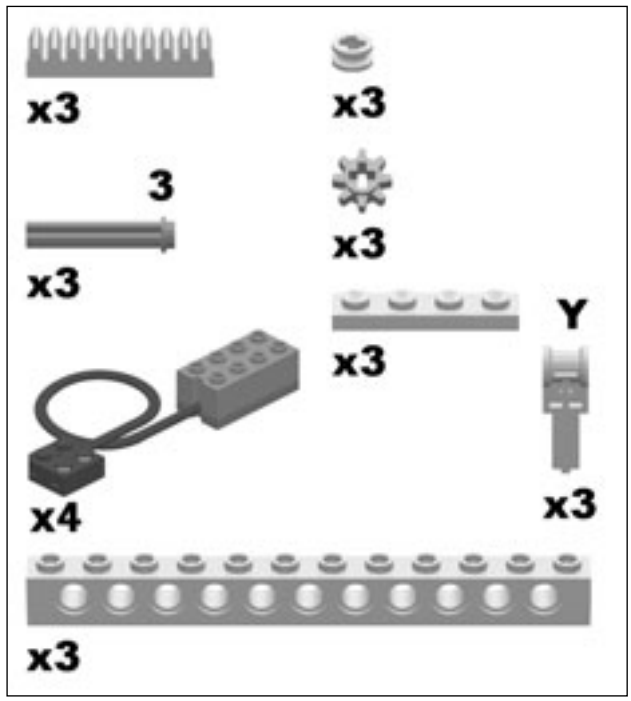


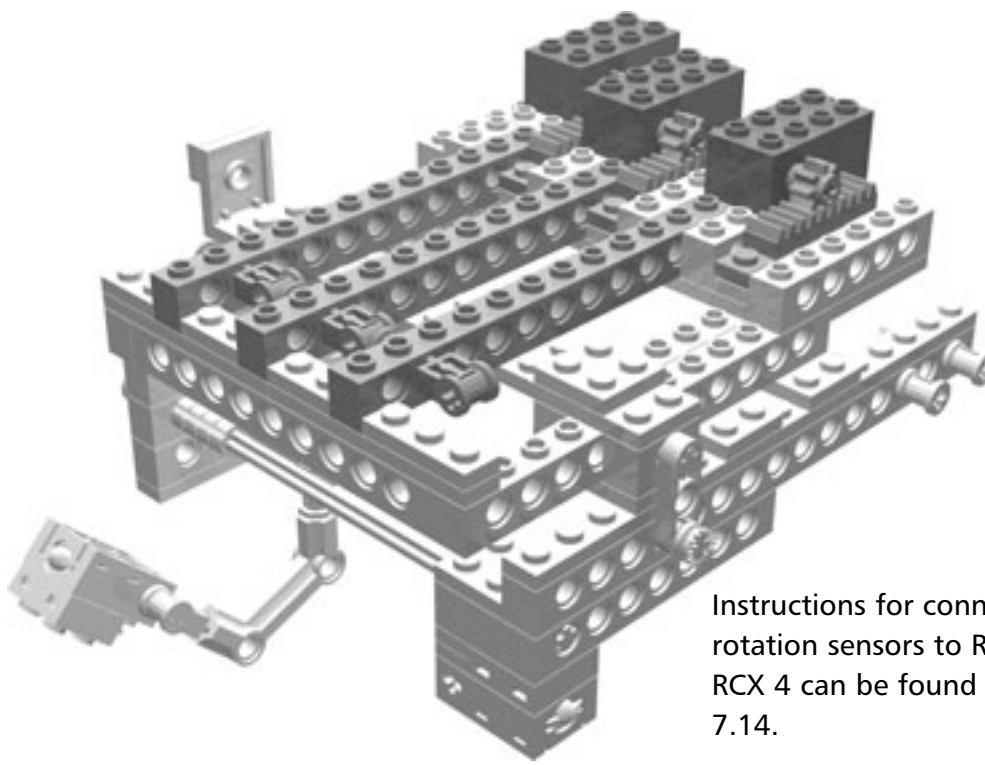
Hand Step 5





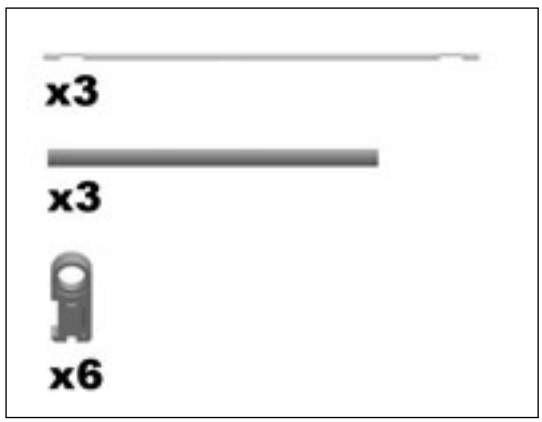
Hand Step 6

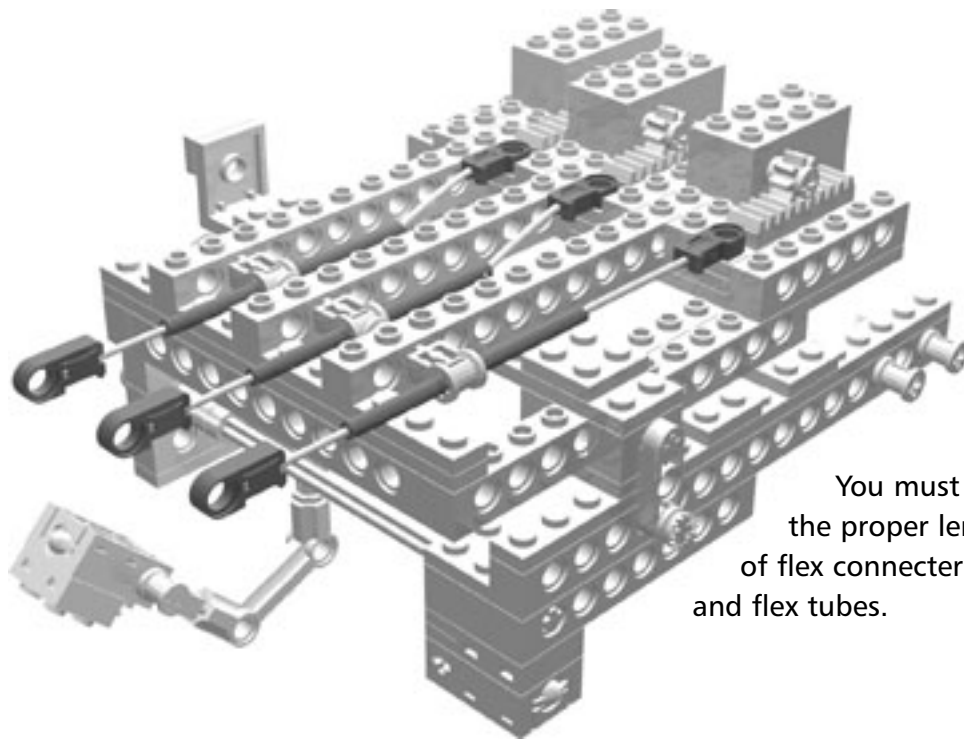




Instructions for connecting the rotation sensors to RCX 3 and RCX 4 can be found in Figure 7.14.

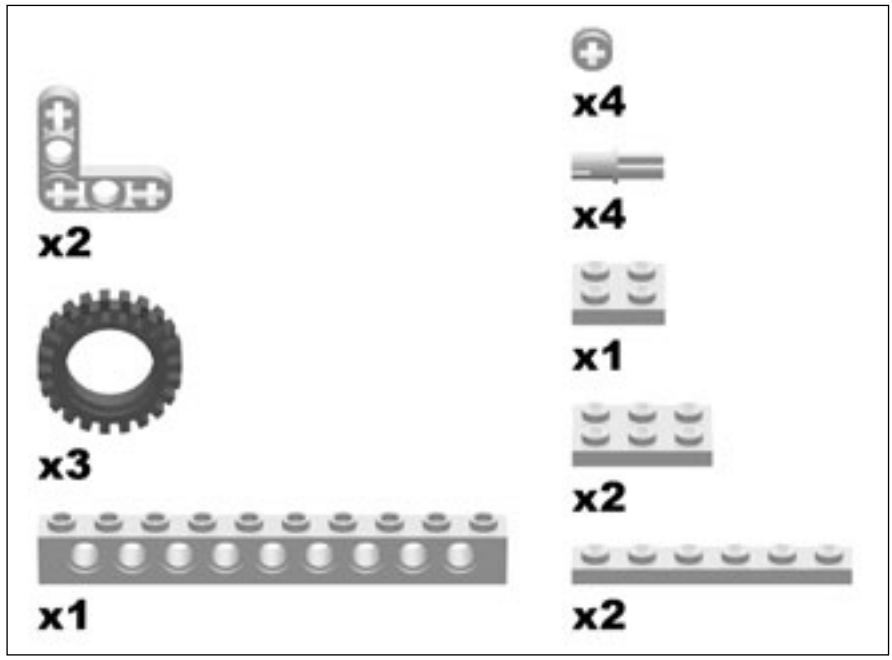
Hand Step 7



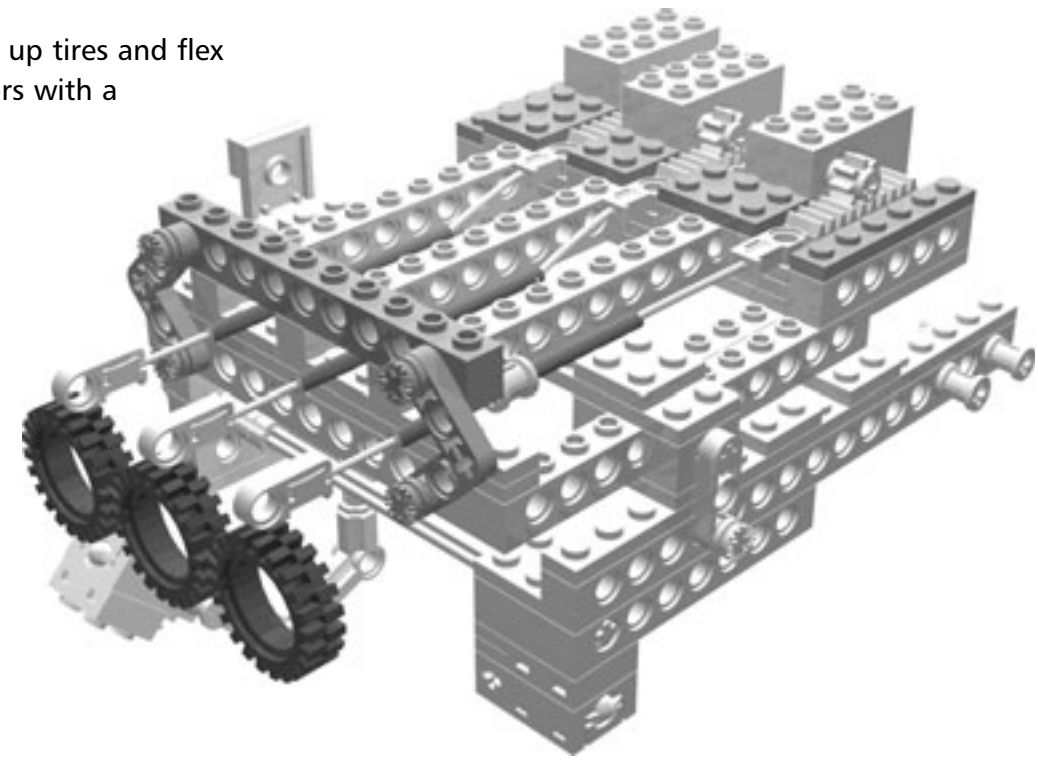


You must select the proper length of flex connecter cables and flex tubes.

Hand Step 8

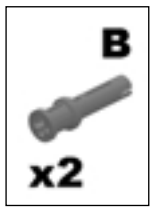


Please tie up tires and flex connectors with a thread.

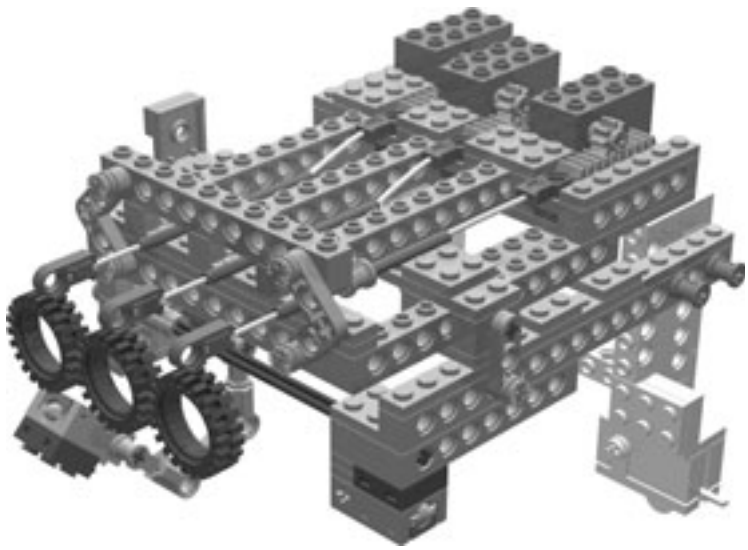


Putting It All Together

Final Step 0



Locate the Wrist sub-assembly and orient it as shown.

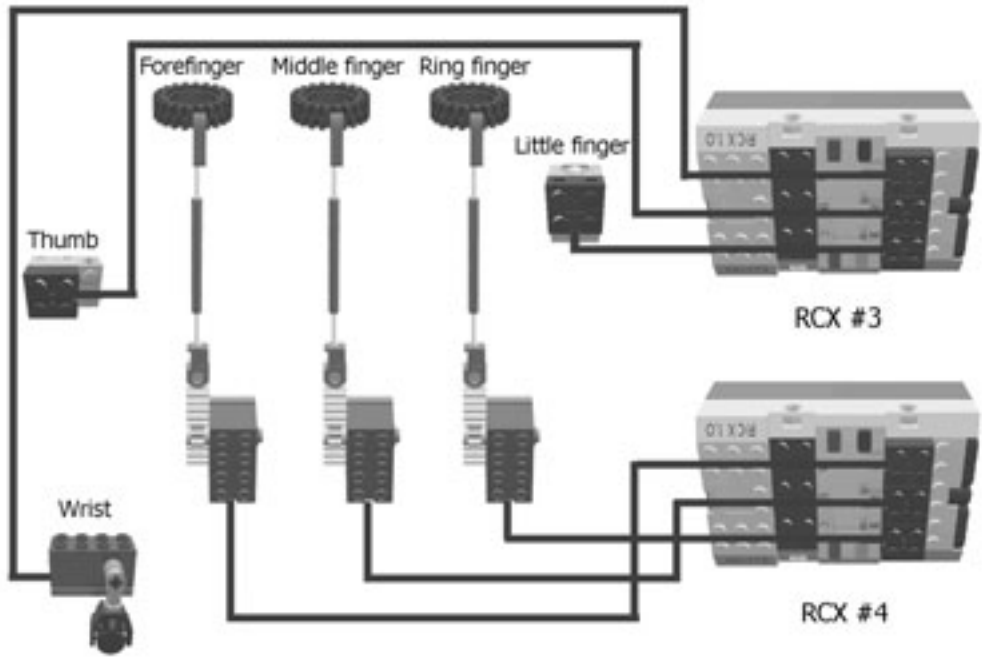


Next, connect the two assemblies using four blue TECHNIC snap connectors.

Programming the Power Glove

I created the Power Glove out of a personal preference for a more human interface instead of using a joystick or game pad. Using LEGO Flex System parts, six simultaneous operations are possible (see Figure 7.14 for a diagram of the Power Glove System). Adding to the complexity of the design, I had to adjust my glove to fit my hand (a true tailor-made task). It is very difficult to get the LEGO Flex System parts like the LEGO service pack #5118 at the present. However, this is just for reference.

Figure 7.14 Power Glove System Diagram



I created the following program in NQC to operate the Power Glove. The program for the Power Glove is fairly straightforward. However, it is comprised of two RCXs: RCX 3 (by glove1.nqc) controls RCX 2 (by recieve1.nqc) – turntable, turning wrist and grabber open/close.

- **Turntable** *IN_1* (rotation sensor) by twisting your wrist
- **Grabber (open/close)** *IN_2* (touch sensor) by pushing with your thumb
- **Grabber (turn wrist)** *IN_3* (touch sensor) by pushing with your little finger

RCX 4 (by glove2.nqc) controls RCX 1 (by recieve2.nqc) upper arm, forearm and wrist.

- **Grabber (wrist up/down)** *IN_1* (rotation sensor) by using the forefinger
- **Forearm** *IN_2* (rotation sensor) by using the middle finger
- **Upper arm** *IN_3* (rotation sensor) by using the ring finger

This program can be found on the CD-ROM that accompanies this book.

```
/*
   The power glove RCX #3: this is for the send RCX
   NQC 2.4r3
   glove1.nqc
```

```
   Hideaki Joda Yabuki, 2002, Japan
*/
int flag = 0;
task main()
{
  SetSensor(SENSOR_1,SENSOR_ROTATION);
  SetSensor(SENSOR_2,SENSOR_TOUCH);
  SetSensor(SENSOR_3,SENSOR_TOUCH);
  SetSensorMode(SENSOR_1, SENSOR_MODE_ROTATION);
  SetSensorMode(SENSOR_2, SENSOR_MODE_BOOL);
  SetSensorMode(SENSOR_3, SENSOR_MODE_BOOL);
  ClearSensor(SENSOR_1);
  Wait(100);
  while (true)
  {
    if (SENSOR_1 < -2)
    {
       SendMessage(100); // turntable_cw
    }
    if (SENSOR_1 > -1 && SENSOR_1 < 1)
    {
       SendMessage(101); // stop
    }
    if (SENSOR_1 > 2)
    {
       SendMessage(102); // turntable_ccw
    }
    if (SENSOR_2 == 1)
    {
       SendMessage(104); // Grab operation
    }
    if (SENSOR_3 == 1)
    {
       SendMessage(103); // turn_wrist
       Wait (50);
    }
  }
}
/*
   The power glove for RCX #2 on CyberArm IV
```

```
   NQC 2.4r3
   receive1.nqc
   Hideaki Joda Yabuki, 2002, Japan
*/
int g_flag = 0;
task turntable_ccw()
{
     SetPower(OUT_C,7);
     OnFwd(OUT_C);
     while (true)
     {
     if (SENSOR_3 < -70)
        {
         Off(OUT_C);
         break;
         }
     }
}
task turntable_cw()
{
     SetPower(OUT_C,7);
     OnRev(OUT_C);
     while (true)
     {
     if (SENSOR_3 > 70)
        {
         Off(OUT_C);
         break;
         }
     }
}
task turn_wrist()
{
     OnRev(OUT_B); Wait(200);
     Off(OUT_B); Wait(200);
     OnFwd(OUT_B);
     while (true)
     {
      if (SENSOR_2 == 1)
      {
```

```
          Off(OUT_B);
          break;
       }
      }
}
sub grab_open()
{
  OnRev(OUT_A); Wait(30);
  Off(OUT_A); Wait(50);
  g_flag = 0;
}
sub grab_close()
{
  OnFwd(OUT_A); Wait(30);
  Off(OUT_A); Wait(50);
  g_flag = 1;
}
task grab_operation()
{
  if (g_flag == 0)
  {
   grab_close();
  }
  else
  {
   grab_open();
  }
}
task main()
{
 SetSensor(SENSOR_2,SENSOR_TOUCH);
 SetSensor(SENSOR_3,SENSOR_ROTATION);
 SetSensorMode(SENSOR_2, SENSOR_MODE_BOOL);
 SetSensorMode(SENSOR_3, SENSOR_MODE_ROTATION);
 ClearSensor(SENSOR_3);
 Wait(100);
  while (true)
  {
    ClearMessage();
    until (Message() != 0);
```

```
    if (Message() == 100)
    {
       if (SENSOR_3 < 70)
       {
       start turntable_cw;
       }
    }
    if (Message() == 101)
    {
    Off(OUT_C);
    }
    if (Message() == 102)
    {
       if (SENSOR_3 > -70)
       {
       start turntable_ccw;
       }
    }
    if (Message() == 103)
    {
    start turn_wrist;
    }
   if (Message() == 104)
    {
    start grab_operation;
    }
  }
}
/*
   The power glove RCX #4: this is for the send RCX = glove
   NQC 2.4r3
   glove2.nqc
   Hideaki Joda Yabuki, 2002, Japan
*/
int flag = 0;
task main()
{
  SetSensor(SENSOR_1,SENSOR_ROTATION);
  SetSensor(SENSOR_2,SENSOR_ROTATION);
  SetSensor(SENSOR_3,SENSOR_ROTATION);
```

```
SetSensorMode(SENSOR_1, SENSOR_MODE_ROTATION);
SetSensorMode(SENSOR_2, SENSOR_MODE_ROTATION);
SetSensorMode(SENSOR_3, SENSOR_MODE_ROTATION);
ClearSensor(SENSOR_1);
ClearSensor(SENSOR_2);
ClearSensor(SENSOR_3);
Wait(100);
while (true)
{
  if (SENSOR_3 < -3)
  {
     SendMessage(200); // upperarm_up
  }
  if (SENSOR_3 > -2 && SENSOR_3 < 2)
  {
     SendMessage(201); // stop
  }
  if (SENSOR_3 > 3)
  {
     SendMessage(202); // upperarm_down
  }
  if (SENSOR_2 < -3)
  {
     SendMessage(203); // forearm_up
  }
  if (SENSOR_2 > -2 && SENSOR_2 < 2)
  {
     SendMessage(204); // stop
  }
  if (SENSOR_2 > 3)
  {
     SendMessage(205); // forearm_down
  }
  if (SENSOR_1 < -3)
  {
     SendMessage(206); // wrist_up
  }
  if (SENSOR_1 > -2 && SENSOR_1 < 2)
  {
     SendMessage(207); // stop
```

```
    }
    if (SENSOR_1 > 3)
    {
       SendMessage(208); // wrist_down
    }
  }
}
/*
   The power glove for RCX #1 on CyberArm IV
   NQC 2.4r3
   receive2.nqc
   Hideaki Joda Yabuki, 2002, Japan
*/
int g_flag = 0;
task upperarm_up()
{
  SetPower(OUT_A,7);
  OnFwd(OUT_A);
  while (true)
  {
    if (SENSOR_1 > 110)
    {
      Off(OUT_A);
      break;
    }
  }
}
task upperarm_down()
{
  SetPower(OUT_A,7);
  OnRev(OUT_A);
  while (true)
  {
    if (SENSOR_1 < 0)
    {
      Off(OUT_A);
      break;
    }
  }
}
```

```
task forearm_up()
{
  SetPower(OUT_B,7);
  OnFwd(OUT_B);
  while (true)
  {
    if (SENSOR_2 > 70)
    {
      Off(OUT_B);
      break;
    }
  }
}
task forearm_down()
{
  SetPower(OUT_B,7);
  OnRev(OUT_B);
  while (true)
  {
    if (SENSOR_2 < 0)
    {
      Off(OUT_B);
      break;
    }
  }
}
task wrist_up()
{
  SetPower(OUT_C,7);
  OnFwd(OUT_C);
  while (true)
  {
    if (SENSOR_3 < -50)
    {
      Off(OUT_C);
      break;
    }
  }
}
task wrist_down()
```

```
{
  SetPower(OUT_C,7);
  OnRev(OUT_C);
  while (true)
  {
    if (SENSOR_3 > 0)
    {
      Off(OUT_C);
      break;
    }
  }
}
task main()
{
  SetSensor(SENSOR_1,SENSOR_ROTATION);
  SetSensor(SENSOR_2,SENSOR_ROTATION);
  SetSensor(SENSOR_3,SENSOR_ROTATION);
  SetSensorMode(SENSOR_1, SENSOR_MODE_ROTATION);
  SetSensorMode(SENSOR_2, SENSOR_MODE_ROTATION);
  SetSensorMode(SENSOR_3, SENSOR_MODE_ROTATION);
  ClearSensor(SENSOR_1);
  ClearSensor(SENSOR_2);
  ClearSensor(SENSOR_3);
  Wait(100);
  while (true)
  {
    ClearMessage();
    until (Message() != 0);
    if (Message() == 200)
    {
       if (SENSOR_1 < 110)
       {
       start upperarm_up;
       }
    }
    if (Message() == 201)
    {
    Off(OUT_A);
    }
    if (Message() == 202)
    {
       if (SENSOR_1 > 0)
```

```
        {
        start upperarm_down;
        }
    }
    if (Message() == 203)
    {
        if (SENSOR_2 < 70)
        {
        start forearm_up;
        }
    }
    if (Message() == 204)
    {
    Off(OUT_B);
    }
    if (Message() == 205)
    {
        if (SENSOR_2 > 0)
        {
        start forearm_down;
        }
    }
    if (Message() == 206)
    {
        if (SENSOR_3 > -50)
        {
        start wrist_up;
        }
    }
    if (Message() == 207)
    {
    Off(OUT_C);
    }
    if (Message() == 208)
    {
        if (SENSOR_3 < 0)
        {
        start wrist_down;
        }
    }
  }
}
```

Summary

CyberArm IV is a genuine robotic arm. Of course, it has several limitations: no variable speed control like an expensive servomotor, less precision because of gear backlash, and so on. The possibilities inherent in the arm, however, have really gone beyond my programming abilities. Recall the degrees of freedom shown in Figure 7.4 that shows the working capabilities of the arm. How does one move the arm from one position to another (besides the shortest path)? How can one control the posture of the grabber so that it maintains a level surface of the water in a cup it's holding? Another possibility exists in creating even a complicated leg (even though it wouldn't walk) without using a linkage mechanism of Chebyshev. I started to learn true robotic concepts like coordinate transformation and kinematics with mathematics and physics while learning about the full potential of CyberArm IV.

A quarter of a century has passed since the LEGO Group launched "Expert Builder sets" for children to discover mechanical operation principles and mechanisms. Amazingly, most of these parts are still included as part of today's TECHNIC set. The endurance and lifespan of LEGO is amazing in the bustling world of toys. LEGO MINDSTORMS, as a high-tech toy, broke the perceived age limit for people playing with LEGOs. Sadly, few sophisticated people have the well-balanced knowledge needed for inventing, building, and programming robotic LEGO MINDSTORMS creations. I wish to offer my CyberArm IV as a platform for all who love LEGO MINDSTORMS. Often, master builders work alone through their entire creation process. However, in the age of the Internet, I have been thinking about the possibility of collaborating with others on a LEGO project, with persons who have abilities in concept, system design, mechanics, electronics, and programming. My CyberArm IV is a generic robotic arm that is incomplete and full of unexplored possibilities. My hope is that readers who build copies of CyberArm IV will expand its capabilities. It's your turn now! Your success in exploring the latent abilities of the CyberArm IV will depend on your unceasing curiosity and creativity, but I fully expect you to succeed. The immense LEGO MINDSTORMS universe is right in front of you.

"A journey of a thousand miles begins with a single step." Lao Tzu (the 6th century B.C.)