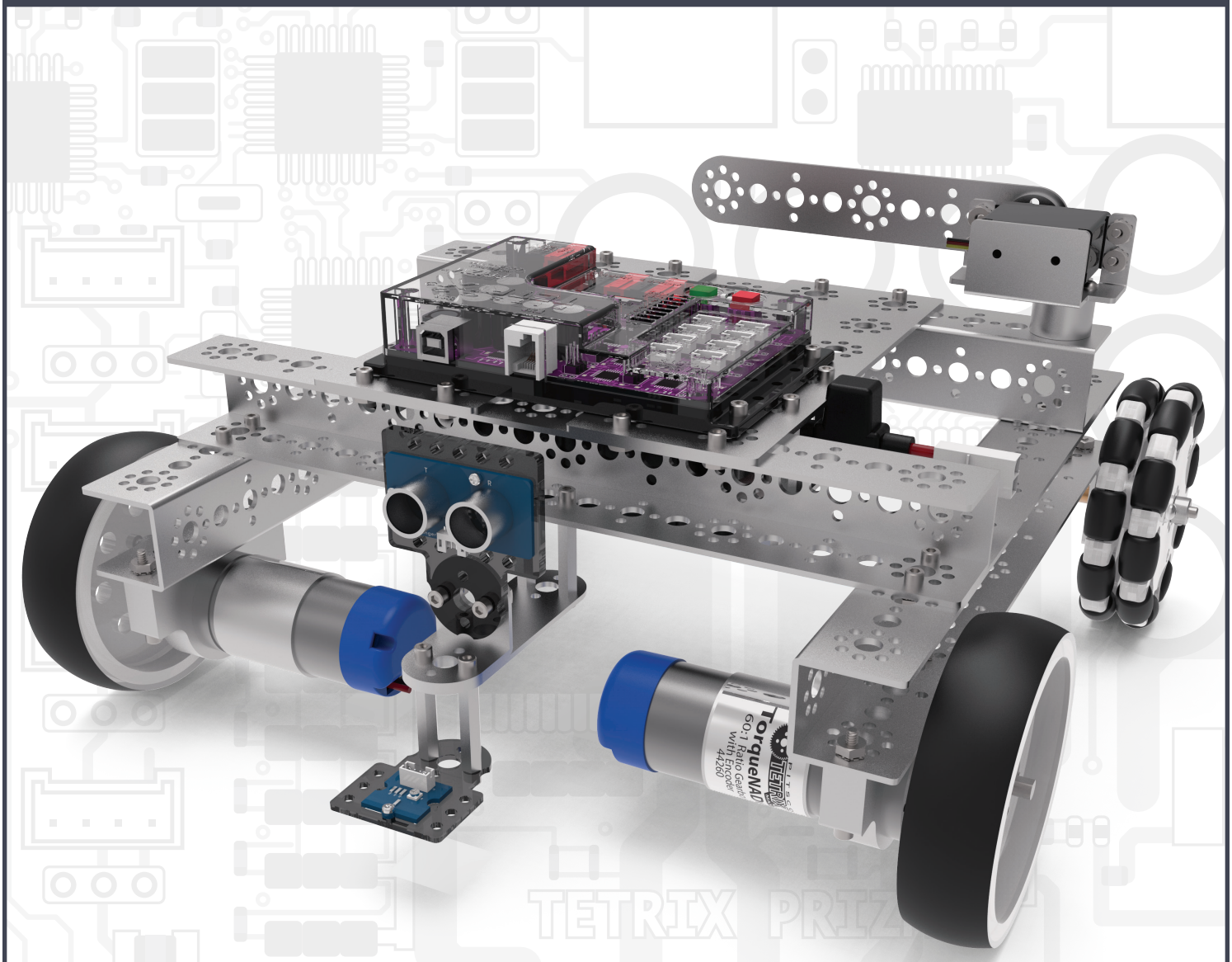
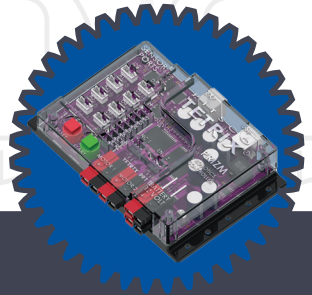
PITSCO

TETRIX® MAX





TETRIX® PRIZM® Robotics Controller Programming Guide

PITSCO EDUCATION

44716

Content advising by Paul Uttley, Pamela Scifers, Tim Lankford, Nevin Jones, and Bill Holden.

Build creation and *SolidWorks*® *Composer*™ and *KeyShot*® renderings by Tim Lankford, Brian Eckelberry, and Jason Redd. Desktop publishing by Todd McGeorge.

V1.2
08/18
44716

Check Pitsco.com for PDF updates of this guide.

WARNING

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

Table of Contents

Welcome, Coders, Roboticists, Engineers, Students, Teachers, Mentors, and Coaches,

Pitsco Education is pleased to bring you the *TETRIX® PRIZM® Robotics Controller Programming Guide* – an exciting and progressive series of activities that teaches the essentials of learning to program your TETRIX MAX creations using the PRIZM controller and the *Arduino Software (IDE)*.

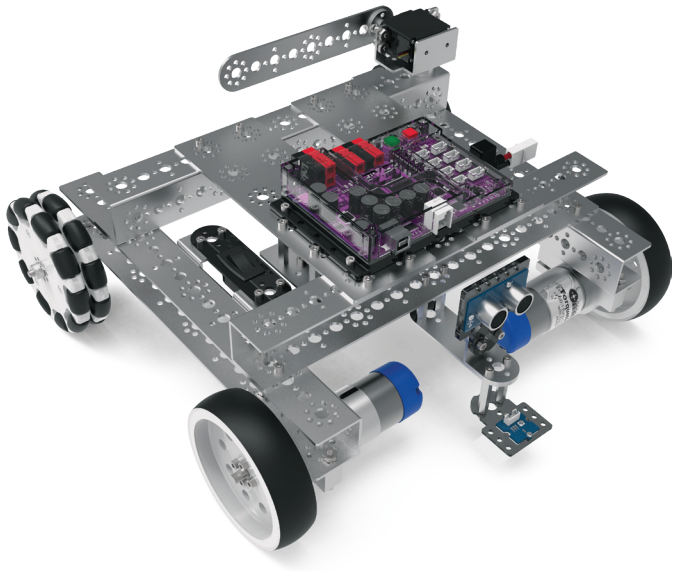
This programming guide offers a valuable tool for teaching students and teachers how to use the PRIZM controller (the brain) and the TETRIX MAX system to build and code smart, precise robots that are as real-world as it gets. The guide comes with five getting started activities, step-by-step building instructions for creating a TaskBot, 10 complete TaskBot-oriented lessons, and Hacking the Code extension activities, making it a great tool for exploring the functionality of the PRIZM controller, TETRIX hardware components, and *Arduino Software (IDE)* and offering students a great foundation to build on.

By combining the plug-and-play PRIZM controller with the intuitive MAX building system and an easy-to-use, text-based software environment, this solution offers a great entry into teaching and learning through robotics. The progressive nature of the activities enables robotic creations to come to life quickly and easily, meaning students can experience instant success and focus more classroom time on problem-solving and applying their STEM knowledge.

Plus, PRIZM is not just a great tool for teaching programming. It can bring to life lessons on sensors, power, gear ratios, and more. Even the controller's clear polycarbonate shield was designed to maximize educational value – letting users see the inner architecture of the controller.

We have also included some STEM Extensions (concepts beyond the scope of this guide) that can be covered in each lesson if you choose to do so. These extensions can be incorporated if you have content knowledge of these concepts or if you can work with other teachers to integrate these concepts.

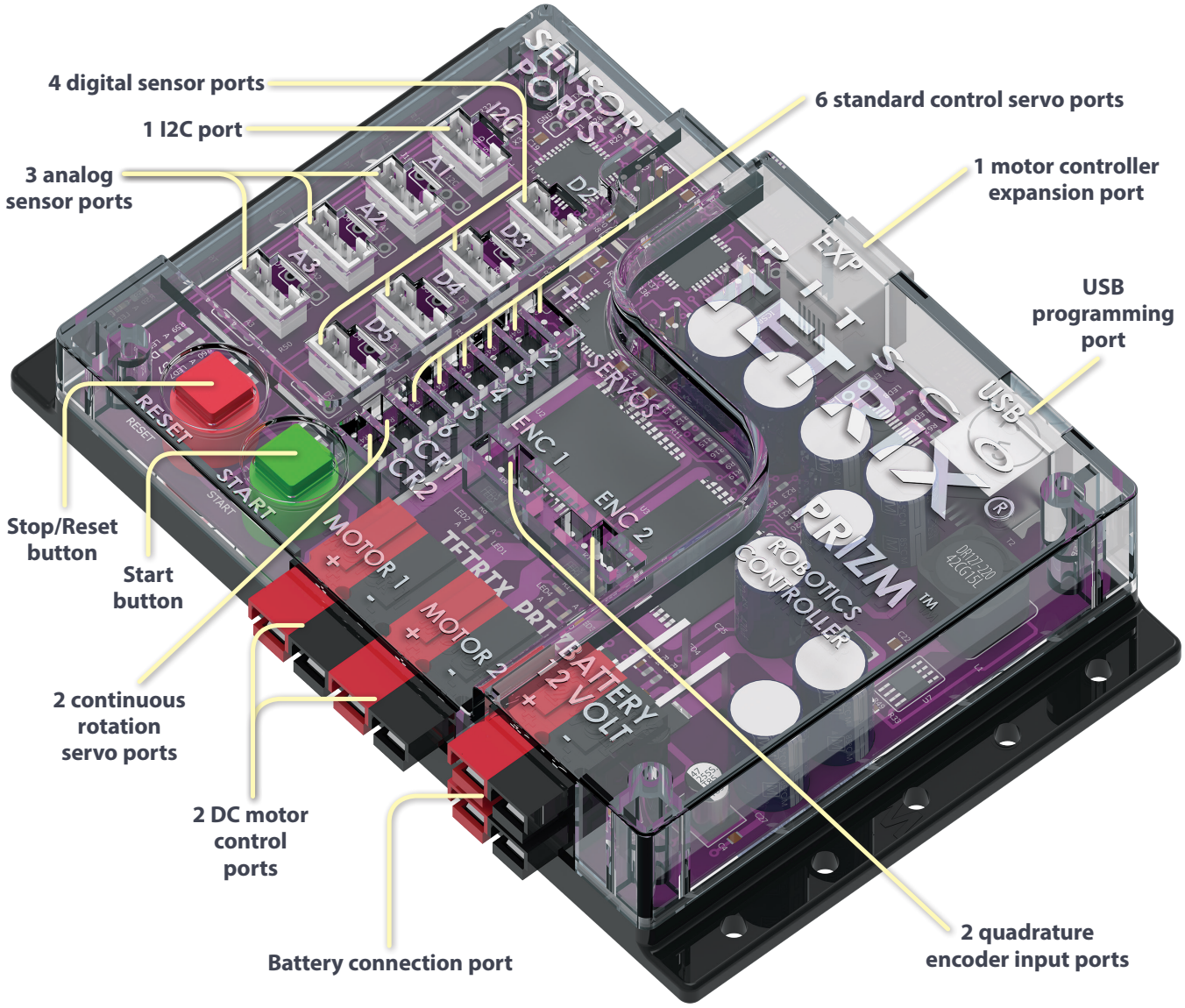
We hope this guide offers a great jumping-off point to learning with PRIZM. We cannot wait to see the innovative projects and robotic creations that result.



Happy coding, building, problem-solving, engineering, and collaborating!

PRIZM Controller Technology Overview

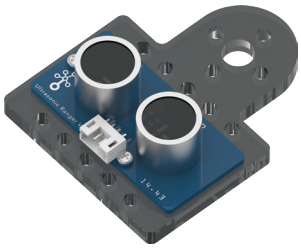
PRIZM Robotics Controller:
A programmable device that is the brain of a TETRIX MAX robot.



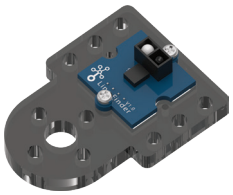
For complete, detailed specifications, please refer to the TETRIX PRIZM Robotics Controller Technical Specifications located in the appendix on page 124.

Sensor:

A device that detects and responds to surrounding environmental factors



Ultrasonic Sensor:
Enables a robot to measure distance to an object and respond to movement



Line Finder Sensor:
Enables a robot to follow a black line on a white background or vice versa

Motor:

A machine that produces motion or power for completing work

TorqueNADO® Motor:
12-volt DC motor that features 100 rpm and 700 oz-in. of torque

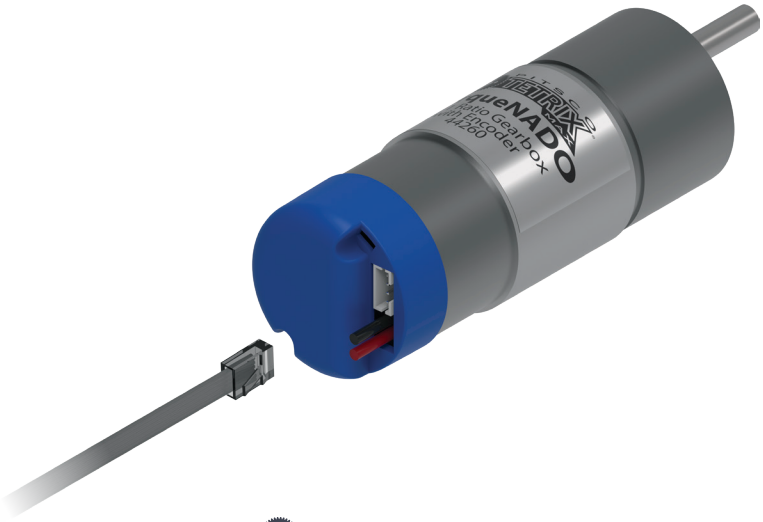
Powerpole Motor Cable:
Connects the DC motor to the power source

Standard Servo Motor:
Allows for exact positioning within a 180-degree range of motion

Continuous Rotation Servo Motor:
Allows for continuous direction of movement, both forward and backward

Encoder:

A sensing device that converts motion to an electrical signal that is read by the PRIZM controller to determine position, count, speed, or direction of a DC motor.



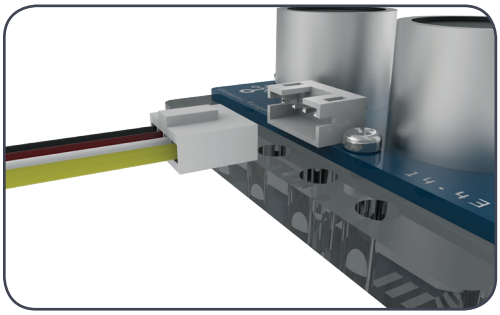
Motor Shaft Encoder:
Allows you to program the DC motors using both rotation and degrees.

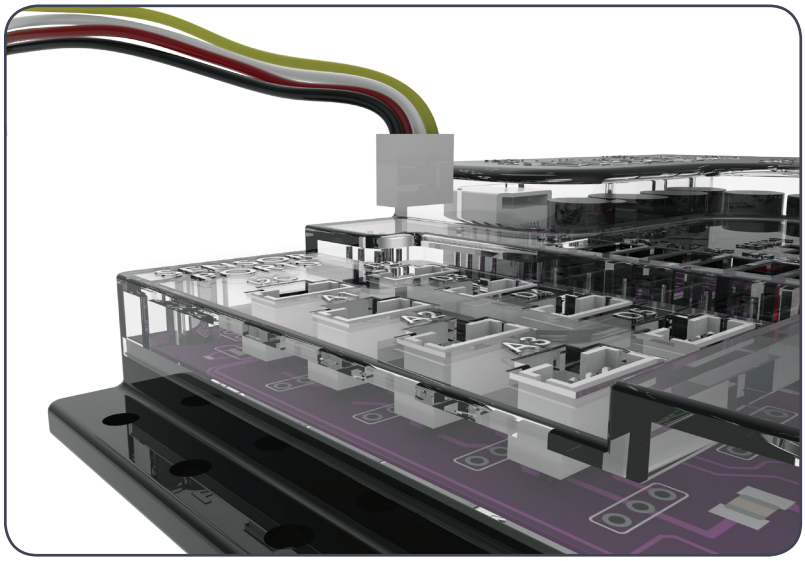
The TorqueNADO motor comes equipped with a motor shaft encoder. However, the encoder will not be used for the activities in this guide. The four-wire encoder cable can be removed.

Note: The TorqueNADO Motor (44260) is the recommended DC motor the activities and challenges in this guide. However, other 12 VDC motors can be substituted and connected to PRIZM using a TETRIX MAX Powerpole Motor Cable (41352).

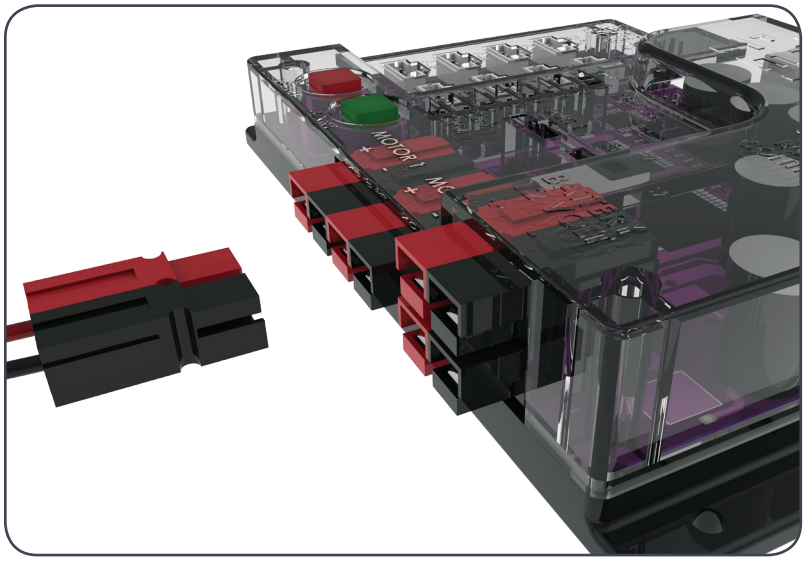
PRIZM Setup

Attaching the Sensors:
To connect a sensor to the PRIZM, plug one end of the sensor adapter wire into the sensor. Plug the other end into ports labeled D2-D5 or A1-A3.

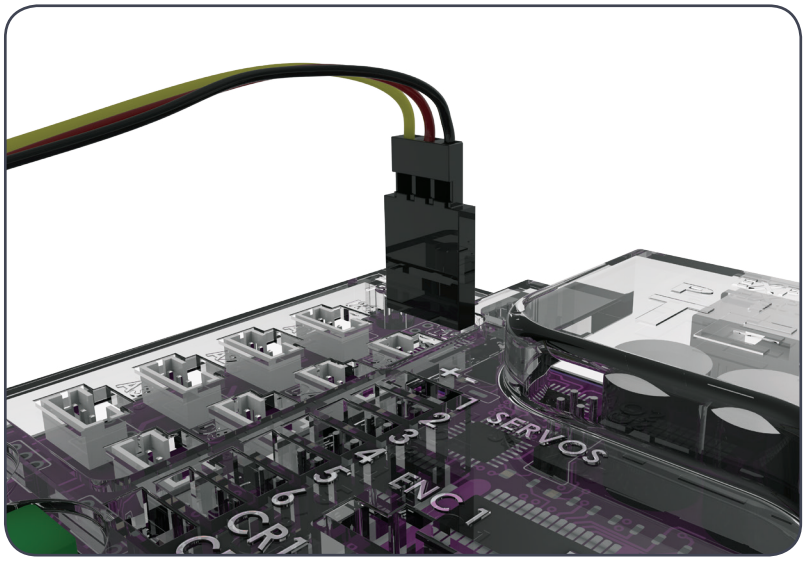




Attaching the DC Motors:
To connect a DC motor to the PRIZM, plug the black and red Powerpole connectors from the motor cable into one of the two motor ports on the PRIZM. Connectors should be plugged in RED to RED and BLACK to BLACK for proper operation.



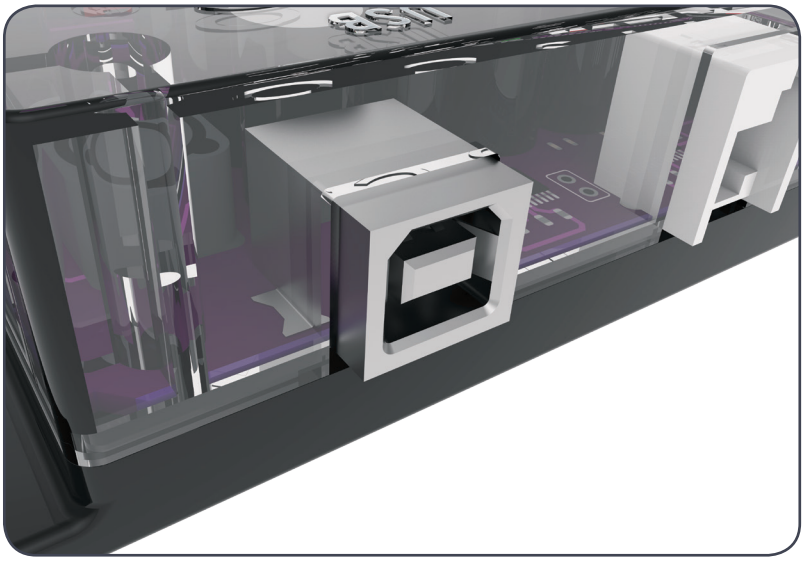
Attaching the Servo Motors:
To connect a standard servo motor to the PRIZM, plug one end of the servo wire into servo ports 1-6. To connect a continuous rotation servo motor to the PRIZM, plug one end of the servo wire into servo ports labeled CR1 or CR2.



Downloading and Uploading:
The PRIZM USB port is used for communication between PRIZM and a Windows, Mac, or Linux device.

The port enables users to download and upload data from the computer to the PRIZM controller.

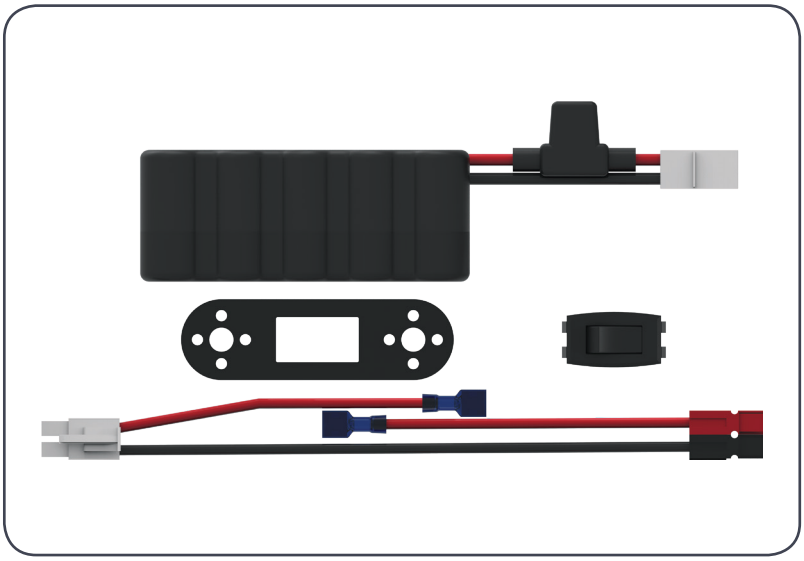
To upload a program to the PRIZM, plug the B Connector of the USB cable into the controller's USB labeled port and plug the A Connector into your device.



Powering the PRIZM:
The PRIZM controller is powered by a TETRIX 12-Volt Rechargeable NiMH Battery Pack.

Included with the PRIZM controller is an on/off power switch assembly designed to connect the battery pack connector to the PRIZM-style Powerpole battery connection port.

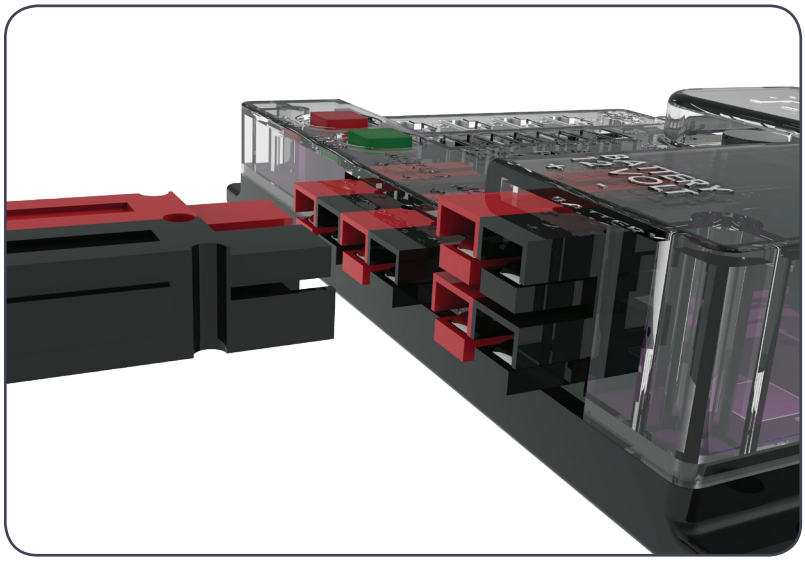
Instructions for assembling the battery and on/off kit are included in the Building the TaskBot activity in this guide.



Warning: Do not attempt to use third-party battery packs with the PRIZM controller. The TETRIX battery packs are equipped with a safety fuse and are the only packs approved for use with the system. Damage to the product as a result of doing so will void your warranty.

Attaching the Battery to the PRIZM:
To connect the battery pack to the PRIZM, plug the colored battery connectors into one of the two battery ports located on the controller. Connectors should be plugged RED to RED and BLACK to BLACK for proper operation. Do not connect in reverse polarity.

The battery pack can be plugged into either the top or bottom row of the port.



Warning: Do not attempt to connect two battery packs to one PRIZM controller. Damage and failure will occur if this is attempted. Use only one battery pack to power the PRIZM system and any extra daisy-chained expansion controllers.

Software Overview

The Arduino Integrated Development Environment (IDE) is open-source software that compiles code and sends it to the hardware. As a text-based and developer-friendly software with a huge existing user base, the *Arduino Software (IDE)* is supported for a variety of Windows, Macintosh, and Linux devices. This makes the *Arduino Software (IDE)* flexible, classroom friendly, and competition ready and offers all users, even the hobbyist, a low entry and high ceiling. All these factors make it perfect for beginner and veteran coders alike.

The *Arduino Software (IDE)* uses a C-based programming language to communicate with the PRIZM. Within the *Arduino Software (IDE)* an individual program is referred to as a *sketch*. Each of the activities in this guide will involve creating a sketch that gives instructions to the robot.

For the purposes of this guide, we will focus on the basics of using the *Arduino Software (IDE)* as it applies to the PRIZM controller. Working through examples and hands-on application of code using a small TaskBot constructed with the TETRIX MAX robotics building system will show you how easy it is to use Arduino with PRIZM.

Note: This is not meant to be a complete tutorial on programming with the Arduino C-based language. There are many excellent resources available on the web to learn more about advanced programming skills. If you are interested in such resources, a good place to start would be the Arduino website at **www.arduino.cc**.

Software Setup

The first thing we need to do is install the *Arduino Software (IDE)*. The software can be found at the Arduino website (**www.arduino.cc**) for Windows, Macintosh, and Linux operating systems. From the Arduino homepage, click the Download tab. On the Download page, select the download for your operating system and follow any additional instructions.

Note: All instructions and screen shots used throughout this guide are based on the 1.6.11 version of the *Arduino Software (IDE)*. Instructions and views might slightly vary based on the platform and version you are using.

Installing the PRIZM Controller Arduino Library

Adding custom libraries can expand the usability of the *Arduino Software (IDE)*. Libraries are collections of code that make it easier to create a program, or, as Arduino calls it, a sketch. After you have successfully installed the *Arduino Software (IDE)*, you can add the Arduino PRIZM controller library. The PRIZM controller library contains special functions written for the TETRIX PRIZM controller. Many, but not all, of the functions will be explored in our activities.

Teacher's Note:
Depending on your classroom and IT situation, you might want to download and install the *Arduino Software (IDE)* and the TETRIX PRIZM Arduino Library on the computers you and your students will be using.

It is recommended that you organize the class into teams of two to four. It is recommended that you go through each process before students do. This will enable you to have a good understanding of what student questions might arise and how to answer those questions.

For a deeper look at all available functions in the PRIZM library, please refer to the appendix section titled TETRIX PRIZM Arduino Library Functions.

The PRIZM library is distributed as a .zip file: TETRIX_PRIZM.zip. The first step is to download the PRIZM library from the Pitsco website. We can find the library at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

After the library has been downloaded, there are two ways to install the PRIZM library into the *Arduino Software (IDE)*.

Importing a .zip Library

One way is to import it using the *Arduino Software (IDE)* Add .ZIP Library menu option.

In the *Arduino Software (IDE)*, navigate to **Sketch > Include Library**. From the drop-down menu, select **Add .ZIP Library** (Figure 1).

Tip: Figures within this section show typical installation within Windows. The look and file locations within the Mac and Linux operating systems might vary.

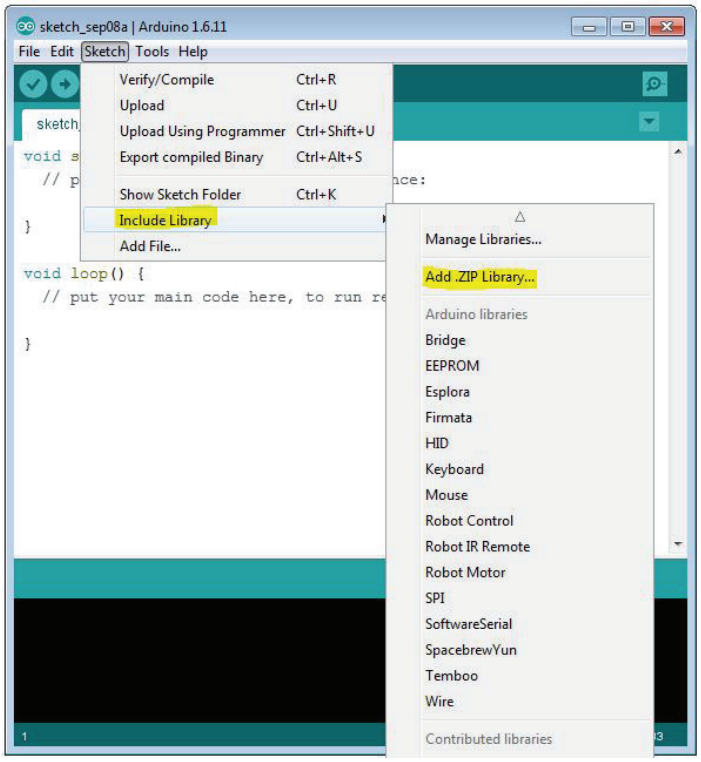


Figure 1

You will be prompted to select the library you would like to add. Navigate to the location where you saved the TETRIX_PRIZM.zip library file, select it, and open it (Figure 2).

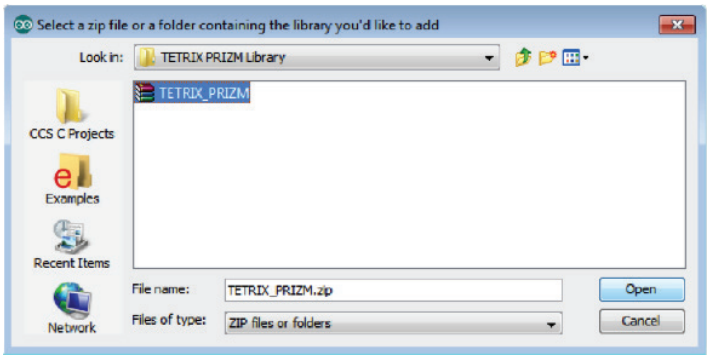


Figure 2

Return to the **Sketch > Include Library** menu. You should now see the library at the bottom of the drop-down menu. It is ready to be used in our sketches; however, example sketches for the library will not appear in the **File > Examples** menu until after the *Arduino Software (IDE)* has been restarted.

Manual Installation

To install the PRIZM library manually, first close the *Arduino Software (IDE)* application. Then, extract the .zip file TETRIX_PRIZM.zip containing the library. After the folder is extracted, drag or copy the TETRIX_PRIZM folder into the Arduino libraries folder.

For Windows users, it will likely be called **Documents\Arduino\libraries**.

For Mac users, it will likely be called **Documents/Arduino/libraries**.

For Linux users, it will be the **libraries** folder in the Arduino sketchbook.

Restart the *Arduino Software (IDE)*. Make sure the TETRIX_PRIZM library appears in the **Sketch > Include Library** menu of the software.

In addition, several PRIZM sketch examples will now appear in the **File > Examples > TETRIX_PRIZM** drop-down menu.

That's it! We have successfully installed the PRIZM Arduino library.

Configuring USB Communication

PRIZM and the *Arduino Software (IDE)* will communicate with each other through the computer's USB port.

Therefore, before we can begin programming, we first need to be sure that the PRIZM controller is properly set up in the *Arduino Software (IDE)* for communication over the USB port.

The easiest way to do this is to first start the *Arduino Software (IDE)* and navigate to **Tools > Board** and select **Arduino/Genuino Uno** (Figure 3). The PRIZM controller uses the same processor chip as a genuine Arduino UNO, so this is the board you will select.

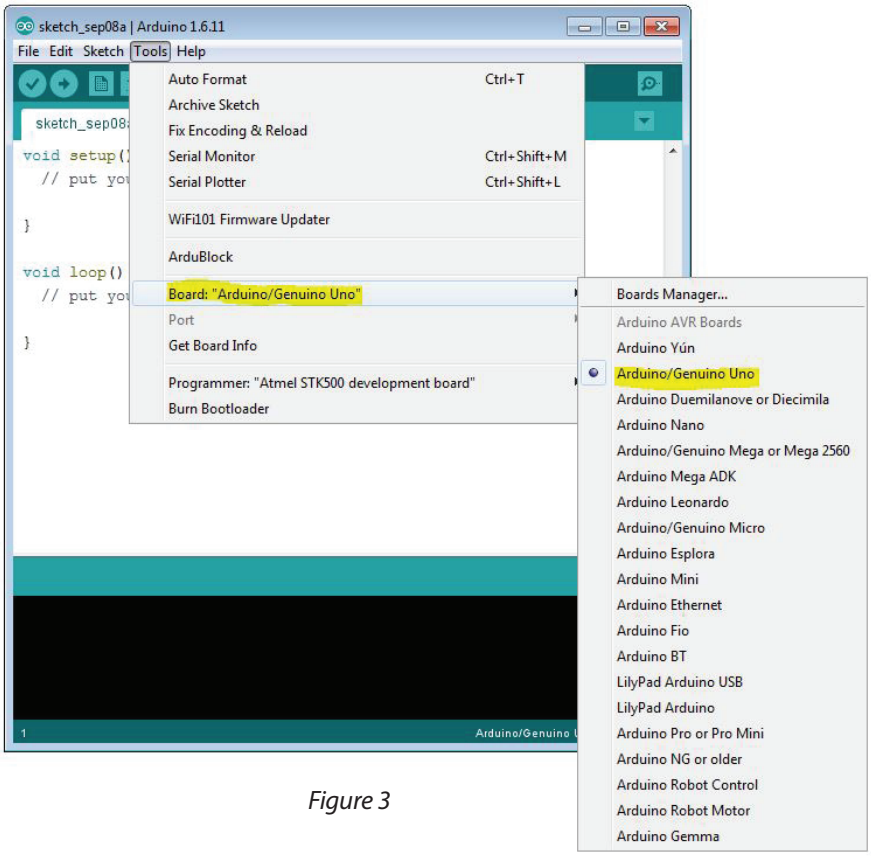


Figure 3

Next, **without the PRIZM connected**, navigate to **Tools > Port** (Figure 4) and check the current connections. If there are no current connections detected, the word *Port* will be grayed out. If there are connections detected, take note of the COM ports that are listed.

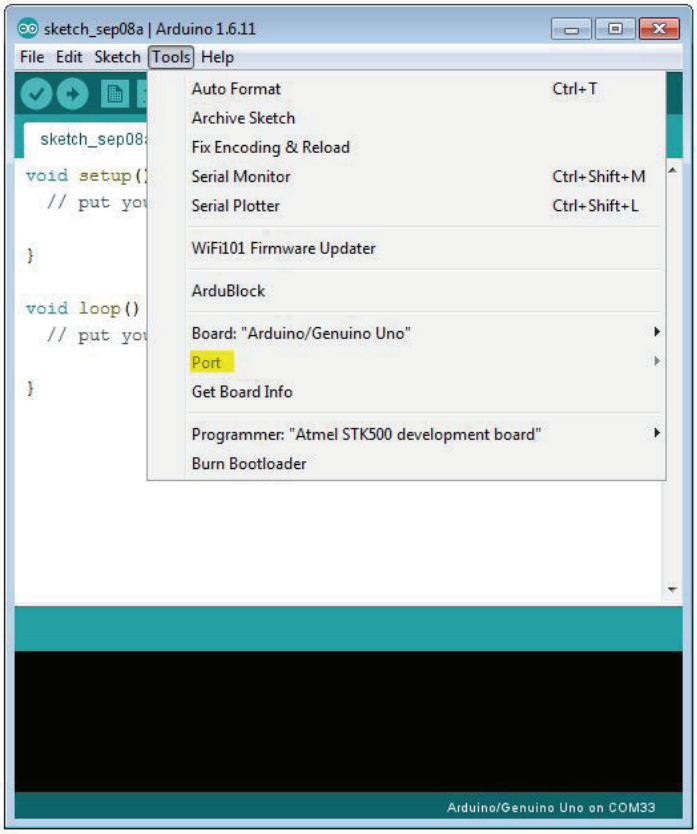
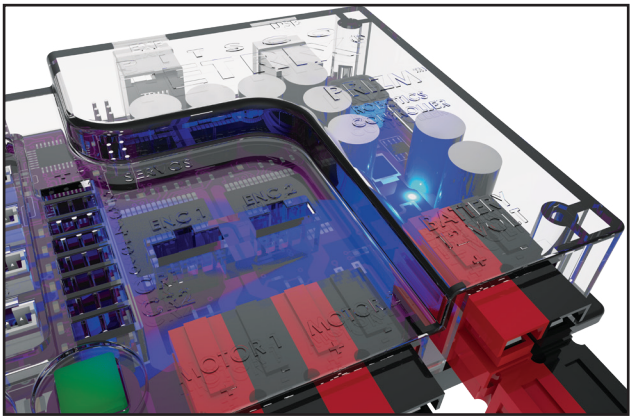


Figure 4: Port drop-down menu with PRIZM not connected. Please note that lists might vary.

Next, plug the PRIZM controller into a USB port and power it up by connecting the TETRIX battery pack and turning the power switch on.



With power applied, the blue power indicator LED will be lit. Be sure to give the PRIZM controller time to complete the first-time connect installation. This could take 5-10 seconds. After the PRIZM has been connected and installed, it will be assigned a COM port by the computer system.

Navigate to **Tools > Port** and select the newly installed COM port. The new COM port will be the PRIZM. By selecting the new COM port, you are telling the *Arduino Software (IDE)* to use this port for communications. The COM port you use will likely be different from the one in Figure 5.

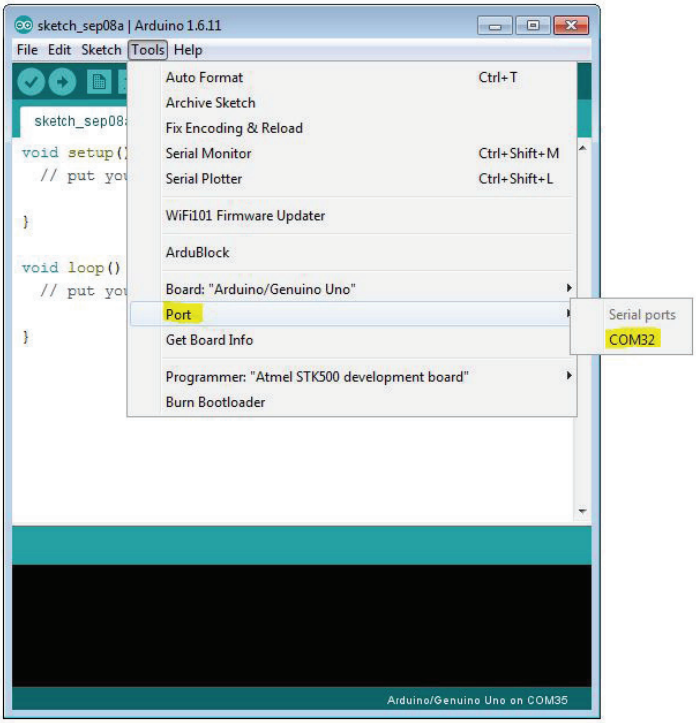


Figure 5: Port drop-down menu after PRIZM is connected to USB port and powered on.

The new port that appears in this example is COM32.

Select the new port item to tell the *Arduino Software (IDE)* to use this port for communications. Our port will likely be different and that is OK. When the communications port has been set up, communications with the PRIZM controller have been enabled and coding can begin.

When this step is complete, our computer system will automatically default to this selected port each time we plug in our PRIZM controller and start the *Arduino Software (IDE)*.

Tip: Each PRIZM unit will use a different COM port on the same computer. For each new PRIZM that is connected, follow the steps for numbering controllers and matching to the computer as detailed above. One potential way around this issue is to number each PRIZM controller and assign it to a corresponding computer. This will facilitate the computer selecting the correct port for PRIZM each time it is connected and powered up.

Getting Started Activities

Now, it is time to get started with the activities. Each of the five getting started activities is designed to introduce you to the *Arduino Software (IDE)* and how it works with the PRIZM and select basic hardware. Success with these first five activities will demonstrate how easy it is to use *Arduino Software (IDE)* with the PRIZM and prepare you for coding the PRIZM TaskBot.

The Arduino Sketch

As we begin creating our first sketch, it is necessary to understand some basic rules about sketches. It is best to think of a sketch as a list of instructions to be carried out in the order that they are written down. Each sketch will have several instructions, and typically each instruction will be one line of text within the sketch. Lines of text within a sketch are also known as code, which is why programming is sometimes called coding.

Most of the rules that affect text-based programming are known as syntax, which is similar to grammar and punctuation in English.

For instance, each line of code must end with a semicolon just like a period marks the end of a sentence. As we come across more coding rules in the activities, we will make sure to explain them.

Many times the best way to learn how to code is by following an example. In the following sections we will work through several coding examples to better explain how to create sketches and upload them to the PRIZM controller.

An example sketch of each function can be found in the **File > Examples > TETRIX_PRIZM** drop-down menu.

Note: If you are already comfortable with coding in Arduino sketches and want to jump ahead, an overview of each library function can still be a helpful starting point. We have created and included definitions of the functions along with descriptions that show how each would appear in an Arduino sketch. You can find these in the appendix of this guide on pages 133-145. More library functions might be added in the future as the library is updated to newer versions.

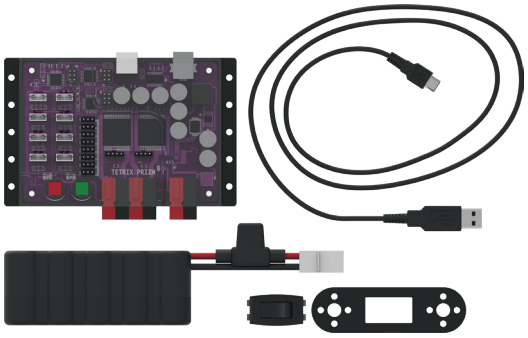
Note: In addition to the PRIZM library, there is an entire collection of Arduino language commands that are necessary to understand before we can create functional programs. The Arduino language reference as well as numerous language learning tutorials can be found by visiting the Arduino homepage at **www.arduino.cc**.

Activity 1: Hello World!

Let's begin with a very simple sketch that will blink the onboard PRIZM red LED. This activity will be the PRIZM equivalent of a Hello World! program, which is usually the intro activity for any beginning programmer. The sketch we will create is one of the simplest and most basic PRIZM functions and requires only the PRIZM, a power source, and a USB connection to the computer.

Parts Needed

- PRIZM controller
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- PRIZM Controller On/Off Battery Switch Adapter
- Computer



Opening the Sketch

Let's start by looking at our first example sketch. Open the sketch by selecting **File > Examples > TETRIX_PRIZM > GettingStarted_Act1_Blink_RedLED**. A new sketch window will open titled GettingStarted_Act1_Blink_RedLED (Figure 6).

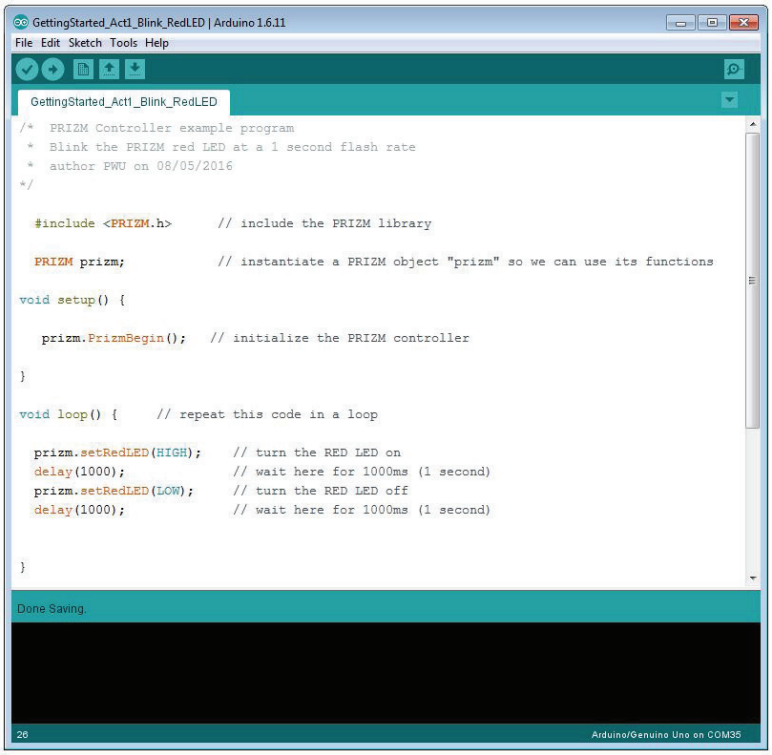


Figure 6

Building the Knowledge Base

Before we can upload the sketch to the PRIZM, we need to make sure the PRIZM has power, is connected to the computer, and is detected by the computer (Figure 7). When the PRIZM is connected as shown, power on the PRIZM with the on/off switch. You will know the PRIZM has power by the glowing blue light. To see if the PRIZM is detected by the computer, check the port as we did in the Configuring USB Communication section.

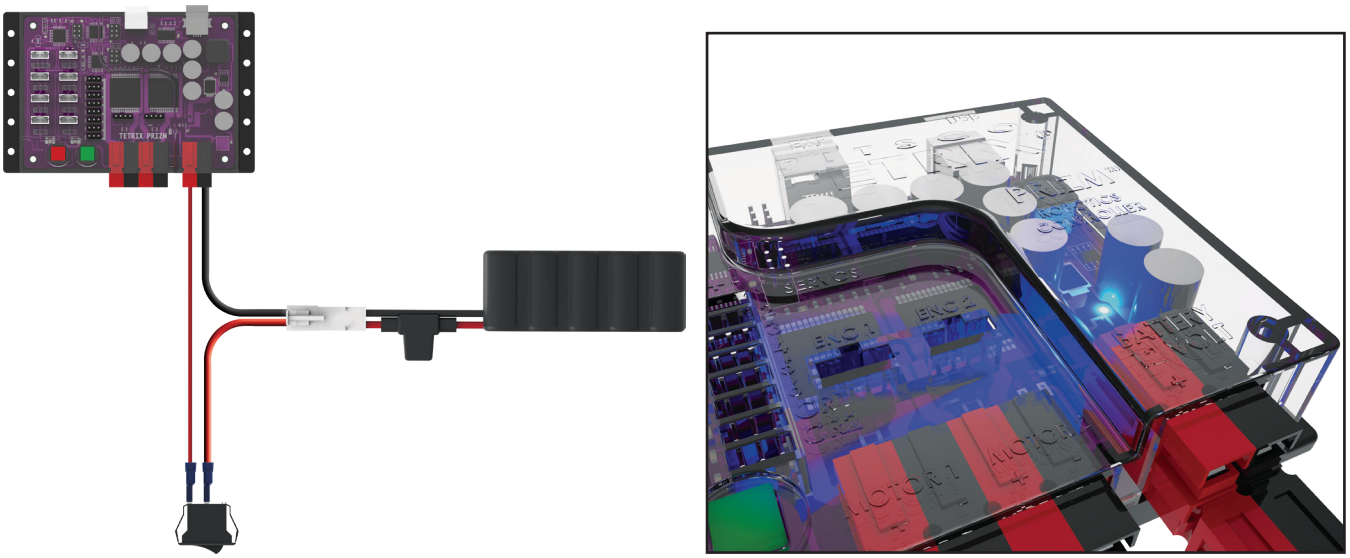


Figure 7

Executing the Code

To upload the sketch to the PRIZM, click **Upload** (Figure 8).

Figure 8

As the data uploads, the yellow LEDs to the side of the USB connection will flash. When the upload is finished, there will be a solid green LED beside the red Reset button. The green LED means the code is ready to execute. Press Start to execute the code. The red LED next to the Reset button will blink off and on in one-second intervals. To stop the program, press the Reset button.

Congratulations! You have successfully uploaded your first sketch to the PRIZM and demonstrated the results to the world.

Moving Forward

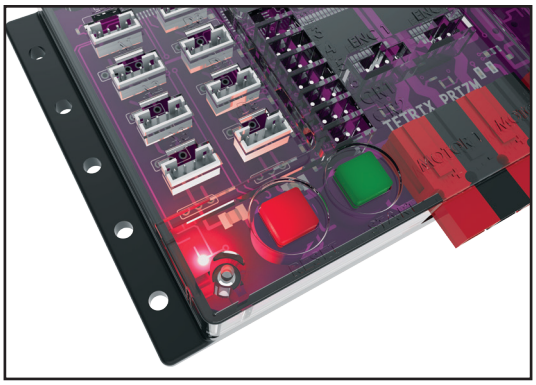
For more detailed information about the sketch process and the PRIZM library functions involved in blinking the LED, refer to the following sections within the appendix: TETRIX PRIZM Arduino Library Functions:

- Page 133 about include statements
- Page 134 about initialization functions
- Page 136 about the Start button
- Page 137 about onboard LEDs

For now, let's change some parameters in our code and see how it affects the behavior of the red LED. According to the comments in the example, the delay function defines the duration the LED is on or off. This is a parameter we can change in our code. Experiment with changing those values to create new blinking behaviors for the LED. Try making the LED blink faster or slower.

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.



Real-World Connection

Many things within the electronic world around us blink, such as caution lights for road construction warnings, waffle irons (the blinking light turns solid when the waffles are ready), or notifications on our phones indicating that we have incoming calls. The rate at which these items blink – or when they blink – is controlled by electronics. This control can be from simple timing circuits – or it can be from computers or other devices with microprocessor chips.

STEM Extensions

Science

- Electricity terms (voltage, current, and resistance)
- How an LED works
- What determines the color of an LED

Technology

- How computers work
- How computers are programmed

Engineering

- Problem-solving process

Math

- Frequency
- Duration
- Sequence

Hacking the Code Activity

With the example as a reference, try creating the blinking LED in a new sketch. Instead of just blinking the red LED, try to blink the green LED too. Flashing or blinking lights have a rich tradition as a method of signaling or long-distance communication. You could challenge yourself to communicate "Hello World!" in blinking Morse code.

To start a new sketch, select **File > New**. Be sure to use the appendix: TETRIX PRIZM Arduino Library Functions in the back of the guide for help with sketch structure and syntax. An example library of code can also be found in the appendix for those that need extra help. Do not forget to use basic software skills such as copy and paste.

When creating your own sketch, there is a built-in software tool to help ensure your code is free of syntax errors. You can check your syntax by clicking **Verify** (Figure 9). This will cause the code to compile but not upload. You will be prompted to save your sketch before verification.

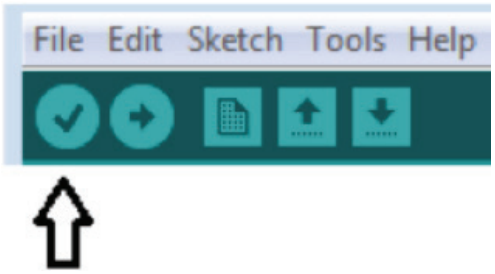


Figure 9

If there are errors in the code, they will be displayed in the compiler error window at the bottom of the sketch window (Figure 10).

Errors will need to be corrected before code can be uploaded to the PRIZM controller. If there are no errors, the compiler will complete and indicate that it is done compiling, and you can upload your code.

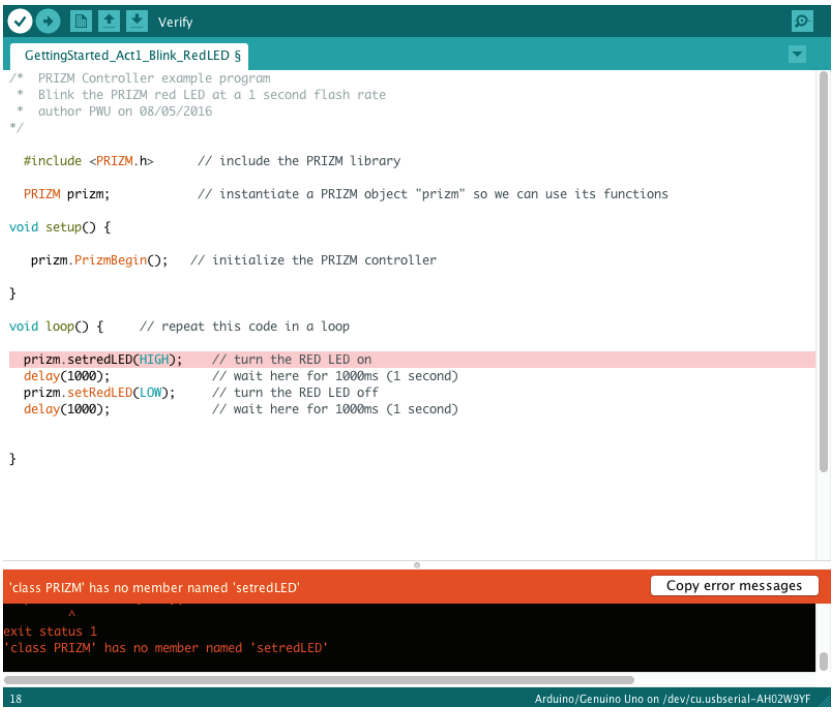


Figure 10

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

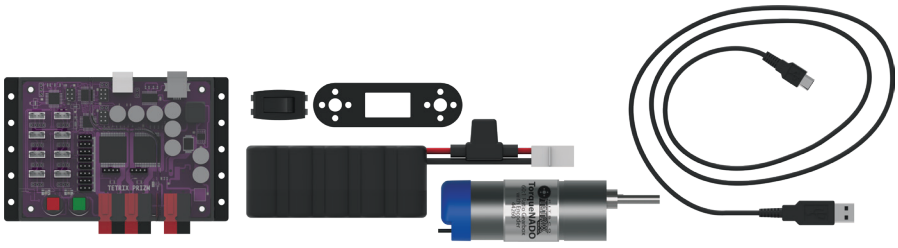
Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

Activity 2: Moving Your DC Motors

For our second activity we want to keep things simple but add an element of motion. We will create a sketch that will rotate a TETRIX DC motor.

Parts Needed

- TETRIX DC motor
- PRIZM controller
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- PRIZM Controller On/Off Battery Switch Adapter
- Computer



Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later. Open the sketch by selecting **File > Examples > TETRIX_PRIZM > GettingStarted_Act2_Move_DCMotor.** A new sketch window will open titled GettingStarted_Act2_Move_DCMotor (Figure 11).

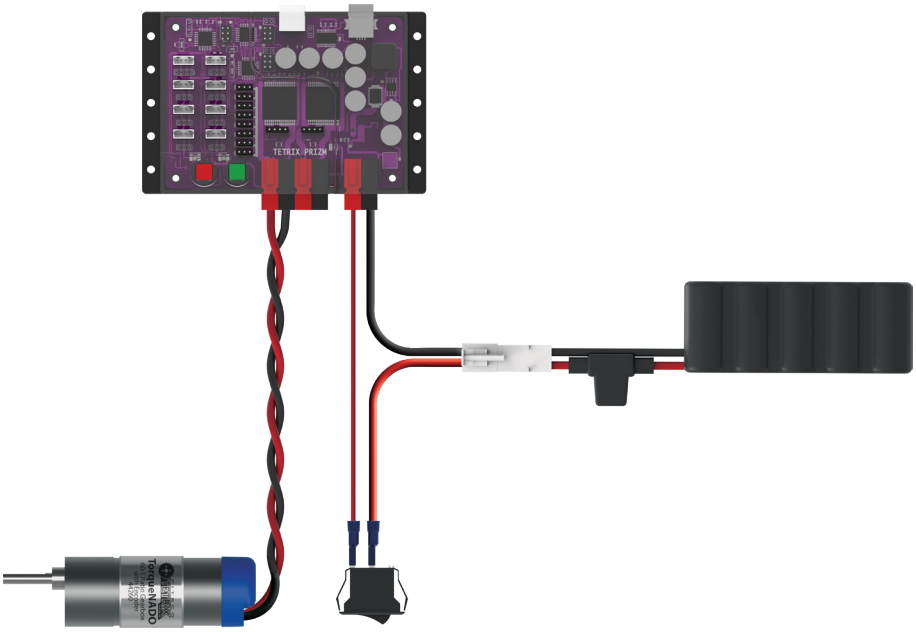
Figure 11

Building the Knowledge Base

In this second sketch we want to take a closer look at the sketch syntax, specifically comments. Comments are lines in the program that are used to inform yourself or others about the way the program works. They are for informational purposes only and do not affect the code.

There are two types of comments: single line and multiline. Single-line comments are preceded by // and anything after // is a comment to the end of the line. Multiline comments are preceded by /* and can be several lines long but must be closed with */.

When you look at this second sketch, the comments explain how the sketch author intended this program to work. The intent of this sketch is to spin the DC motor channel 1 for five seconds and then coast to a stop. After two seconds of coasting, the motor will spin in the opposite direction. This behavior will continue until the red Reset button is pressed.



Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections. Keep in mind that we added a new connection with the motor.

Upload the sketch. The green LED will light up, indicating that the code is ready to execute. When this has happened, press the green Start button on the PRIZM controller.

Observe the direction and duration of the motor rotation. Based on the sketch comments, did the behavior match expectations?

Press the red Reset button when you are ready to stop the motor.

Moving Forward

The function used in this sketch is prizm.setMotorPower. It has two parameters: motor channel and motor power. In the example, prizm.setMotorPower(1,25) means motor 1 will spin at 25% power clockwise. The first value in parentheses defines the motor channel, and the second value in parentheses defines power percentage and direction.

We can alter the direction and stopping behavior by changing the second value in parentheses. If the second value is negative, the motor rotates counterclockwise, as shown in the sketch. If we change the value to 125 instead of 0, the stopping behavior will change from coast to brake.

For more detailed information about the sketch process and the PRIZM library functions involved in moving the motor, refer to the following section within the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 137-142 about DC motor functions

Practice changing the parameters in the sketch. We can change the motor power, motor direction, stopping behavior, and delay between functions. Observe the effect these changes have on the motor.

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

Real-World Connection

Controlling motors is not a new thing. Controlling them by an onboard computer in an electric car (such as a Tesla) at 70 mph down the road or during a sharp turn on a curvy highway – that is new! For these cars that are driven by electric motors, the speed and power levels of all the drive motors must be coordinated to make the turn in the highway as the driver is turning the steering wheel. All this has been programmed into the brains of these cars to make it simple and easy for the driver.

STEM Extensions

Science

- How DC motors work
- Angular velocity

Technology

- Relationship between power, voltage, and current
- Torque

Engineering

- Determining load and torque

Math

- Pulse width modulation (PWM)
- Revolutions per minute (rpm)

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Hacking the Code Activity

With the example as a reference, try creating a new sketch to move your DC motor. Remember what we learned from our first activity and think of creative ways to include blinking LEDs with your rotating motor.

Be sure to use the appendix: TETRIX PRIZM Arduino Library Functions on page 133 for help with sketch structure and syntax.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

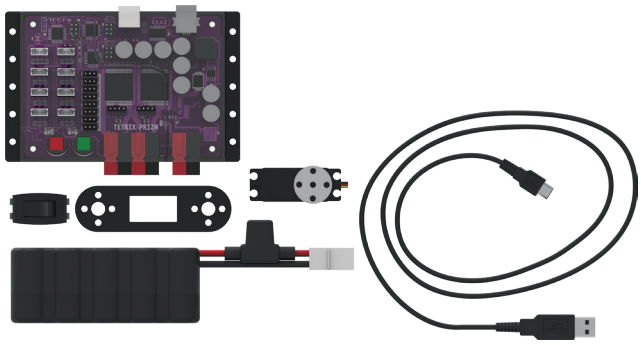
Activity 3: Moving Your Servo Motors

Our third activity will explore another element of motion with servo motors. We will create a sketch to rotate a servo motor.

Servo motors offer the benefit of being able to move to a set position regardless of the start position within a limited range of motion. Servo motors have an approximate range of motion from 0 to 180 degrees. For example, we can tell a servo to go to position 45 degrees regardless of where it starts. If it starts at 0 degrees, it will move clockwise to 45 degrees. If it starts at 120 degrees, it will move counterclockwise to 45 degrees.

Parts Needed

- TETRIX MAX Standard-Scale Servo Motor
- PRIZM controller
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- PRIZM Controller On/Off Battery Switch Adapter
- Computer



Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > GettingStarted_Act3_Move_Servo**. A new sketch window will open titled GettingStarted_Act3_Move_Servo (Figure 12).

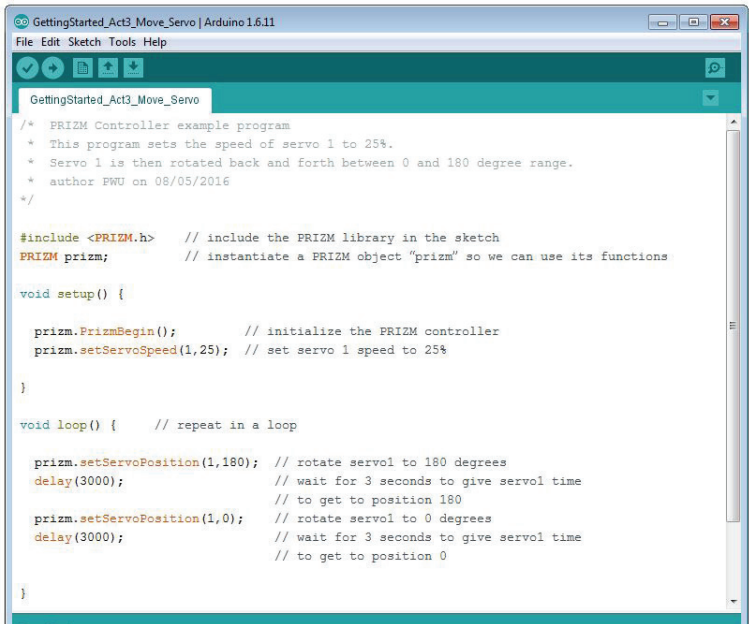


Figure 12

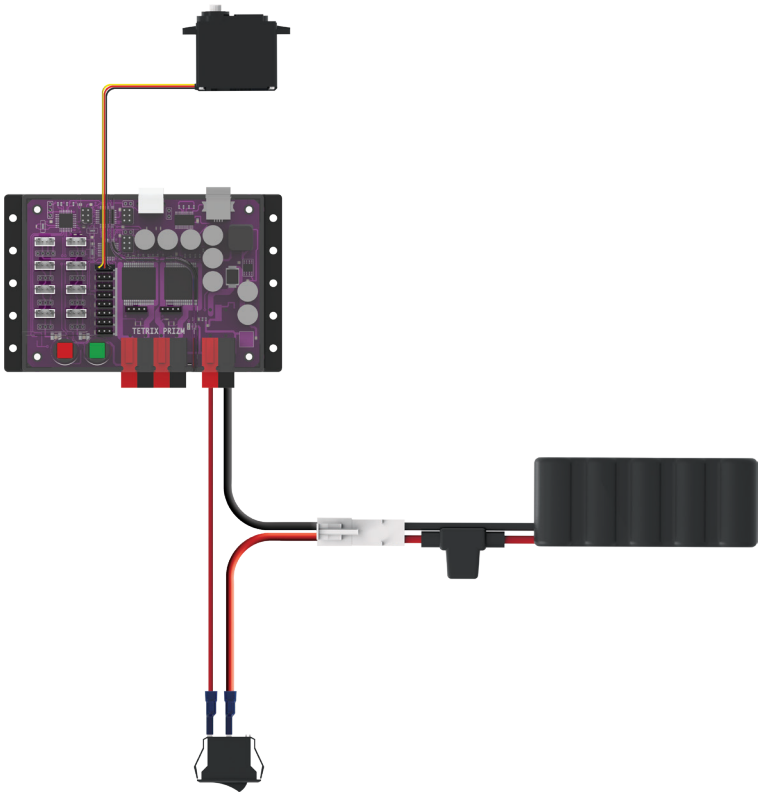
Building the Knowledge Base

In this third sketch we want to continue looking at sketch syntax, specifically the include statement and the object declaration. The include statement is used to add the functionality of the PRIZM software library into the sketch. The object declaration is a technique used when writing programs to make them easier to manage as they grow in size and complexity.

The PRIZM software library is a collection of special mini programs each with its own distinct function name. These mini programs are designed to make writing sketches for PRIZM easy and intuitive. By using the include statement, we add the functionality of the library to our sketch. The include statement for PRIZM is #include <PRIZM.h>.

The object declaration is an important statement when using the PRIZM controller along with the PRIZM software library. In order to use the functions contained in the PRIZM software library, we must first declare a library object name that is then inserted as a "prefix" before each library function. The object declaration we use is PRIZM prizm;. We use this statement just after the include statement.

In all of our sketch examples, we use an include statement and object declaration to add the functionality of the PRIZM software library. Include statements and object declarations are common for most forms of C-based language.



Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections. Keep in mind that we added a new connection with the servo motor.

Upload the sketch. The green LED will light up, indicating the code is ready to execute. When this has happened, press the green Start button on the PRIZM controller.

Observe the direction and duration of the servo motor rotation. Based on the sketch comments, did the behavior match expectations?

Press the red Reset button when you are ready to stop the motor.

Moving Forward

In this sketch we introduce two new PRIZM functions, prizm.setServoSpeed and prizm.setServoPosition. Both functions have two parameters, but they are different.

The two parameters of the prizm.setServoSpeed function are servo channel and servo speed. In the example, prizm.setServoSpeed(1,25) means servo 1 will spin at 25% power while it rotates to the position commanded by the prizm.setServoPosition function. This function needs to be called only once at the beginning of the program.

The two parameters of the prizm.setServoPosition function are servo channel and target position. In the example, prizm.setServoPosition(1,180) means servo 1 will rotate to the target position of 180 degrees.

In the sketch both of these functions work together to tell the servo motor not only the target position but also the speed to use while moving to the target position. We can alter the position and speed of the servo by changing the values of both functions.

For more detailed information about the sketch process and the PRIZM library functions involved in moving the servo motor, refer to the following section within the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 144-145 about servo motor functions

Practice changing the parameters in the sketch. Observe the effect these changes have on the servo motor.

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

Real-World Connection

Historically, servo motors were used mostly with a remote-controlled (R/C) transmitter for R/C model cars to control steering or R/C model airplanes to control flaps and rudders. Robots can use servos controlled by R/C transmitters – but they can also use servos controlled by PRIZM to operate robotic arms, grippers, tilting and rotating mounts for cameras, or many other applications in which precise movement is needed.

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

STEM Extensions

Science

- Levers
- Centripetal force

Technology

- Mechanical linkages
- Transmission of force

Engineering

- Applying torque

Math

- Radian vs Cartesian measurements
- Arc length

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

Hacking the Code Activity

With the example as a reference, try creating a new sketch to move your servo motor. Remember what we learned from our previous activities and think of creative ways to combine the functions you have learned.

Activity 4: Introduction to the Line Finder Sensor

For the fourth activity we will change focus from motor outputs to sensor inputs by introducing and exploring sensors. In this example we will connect a Line Finder Sensor to digital sensor port D3, and we will create a sketch to read a digital input from the Line Finder Sensor.

Sensors enable us to gather information from the world around us. The type of information depends on the type of sensor. The Line Finder Sensor uses reflected infrared light to distinguish between light and dark surfaces.

Parts Needed

- Contrasting light and dark surface
- Grove Line Finder Sensor and cable
- PRIZM controller
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- PRIZM Controller On/Off Battery Switch Adapter
- Computer

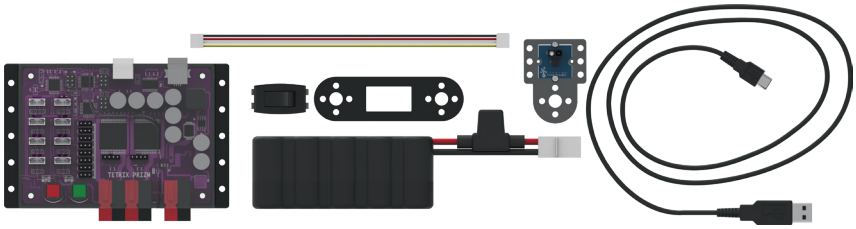


Figure 13: Contrasting light and dark surface

Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > GettingStarted_Act4_Intro_LineFinder**. A new sketch window will open titled GettingStarted_Act4_Intro_LineFinder (Figure 14).

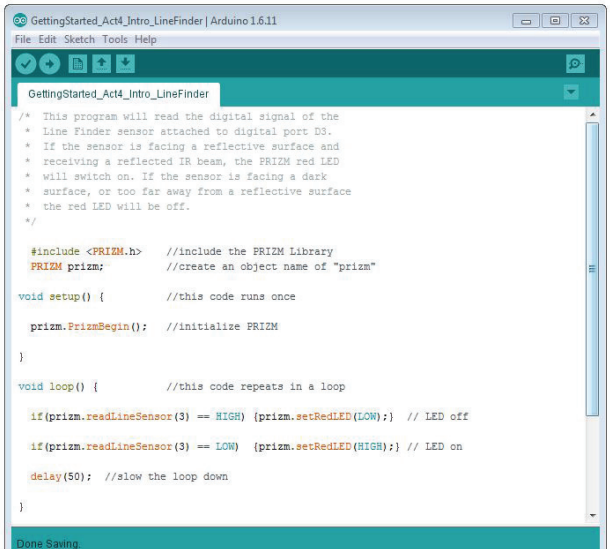


Figure 14

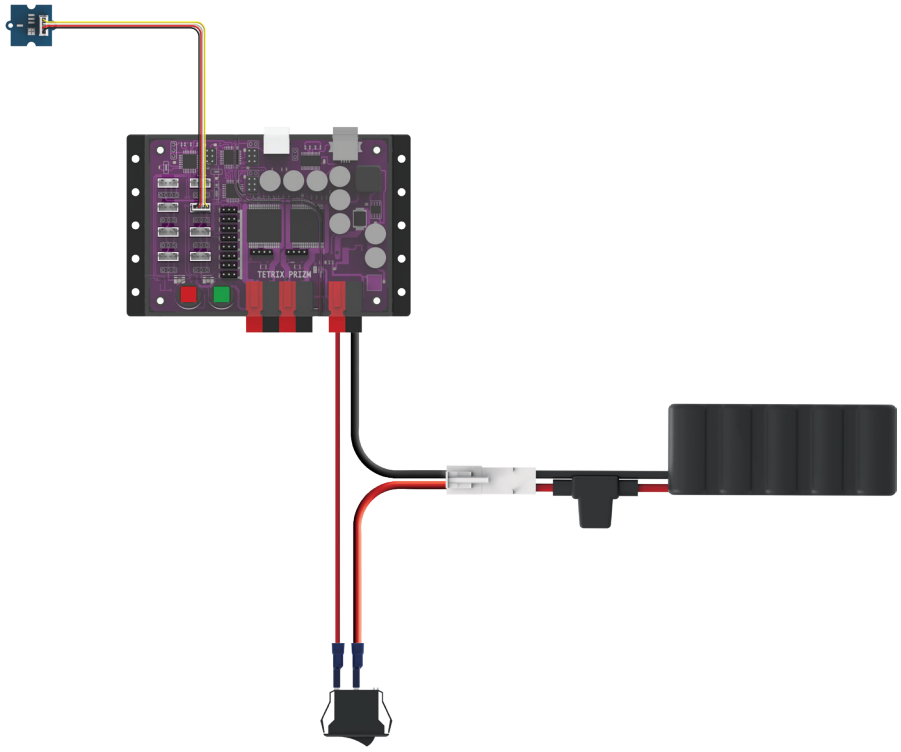
Building the Knowledge Base

For the fourth sketch we want to take a closer look at two of the fundamental structure elements that make up a sketch. In every sketch you write, there will be a setup() and a loop() function.

The setup() function follows right after the include statement and object declaration as part of the beginning of our code. The setup() function contains items that need to be run only once as part of the sketch. Many functions can go here, but we always use at least the PRIZM initialization statement, prizm. PrizmBegin(). The main purpose of this function is to configure the Start button.

The loop() function does exactly what its name suggests. Everything contained within the brackets of the loop will repeat consecutively until the sketch is ended with a command or the Reset button. The loop() contains the main body of our code.

The contents of the setup() and the loop() are contained between curly braces. A left curly brace { begins a group of statements, and a right curly brace } ends a group of statements. They can be thought of as bookends, and the code in between is said to be a block of code.



Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections. Keep in mind that we added a new connection in digital sensor port 3 with the Line Finder Sensor.

Upload the sketch. The green LED will light up indicating the code is ready to execute. When this has happened, press the green Start button on the PRIZM controller.

Hold the sensor over the contrasting surface. As the sensor moves from light to dark, observe the red LED on the PRIZM. When the sensor is over a line, non-reflective, or dark surface, the red LED will be off. When the sensor is over a white or reflective surface, the red LED will be on.

Press the red Reset button when you are ready to stop the sensor.

Note: The Line Finder Sensor has an adjustable range that we can tweak by turning the small adjustment screw on the back side of the sensor board.

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

Moving Forward

This sketch introduces a program structure, a new function, and a comparison statement. The program structure is an "if" statement, the new function is prizm.readLineSensor, and the comparison statement is == (equal to).

The basic "if" statement enables us to test for a certain condition. If this condition is met, then the program can perform an action. If the statement in the parentheses is true, the statements within brackets are run; if the statement is not true, the program will skip over the code.

The function prizm.readLineSensor reads the state of the Line Finder Sensor and returns a value of "1" (HIGH) or "0" (LOW). A value of "1" is returned when the Line Finder Sensor detects a dark line or a non-reflective surface; a value of "0" is returned when the Line Finder Sensor detects a white or reflective surface.

The comparison statement == (equal to) defines a type of test.

When these three elements are combined in the sketch, we create a test condition based on the input of the Line Finder Sensor that turns the red LED on or off. In simple terms, if the Line Finder Sensor detects a line or a non-reflective surface, then it turns the red LED off. If the Line Finder Sensor detects a white or reflective surface, it turns the LED on.

For more detailed information about the sketch process and the PRIZM library functions involved in using the Line Finder Sensor, refer to the following section within the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 134-135 about sensor port functions

Experiment with the Line Finder Sensor on different surfaces and different heights to see how the sensor reacts.

Real-World Connection

Finding our way in this world can be challenging. Telling a robot how to find its way can be challenging as well. One way that robots within a warehouse can find their way to the right location is by having them follow lines. But to follow lines, they have to detect the lines. One way to accomplish this is by using a sensor that detects dark and light surfaces. This, through the computer code, provides information to the robot about where the line is. Other code controls the DC motors to change course if the robot strays from the line.

STEM Extensions

Science

- Light – reflection and absorption
- Electromagnetic spectrum

Technology

- Digital vs analog
- Calibration

Engineering

- Determining an edge location

Math

- Data analysis

Hacking the Code Activity

With the example as a reference, try creating a new sketch to use your Line Finder Sensor. Remember what we learned from our previous activities, and think of additional creative actions to perform based on the condition of the Line Finder Sensor.

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

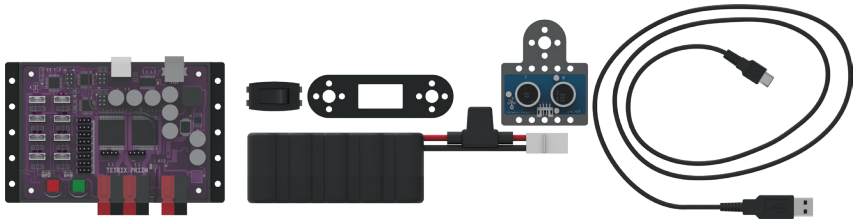
Activity 5: Introduction to the Ultrasonic Sensor

For the final getting started activity, we will finish up our exploration of sensors by creating a sketch using the Ultrasonic Sensor. In this activity we will connect an Ultrasonic Sensor to digital sensor port D3 and display the distance to an object we place in front of it using the serial monitor window.

Like all sensors, the Ultrasonic Sensor enables us to gather information. For the Ultrasonic Sensor, the information gathered communicates distance. The sensor works by sending a sonic pulse burst and then waiting on its return to the sensor as it is reflected off an object in range. The reflected sonic pulse time period is measured to determine the distance to the object. The sensor has a measuring range of approximately 3-400 centimeters.

Parts Needed

- Grove Ultrasonic Sensor and cable
- PRIZM controller
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- PRIZM Controller On/Off Battery Switch Adapter
- Computer



Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > GettingStarted_Act5_Intro_UltraSonic**. A new sketch window will open titled GettingStarted_Act5_Intro_UltraSonic (Figure 15).

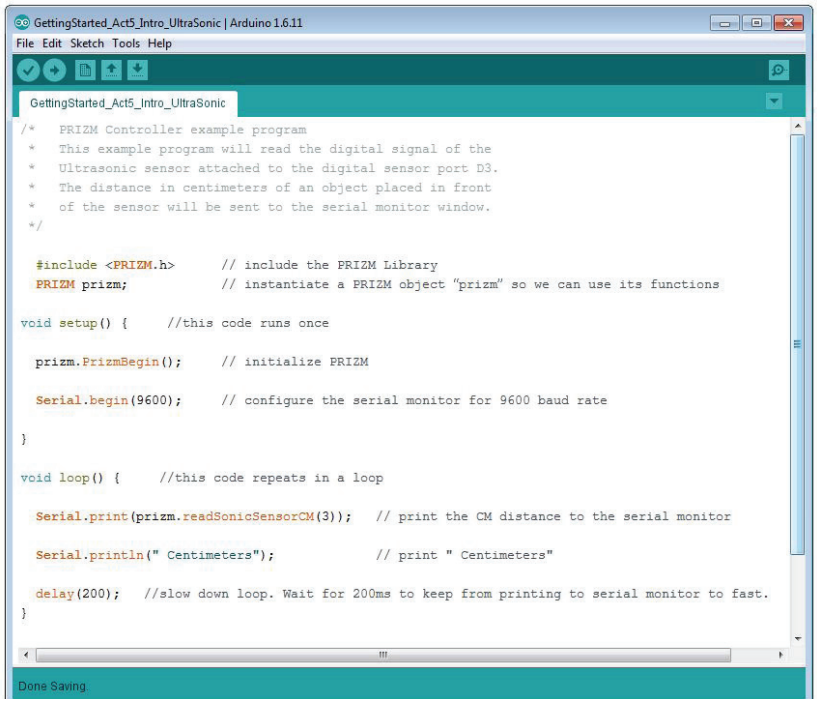
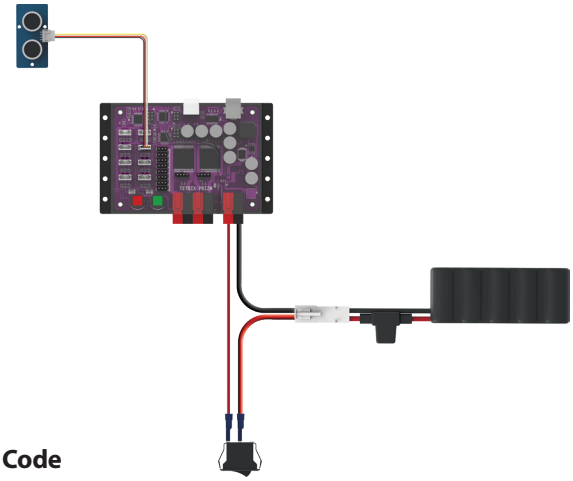


Figure 15

Building the Knowledge Base

For our fifth sketch of the getting started activities, we want to look at a useful tool for viewing data as a dynamic text output. The serial monitor displays serial data being sent from the PRIZM via the USB connection.

The serial monitor's value as a tool lies in its ability to display data in real time that enables you to make better-informed design decisions about robot builds and programming. It can be used to display data from sensors connected to PRIZM or to examine any program data collected by the PRIZM – for example, encoder count data, DC motor current data, or servo position data.



Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections. Keep in mind that we added a new connection in digital sensor port 3 with the Ultrasonic Sensor.

Upload the sketch. Before we execute the sketch, we need to open the serial monitor from the sketch window. To open the serial monitor, click the magnifying glass in the top-right corner of the sketch window (Figure 16).

Figure 16

The serial monitor will open in a separate window (Figure 17).

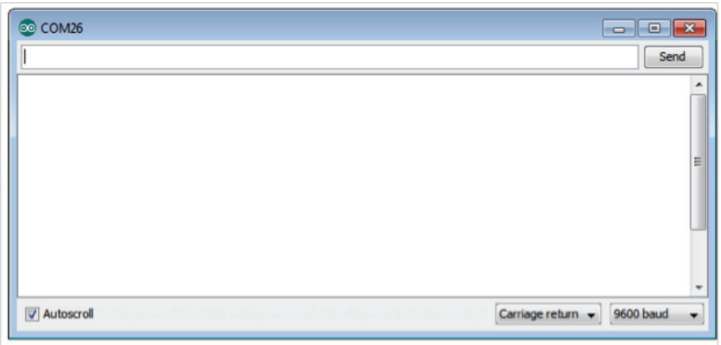


Figure 17

With the sensor lying flat on the desk pointed up, press the green Start button to execute the code.

Hold an object above the sensor at varying distances and observe the serial monitor to see the real-time data.

Press the red Reset button when you are ready to stop the sensor.

Moving Forward

This sketch introduces several new functions: Serial.begin(), Serial.print(), prizm.readSonicSensorCM(), and Serial.println().

Serial.begin() enables the use of serial communication within the sketch. As an initialization function that needs to be run only once, it belongs in the setup of the sketch. An important part of Serial.begin() is the speed of communication in bits per second, which is called baud rate. The default setting for baud rate in the serial window is 9600, so that is what we use in Serial.begin().

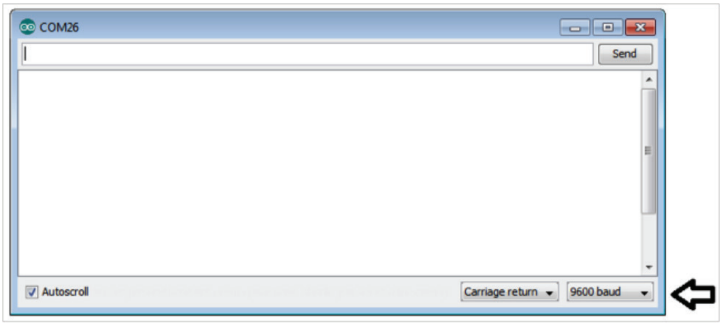


Figure 18

Serial.print() prints data to the serial port as readable text. This can take the form of dynamic information from another device or static information from the programmer.

prizm.readSonicSensorCM() reads the state of the Ultrasonic Sensor and returns a digital value within the designated measurement range. For our sensor, that range is between approximately 3-400 centimeters. This value should reflect the distance the Ultrasonic Sensor is from a detectable object.

Serial.println() prints data to the serial port as readable text followed by a built-in command for a new line.

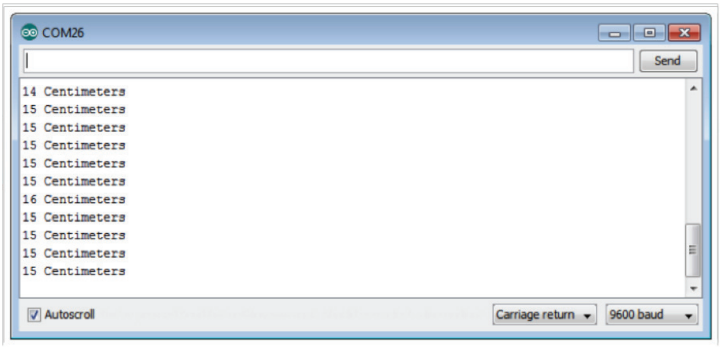


Figure 19

These four functions might seem complicated, but they actually work together simply. In this sketch, Serial.begin(9600) enables and defines the speed of communication in the setup. Serial.print() tells what type of data to print. prizm.readSonicSensorCM() provides the type of data to print because it is within the parentheses of Serial.print(). And Serial.println(" Centimeters") clarifies the type of data being printed – in this case, with the modifier " Centimeters."

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

For more detailed information about the sketch process and the PRIZM library functions involved in using the Ultrasonic Sensor and the serial monitor, refer to **www.arduino.cc** and the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 134-135 about sensor port functions

Real-World Connection

When you were very, very young, you were probably scanned by an ultrasonic sensor. If you do not remember this, it is because it was prior to you being born. One of the great applications of ultrasonic technology is within the medical field. Doctors can use the reflection of sound waves back to sensor waves to see inside a living person. Doctors can see a baby's size and development – and determine when they think he or she might be born!

STEM Extensions

Science

- Sound wave terminology (frequency, amplitude, crest, trough)
- Reflection of sound waves

Technology

- Measuring frequency
- Sound digitization

Engineering

- Applications of sonic measurements

Math

- Inverse square law

Hacking the Code Activity

With the example as a reference, try creating a new sketch to use the Ultrasonic Sensor with the serial monitor. Remember what we learned from our previous activities and experiment with different objects in front of the Ultrasonic Sensor to see if they are detectable and if the distances can be measured accurately.

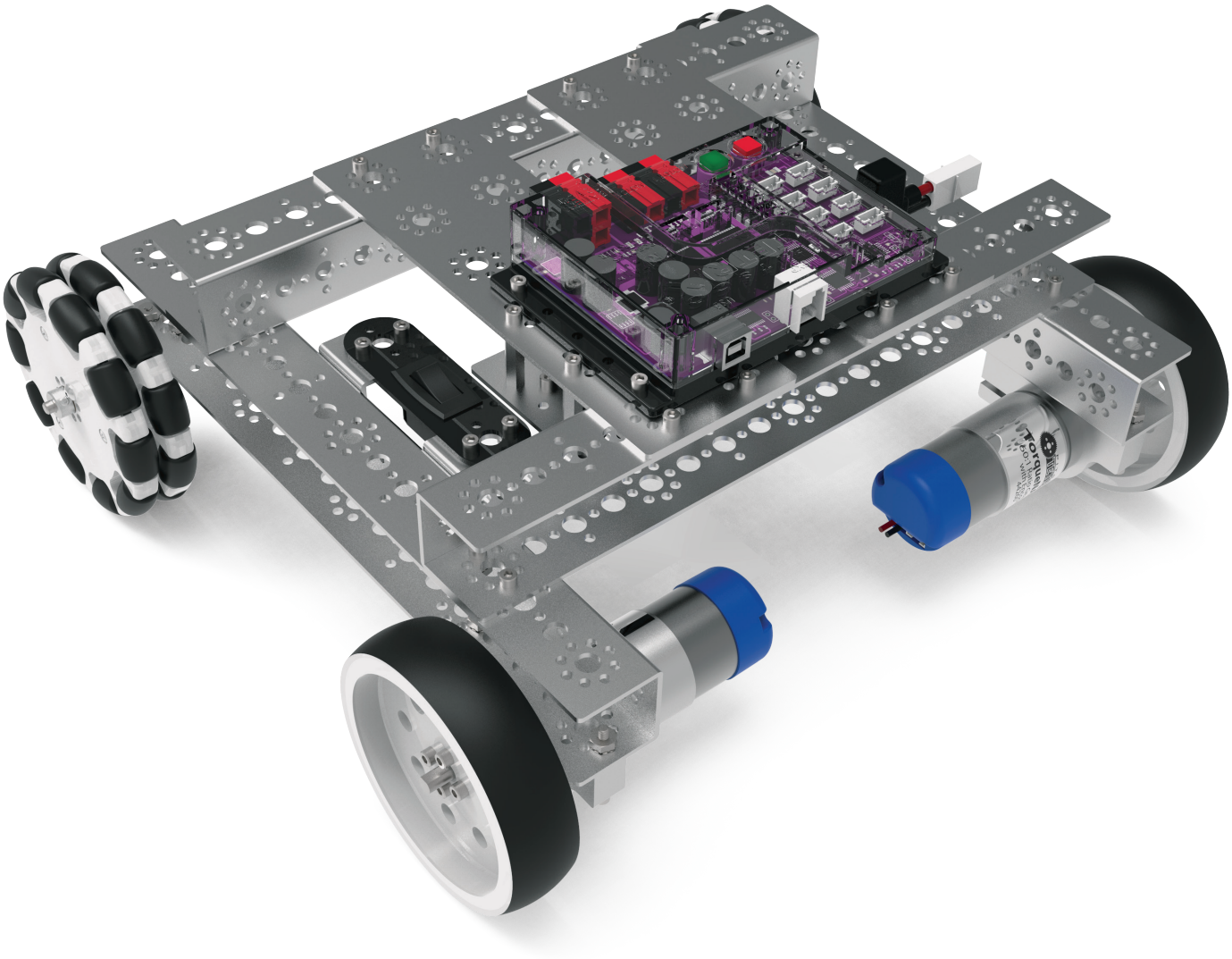
Try changing the code for prizm.readSonicSensorCM() to prizm.readSonicSensorIN() to display the distance from an object in inches. Also be sure to change the Serial. println from " Centimeters" to " Inches" so that the unit is correctly labeled in the serial monitor window. Understanding how to use the Ultrasonic Sensor will give your robot vision to be able to steer around objects and obstacles.

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

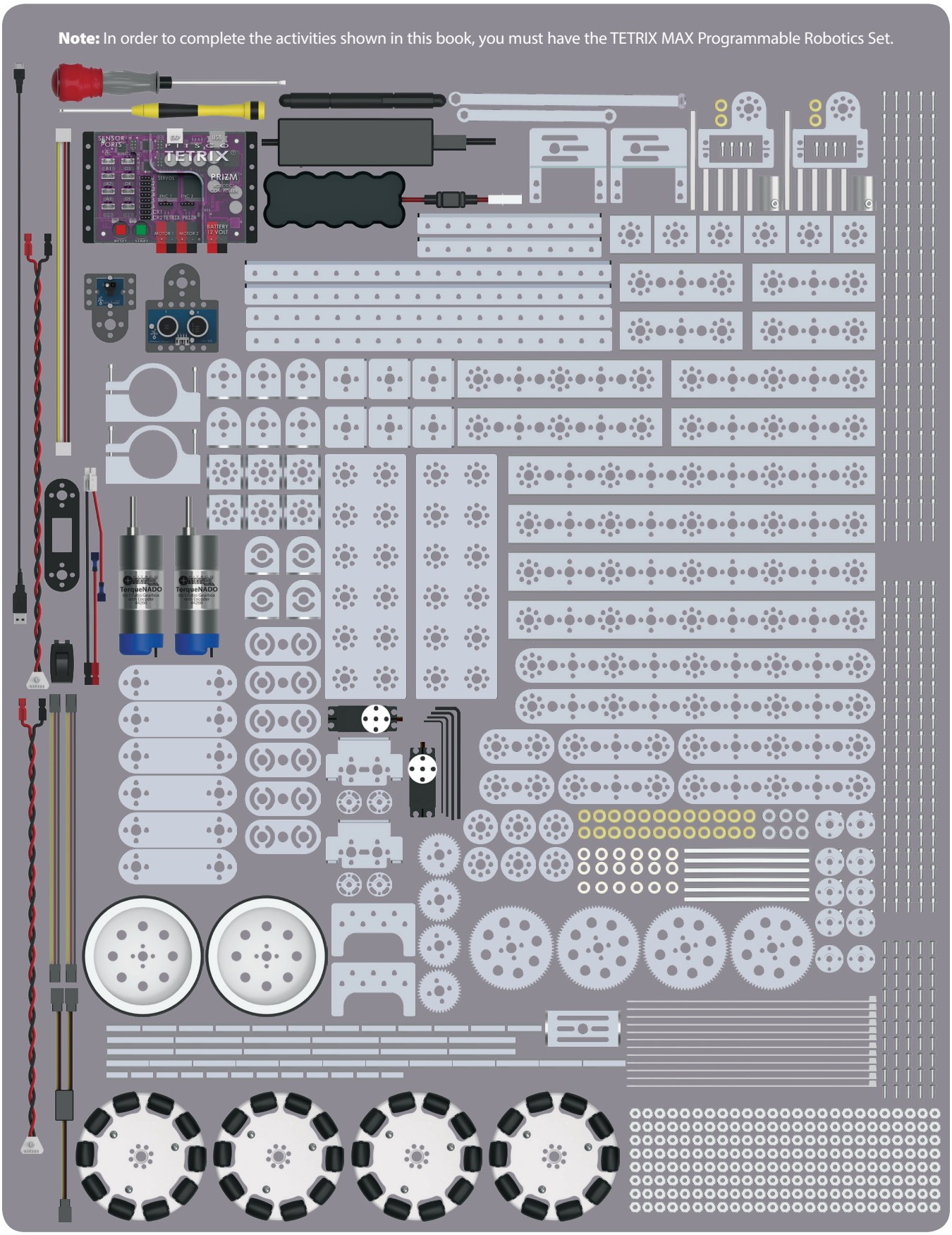
Building and Coding the PRIZM TaskBot

This is what you have been waiting for. It is time to move to the next level. You have worked hard to learn the basics, and now it is time to apply them to an actual robot. The next 10 activities will walk us through building a robot, basic movement, navigation, adding sensors, and more, culminating in an activity that combines everything. It is about to get exciting.

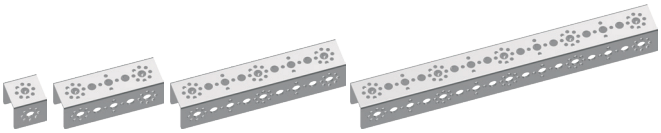


Programmable Robotics Set

Note: In order to complete the activities shown in this book, you must have the TETRIX MAX Programmable Robotics Set.



TETRIX MAX Programmable Robotics Set Parts Index



Channels

Part No.	Part Name	Quantity
39065	TETRIX MAX 32 mm Channel	6
39066	TETRIX MAX 96 mm Channel	4
39067	TETRIX MAX 160 mm Channel	4
39068	TETRIX MAX 288 mm Channel	4

Bars & Angles

Part No.	Part Name	Quantity
39070	TETRIX MAX 288 mm Flat Bar	2
39072	TETRIX MAX 144 mm Angle	2
39071	TETRIX MAX 288 mm Angle	2

Plates & Brackets

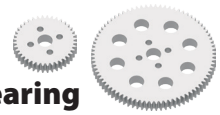
Part No.	Part Name	Quantity
39073	TETRIX MAX Flat Building Plate	2
39061	TETRIX MAX Flat Bracket	6
39062	TETRIX MAX L Bracket	6
39281	TETRIX MAX Inside Corner Bracket	6
39270	TETRIX MAX Inside C Connector	6
41790	TETRIX MAX Adjustable Angle Corner Bracket	4

Flats

Part No.	Part Name	Quantity
39274	TETRIX MAX Flat 64 mm x 27 mm	2
39273	TETRIX MAX Flat 96 mm x 27 mm	2
39272	TETRIX MAX Flat 160 mm x 27 mm	2
39271	TETRIX MAX Flat 288 mm x 27 mm	2
41791	TETRIX MAX Adjustable Angle Flat Bracket	6

Axles, Hubs, & Spacers

Part No.	Part Name	Quantity
39079	TETRIX MAX Motor Shaft Hub	2
39172	TETRIX MAX Axle Hub	6
39092	TETRIX MAX Axle Set Collar	6
39088	TETRIX MAX 100 mm Axle	6
39091	TETRIX MAX Bronze Bushing	24
39090	TETRIX MAX Gear Hub Spacer	2
39100	TETRIX MAX Axle Spacer 1/8"	12
39101	TETRIX MAX Axle Spacer 3/8"	6
39387	TETRIX MAX Flat Round Spacer	6

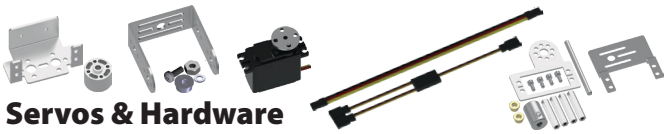


Gearing

Part No.	Part Name	Quantity
39028	TETRIX MAX Gear 40-Tooth	4
39086	TETRIX MAX Gear 80-Tooth	4

Posts & Standoffs

Part No.	Part Name	Quantity
39102	TETRIX Stand-Off Post 6-32 x 1"	12
39103	TETRIX Stand-Off Post 6-32 x 2"	12
39107	TETRIX Stand-Off Post 6-32 x 32 mm	12
41253	TETRIX Stand-Off Post 6-32 x 16 mm	12

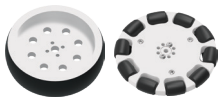


Servos & Hardware

Part No.	Part Name	Quantity
39060	TETRIX MAX Single Standard-Scale Servo Motor Bracket	1
39593	TETRIX MAX Standard-Scale Pivot Arm with Bearing	1
39197	TETRIX MAX Standard-Scale Servo Motor	2
39081	Servo Extension	2
39082	Servo Y Connector	1
41789	TETRIX MAX Standard Servo Mounting Kit	2
39280	TETRIX MAX Adjustable Servo Bracket	2

DC Motors & Hardware

Part No.	Part Name	Quantity
39089	TETRIX MAX Motor Mount	2
44260	TETRIX MAX TorqueNADO Motor	2



Tires & Wheels

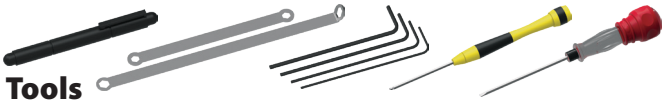
Part No.	Part Name	Quantity
39055	TETRIX MAX 4" Wheel	2
36466	TETRIX MAX 4" Omni Wheel	2

Nuts, Screws, & Fasteners

Part No.	Part Name	Quantity
39094	Kep Nut	100
39097	Socket Head Cap Screw 6-32 x 1/2"	100
39098	Socket Head Cap Screw 6-32 x 5/16"	100
39111	Button Head Cap Screw 3/8"	50
31902	Zip Tie	12

Battery & Charger

Part No.	Part Name	Quantity
38009	TETRIX MAX Battery Clip	2
39057	TETRIX MAX 12-volt 3,000 mAh Battery	1
41399	Global NiMH Battery Pack Charger	1



Tools

Part No.	Part Name	Quantity
36404	4-in-1 Screwdriver	1
38001	TETRIX Wrench Set	2
39104	TETRIX MAX Hex Key Pack	4
40341	Miniature Ball-Point Hex Driver	1
42991	2-in-1 Screwdriver	1

Electronics & Control

Part No.	Part Name	Quantity
43000	TETRIX PRIZM Robotics Controller	1
41352	TETRIX Motor Cable with Powerpoles	2
43169	PRIZM Controller On/Off Battery Switch Adapter	1
43056	Line Finder Sensor Pack	1
43055	Ultrasonic Sensor Pack	1
40967	3-Foot Type A-B USB Cable	1

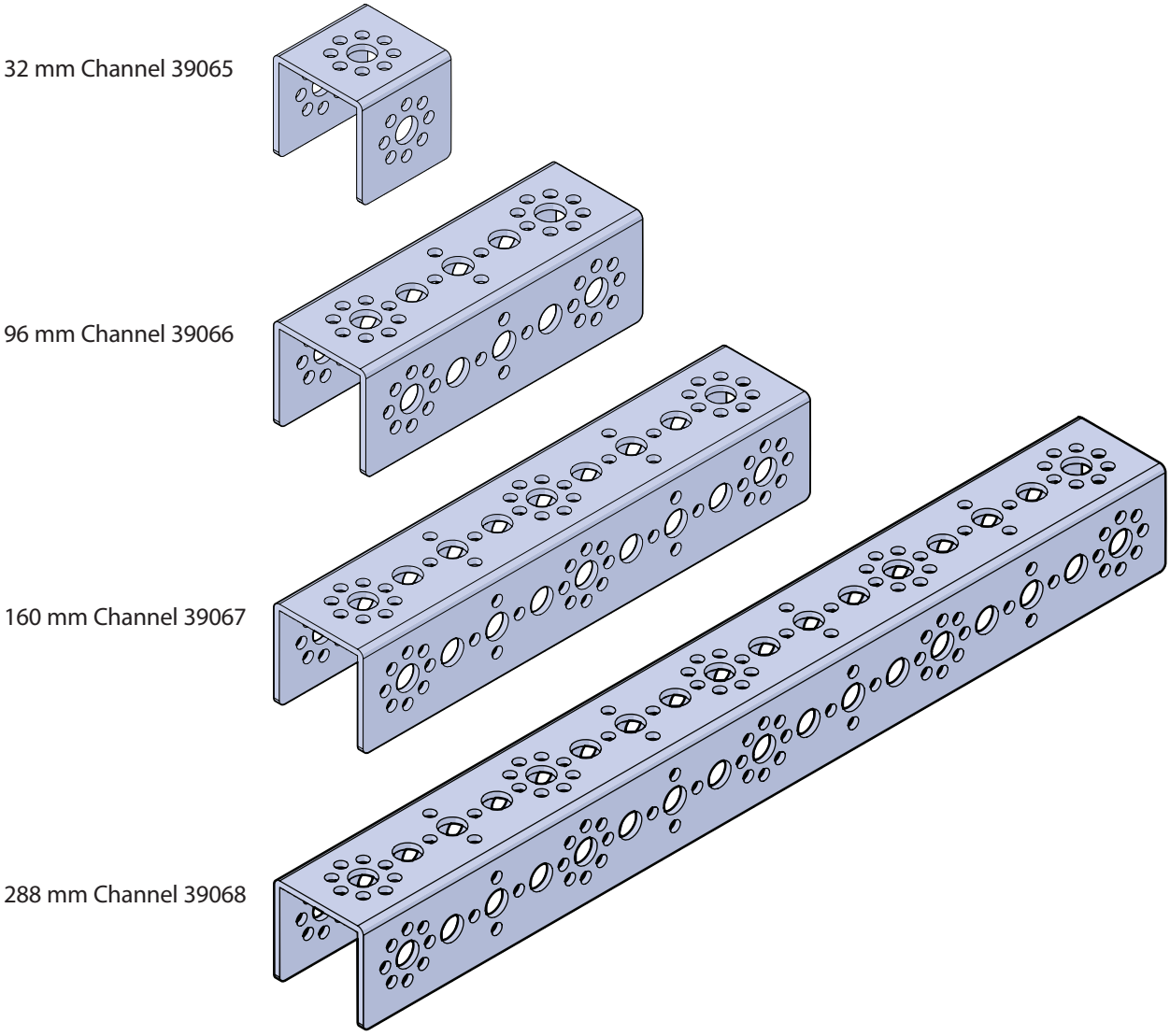
TETRIX MAX Hardware Components Overview

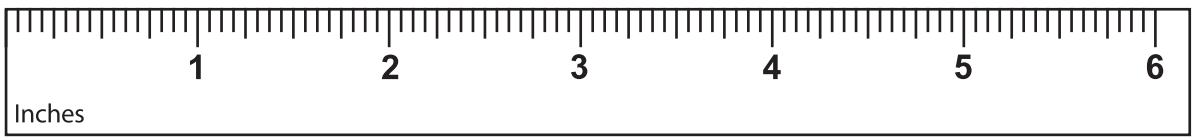
The following mechanical parts overview includes elements from the TETRIX MAX Programmable Robotics Set.

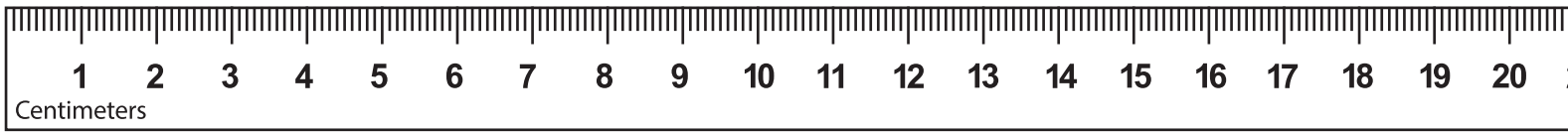
Channels

TETRIX MAX structural elements are identified by length. For example, a 32 mm channel or a 288 mm flat bar. Use the inches ruler below to measure stand-off posts and wheels. Use the centimeters ruler at the bottom of this spread to measure part lengths.

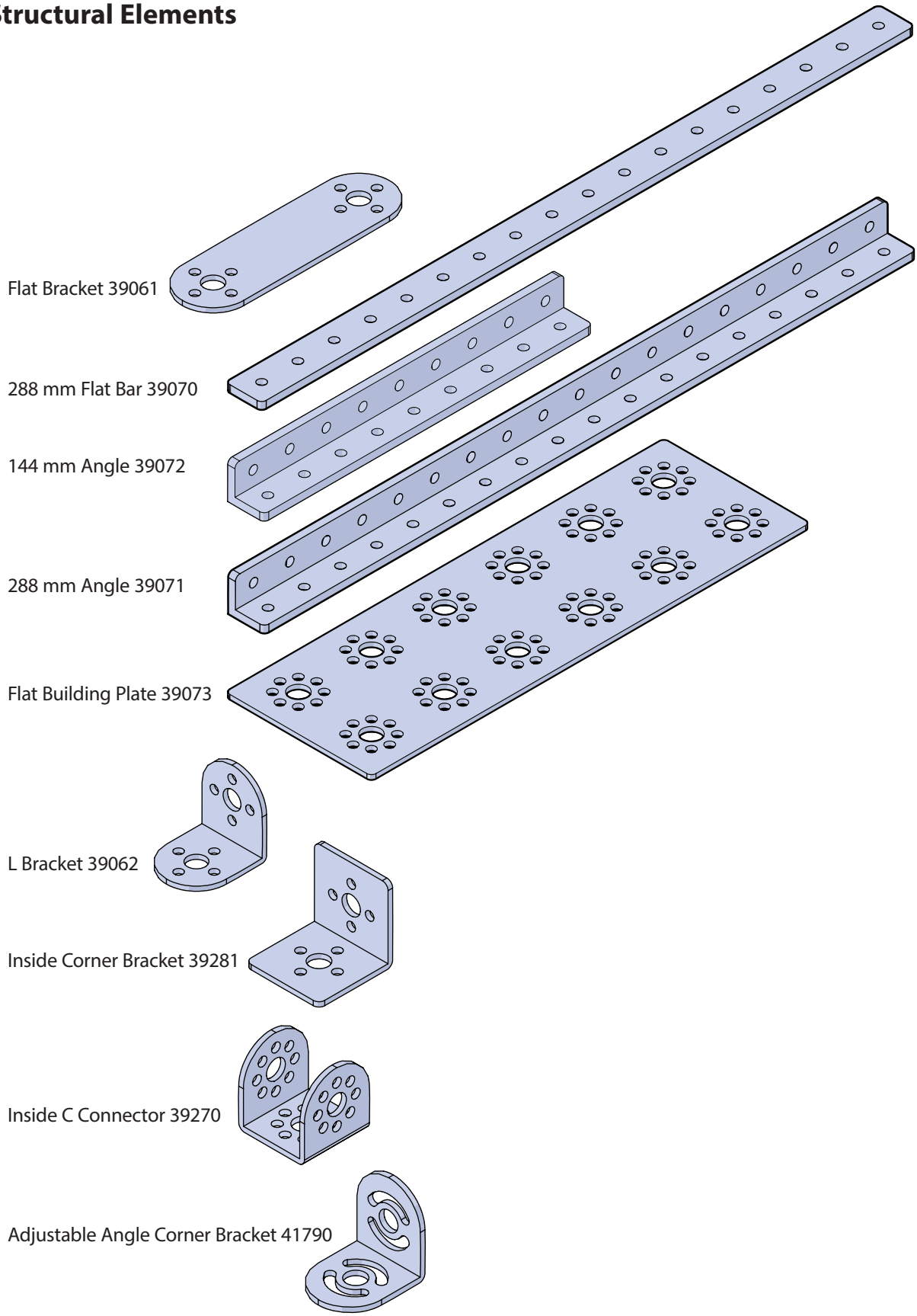
Structural Elements

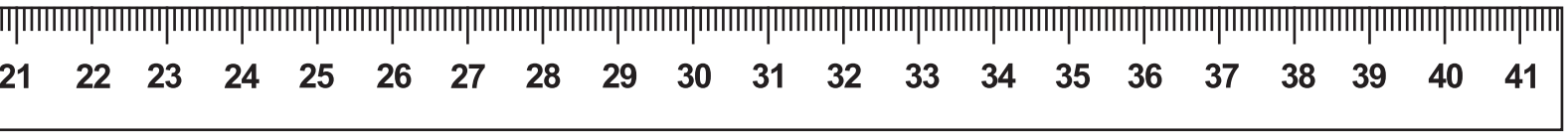






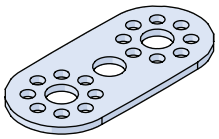
Structural Elements



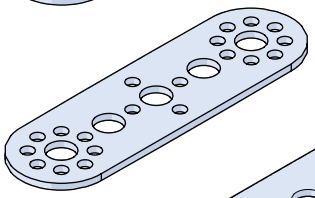


Structural Elements

Flat 64 mm x 27 mm 39274



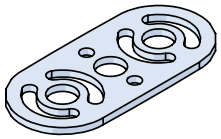
Flat 96 mm x 27 mm 39273



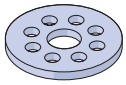
Flat 160 mm x 27 mm 39272

Flat 288 mm x 27 mm 39271

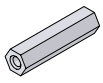
Adjustable Angle Flat Bracket 41791



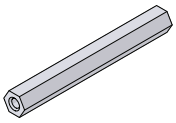
Flat Round Spacer 39387



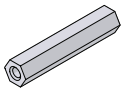
Stand-Off Post 6-32 x 1" 39102



Stand-Off Post 6-32 x 2" 39103



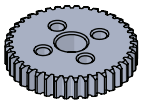
Stand-Off Post 6-32 x 32 mm 39107



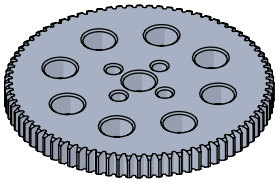
Stand-Off Post 6-32 x 16 mm 41253

Motion Elements

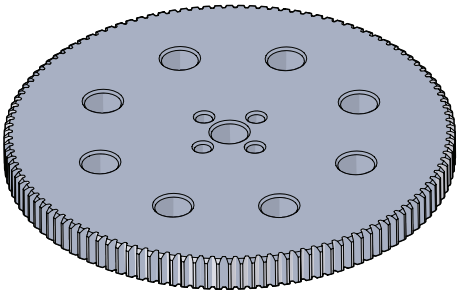
Gear 40-Tooth 39028



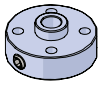
Gear 80-Tooth 39086



Gear 120-Tooth 39085



Axle Hub 39172



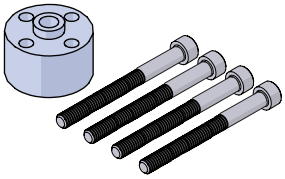
Motor Shaft Hub 39079

Axle Set Collar 39092

100 mm Axle 39088

Bronze Bushing 39091

Gear Hub Spacer 39090

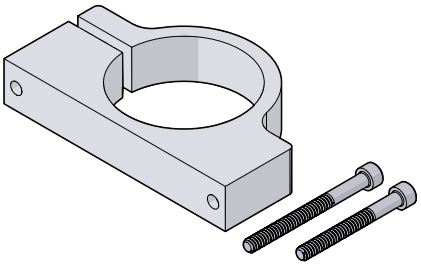


Axle Spacer 1/8" 39100

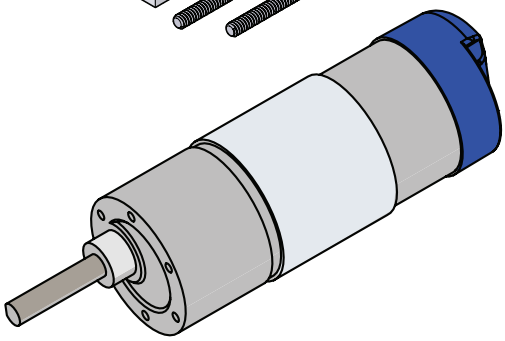
Axle Spacer 3/8" 39101

Motion Elements

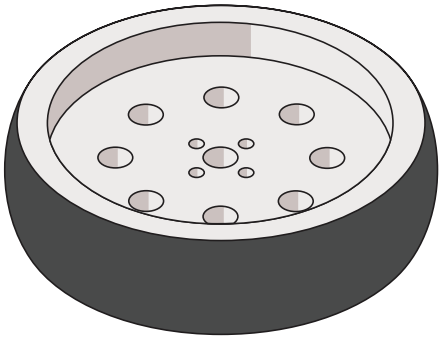
Motor Mount 39089



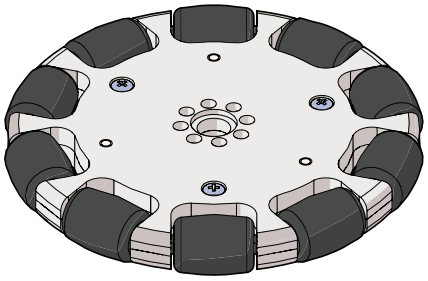
TorqueNADO Motor 44260



4" Wheel 39055



4" Omni Wheel Pack/Assembly 36466



Kep Nut 39094

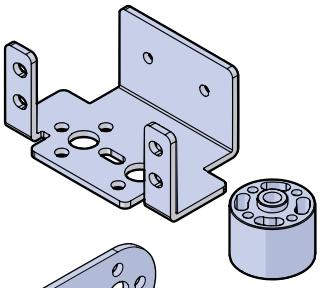
Socket Head Cap Screw 6-32 x 1/2" 39097

Socket Head Cap Screw 6-32 x 5/16" 39098

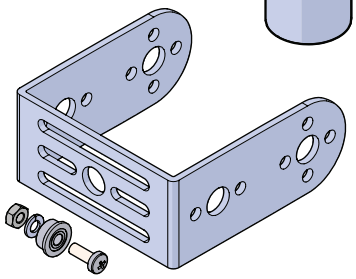
Button Head Cap Screw 3/8" 39111

Motion Elements

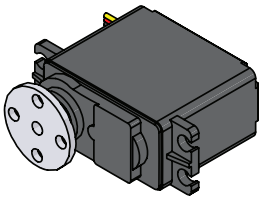
Single Standard-Scale Servo Motor Bracket 39060



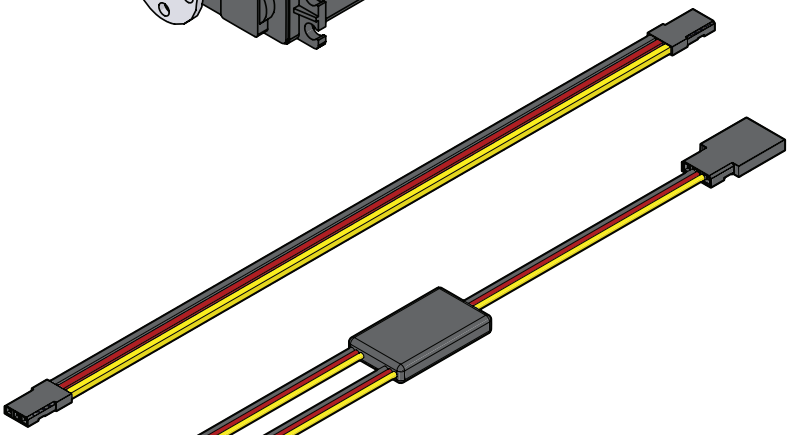
Standard-Scale Pivot Arm with Bearing 39593



Standard-Scale Servo Motor 39197

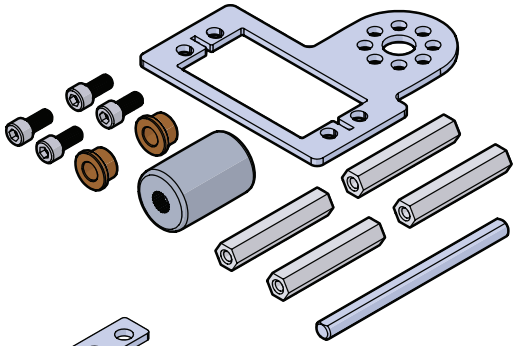


Servo Extension 39081

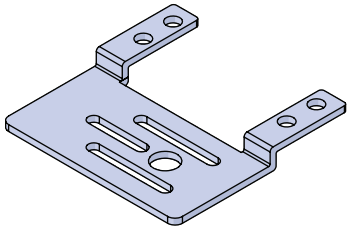


Servo Y Connector 39082

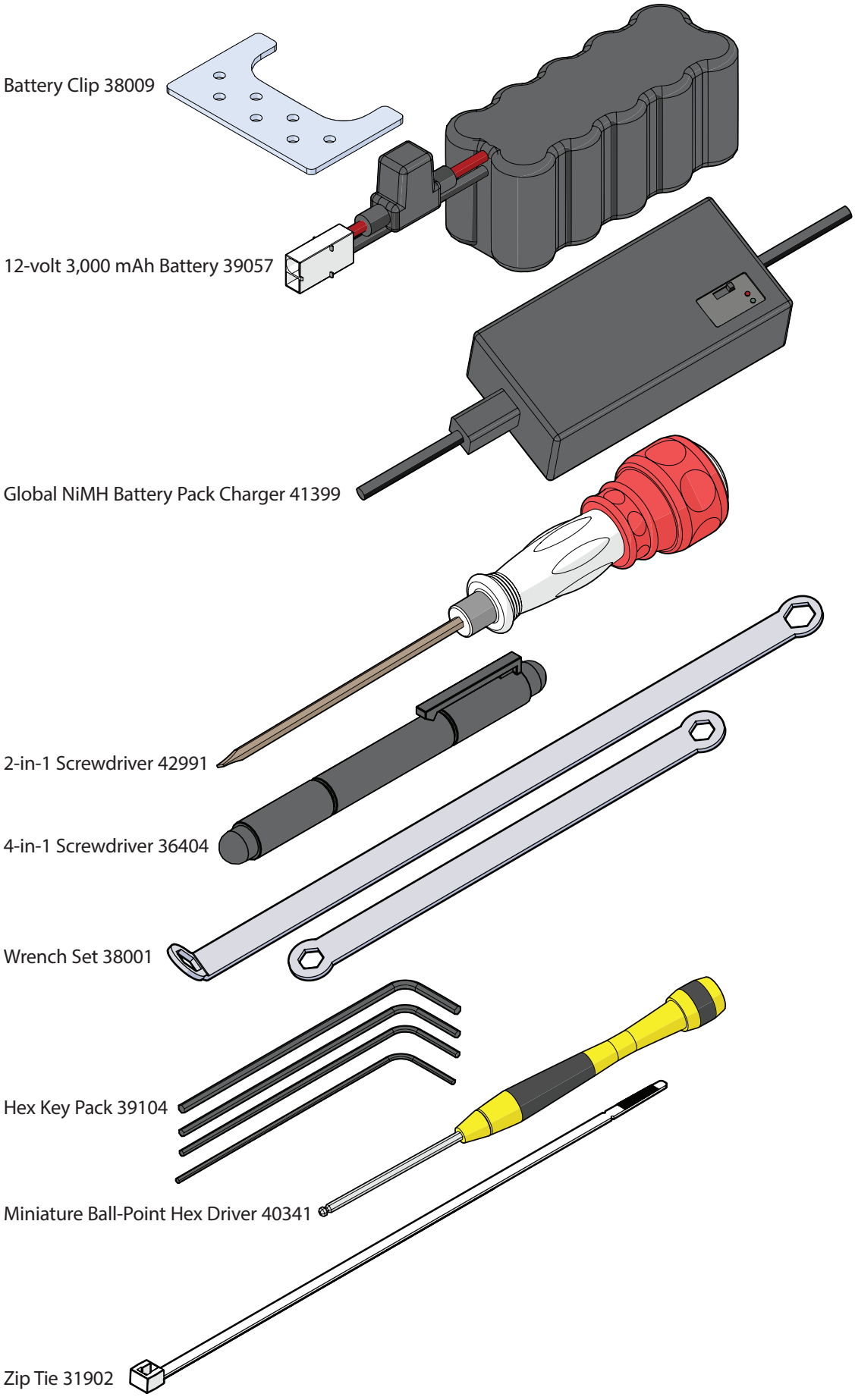
Standard Servo Mounting Kit 41789



Adjustable Servo Bracket 39280

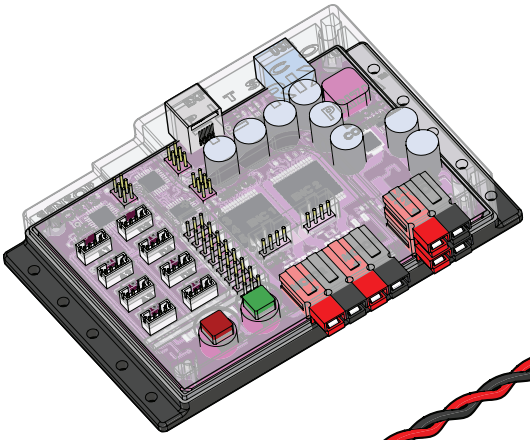


Power, Tools, and Accessories Elements



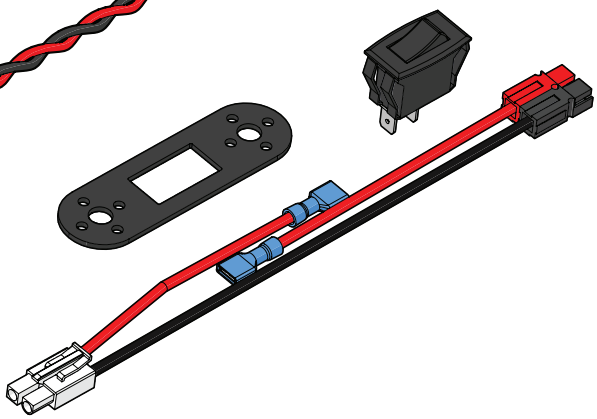
Control Elements

PRIZM Robotics Controller 43000

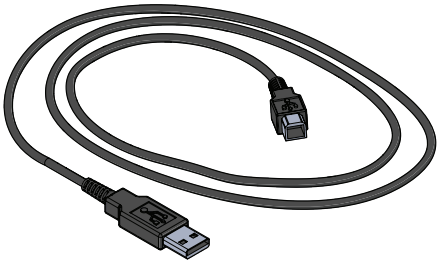


Motor Cable with Powerpoles 41352

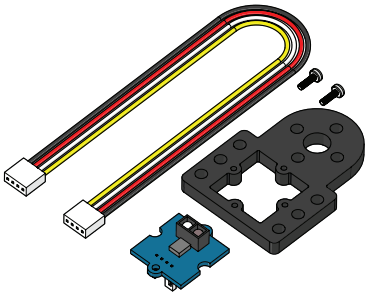
PRIZM Controller On/Off Battery Switch Adapter 43169



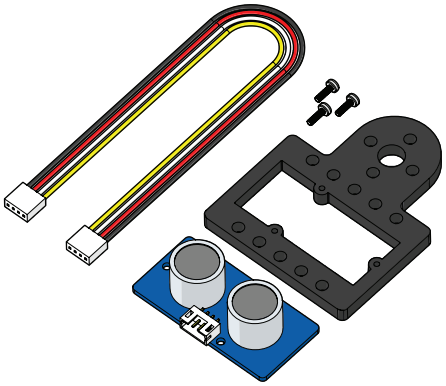
3-Foot Type A-B USB Cable 40967



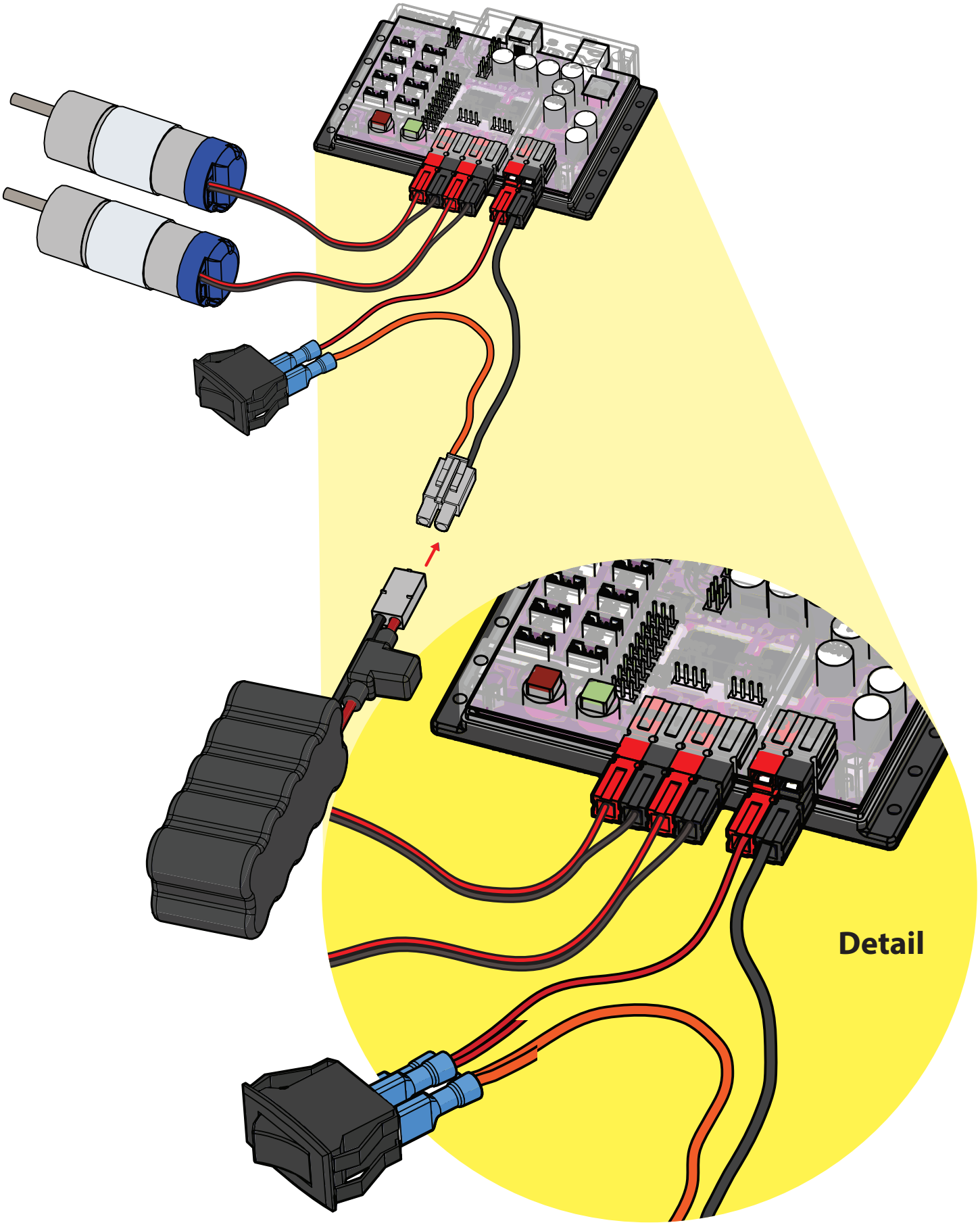
Line Finder Sensor Pack 43056



Ultrasonic Sensor Pack 43055



PRIZM Controller Wiring Illustrated



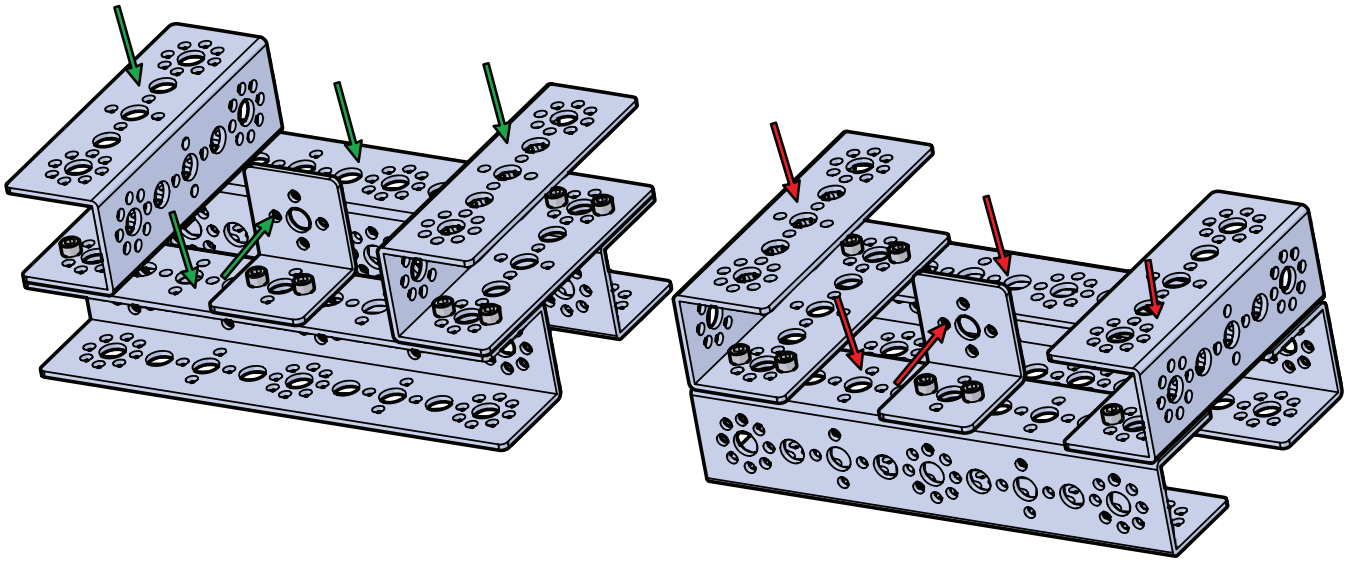
TETRIX MAX Builder's Guide Setup/Construction Tips

Planning for easy access to fasteners will make the building experience better for everyone involved.

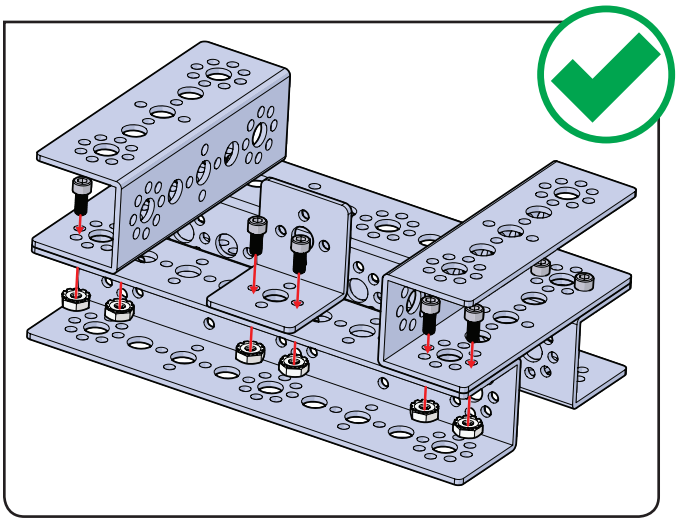
As a general rule, when you create any substructure, it's a good idea to only snug nuts and screws until you are sure all elements fit together. Then, go back and tighten everything before moving on to the next step.

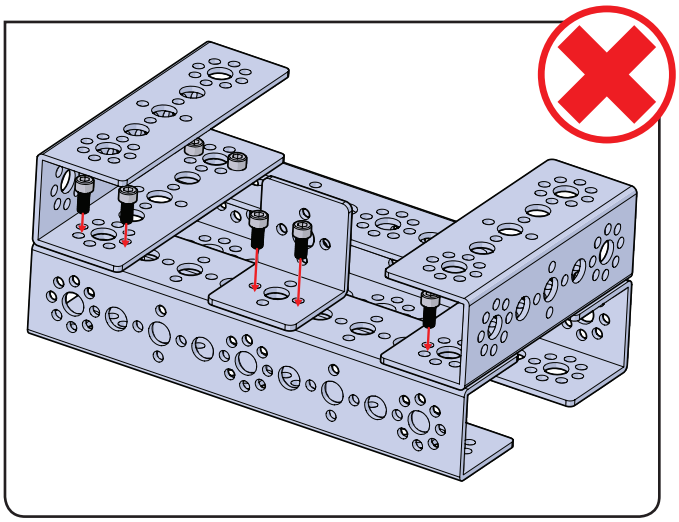
1. Channel Placement

A little planning and forethought about how structural elements go together can make your building experience easier, quicker, and more efficient.



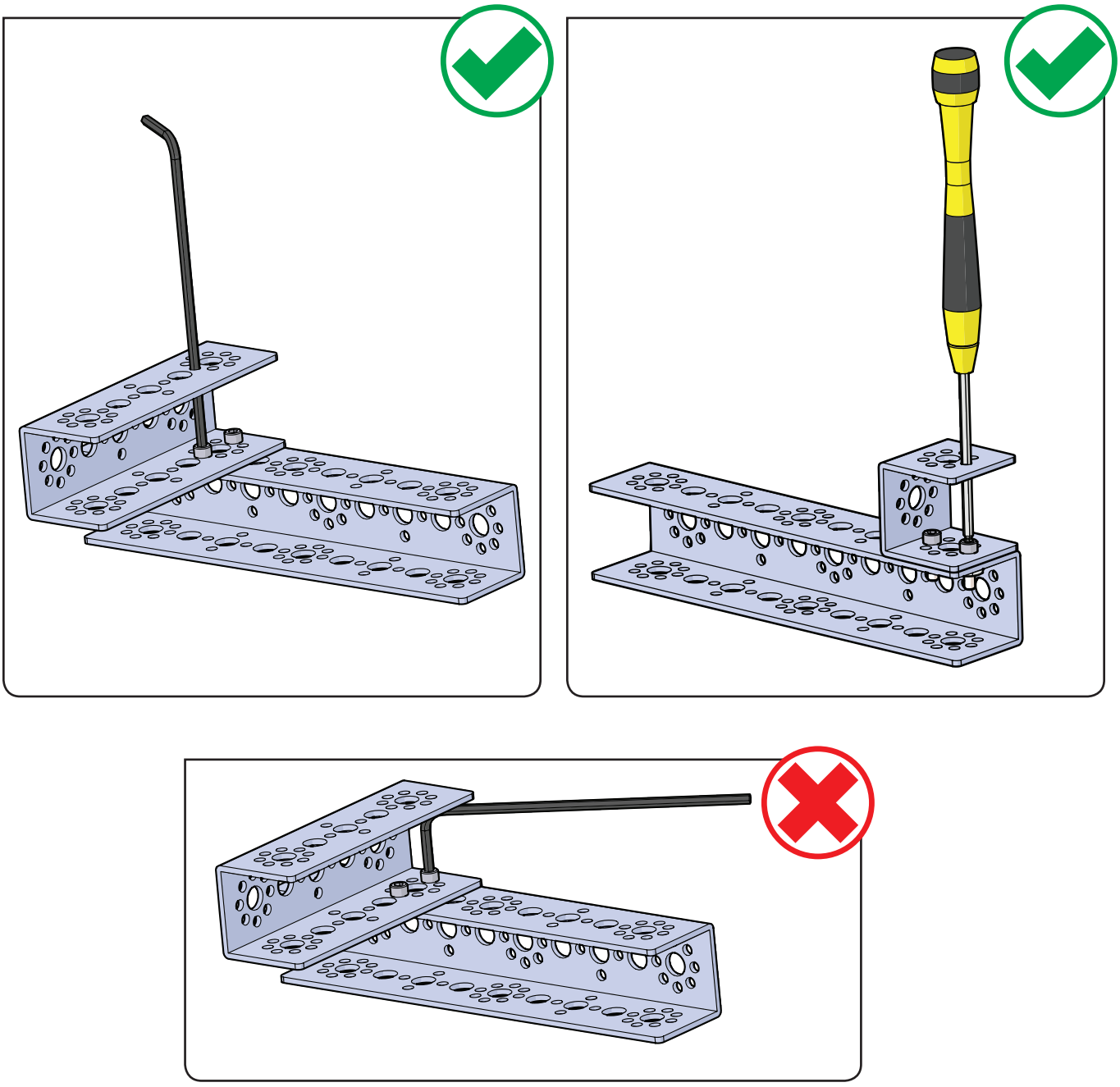
In the figure above, both structures have similar building surfaces to work from but have different accessibility for fastening the kep nuts on the screws. Easy access is preferred. Avoid difficult access if possible.





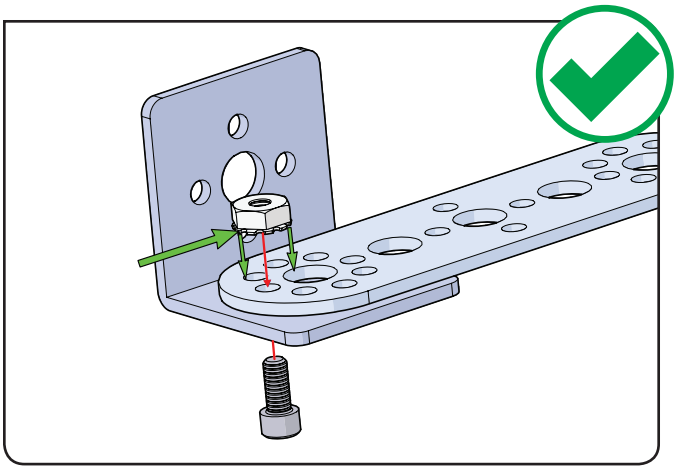
2. Tool Use

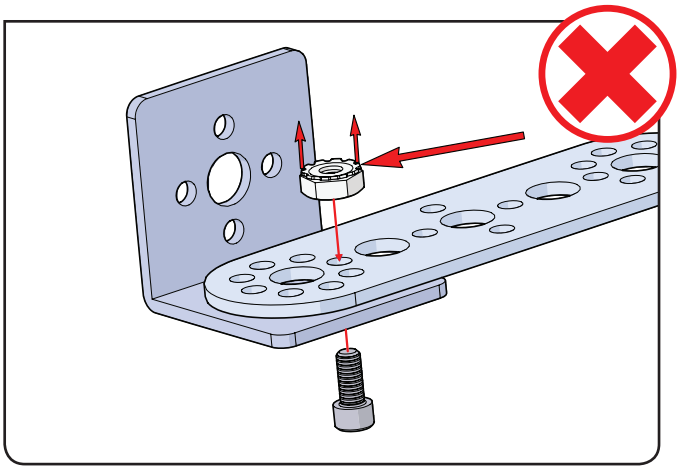
Proper use of the basic tools makes the building process smoother and more enjoyable and saves time.



3. Take Advantage of Design Features

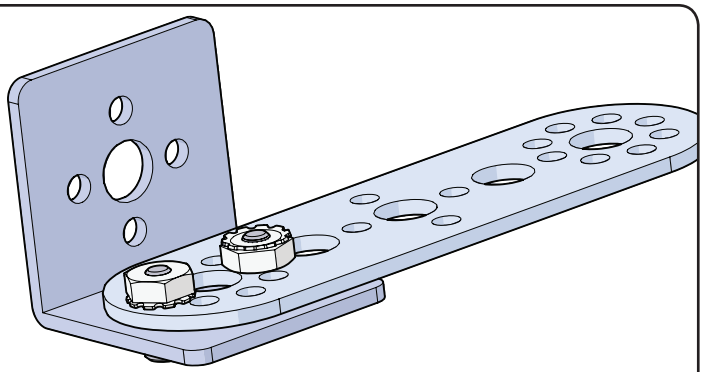
Some elements have design features that either make the element function better or fill a specific role. Recognizing those so you can take full advantage will make your build be stronger, last longer, and function better.





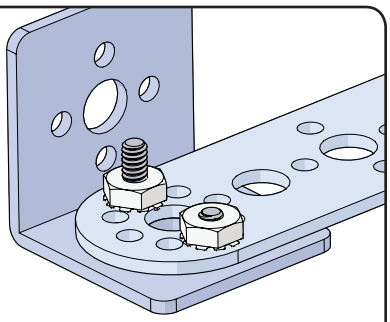
This is the proper orientation for kep nuts. The self-locking feature of the kep nut should always be against the flat surface of the structural element, such as in the example on the left. The image on the right is wrong!

Below is a view of both together for comparison.



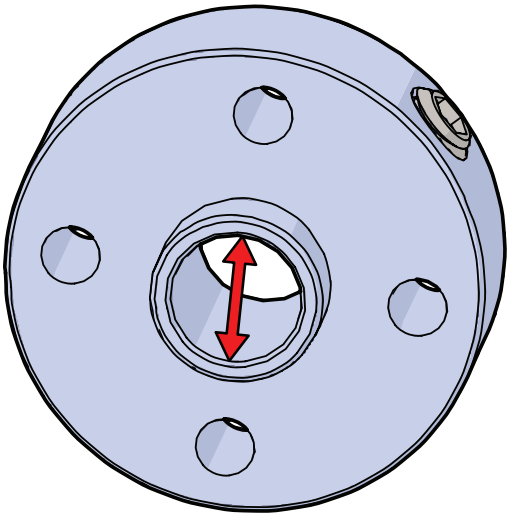
The orientation of the kep nut on the left is correct.
The orientation of the kep nut on the right is wrong!

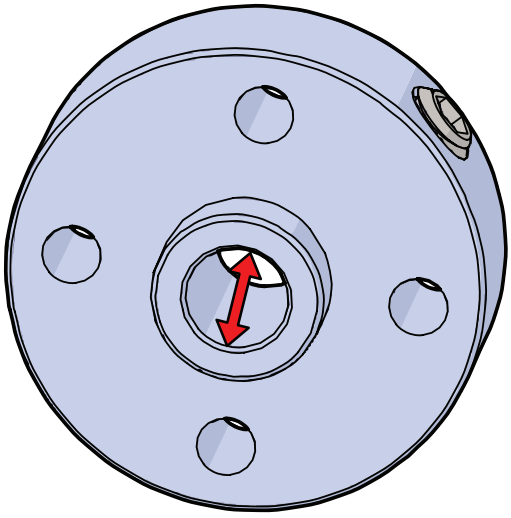
Using the right-length screw for the job will make best use of your available resources. Use the shortest screw possible to get the job done, and save the longer screws for the places where they are necessary.



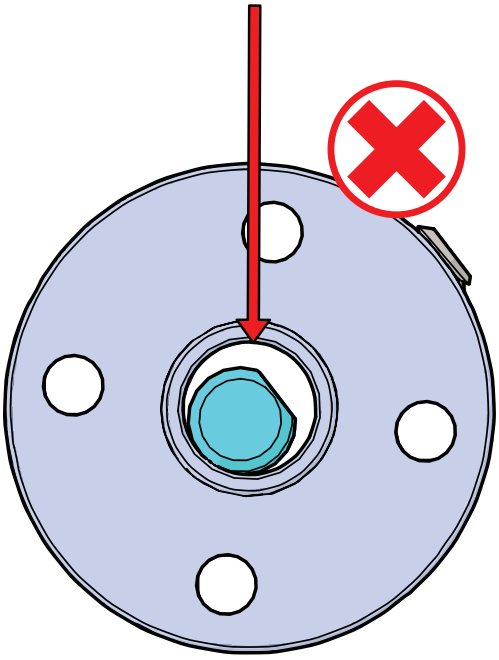
While either screw in the image above would work, the best use of resources would dictate using the one shown on the right.

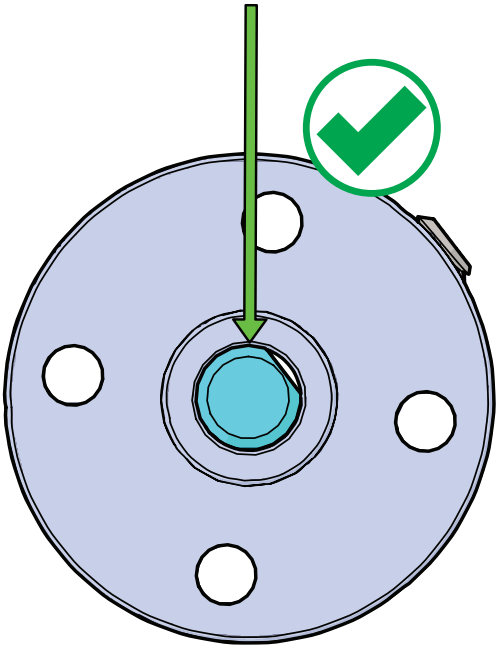
Axle hubs and motor shaft hubs are often confused because they look alike and they serve the same function but in slightly different applications.





The difference between the two elements is the size of the inside diameter of the hole in the center of the element. The axle hub is sized for the axle, while the motor shaft hub is sized for the motor shaft.

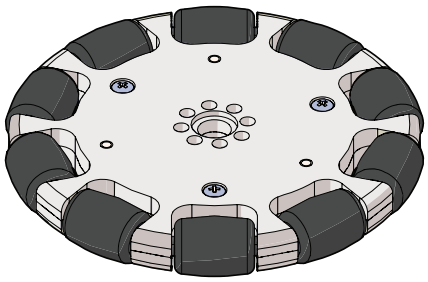




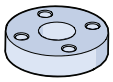
While axle hubs can be used only on axles because of size, the motor shaft hub could be incorrectly used on an axle and cause problems with function and attachment of wheels or gears. The image on the left shows an axle in the center of a motor shaft hub. Notice the difference in size between the outer diameter of the axle and the inside diameter of the motor shaft hub. The image on the right shows an axle in the center of the axle hub. Notice the elements are sized correctly to fit together and function properly.

4. Assembly of Specialty Parts

Omni Wheel Setup Parts Needed



2x
4" Omni Wheel Pack/
Assembly 36466

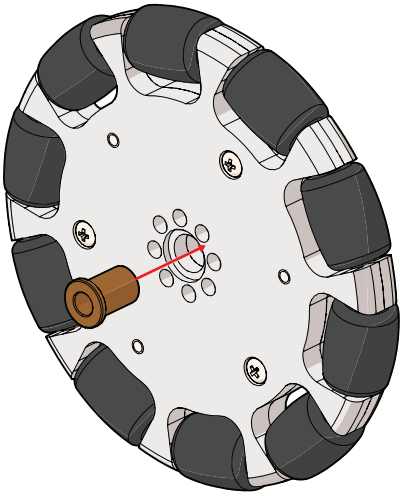


1x
Omni Wheel
Spacer 36466

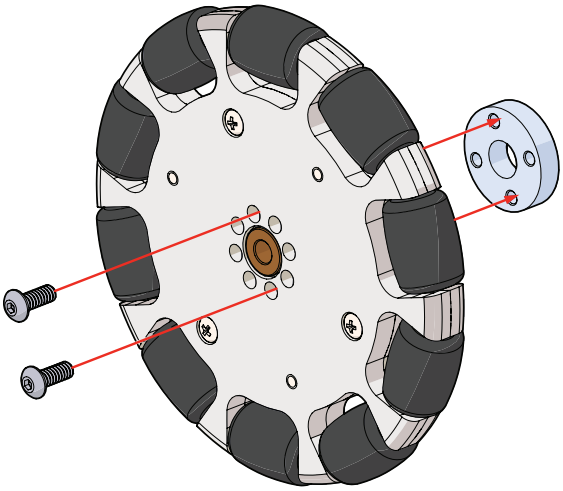
2x
Bronze Bushing
39091

4x
Button Head Cap
Screw 3/8" 39111

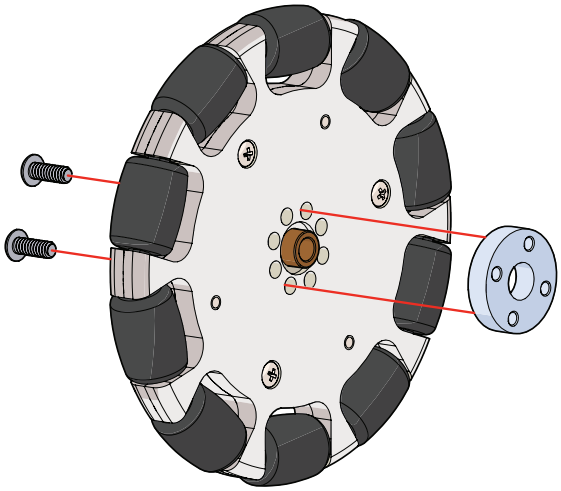
Step 1.0



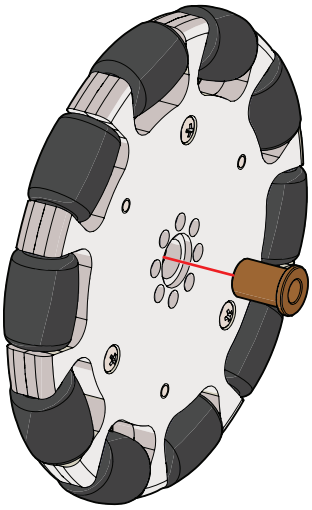
Step 1.1 (front view)



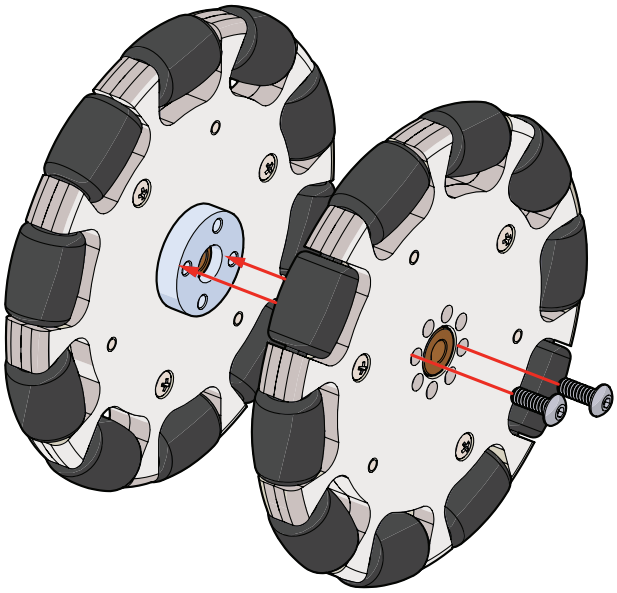
Step 1.1 (back view)

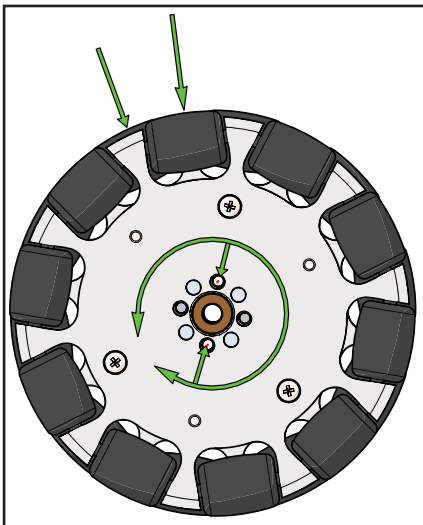


Step 1.2

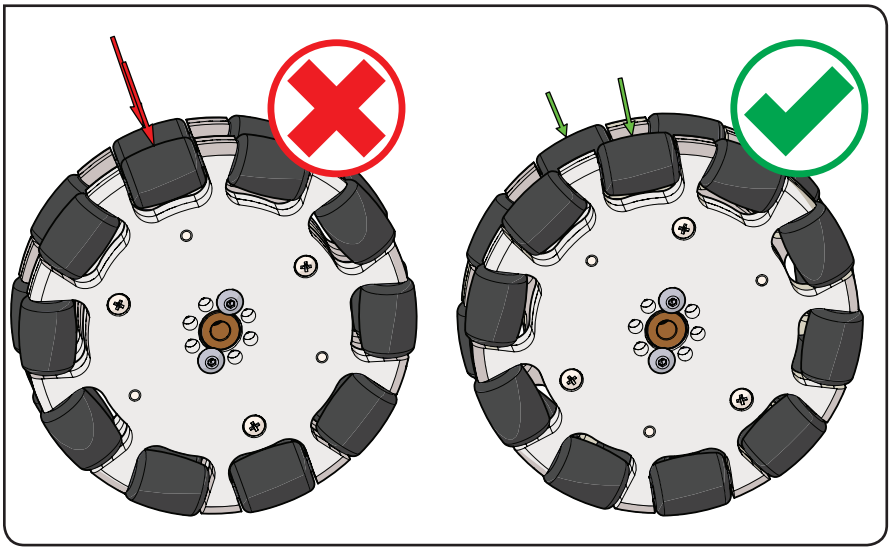


Step 1.3





Tip: To get proper offset of rollers between the two wheels, remove screws from Step 1.3 and rotate the wheel shown in either direction. Reattach screws.



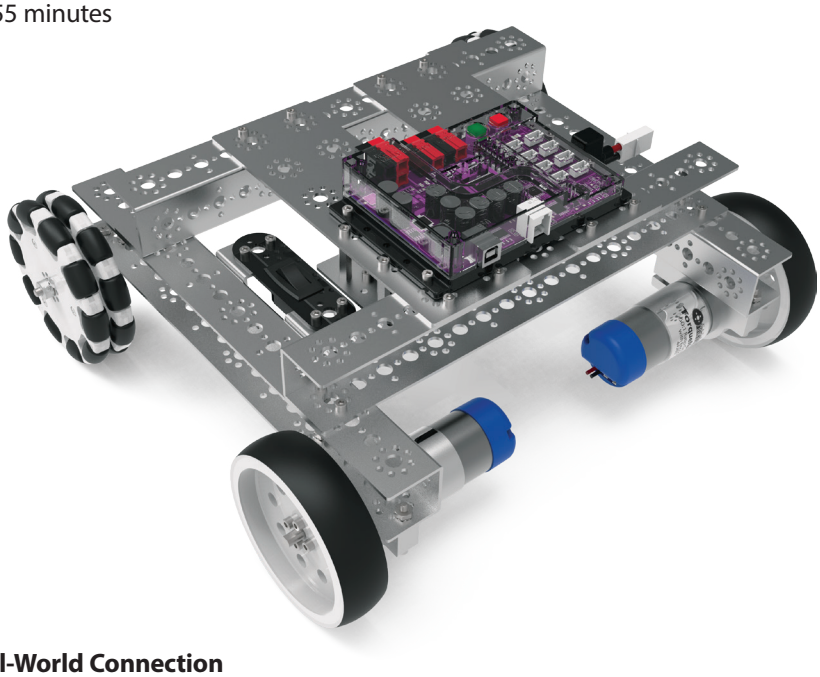
Activity 6: Build the TaskBot

We need a robot. Because the focus of this guide is on working with PRIZM and the *Arduino Software (IDE)*, we do not need a complicated robot. With that in mind, we created the PRIZM TaskBot. The TaskBot is meant to be simple, easy to build, and exactly what we need for the purposes of this guide without any unnecessary parts. That being said, everything you learn with this basic bot can be transferred to a more complicated bot as you continue on your robotics journey.

The TaskBot uses two 12-volt motors mounted back to back for a good example of a differential-drive robot. Two omni wheel assemblies on the opposite end combined with the maneuverable drive make a perfect test vehicle for our work with PRIZM. We will start with a basic drive frame and add more to it in later activities.

This build should be perfect for students who have little to no experience with metal-based building systems.

Building Time Expectations
45-55 minutes



Teacher's Note: Many factors can affect building time, including such things as set organization and whether the builder has a partner. The above time is only an estimate and is based on an individual builder of average experience who is comfortable with hands-on building activities and has access to complete, well-organized sets. Actual time might vary.

Real-World Connection

The construction of a machine will determine its effectiveness as well as its durability. Engineers design machines (including robots) for specific purposes. For instance, a bulldozer is built from very strong and durable materials to be able to withstand the forces involved in pushing over trees and buildings or digging out large holes in the ground.

STEM Extensions

Science

- Structure and function

Technology

- Materials
- Fasteners

Engineering

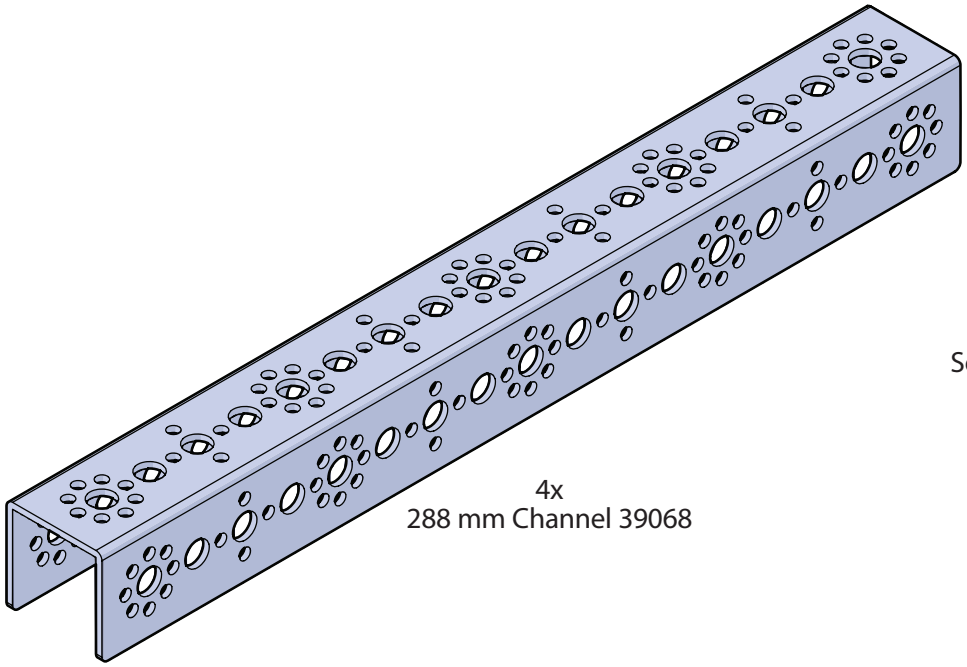
- Machine design

Math

- Measurement
- Modulus

Step 1

Parts Needed



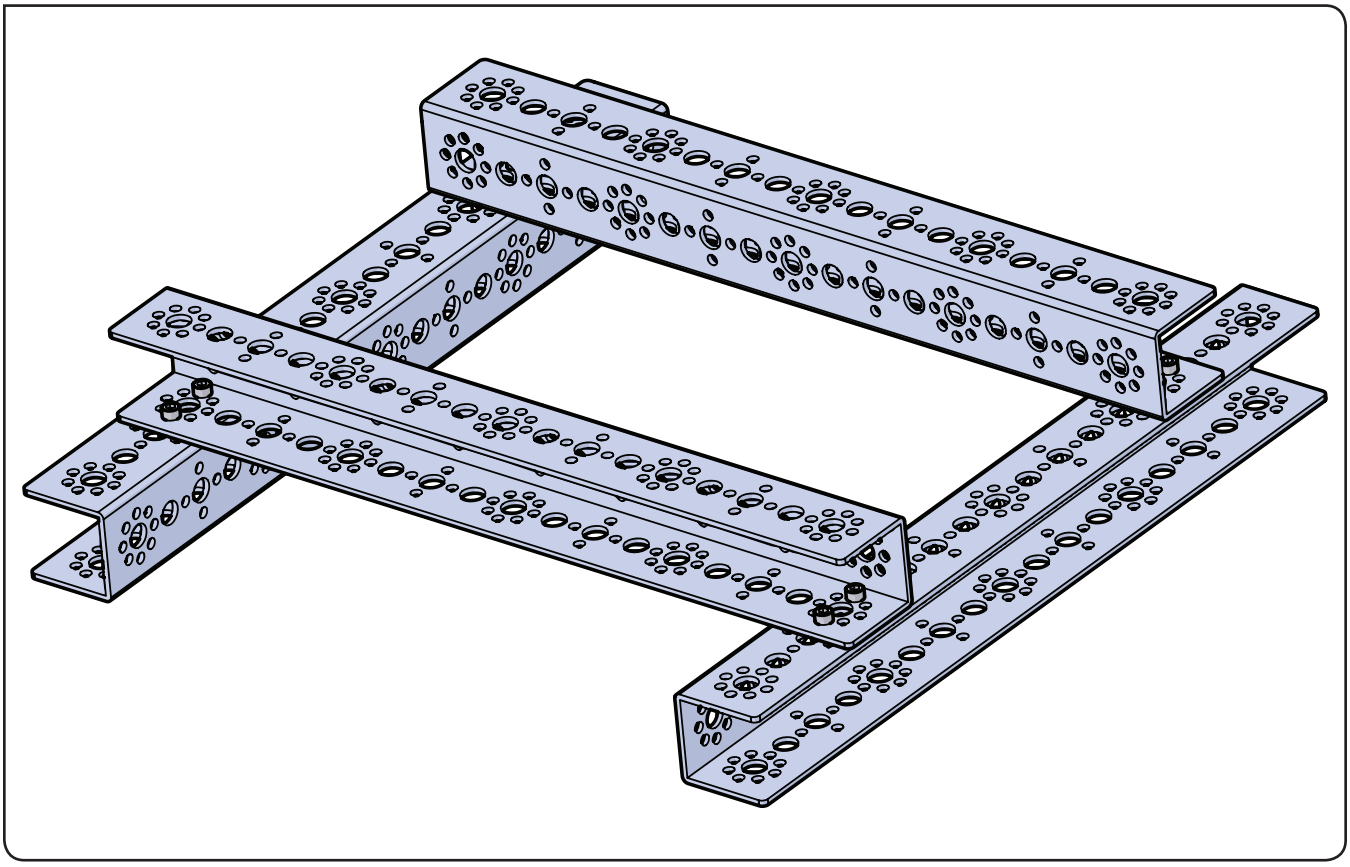
4x
288 mm Channel 39068

8x
Socket Head Cap Screw
6-32 x 5/16" 39098

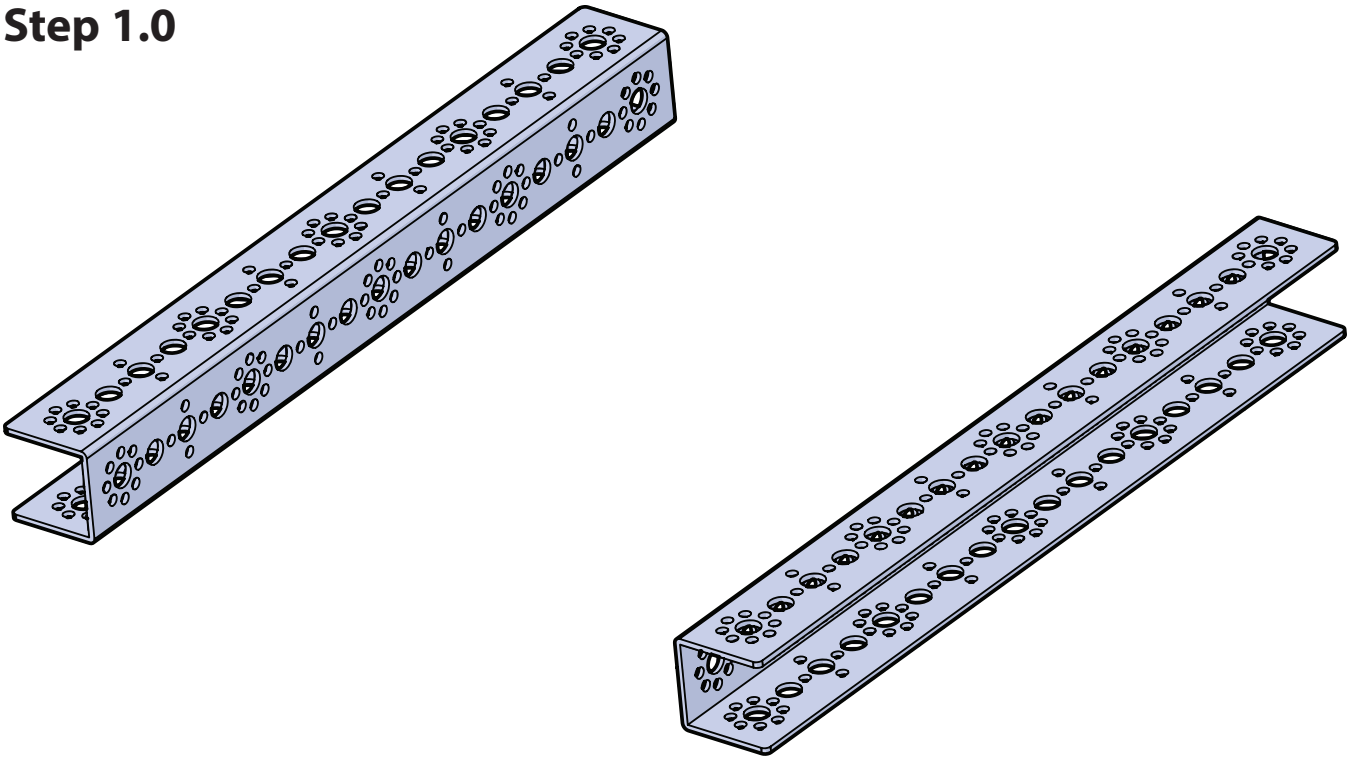
8x
Kep Nut 39094

Tip: See page 36 for help with identifying channel elements. Remember, channels are identified by length.

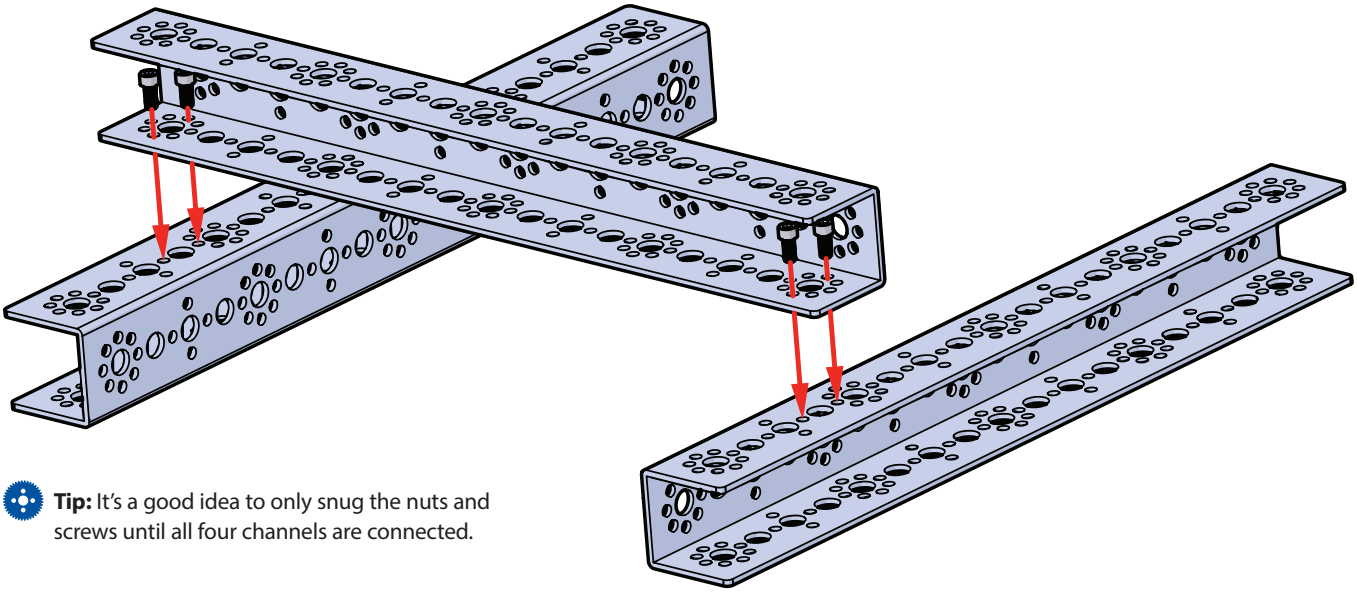
Partial assembly should look like this.



Step 1.0

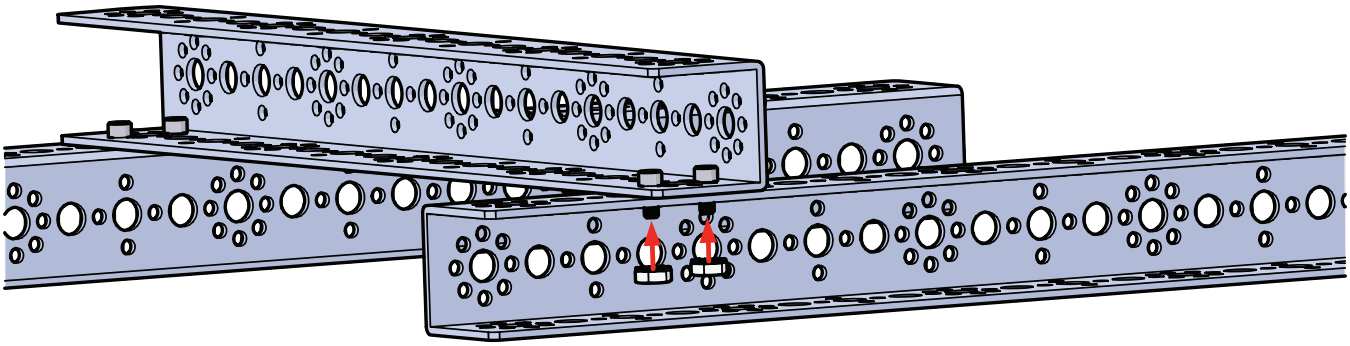


Step 1.1

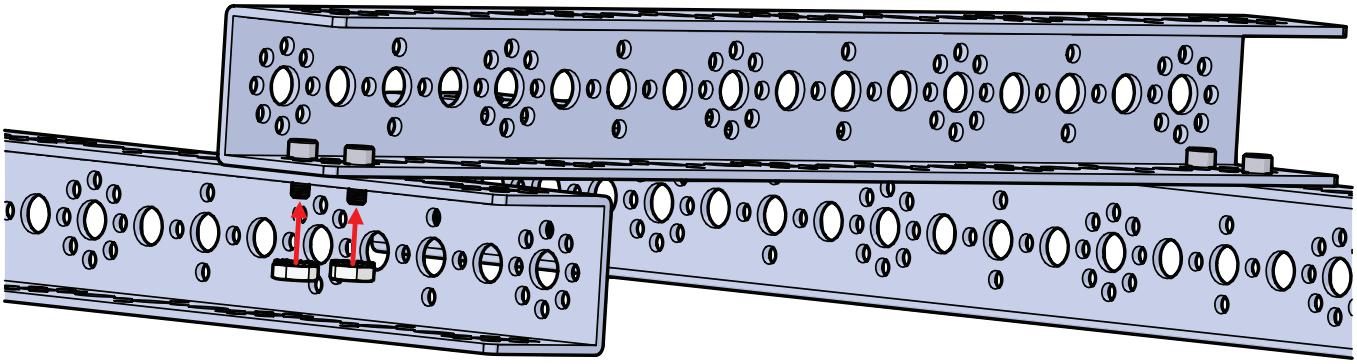


Tip: It's a good idea to only snug the nuts and screws until all four channels are connected.

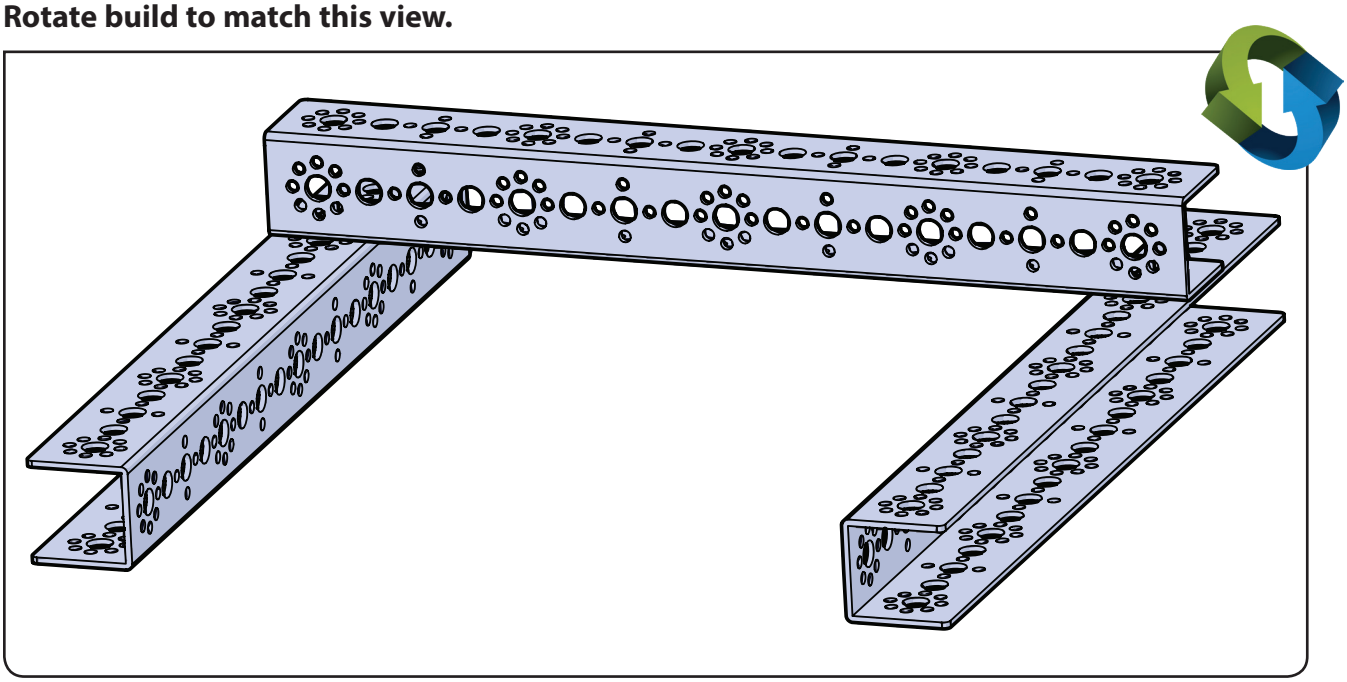
Step 1.2



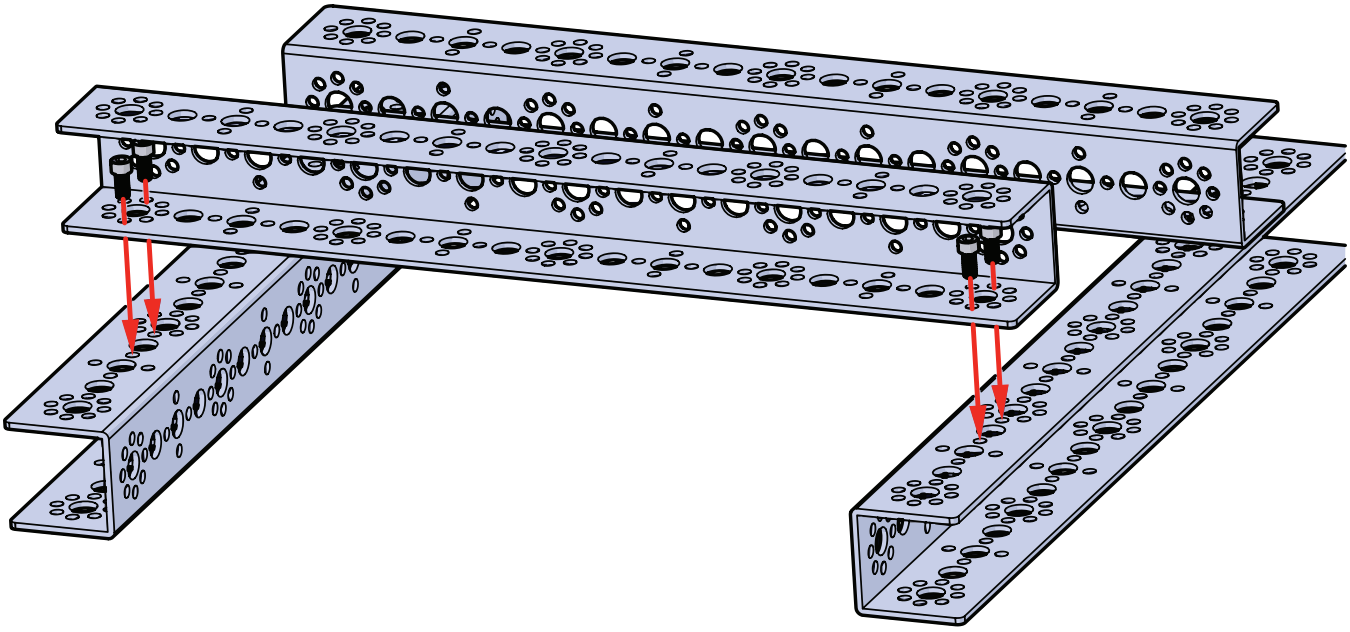
Step 1.3



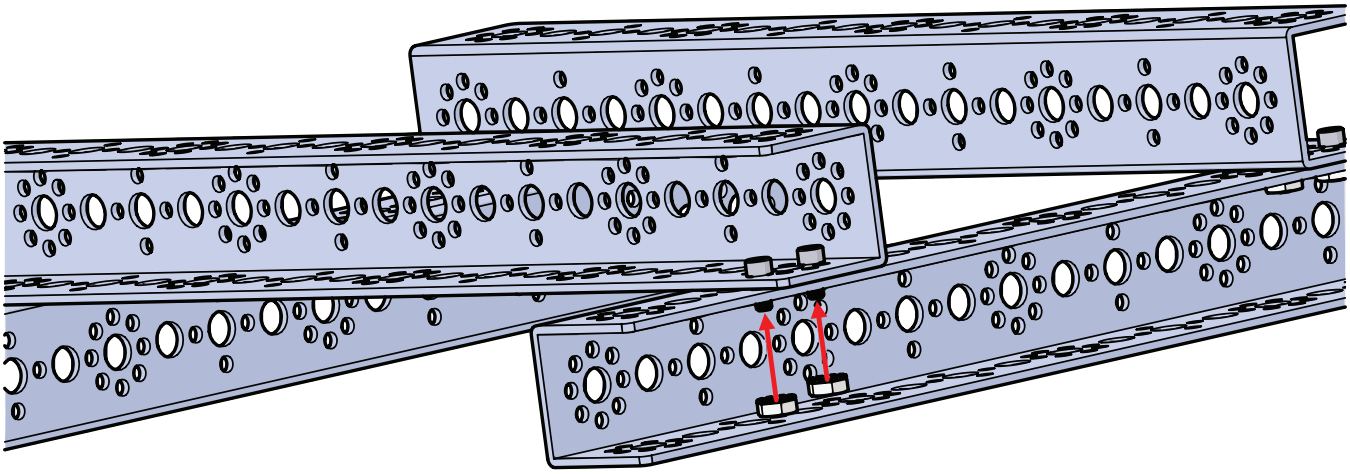
Rotate build to match this view.



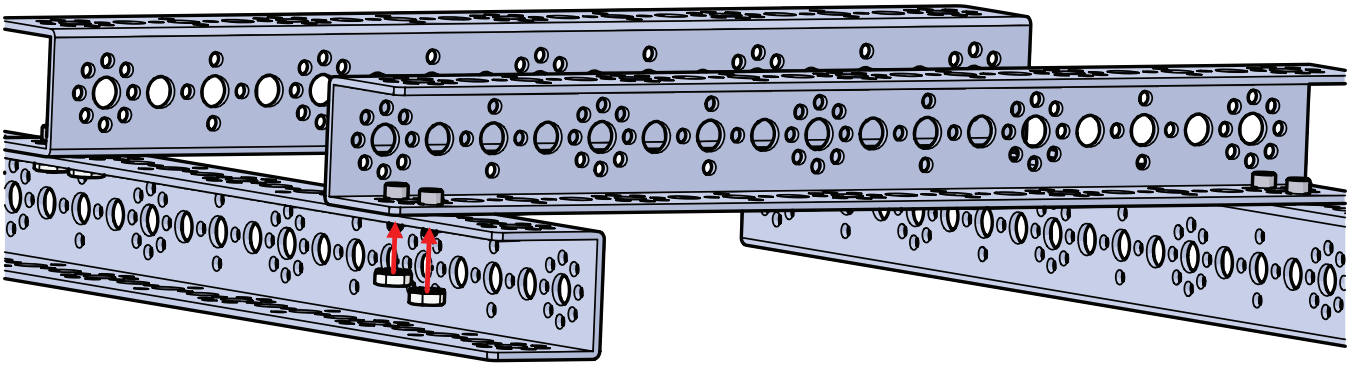
Step 1.4



Step 1.5



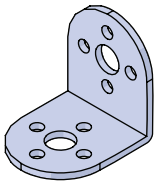
Step 1.6



Tip: After all four channels are connected and the square frame is created, don't forget to go back and tighten all the screws and nuts.

Step 2

Parts Needed

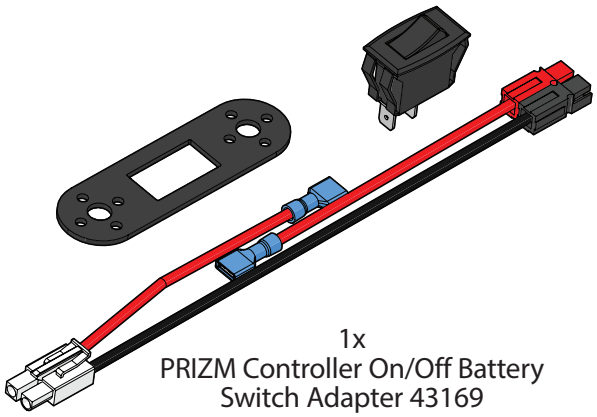


2x
L Bracket 39062

8x
Kep Nut
39094

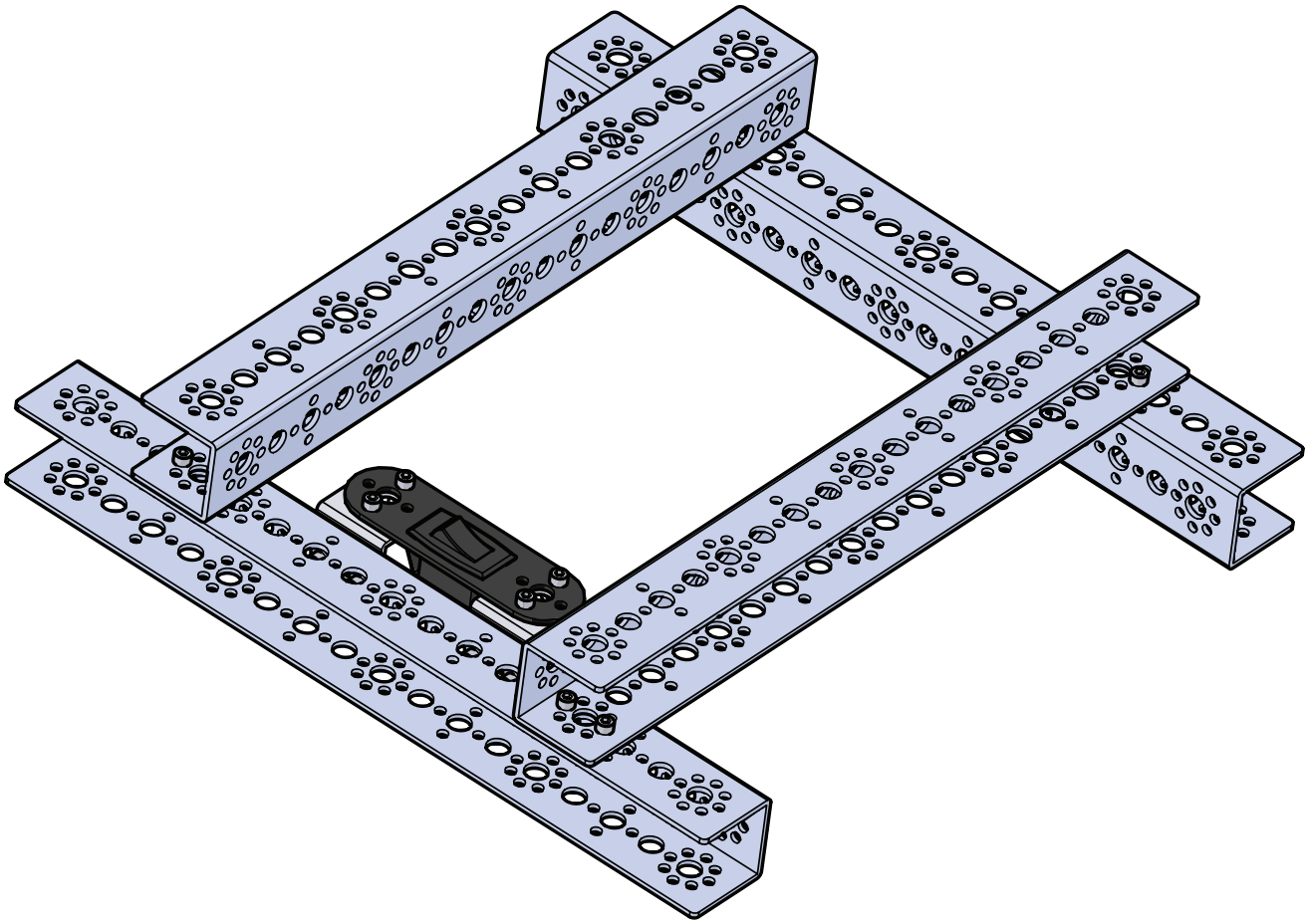
4x
Socket Head Cap Screw
6-32 x 1/2" 39097

4x
Socket Head Cap Screw
6-32 x 5/16" 39098

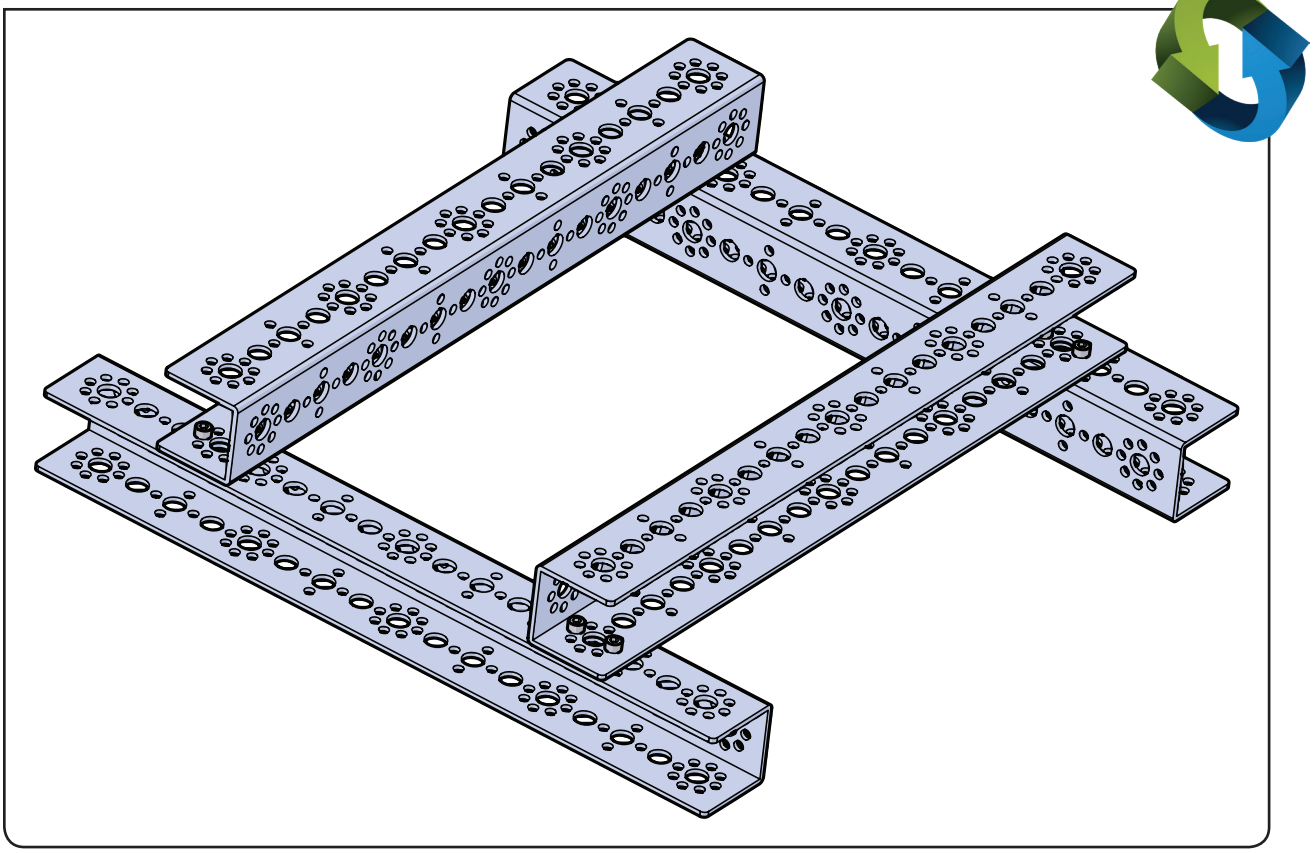


1x
PRIZM Controller On/Off Battery
Switch Adapter 43169

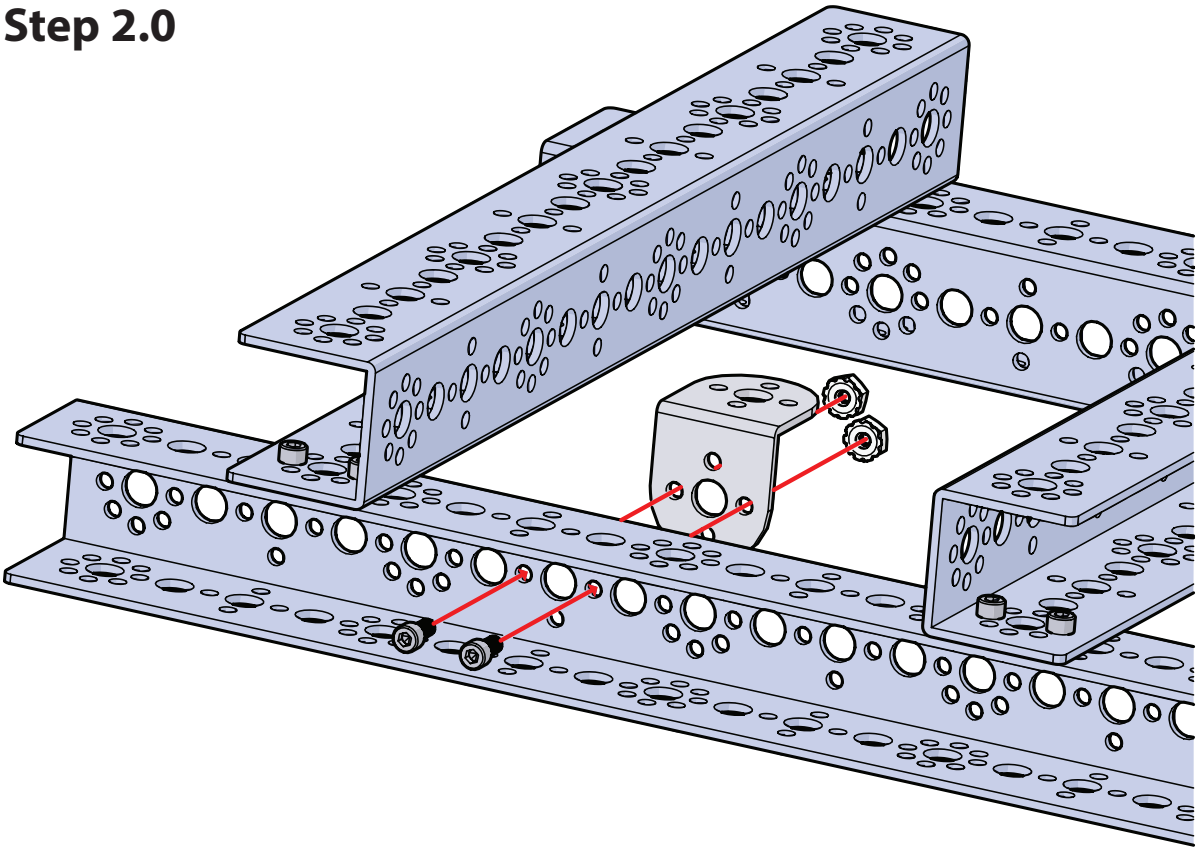
Partial assembly should look like this.



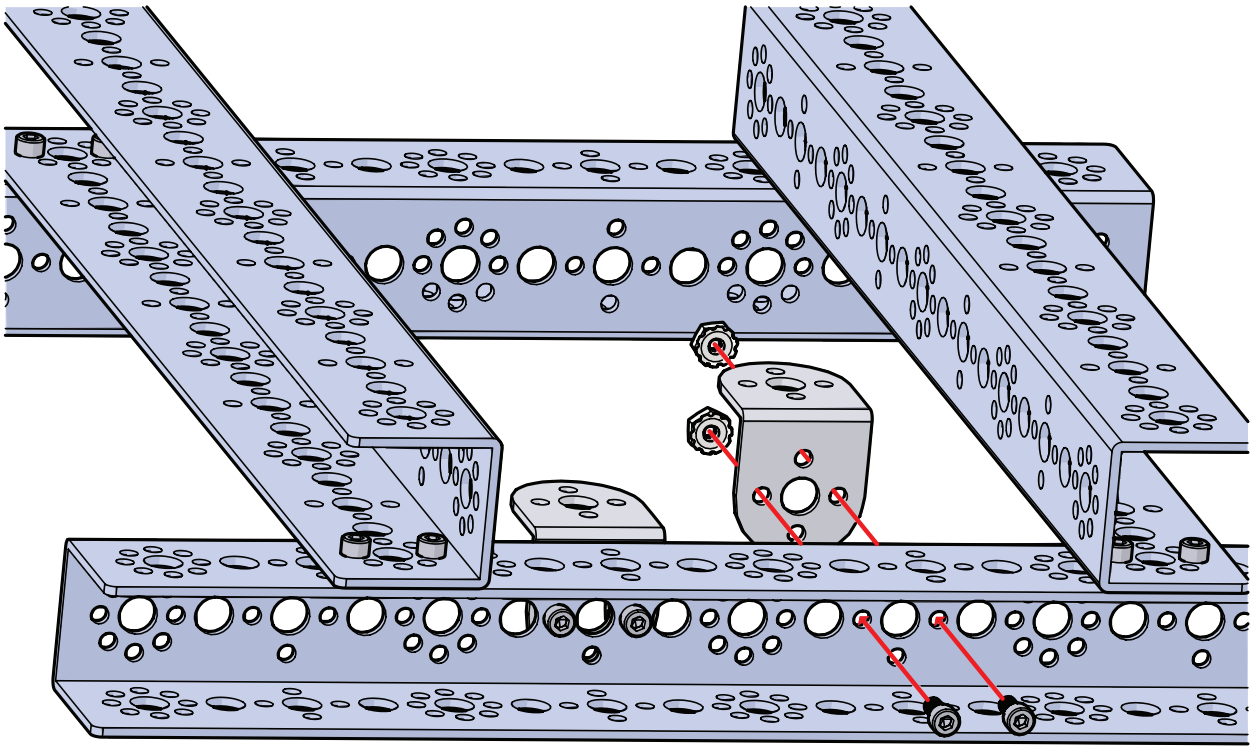
Rotate build to match this view.



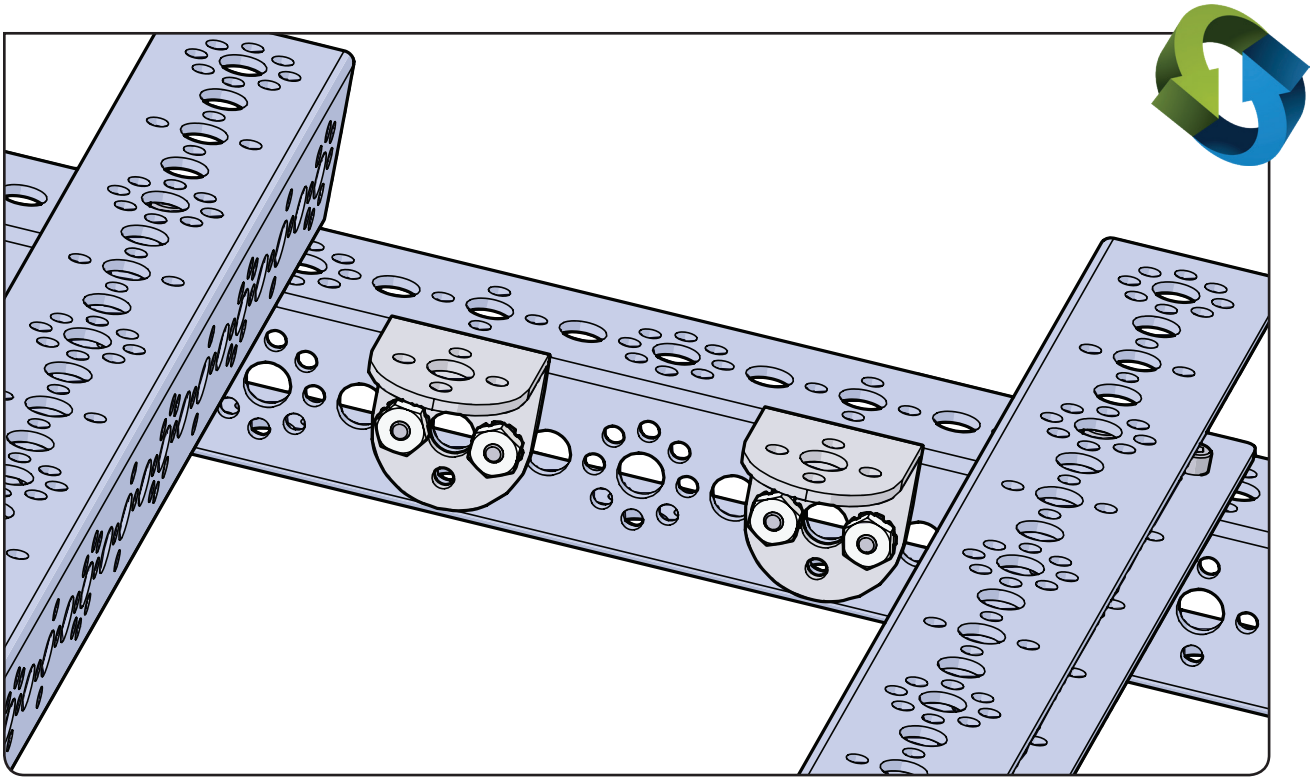
Step 2.0



Step 2.1

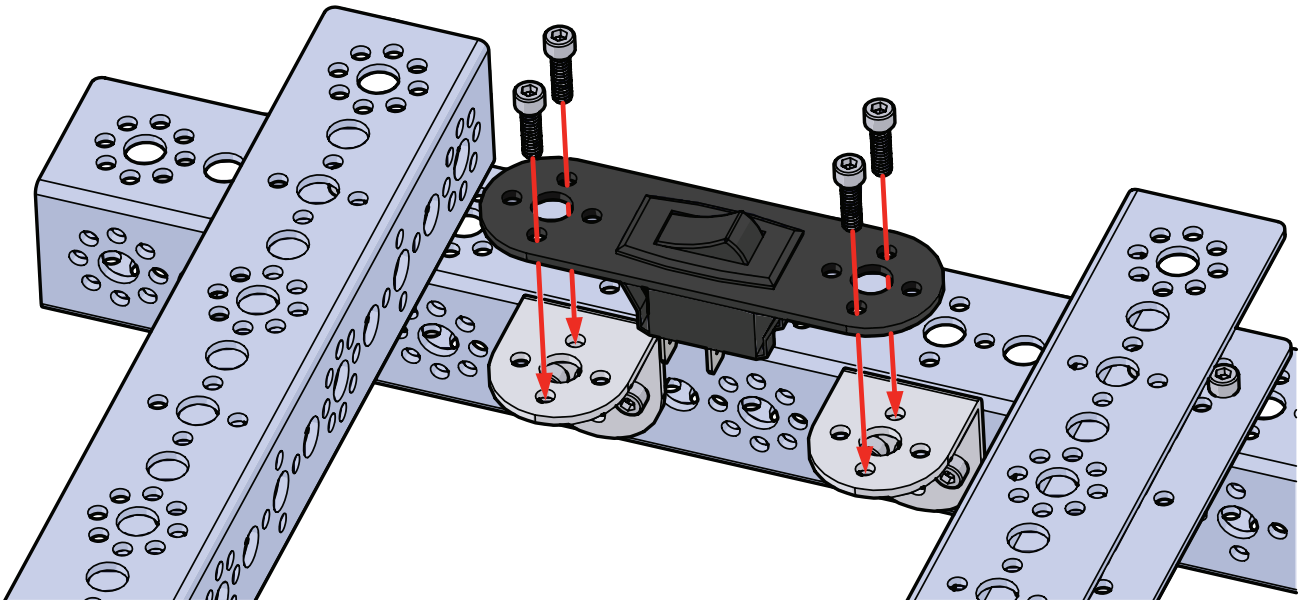


Rotate build to match this view.



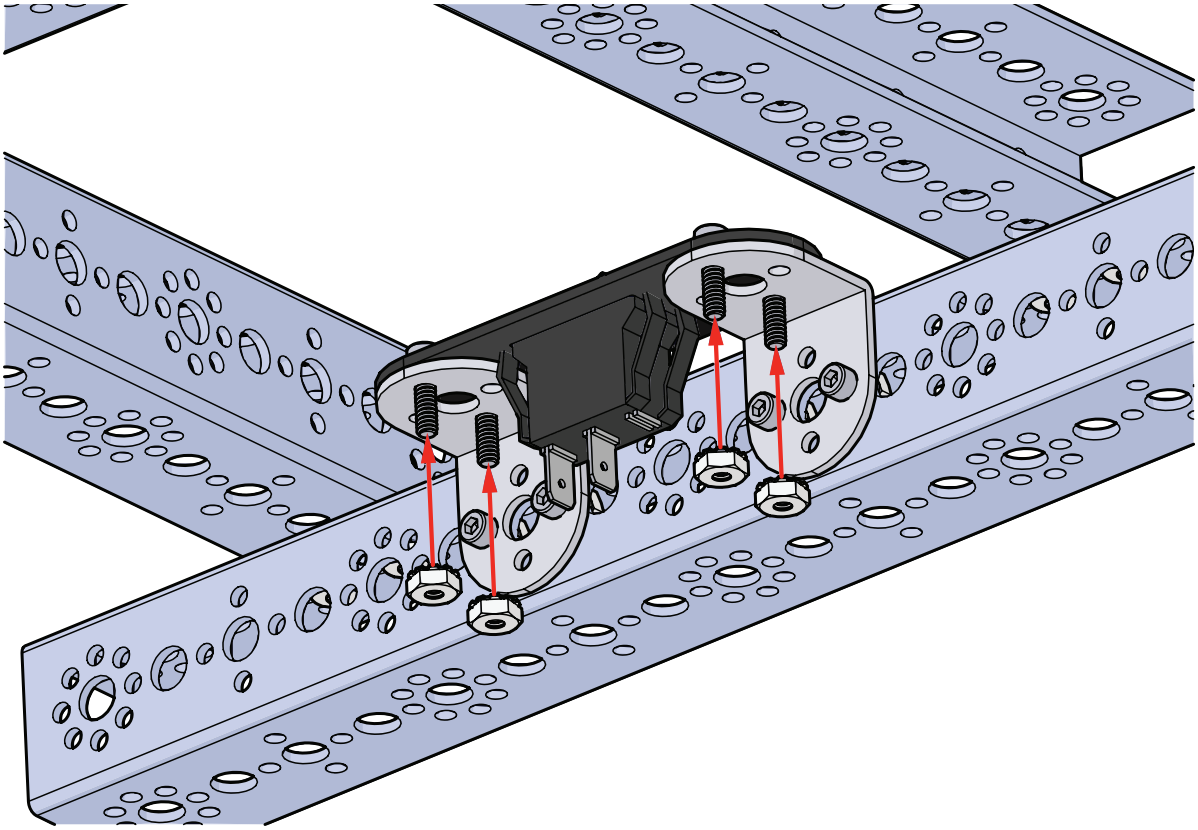
Step 2.2

Tip: This step uses Socket Head Cap Screws (39097).



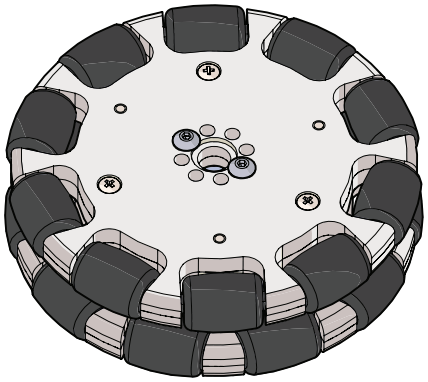
Tip: It does not matter in which direction the on/off power switch is mounted in the plate. It is strictly a matter of personal preference.

Step 2.3

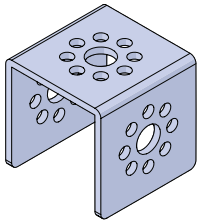


Step 3

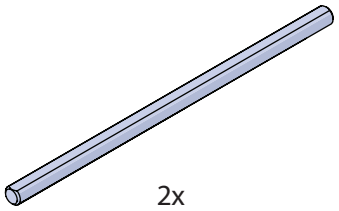
Parts Needed



2x
4" Omni Wheel Pack/Assembly
36466



2x
32 mm Channel
39065



2x
100 mm Axle
39088

4x
Axle Set Collar
39092

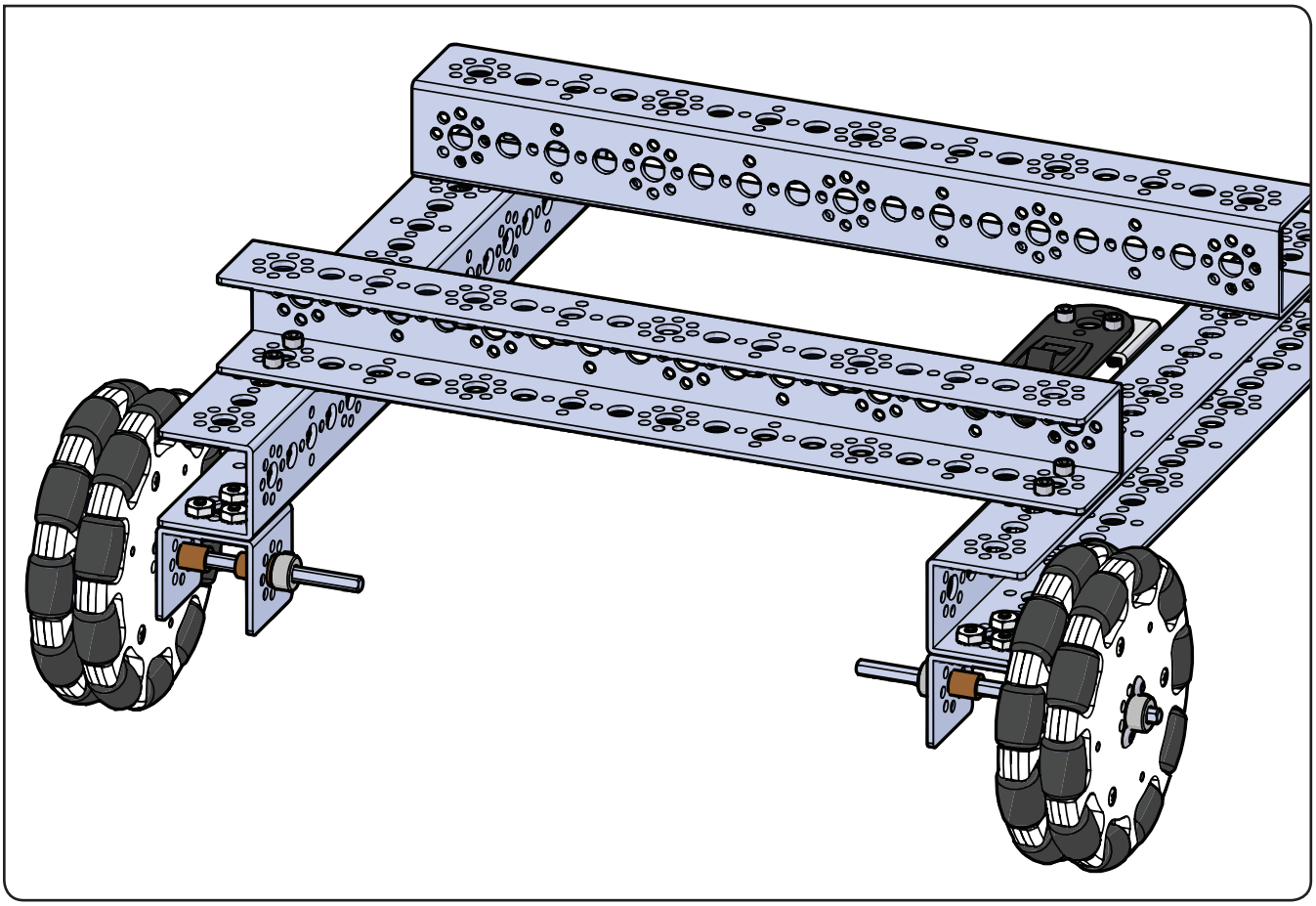
4x
Bronze Bushing 39091

6x
Socket Head Cap Screw
6-32 x 5/16" 39098

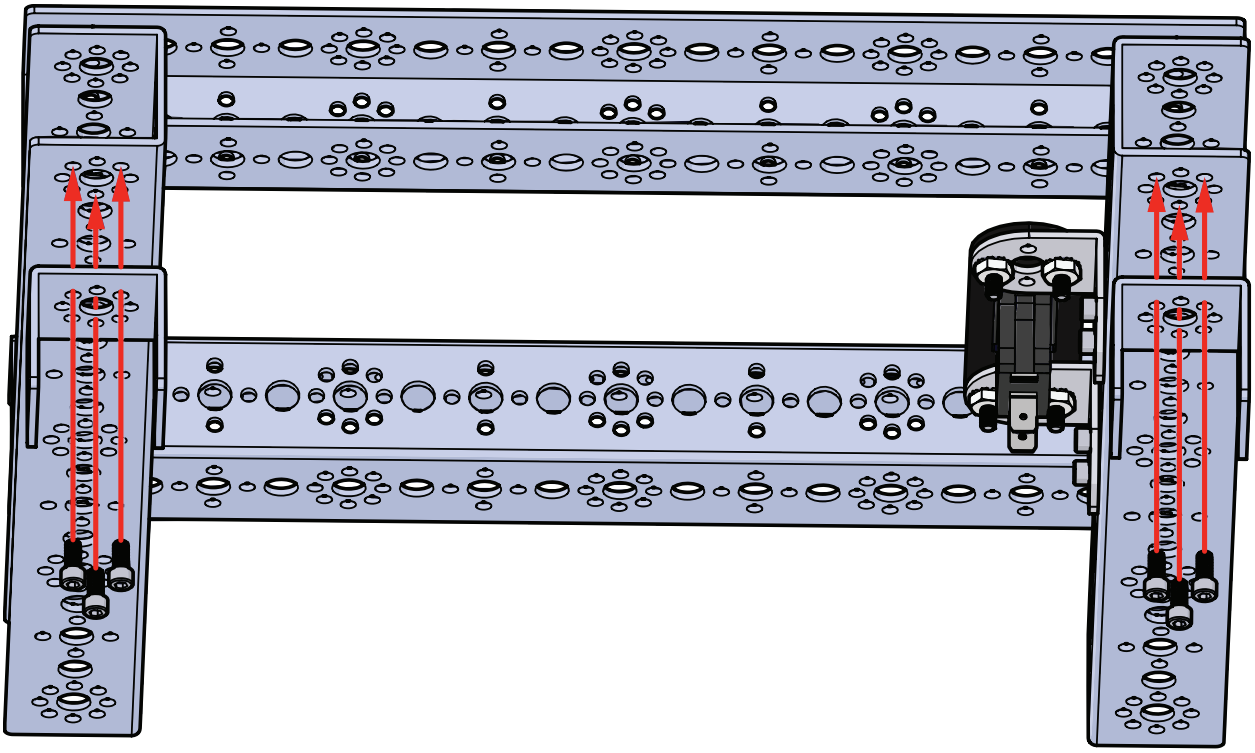
6x
Kep Nut 39094

2x
Axle Spacer 1/8" 39100

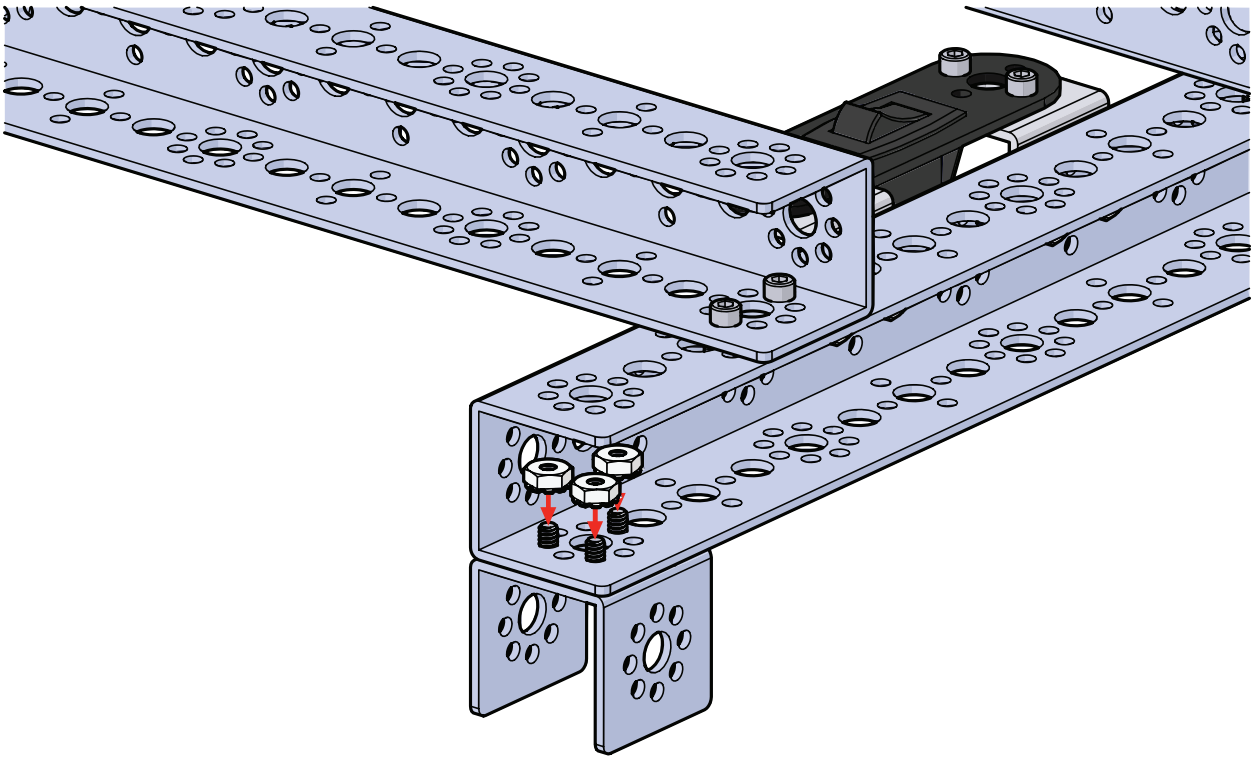
Partial assembly should look like this.



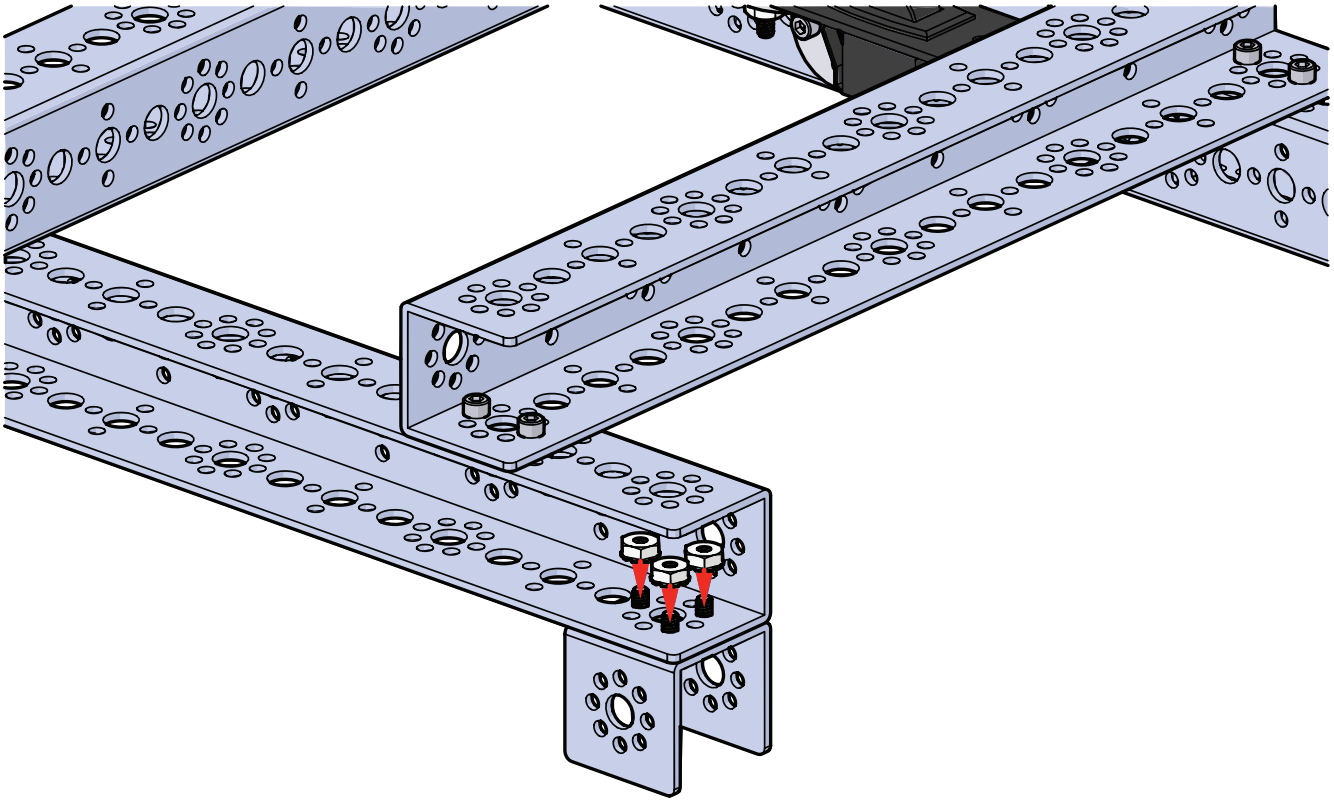
Step 3.0



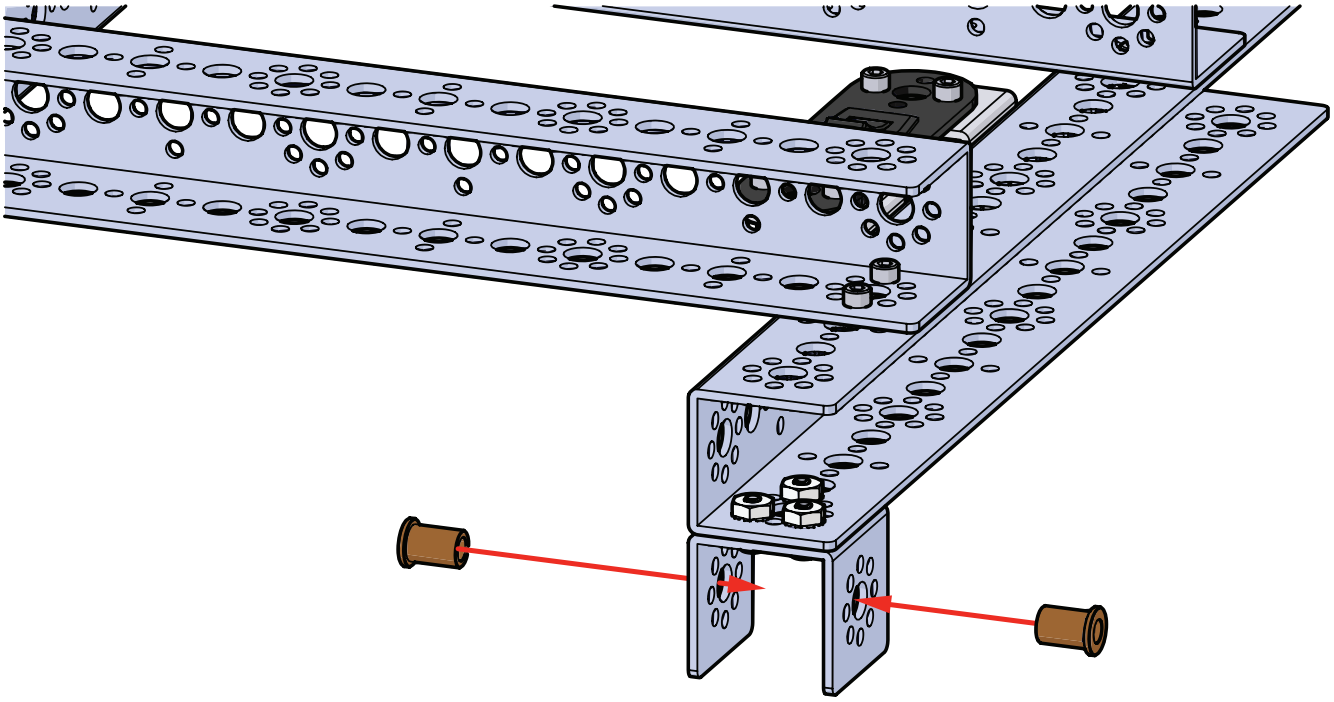
Step 3.1



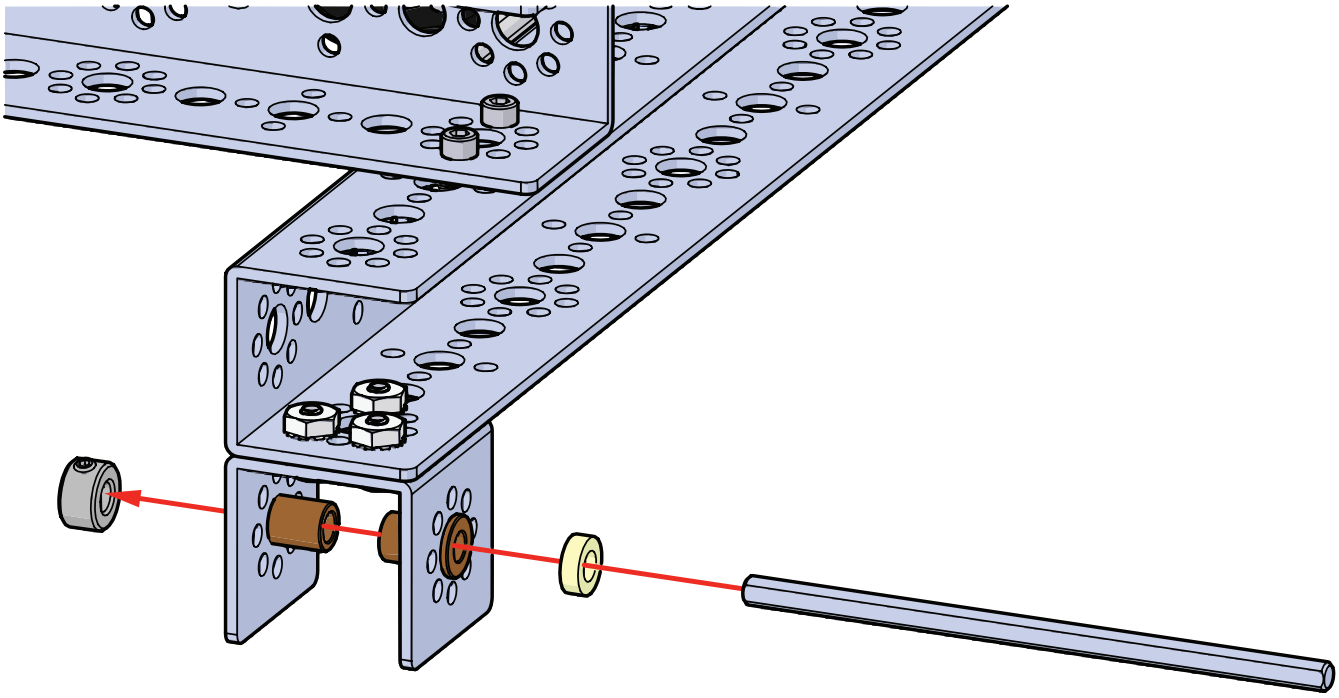
Step 3.2



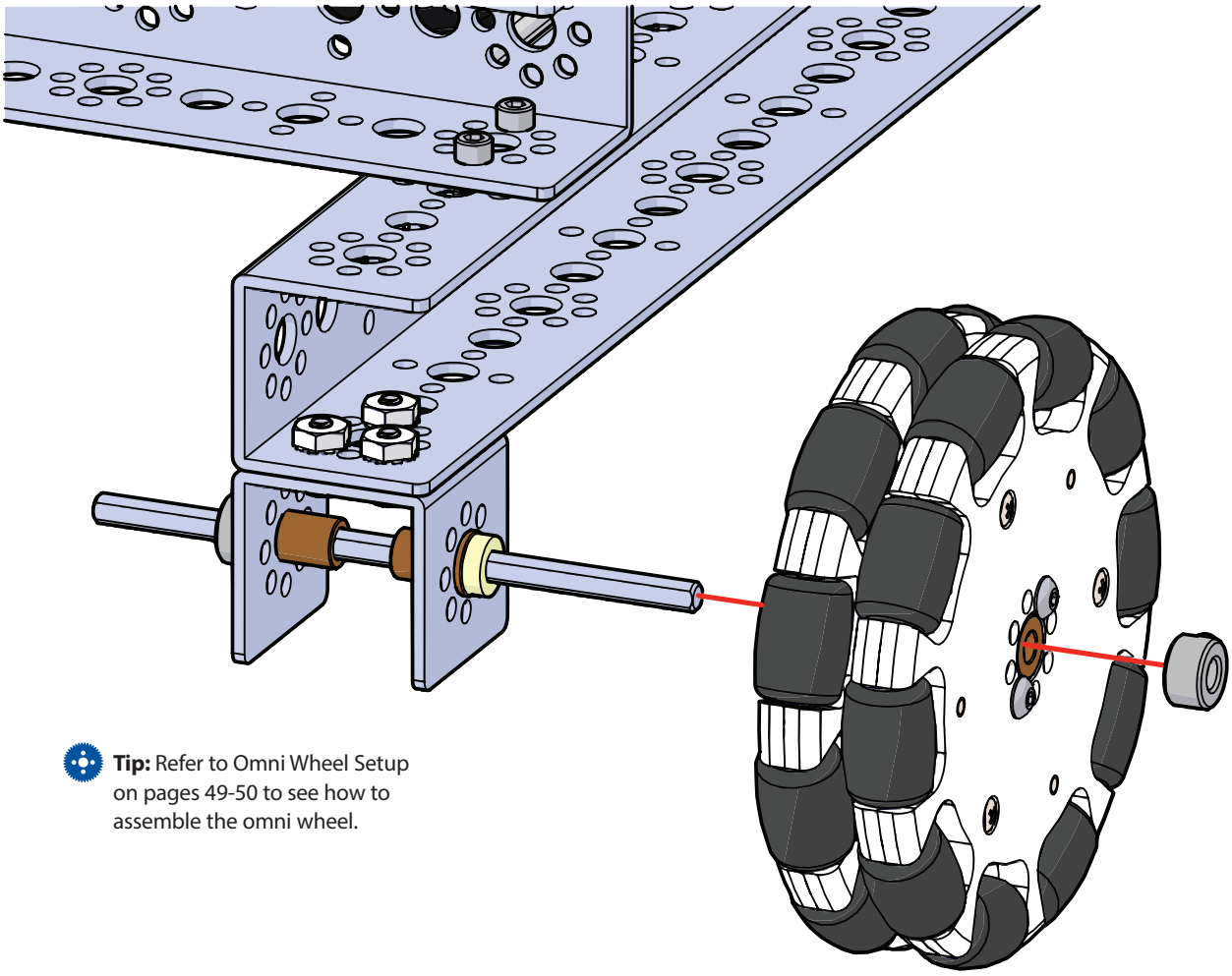
Step 3.3



Step 3.4

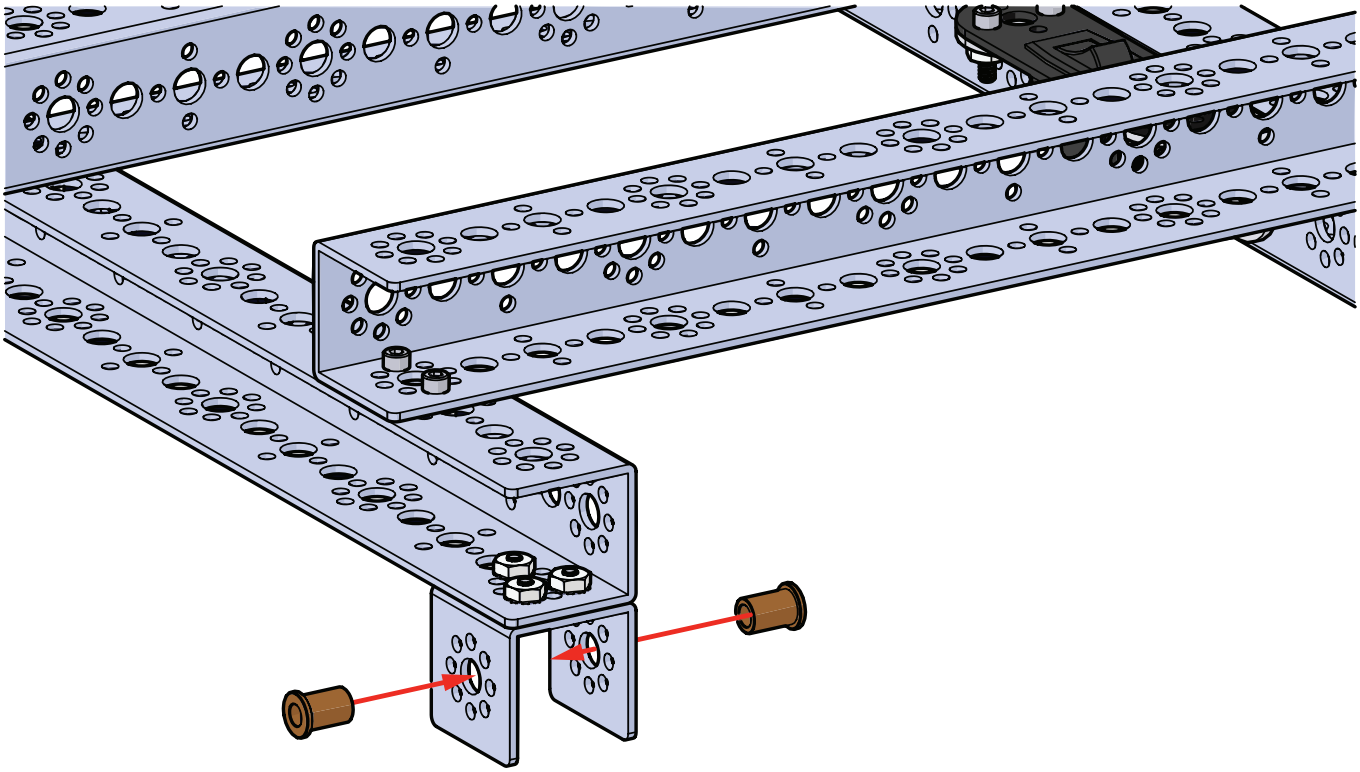


Step 3.5

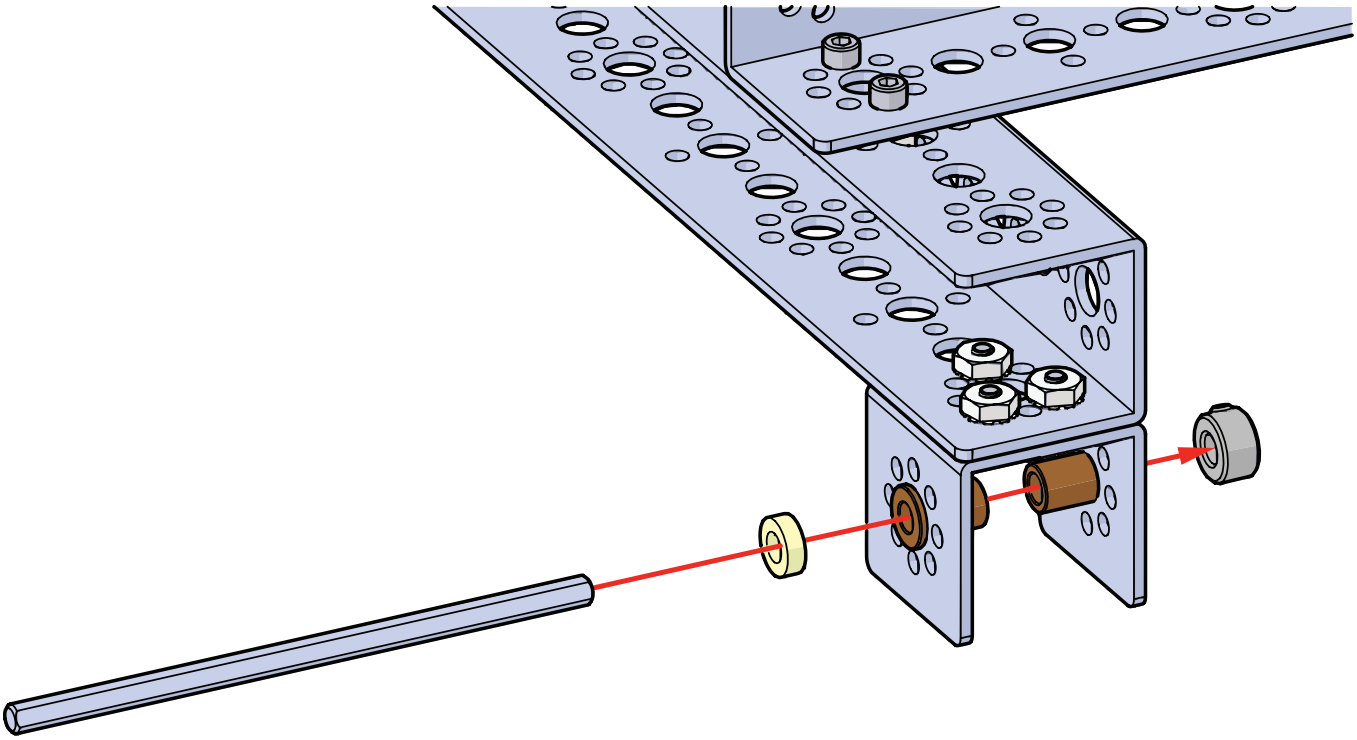


Tip: Refer to Omni Wheel Setup on pages 49-50 to see how to assemble the omni wheel.

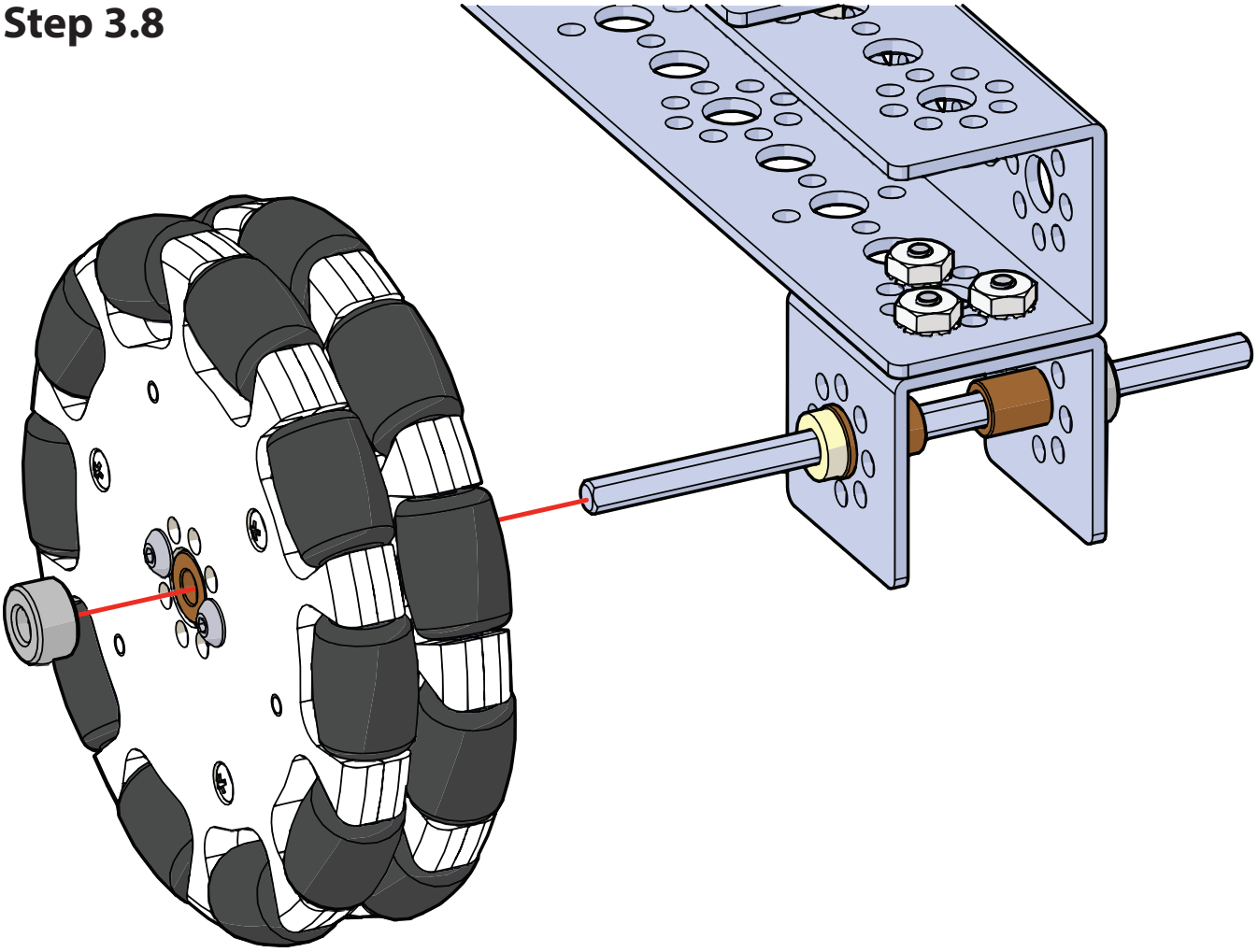
Step 3.6



Step 3.7

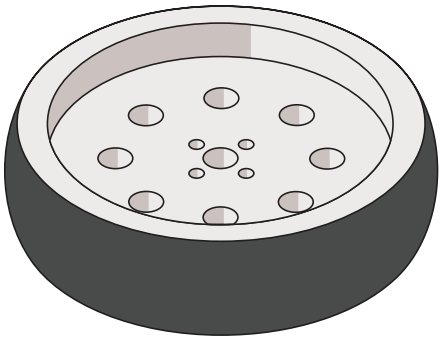


Step 3.8

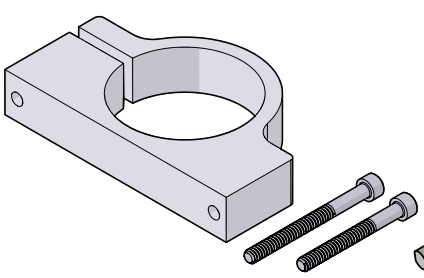


Step 4

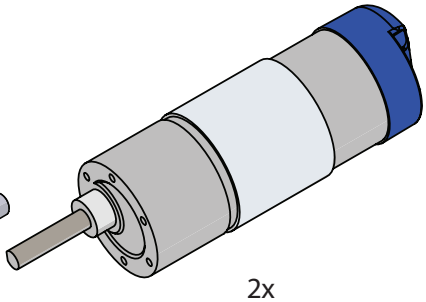
Parts Needed



2x
4" Wheel 39055



2x
Motor Mount 39089



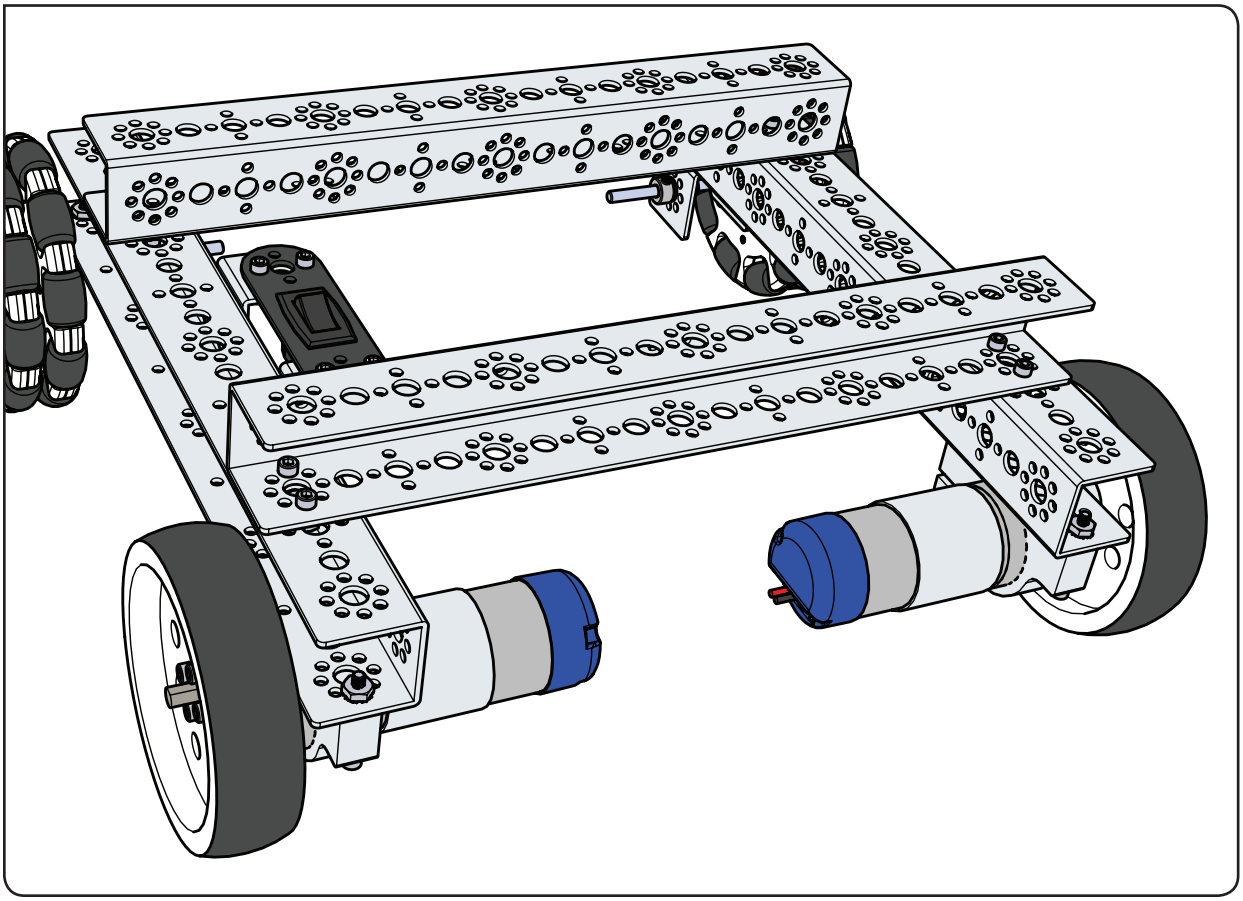
2x
TorqueNADO Motor 44260

2x
Motor Shaft Hub 39079

8x
Socket Head Cap Screw
6-32 x 1/2" 39097

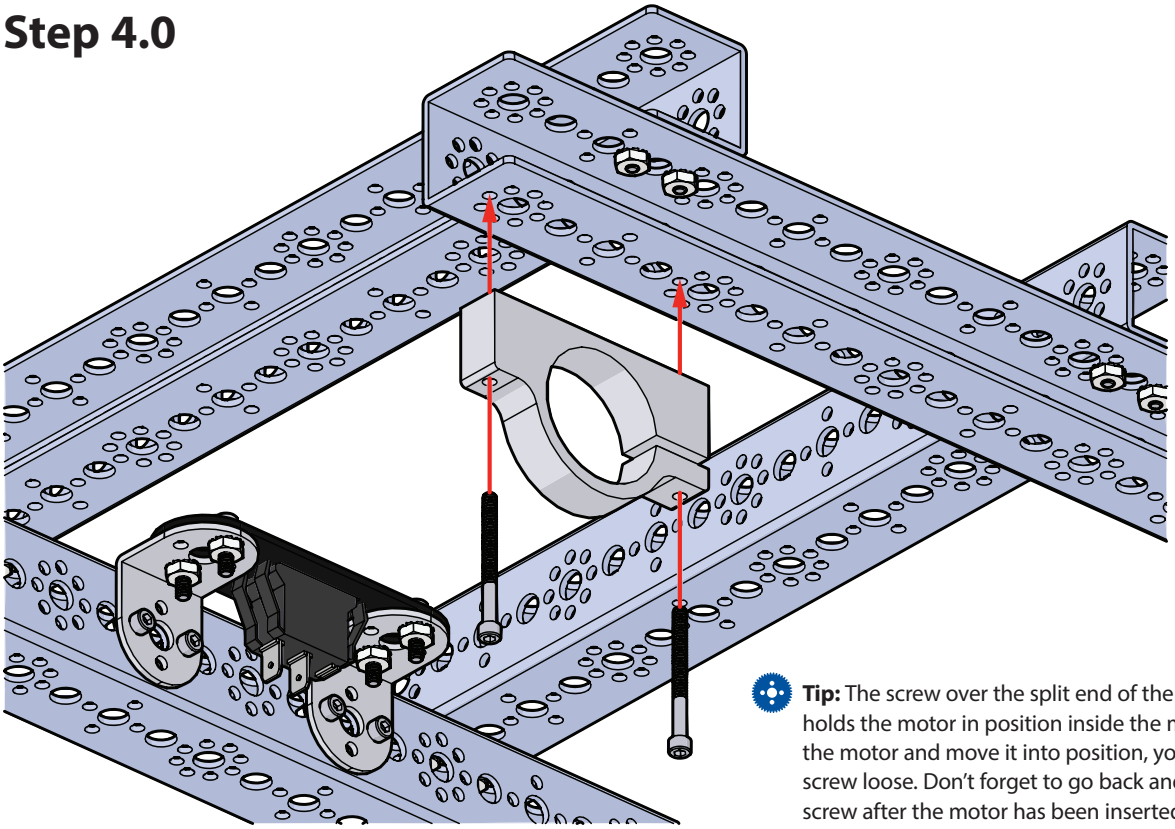
4x
Kep Nut 39094

Finished assembly should look like this.



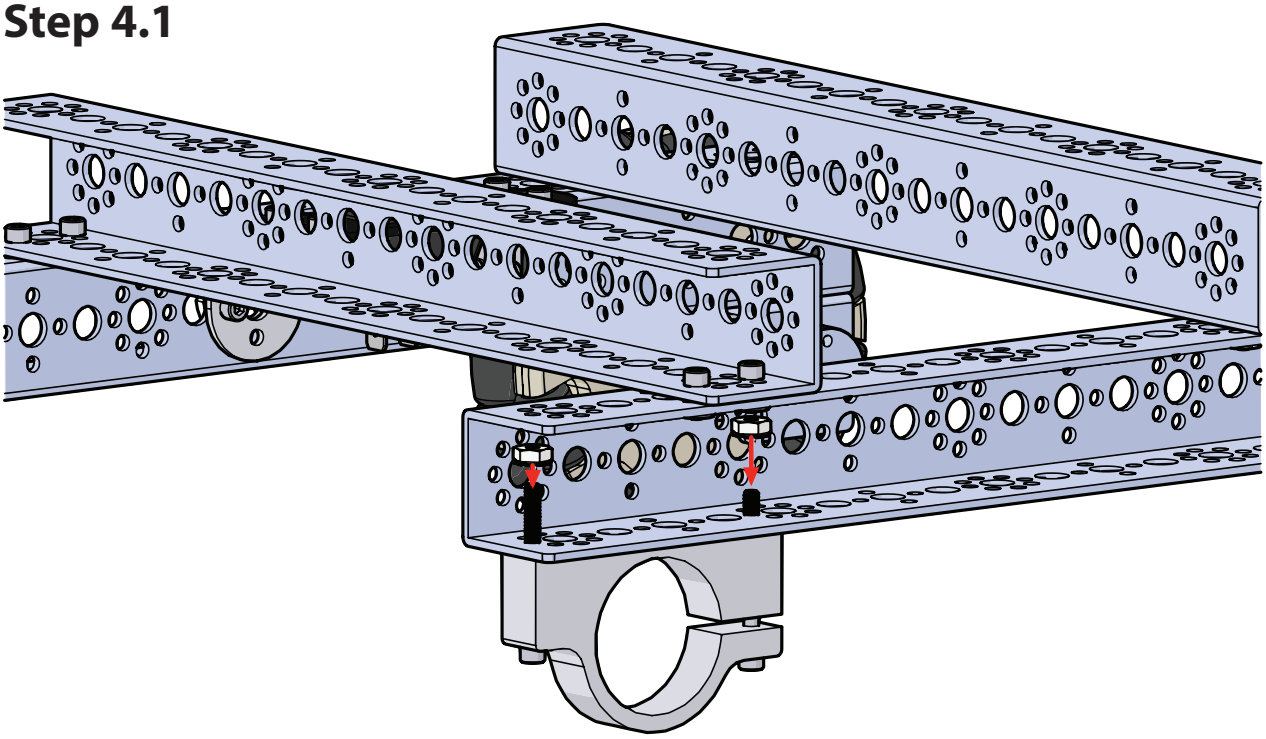
Tip: See the setup tips on page 48 for help distinguishing between the motor shaft hub and axle hub.

Step 4.0

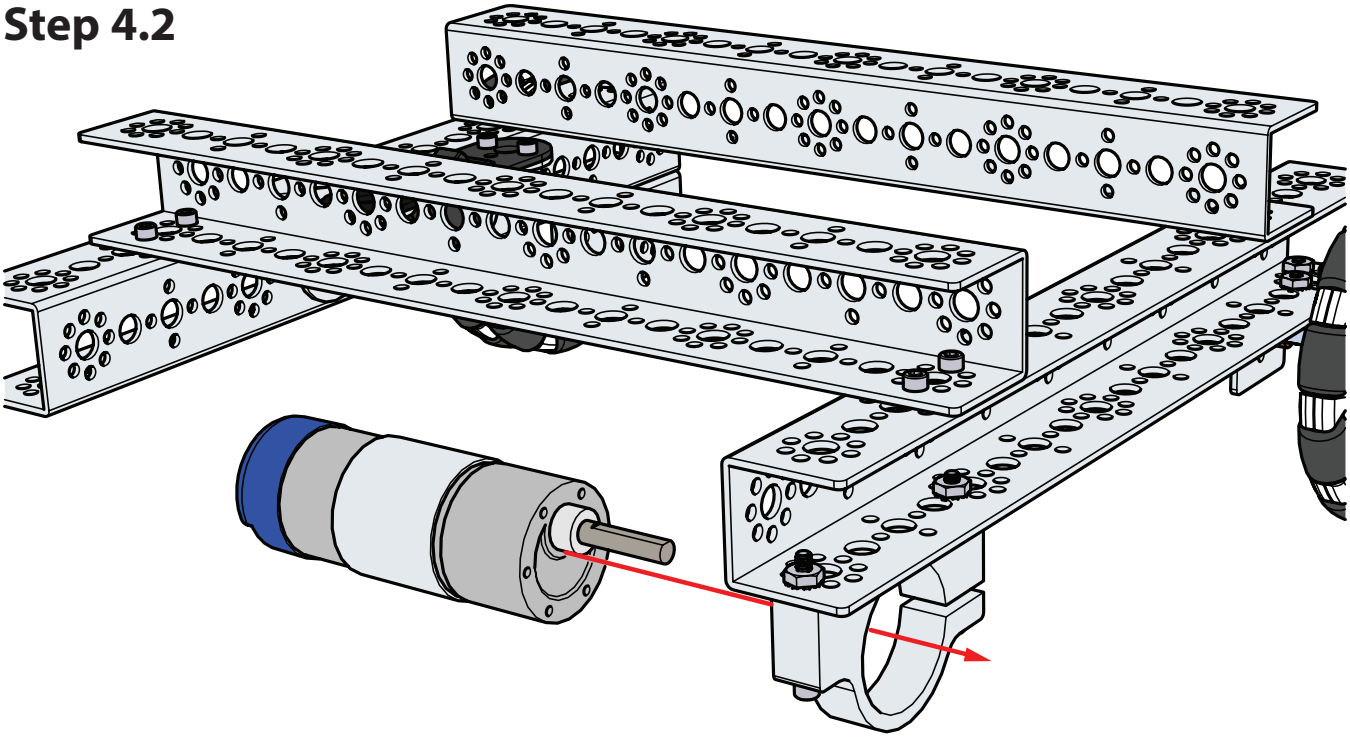


Tip: The screw over the split end of the motor mount holds the motor in position inside the mount. To insert the motor and move it into position, you must keep the screw loose. Don't forget to go back and tighten this screw after the motor has been inserted and moved into the desired position.

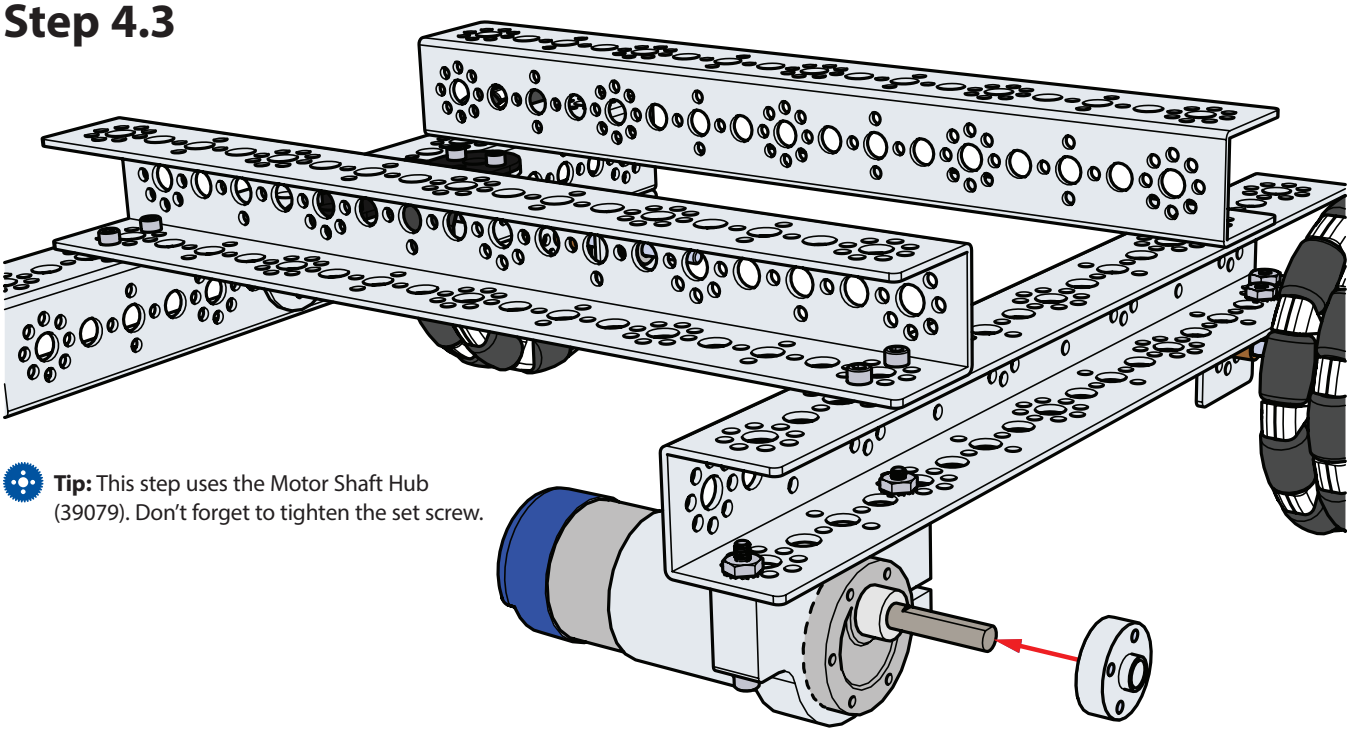
Step 4.1



Step 4.2

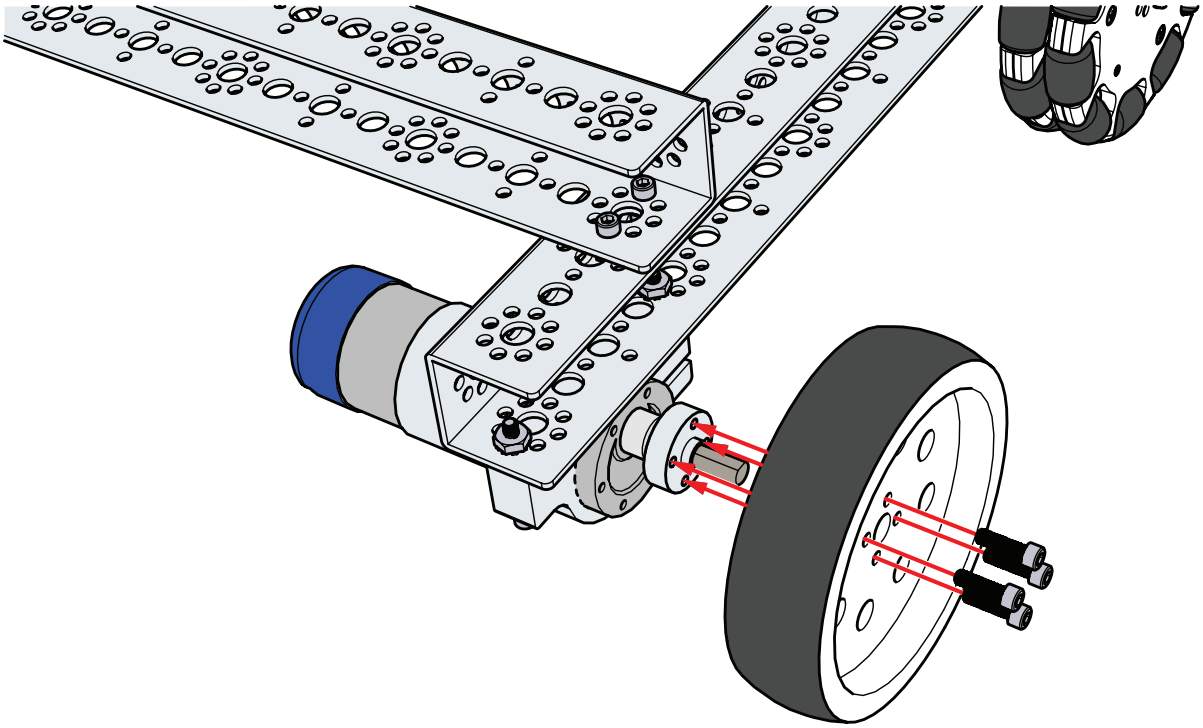


Step 4.3



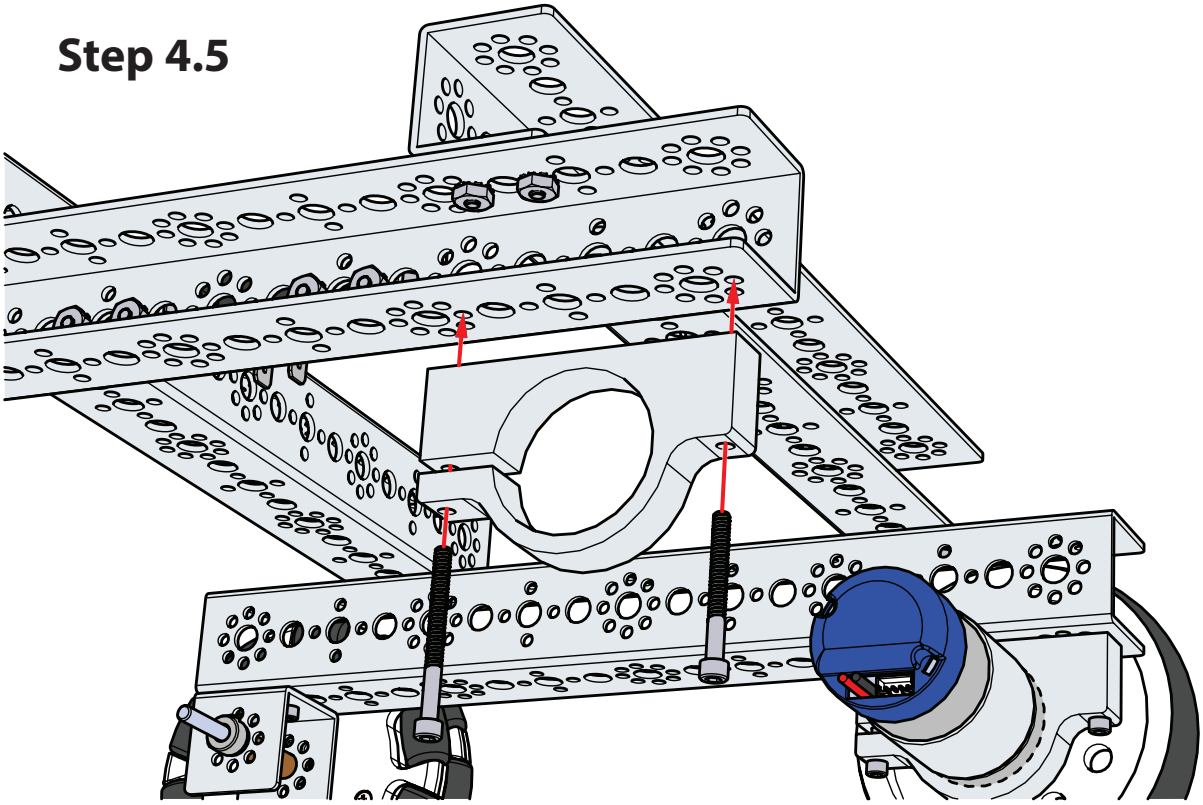
Tip: This step uses the Motor Shaft Hub (39079). Don't forget to tighten the set screw.

Step 4.4

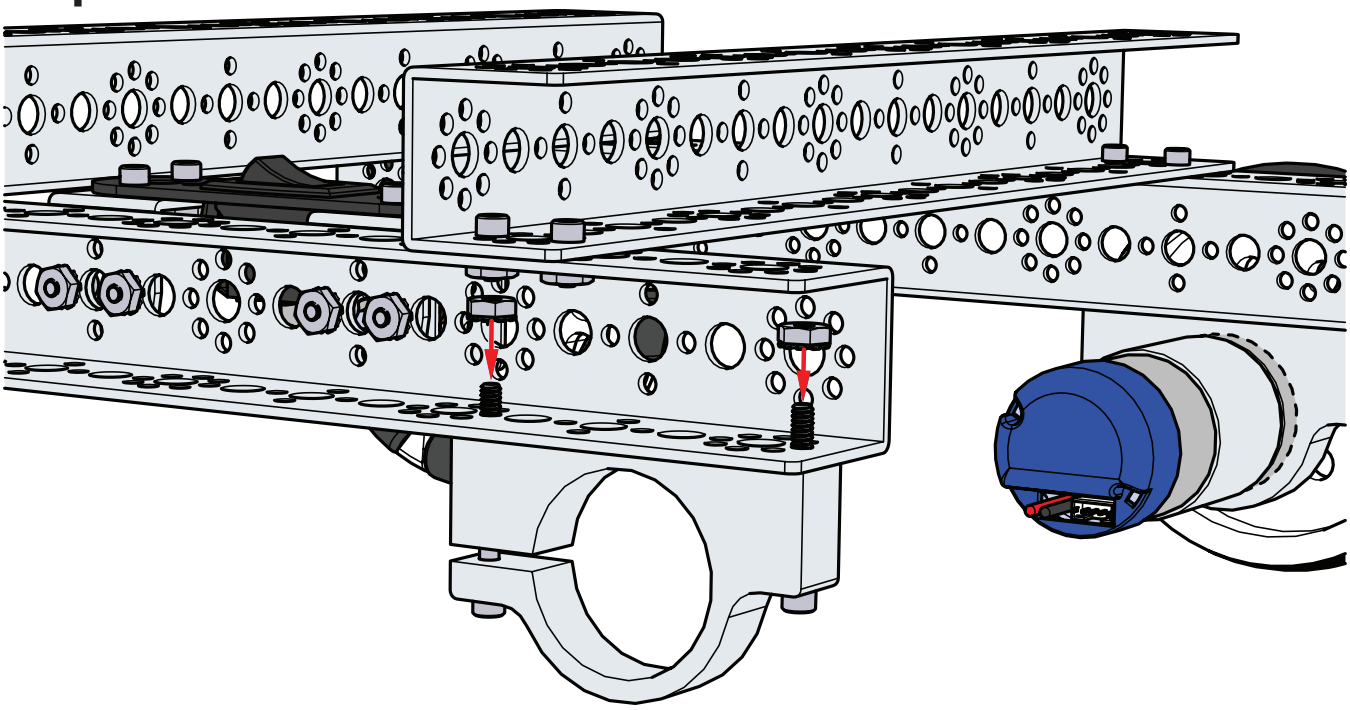


Tip: The screw over the split end of the motor mount holds the motor in position inside the mount. To insert the motor and move it into position, you must keep the screw loose. Don't forget to go back and tighten this screw after the motor has been inserted and moved into the desired position.

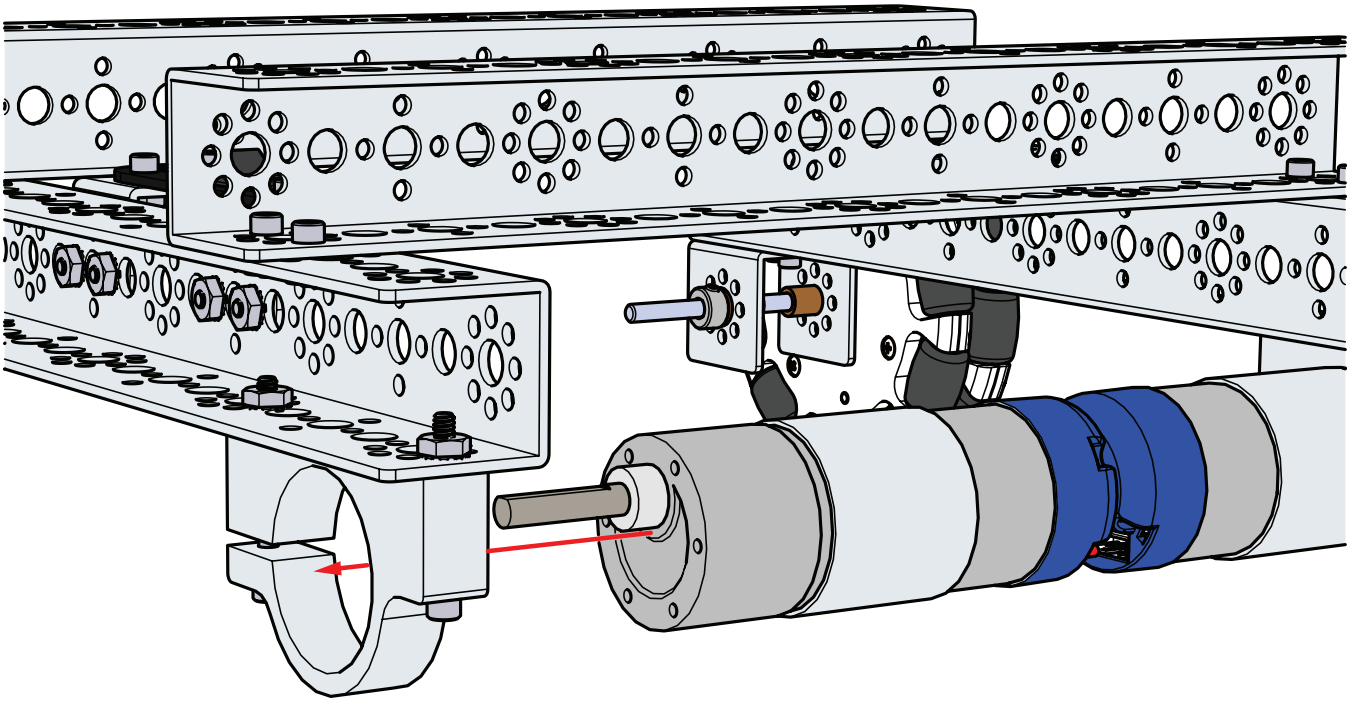
Step 4.5



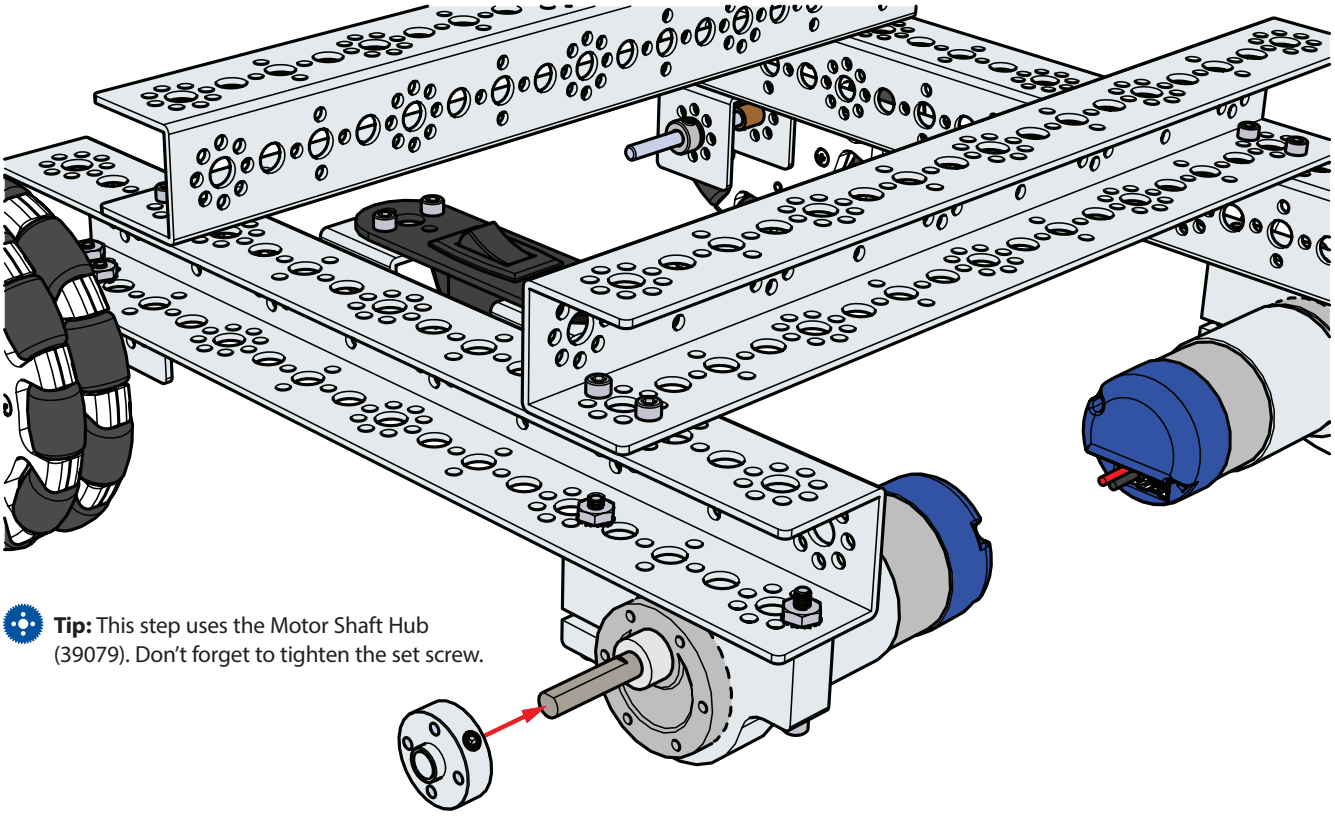
Step 4.6



Step 4.7

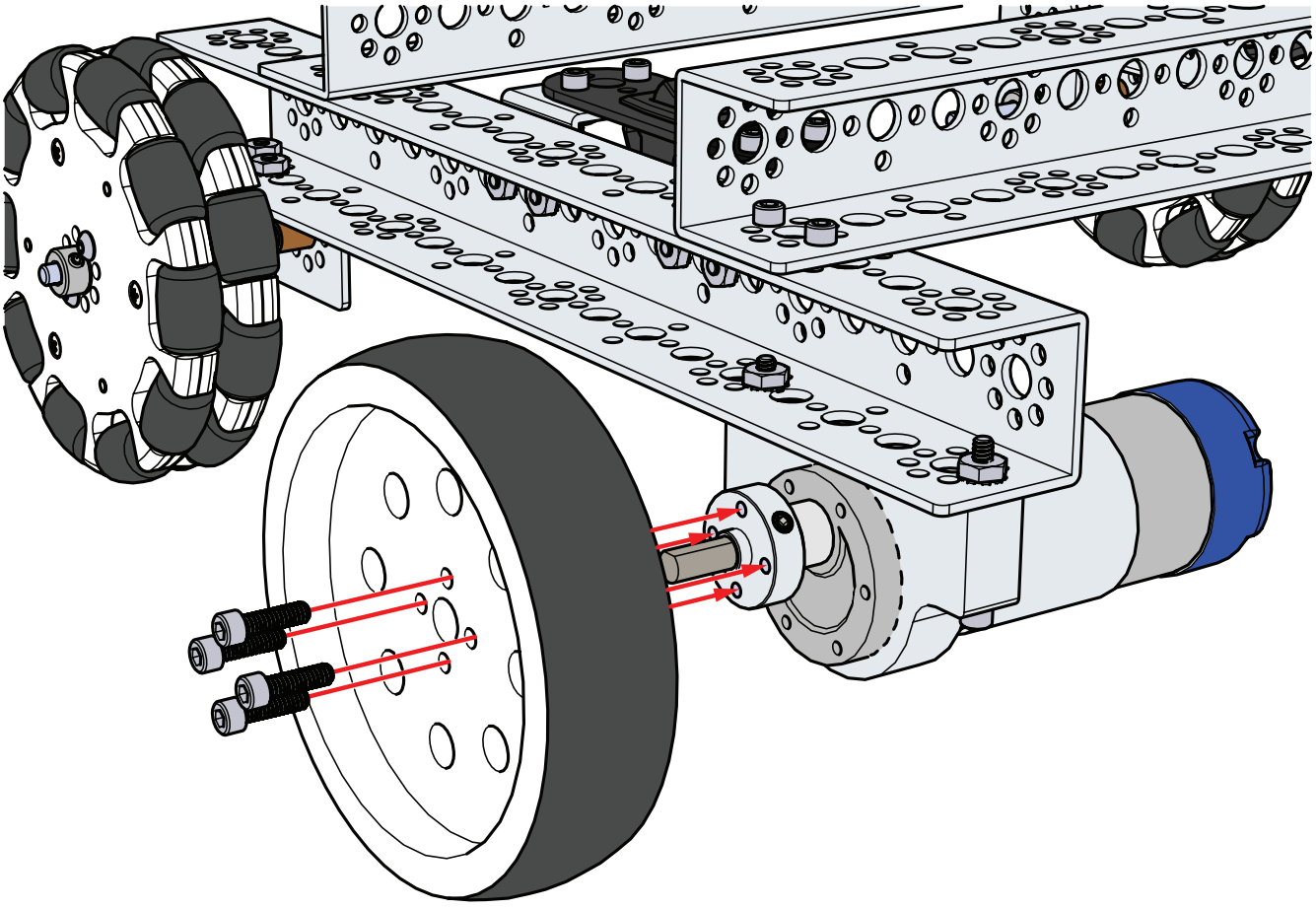


Step 4.8



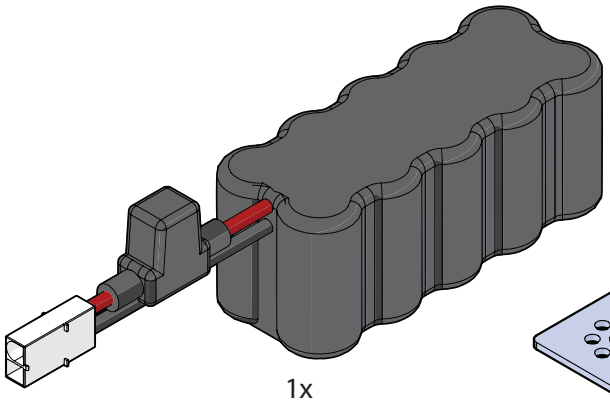
Tip: This step uses the Motor Shaft Hub (39079). Don't forget to tighten the set screw.

Step 4.9

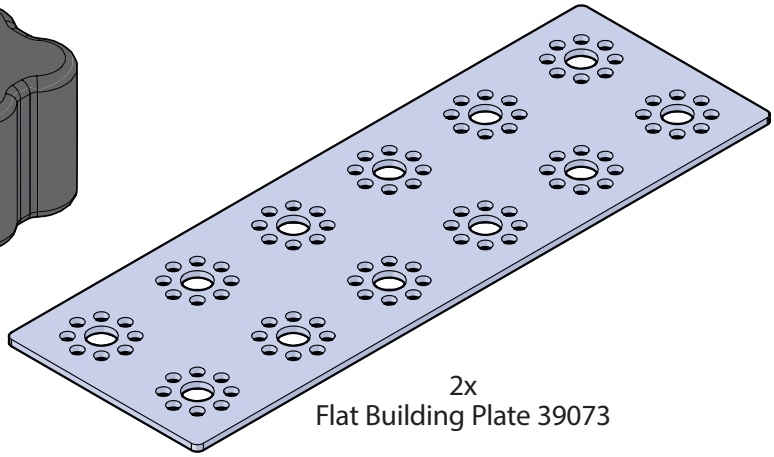


Step 5

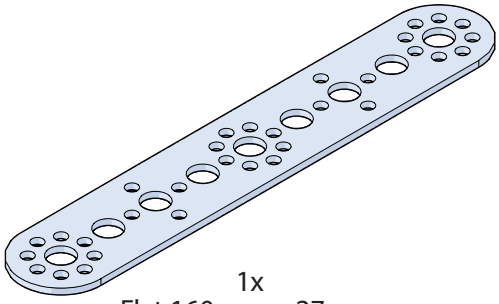
Parts Needed



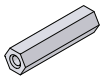
1x
12-volt 3,000 mAh Battery 39057



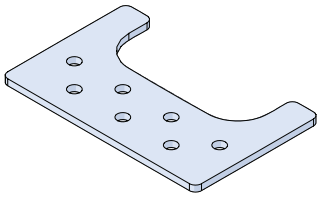
2x
Flat Building Plate 39073



1x
Flat 160 mm x 27 mm
39272



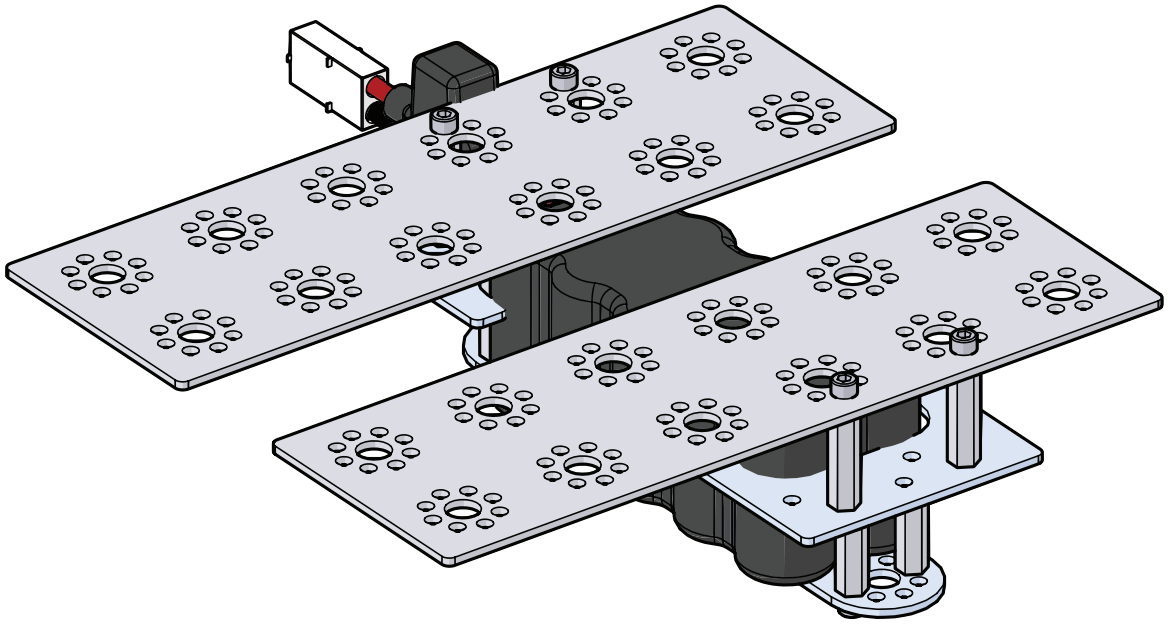
8x
Stand-Off Post 6-32 x 1"
39102



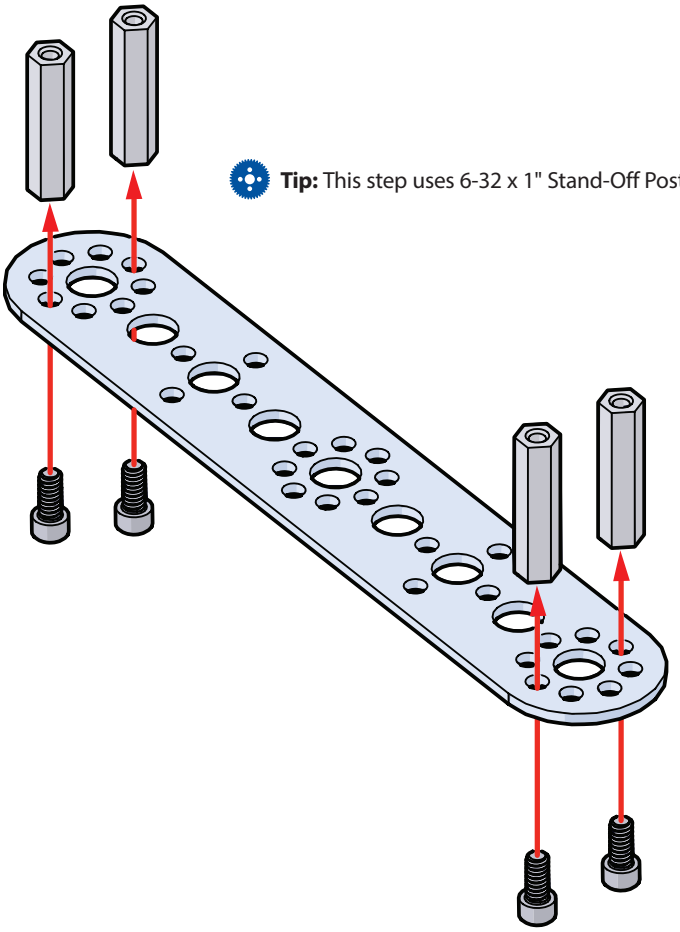
2x
Battery Clip 38009

16x
Socket Head Cap Screw
6-32 x 5/16" 39098

Finished assembly should look like this.

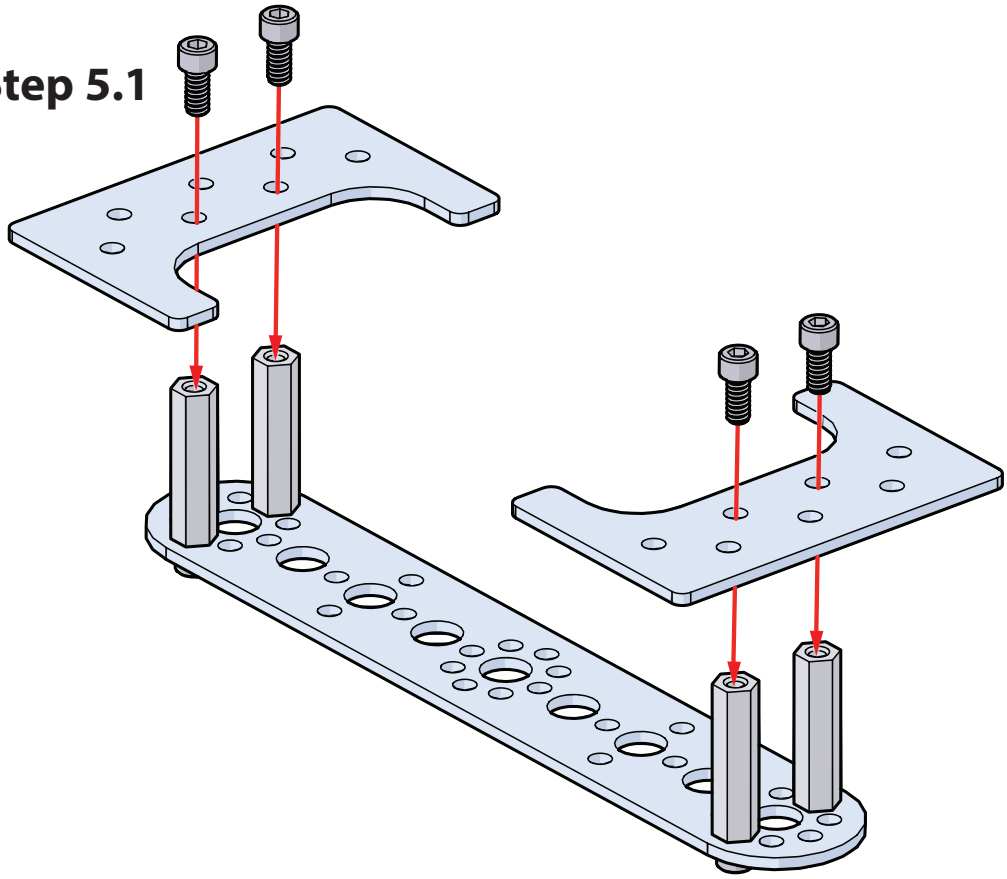


Step 5.0

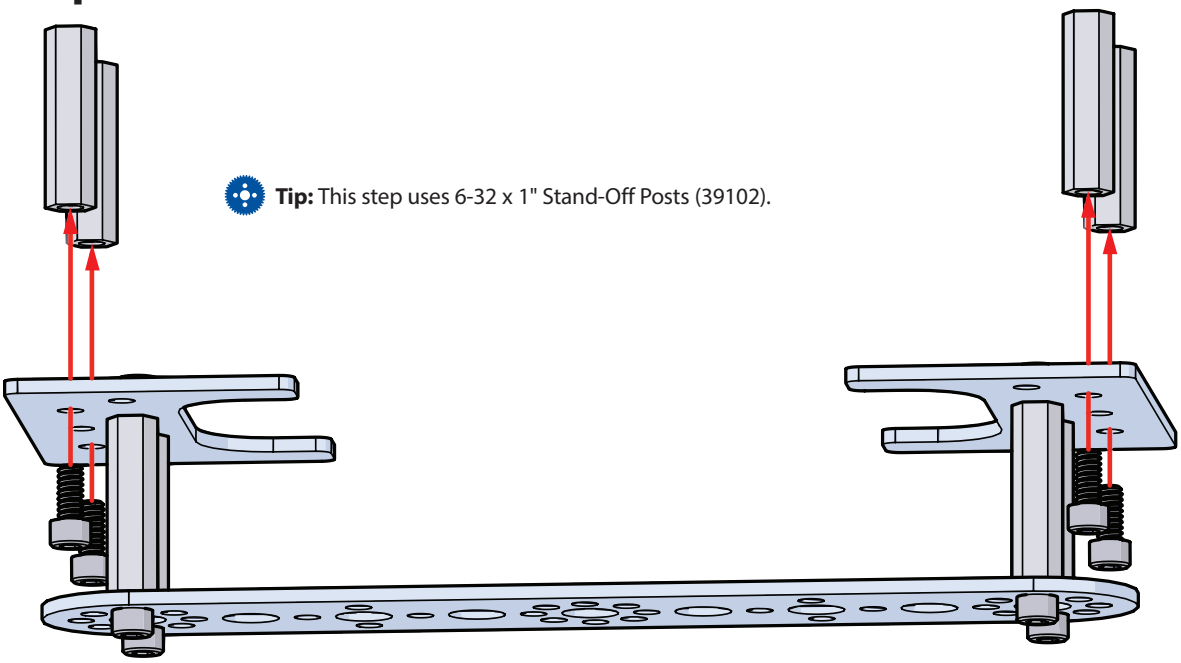


Tip: This step uses 6-32 x 1" Stand-Off Posts (39102).

Step 5.1

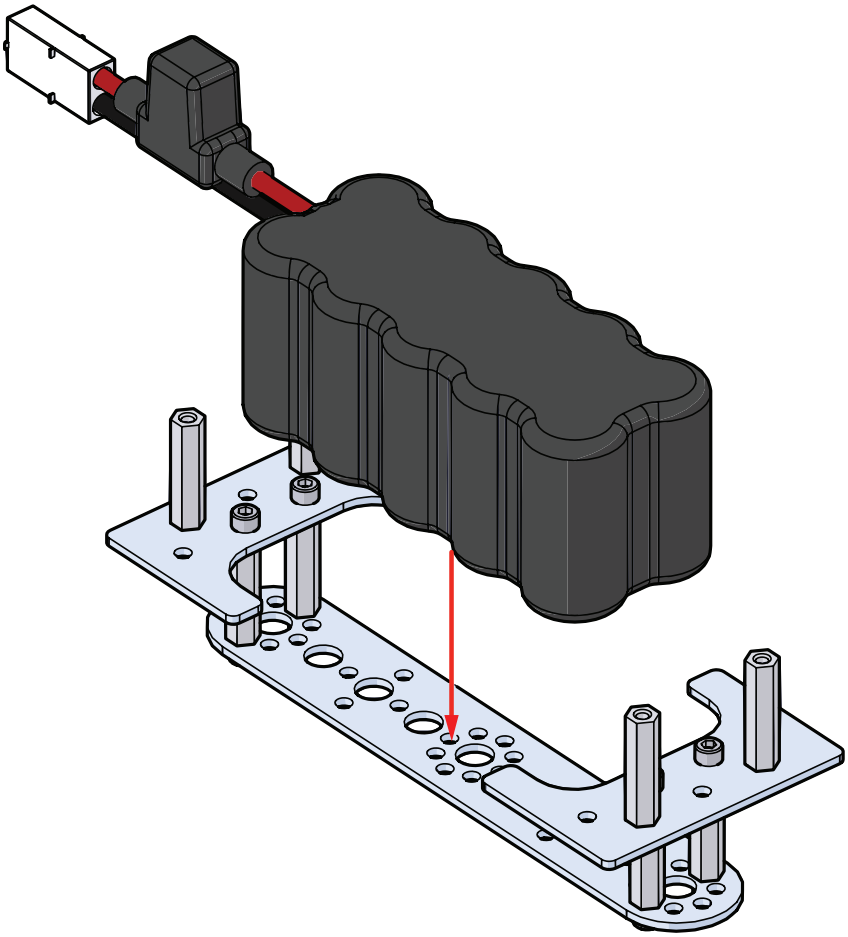


Step 5.2

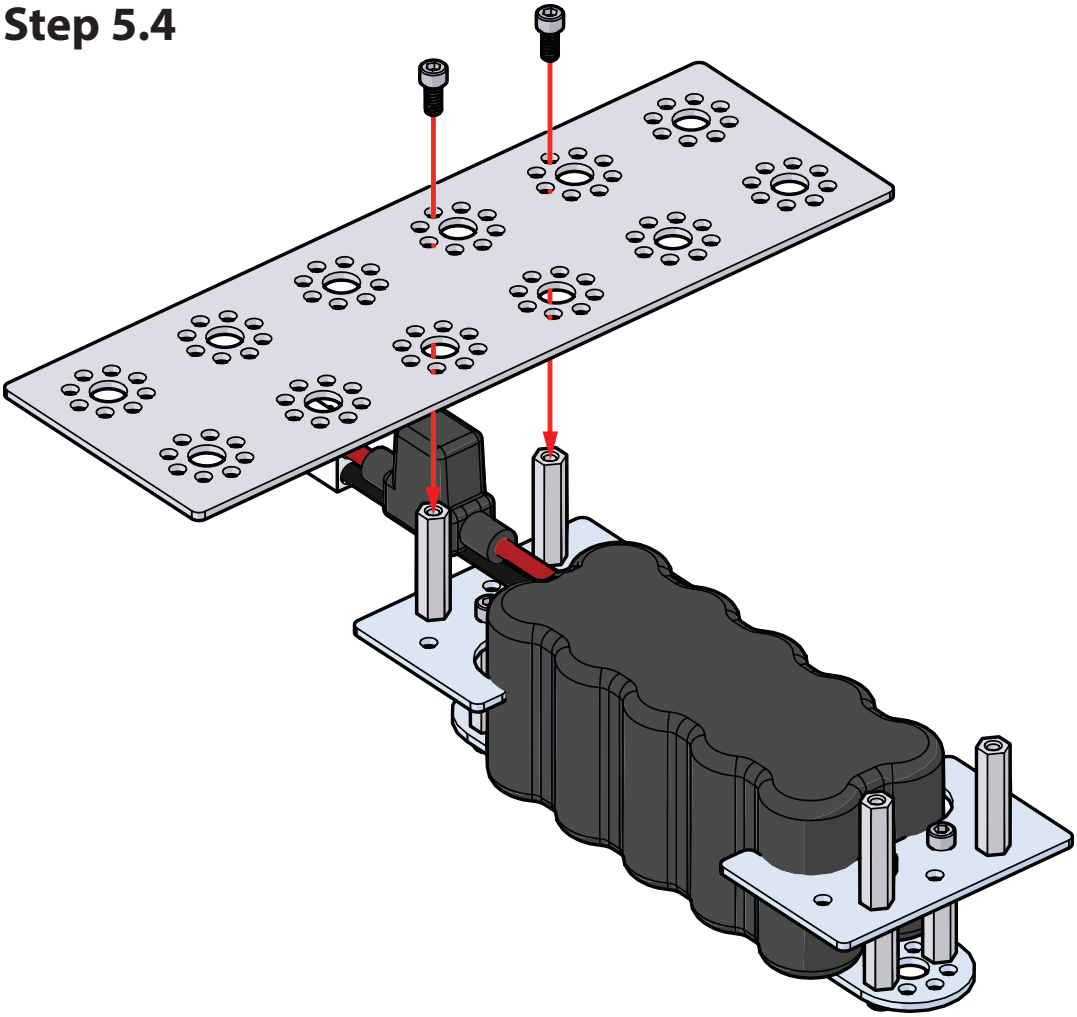


Tip: This step uses 6-32 x 1" Stand-Off Posts (39102).

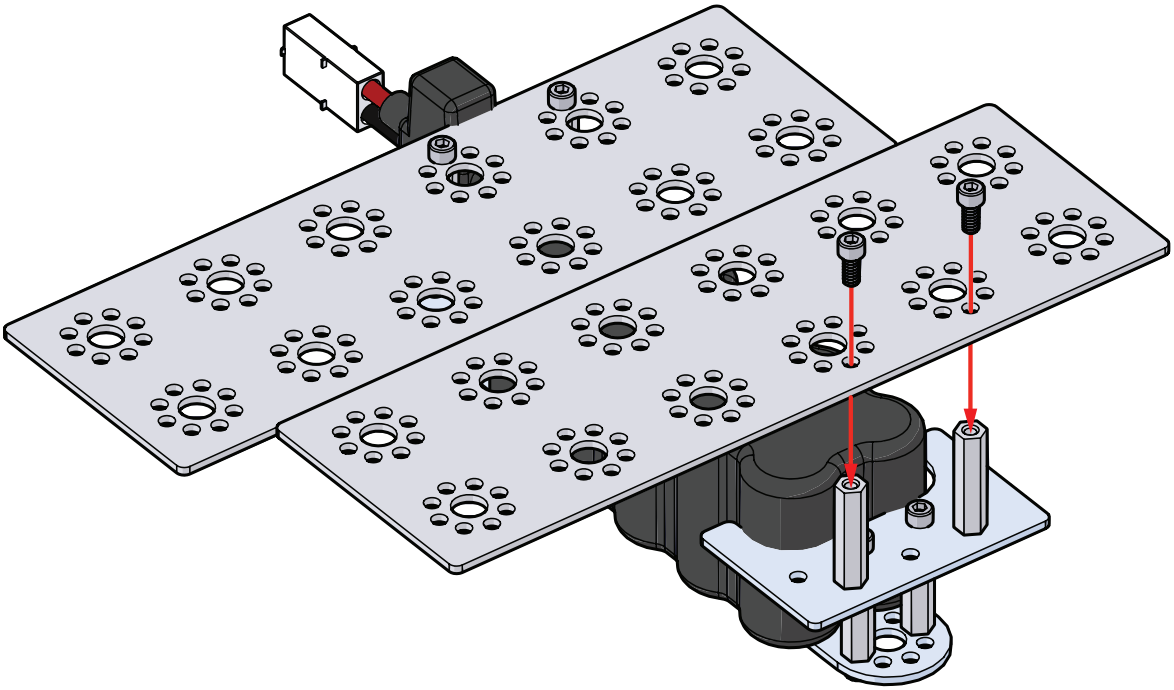
Step 5.3



Step 5.4



Step 5.5



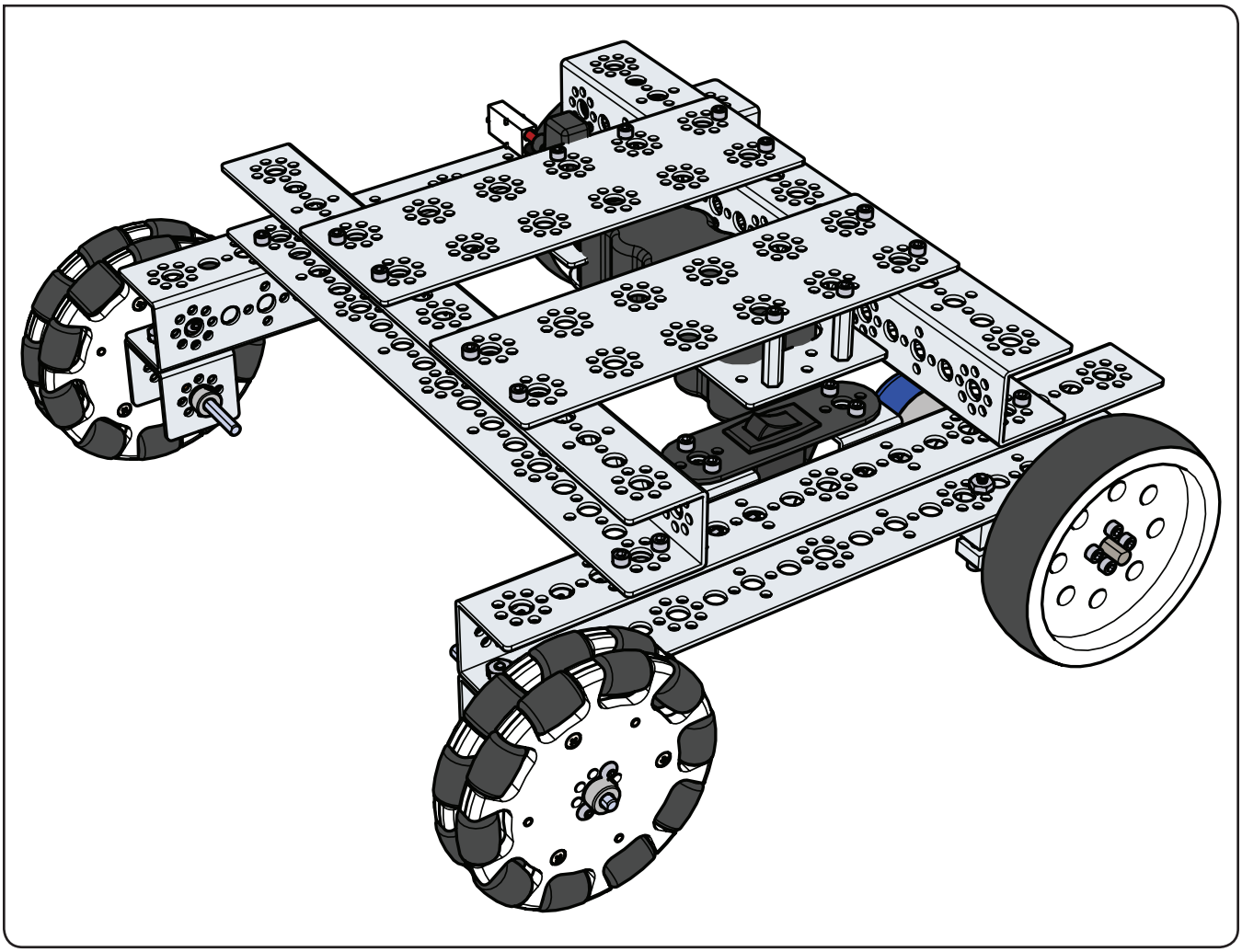
Step 6

Parts Needed

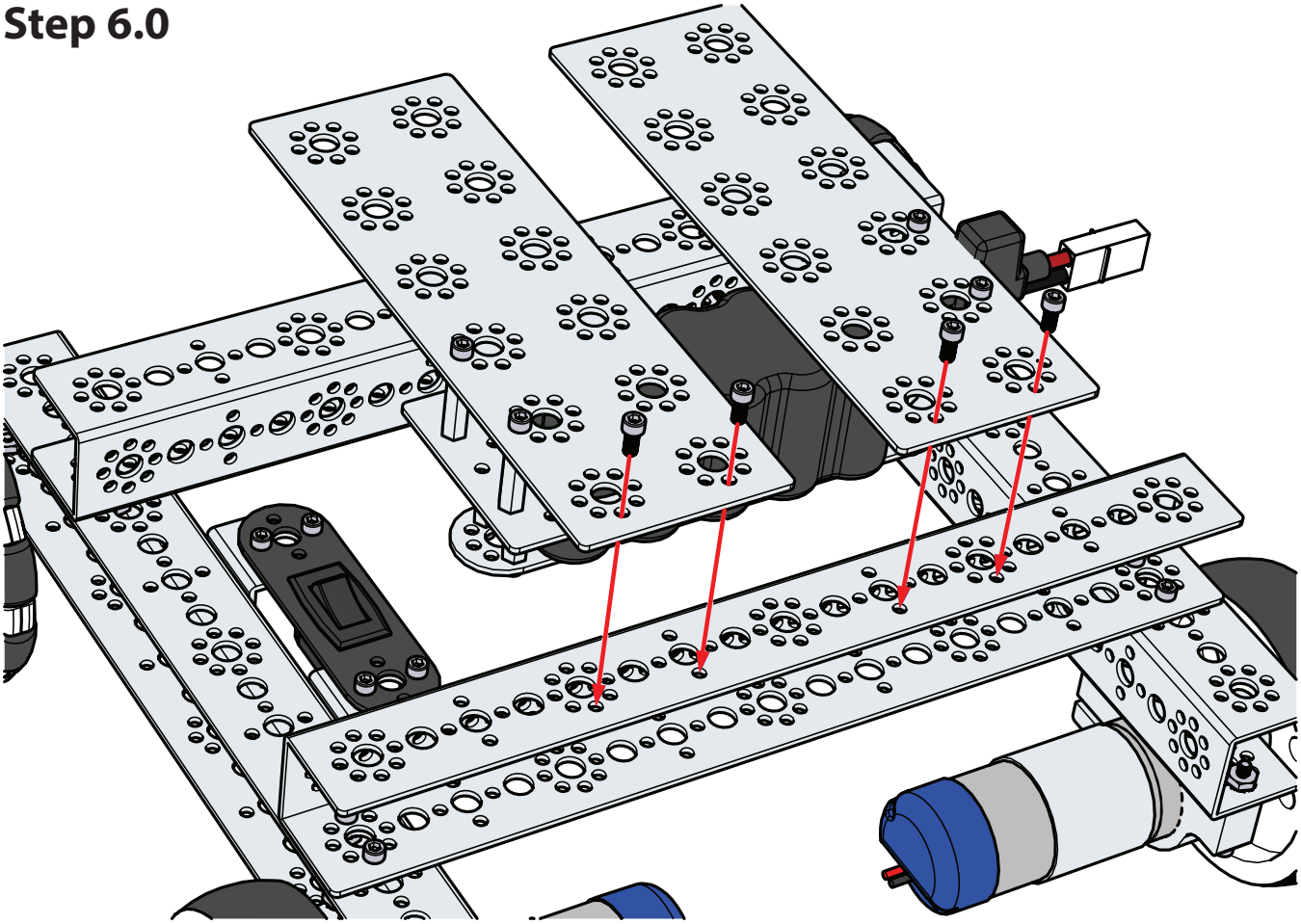
8x
Socket Head Cap Screw
6-32 x 5/16" 39098

8x
Kep Nut 39094

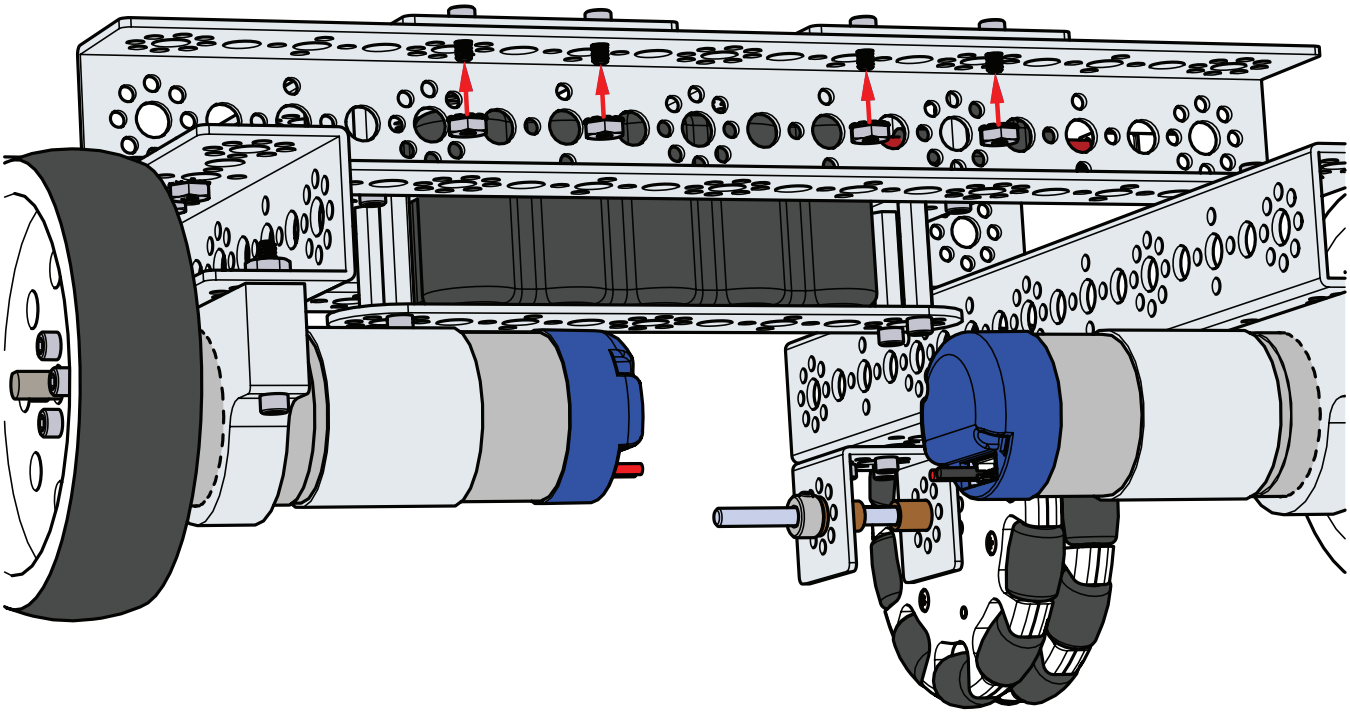
Finished assembly should look like this.



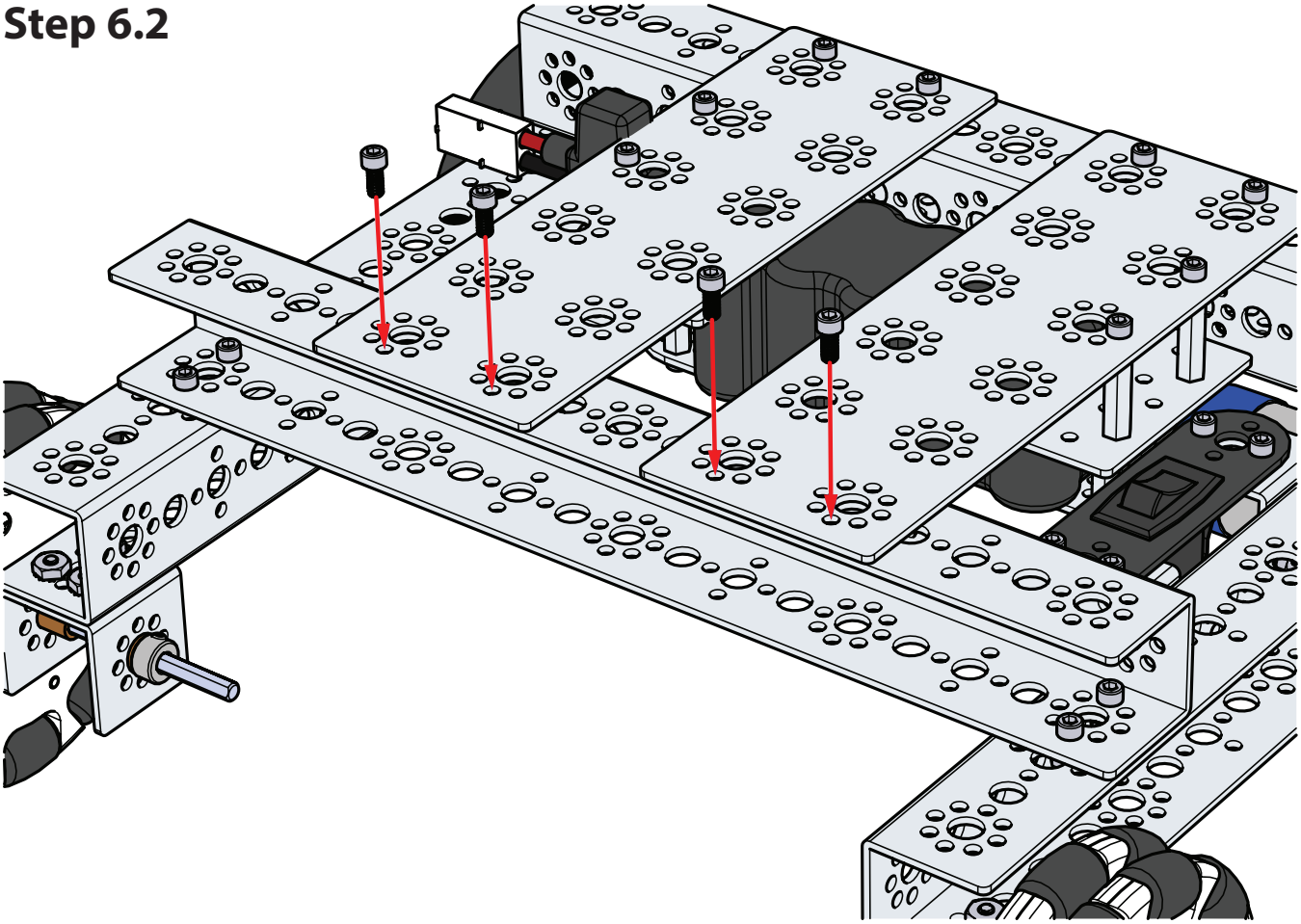
Step 6.0



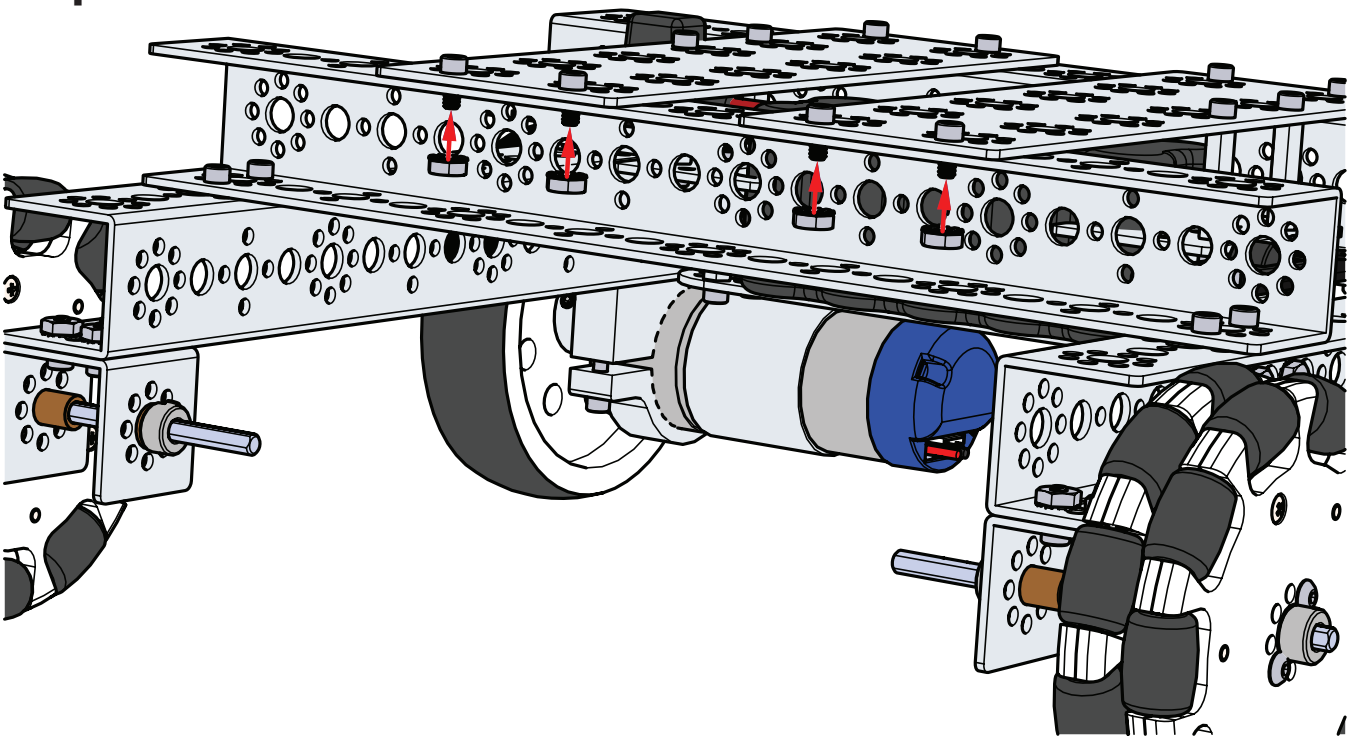
Step 6.1



Step 6.2

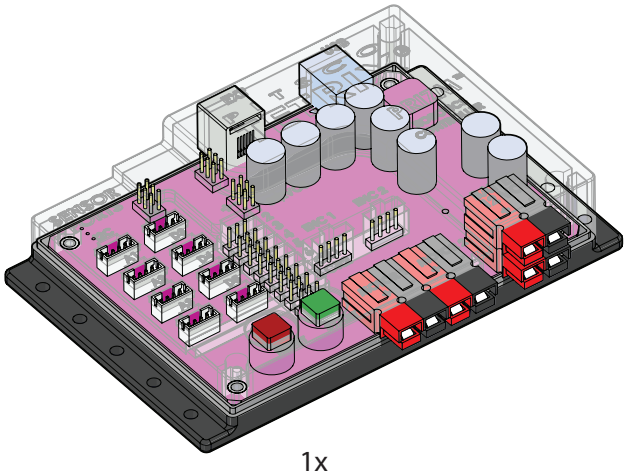


Step 6.3



Step 7

Parts Needed

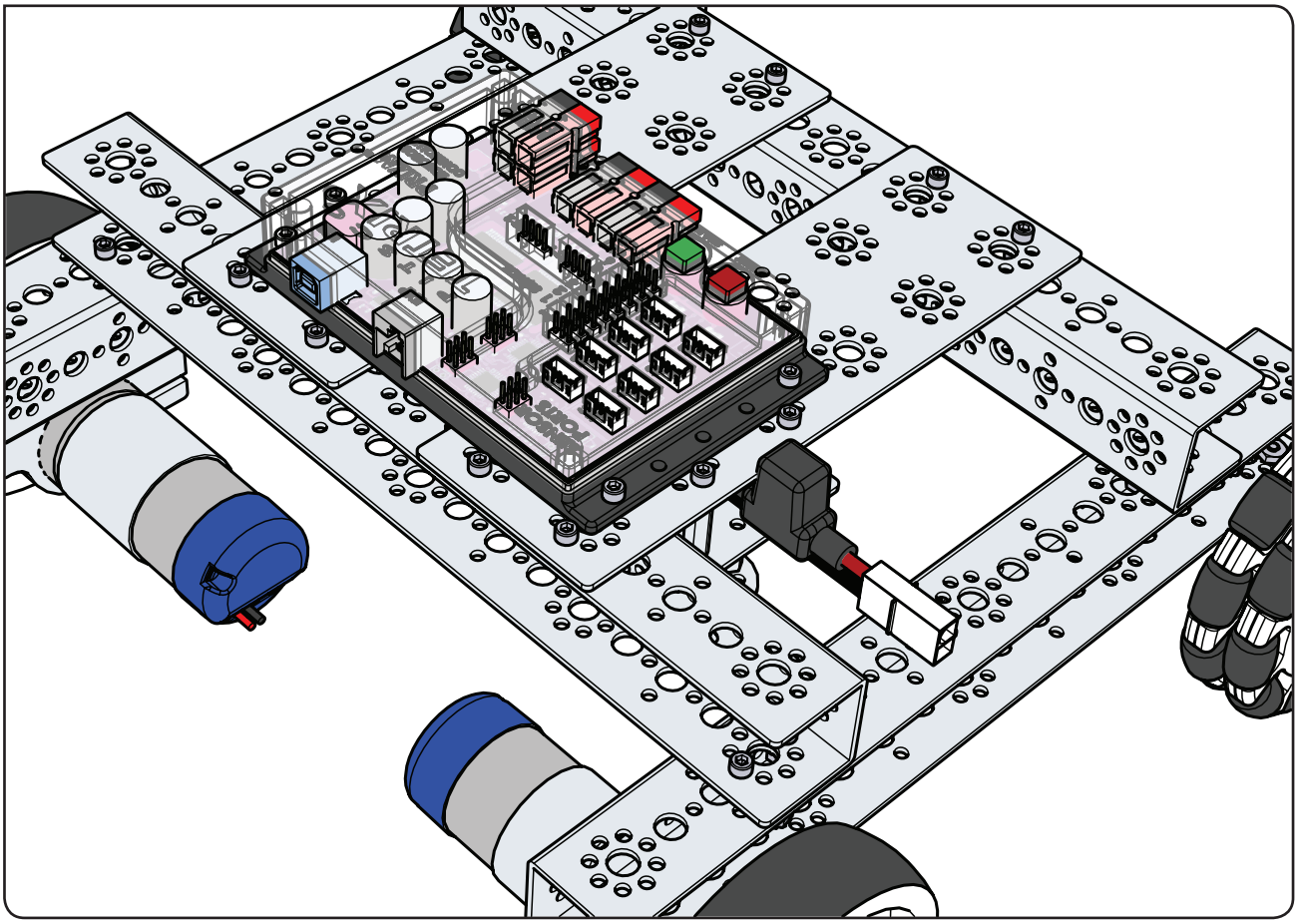


1x
PRIZM Robotics Controller
43000

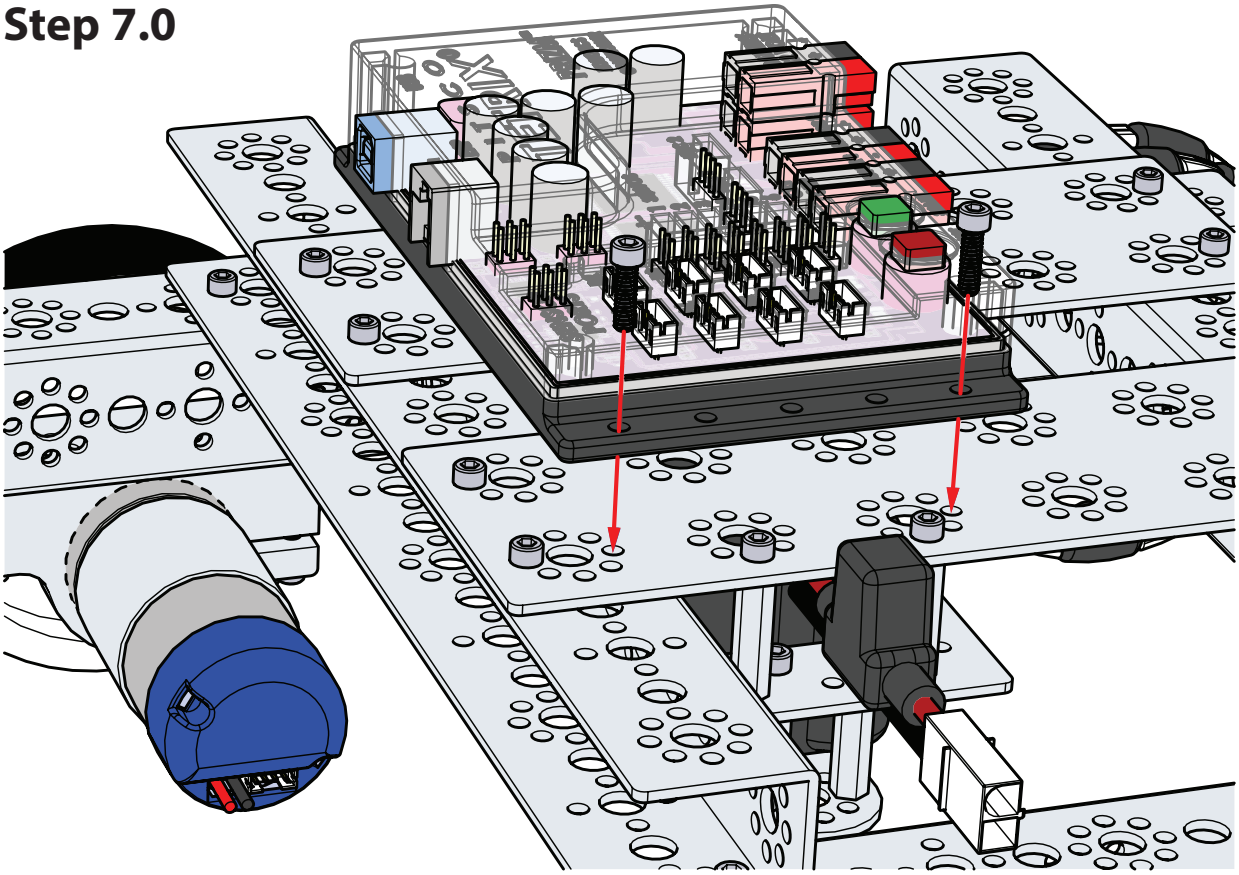
4x
Socket Head Cap Screw
6-32 x 1/2" 39097

4x
Kep Nut 39094

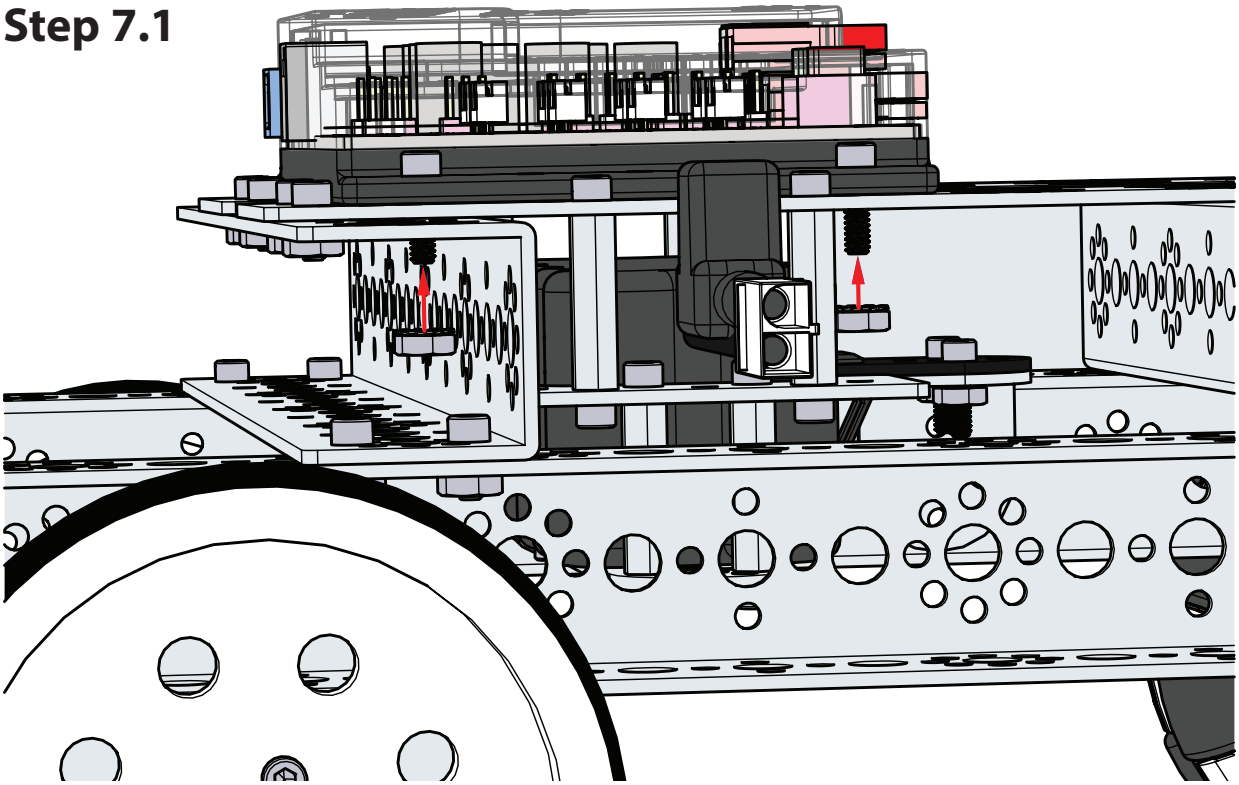
Finished assembly should look like this.



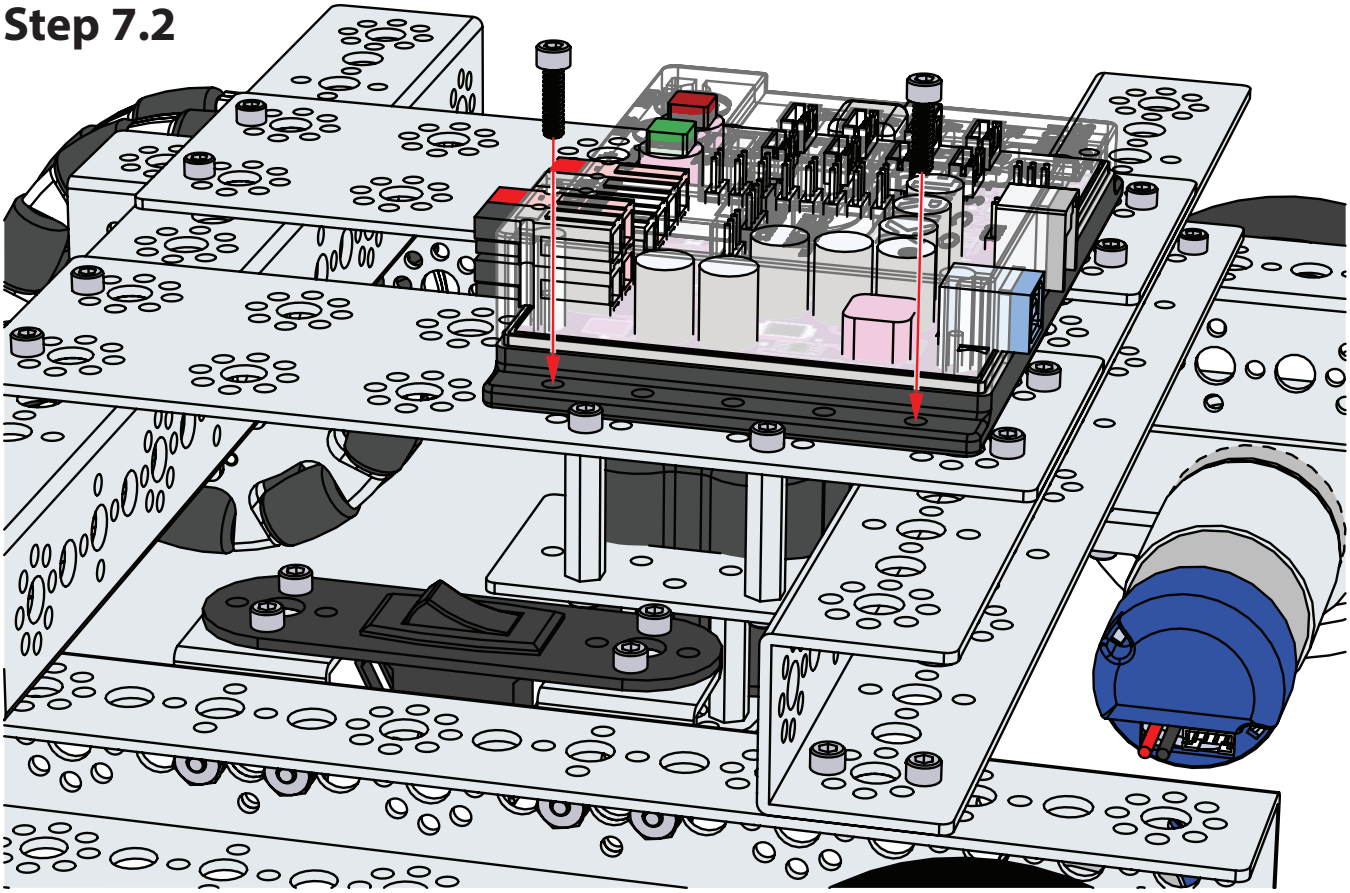
Step 7.0



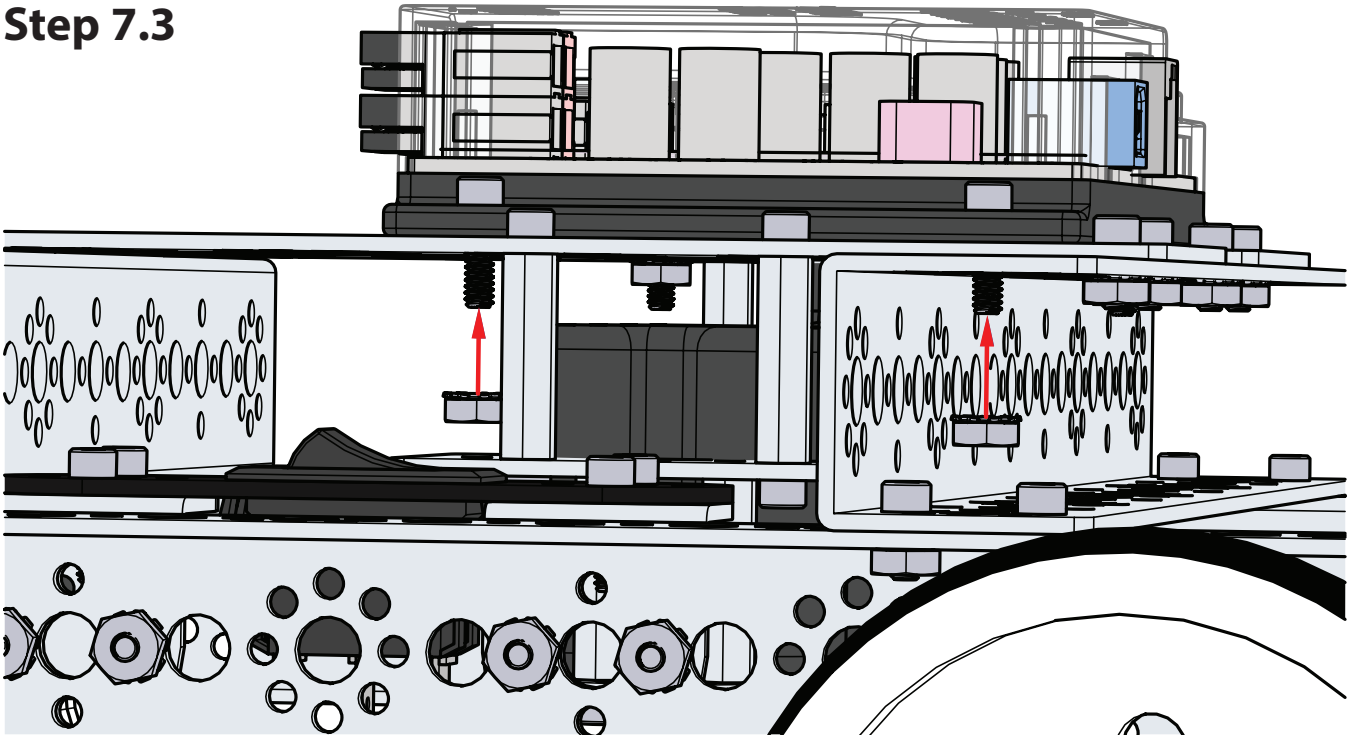
Step 7.1



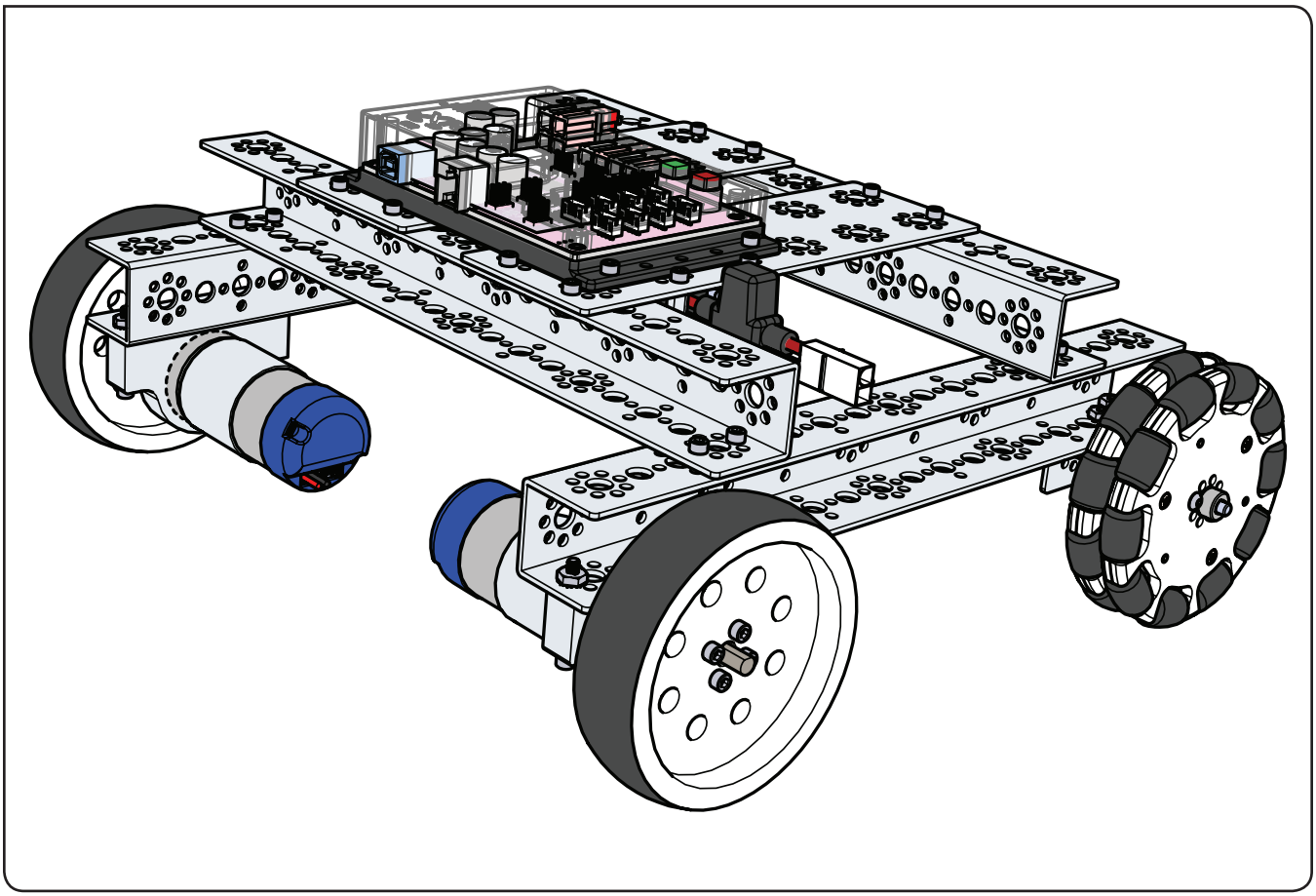
Step 7.2



Step 7.3



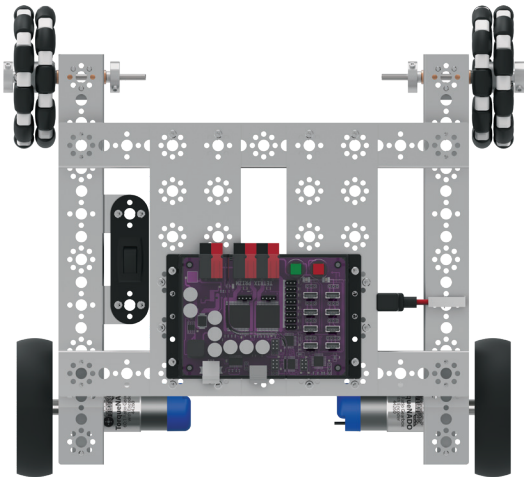
Finished assembly should look like this.



Activity 7: Drive Forward

Activity 7 is the first activity with the TaskBot. We want to start with something simple. In this activity we will create a sketch to move the TaskBot forward for three seconds, stop, and end the program.

Building on what we learned in Activity 2, we will add a second motor and have them work together in unison. Being able to use two motors together in unison is a fundamental requirement for mobile robots.



Parts Needed

- Fully assembled PRIZM TaskBot
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- Computer

Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > TaskBot_ Act7_Drive_Forward**. A new sketch window will open titled TaskBot_Act7_Drive_ Forward (Figure 20).

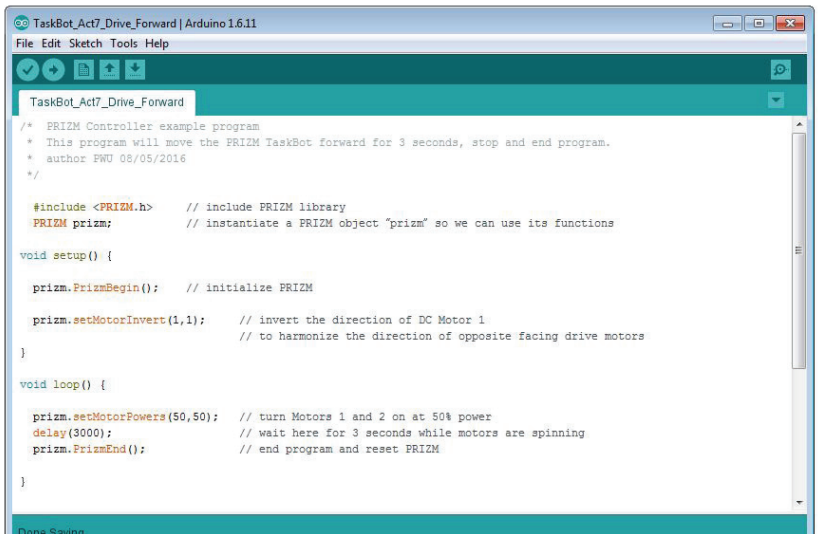


Figure 20

Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections.

Upload the sketch. The green LED will light up, indicating the code is ready to execute. When the LED comes on, disconnect the USB cable and set the TaskBot on the floor.

Press the green Start button to execute the sketch. Observe the direction and duration of the TaskBot's motion. Based on the sketch comments, did the behavior match expectations?

The TaskBot should stop after three seconds.

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

Moving Forward

This sketch introduces three new functions, prizm.setMotorInvert(), prizm.setMotorPowers(), and prizm.PrizmEnd().

The prizm.setMotorInvert() enables you to invert the rotational direction of a motor. When two motors are mounted on opposite sides, this function enables you to give a single direction command to both motors and have them work together. This makes your job as a programmer easier. There are two parameters to the function. The first parameter designates the motor channel, and the second parameter designates no invert or invert (0 or 1).

The prizm.setMotorPowers() enables you to set the power level of motor 1 and motor 2 at the same time. The two parameters set the speed for each motor. In this sketch the motor power for each motor is 50%.

The prizm.PrizmEnd() ends or terminates the sketch.

In this simple sketch all these functions work together so the robot can move forward for three seconds and then stop. Because we want the motors to always work together, prizm.setMotorInvert needs to be used only in the setup. The prizm.setMotorPowers tells both motors to move at 50% power with a single function. To finish the sketch we use prizm.PrizmEnd() instead of having it loop.

For more detailed information about the sketch process and the PRIZM library functions involved in using the DC motors, refer to the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 137-142 about DC motor functions

Real-World Connection

Trains are a very good example of machines that drive forward. Their drive motors (usually diesel powered) do not have to be able to rotate independently to turn corners – they just have to provide forward thrust to get (and keep) the train going. Occasionally they have to go backward – but they just provide reverse thrust in that instance.

STEM Extensions

Science

- Rotational kinetic energy
- Rotational torque

Technology

- Direct drive vs geared drive
- DC motor control functions

Engineering

- Differential, skid, and tricycle steering mechanisms

Math

- Clockwise, counterclockwise rotation
- Wheel circumference vs distance traveled per rotation

Tip: Without the use of encoders, speeds of DC motors using the power commands can vary depending on the charge level of the battery.

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Hacking the Code Activity

With the example as a reference, try creating a new sketch to move the TaskBot forward and stop. Remember what we learned from our previous activities and experiment with trying to make the TaskBot move forward for an amount of time and then reverse to the same spot.

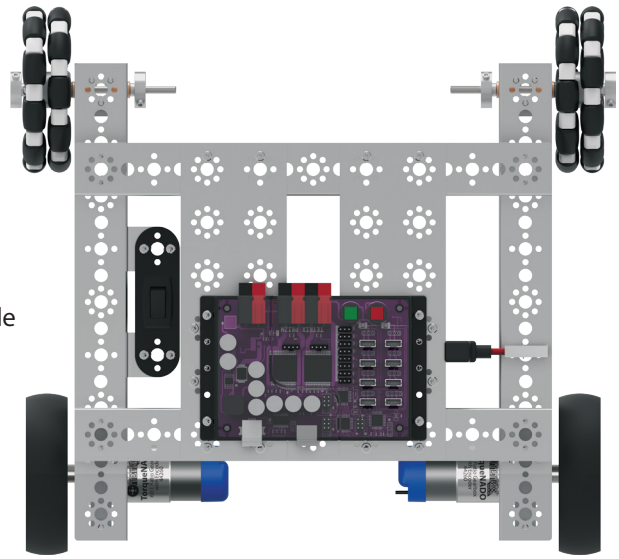
We have the fundamentals to explore speed because we can measure distance over time. In our sketch, when we use a specified time, we can physically measure how far the robot moves. When we change our power parameter, it should affect the distance traveled in the same time frame.

With that in mind, create a challenge by marking a specified distance on the floor. Using the data you have collected, program the TaskBot to get as close to the specified distance as possible without going over.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

Activity 8: Drive in a Circle

For our eighth activity we will apply our knowledge of motors to create a new behavior. While being able to move straight is important, we need to be able to expand on that and make turns. This activity will have your TaskBot driving in circles by varying motor power as the motors work in unison.



Parts Needed

- Fully assembled PRIZM TaskBot
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- Computer

Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > TaskBot_Act8_Drive_Circle**. A new sketch window will open titled TaskBot_Act8_Drive_Circle (Figure 21).

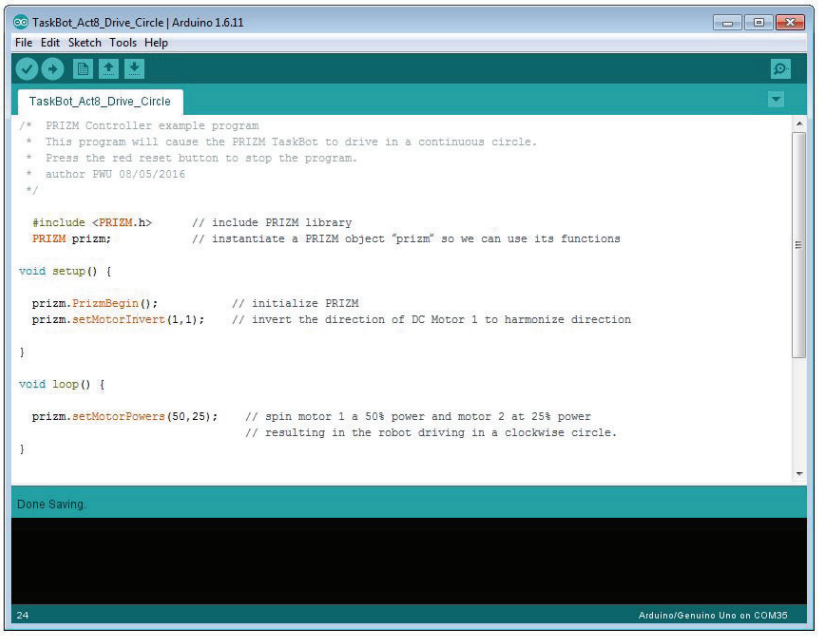


Figure 21

Before we can upload the sketch to the PRIZM, remember to check our connections.

Upload the sketch. The green LED will light up indicating the code is ready to execute. When the LED comes on, disconnect the USB cable and set the TaskBot on the floor.

Press the green Start button to execute the sketch. Observe the direction and duration of the TaskBot's motion.

Press the red Reset button to end the sketch. Based on the sketch comments, did the behavior match expectations?

Tip: You will need about five to six feet of empty floor space for the robot to complete the circle.

Moving Forward

While this sketch does not add any new functions, it should lead to a deeper understanding of how motors work in unison to create a specified behavior.

All the functions in this sketch work together to make the TaskBot move in a circle. Because we want the motors to always work together, prizm.setMotorInvert needs to be used only in the setup. The prizm.setMotorPowers tells both motors to move at different speeds in a single function. One motor is set to 50%, while the other is set to 25%, resulting in a circular motion.

For more detailed information about the sketch process and the PRIZM library functions involved in using the DC motors, refer to the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 137-142 about DC motor functions

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Real-World Connection

If you have ever seen farmland that is watered by sprinkler irrigation systems, you have likely seen an example of a machine that goes around in a circle. The typical sprinkler irrigation system has a center pivot point where the water source is connected, and the beam of the pipe system transports water to sprinkler outlets down the length of the beam. The beam is supported by wheels every 30 or 40 feet, and the wheels are powered by electric motors. Because the center pivot of the system remains in place, as the wheels rotate, they move the entire beam in a circle, irrigating the farmland as the system goes.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

STEM Extensions

Science

- Circular motion
- Tangential motion

Technology

- Motor control and precision
- Problem-solving via coding

Engineering

- Machine design for following geometric paths

Math

- Technical (mathematical) definition of a circle
- Determining radius

Hacking the Code Activity

With the example as a reference, try creating a new sketch to move the TaskBot in a different-size circle. Remember what we learned from our previous activities and experiment with different parameters to make the circle diameter larger or smaller.

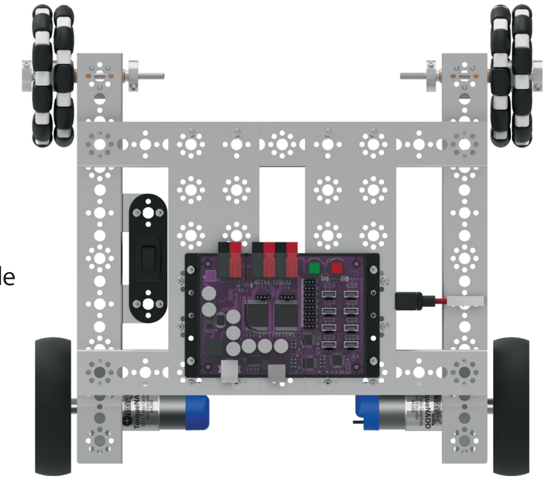
Challenge yourself to add behaviors. What would it take to make your robot drive in an oval or a figure eight?

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

Activity 9: Drive in a Square

Now, we will continue to build our navigational skills with the TaskBot by giving our robot the ability to make 90-degree turns. The ability to make 90-degree turns will be used to make the TaskBot drive in a square.



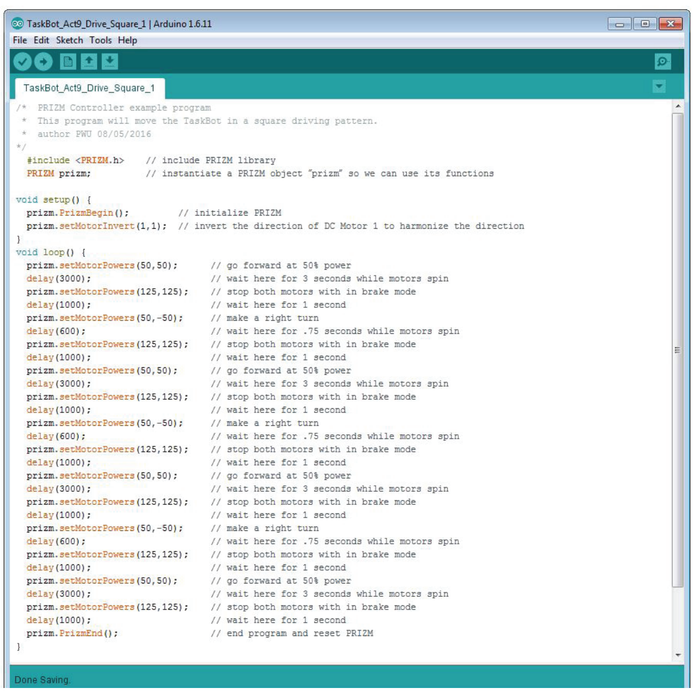
Parts Needed

- Fully assembled PRIZM TaskBot
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- Computer

Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > TaskBot_Act9_Drive_Square_1**. A new sketch window will open titled TaskBot_Act9_Drive_Square_1 (Figure 22).




```
/*  PRIZM Controller example program
 *  This program will move the TaskBot in a square driving pattern.
 *  author PWU 08/05/2016
*/
  #include <PRIZM.h>    // include PRIZM library
  PRIZM prizm;          // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {
  prizm.PrizmBegin();         // initialize PRIZM
  prizm.setMotorInvert(1,1);  // invert the direction of DC Motor 1 to harmonize the direction
}
void loop() {
  prizm.setMotorPowers(50,50);      // go forward at 50% power
  delay(3000);                      // wait here for 3 seconds while motors spin
  prizm.setMotorPowers(125,125);    // stop both motors with in brake mode
  delay(1000);                      // wait here for 1 second
  prizm.setMotorPowers(50,-50);     // make a right turn
  delay(600);                       // wait here for .75 seconds while motors spin
  prizm.setMotorPowers(125,125);    // stop both motors with in brake mode
  delay(1000);                      // wait here for 1 second
  prizm.setMotorPowers(50,50);      // go forward at 50% power
  delay(3000);                      // wait here for 3 seconds while motors spin
  prizm.setMotorPowers(125,125);    // stop both motors with in brake mode
  delay(1000);                      // wait here for 1 second
  prizm.setMotorPowers(50,-50);     // make a right turn
  delay(600);                       // wait here for .75 seconds while motors spin
  prizm.setMotorPowers(125,125);    // stop both motors with in brake mode
  delay(1000);                      // wait here for 1 second
  prizm.setMotorPowers(50,50);      // go forward at 50% power
  delay(3000);                      // wait here for 3 seconds while motors spin
  prizm.setMotorPowers(125,125);    // stop both motors with in brake mode
  delay(1000);                      // wait here for 1 second
  prizm.setMotorPowers(50,-50);     // make a right turn
  delay(600);                       // wait here for .75 seconds while motors spin
  prizm.setMotorPowers(125,125);    // stop both motors with in brake mode
  delay(1000);                      // wait here for 1 second
  prizm.setMotorPowers(50,50);      // go forward at 50% power
  delay(3000);                      // wait here for 3 seconds while motors spin
  prizm.setMotorPowers(125,125);    // stop both motors with in brake mode
  delay(1000);                      // wait here for 1 second
  prizm.PrizmEnd();                 // end program and reset PRIZM
}
```

Figure 22

Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections.

Upload the sketch. The green LED will light up, indicating the code is ready to execute. When this has occurred, disconnect the USB cable and set the TaskBot on the floor.

Press the green Start button to execute the sketch. Observe the direction and duration of the TaskBot's motion. Based on the sketch comments, did the behavior match expectations?

Moving Forward

This sketch has a lot to it, but it is made of functions that we have used before. We have just combined multiple sequential behaviors to create one larger behavior – in this case, a square.

An important thing to understand about this program is that we are using dead reckoning to program the robot's square driving path. Dead reckoning is simply using time as the basis for controlling a motor. For instance, to make a right-hand turn, we are commanding the motors to turn on in opposite directions at 50% power and run for 600 milliseconds.

We estimate that if we spin the motors at a certain rpm for a certain time period, we should come close to making a 90-degree right turn. However, this is not always accurate because the amount that our robot's battery is charged can vary and any wheel slippage on the surface we are working on will cause errors. In essence we are dead reckoning on making a right turn.

All the functions in this sketch work together to make the TaskBot move in a square. Because we want the motors to always work together, prizm.setMotorInvert needs to be used only in the setup. The prizm.setMotorPowers tells both motors to move at different speeds in a single function.

For more detailed information about the sketch process and the PRIZM library functions involved in using the DC motors, refer to the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 137-142 about DC motor functions

Real-World Connection

Some vehicles take the same route time after time. For instance, a vehicle for transporting packages from three different locations to a final destination within a warehouse could have a route that would look like a square. This vehicle is programmed to go an exact distance forward before it turns to go that same distance at a 90-degree angle. After repeating this four times, the vehicle is back where it started, ready for another cycle of package transport.

Tip: This program is one that could be affected by the charge level of the battery. Why? The pattern of this program is generated by a series of delay functions, and as motor rpm changes due to battery levels, the square pattern can change somewhat as batteries drain. That is one of the consequences of using "dead reckoning" as a method of control.

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

STEM Extensions

Science

- Relationship among time, distance, and velocity

Technology

- How time is measured within a microprocessor
- Measuring turning radius

Engineering

- Program design for path following

Math

- Relationship of sides and angles of a square
- Measurement of angles

Hacking the Code Activity

With the example as a reference, try creating a new sketch to move the TaskBot in a square. Remember what we learned from our previous activities and experiment with different parameters to make the square larger or smaller.

Challenge yourself to create complex paths beyond a square. Try tracing out some of the alphabet or navigate through a simple maze using dead reckoning.

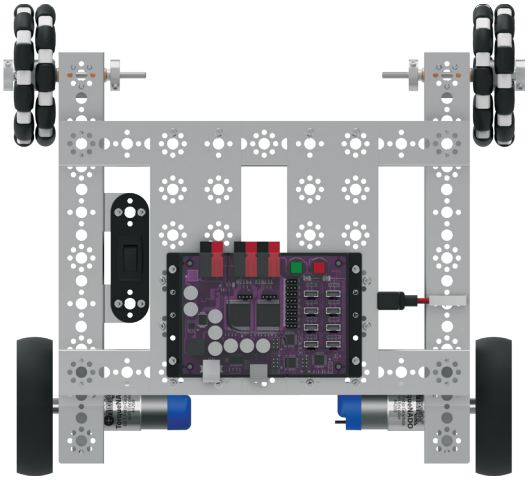
Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

Note: Keep this activity in mind. We will revisit this behavior in the next activity in order to flex our coding muscles.

Activity 10: Simplify the Square

Remember the square activity we did in Activity 9? For our next activity we want to show you a more efficient way to code the same behavior.



Parts Needed

- Fully assembled PRIZM TaskBot
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- Computer

Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > TaskBot_Act10_Drive_Square_2**. A new sketch window will open titled TaskBot_Act10_Drive_Square_2 (Figure 23).

Figure 23

Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections.

Upload the sketch. The green LED will light up, indicating the code is ready to execute. When this has occurred, disconnect the USB cable and set the TaskBot on the floor.

Press the green Start button to execute the sketch. Observe the direction and duration of the TaskBot's motion. Based on the sketch comments, did the behavior match expectations?

Tip: A function can be defined in one area of your sketch and then referenced, or "called," in another area. This creates subroutines that help keep your sketch concise.

Moving Forward

This sketch implements one new program structure in the form of the "for" statement, and it adds a new way to implement functions with called functions. Both of these will combine to make a simpler, easier-to-read sketch.

The "for" statement is used to repeat a block of code enclosed in curly braces. An increment counter is used to count the iterations and terminate the loop. The "for" statement is useful for limiting the number of times a loop will execute.

Called functions occur outside the setup() and loop() functions. They are used most often when the same action needs to occur multiple times in a sketch.

When we made the TaskBot drive in a square in Activity 9 we listed each action line by line, resulting in excessive code that looked more complicated than it was. We can identify specific actions that were repeated several times in the complete behavior. We can define those identified actions in a called function outside the loop(). After the called functions are defined outside the loop(), we can use them in the loop() to simplify the code.

For this sketch we defined a forward() and a rightTurn() outside the loop(). Because we defined these two functions, we can now call them in the loop(). We start the loop() with the "for" statement to define how many times we want the loop to repeat.

There are three parts to the "for" statement. The first part, int x=0, is the initialization. This will happen first and only once.

The second part, x<=3, is the condition. Each time through the loop(), the condition is tested. If it is true, the third part, x++, the increment, is executed, and the condition is tested again. When that condition becomes false, the loop() ends.

In simple terms, the initialization starts the loop(). The condition defines how many times we want the loop to execute, which is three times plus the first initialization. The total times the loop() will execute is four. And the increment defines how it counts each execution of the loop(), which is by one increment.

For more detailed information about the sketch process and the PRIZM library functions involved in using the DC motors, refer to the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 137-142 about DC motor functions

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Real-World Connection

In the process of learning to shoot free throws, you try to train your mind and muscles to repeat a certain set of controlled actions that will cause the basketball to go through the hoop. Repeatability, or the ability to consistently repeat an operation with little or no variation in the results, is critical in many sports but is also critical in robot design and coding. Determining a method (or code) to have a robot precisely follow a square pattern provides evidence that the robot can perform a repetitive action with little or no variation.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

STEM Extensions

Science

- Relationship between time, distance, and velocity

Technology

- How time is measured within a microprocessor
- Measuring turning radius

Engineering

- Program design for path following

Math

- Relationship of sides and angles of a square
- Measurement of angles

Hacking the Code Activity

With the example as a reference, try creating a new sketch to move the TaskBot in a square using the "for" statement and called functions. Or for an additional challenge, create complex paths beyond a square such as rectangles or hexagons.

Remember what we learned from our previous activities. The challenge from here is to move forward with your own unique robot builds and apply the coding knowledge you have learned in new and exciting ways.

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

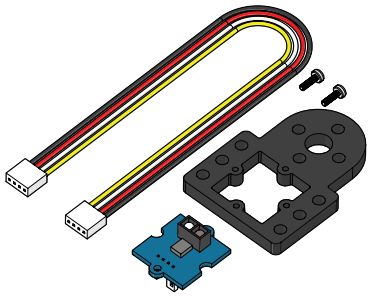
Building Interlude: Make the TaskBot Smart!

Up until now our robot has been good at following instructions, but it has not been able to make any decisions. We know from the getting started activities that sensors enable robots to sense the environment around them. We can give robots the ability to make decisions based on the information that sensors provide, thereby appearing smart.

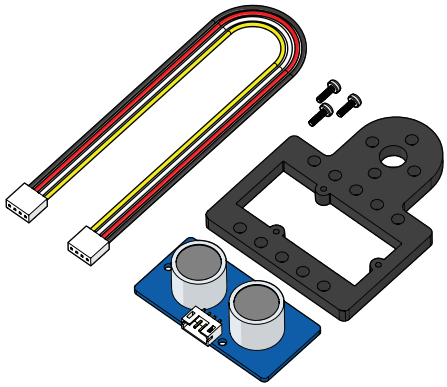
In the next coding examples, we will attach sensors to the PRIZM TaskBot. We will work though two examples using the Line Finder Sensor and two examples using the Ultrasonic Sensor. Instructions for attaching sensors to the PRIZM TaskBot are shown below.

Step 1

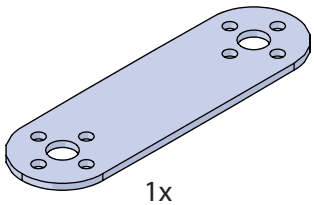
Parts Needed



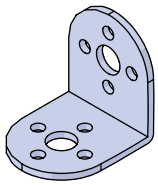
1x
Line Finder Sensor Pack 43056



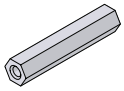
1x
Ultrasonic Sensor Pack 43055



1x
Flat Bracket 39061



1x
L Bracket 39062



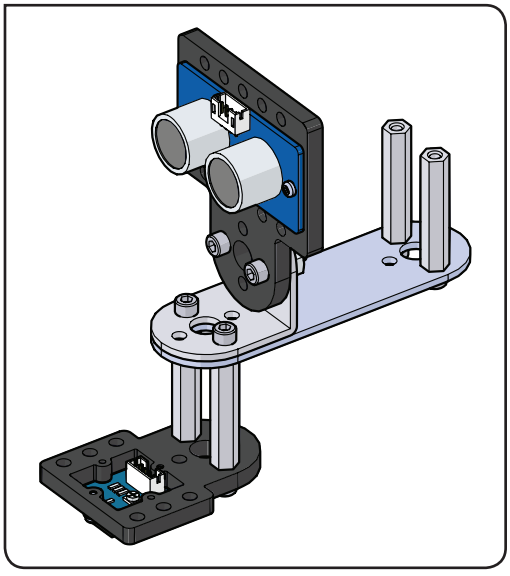
4x
Stand-Off Post
6-32 x 32 mm 39107

6x
Socket Head Cap Screw
6-32 x 1/2" 39097

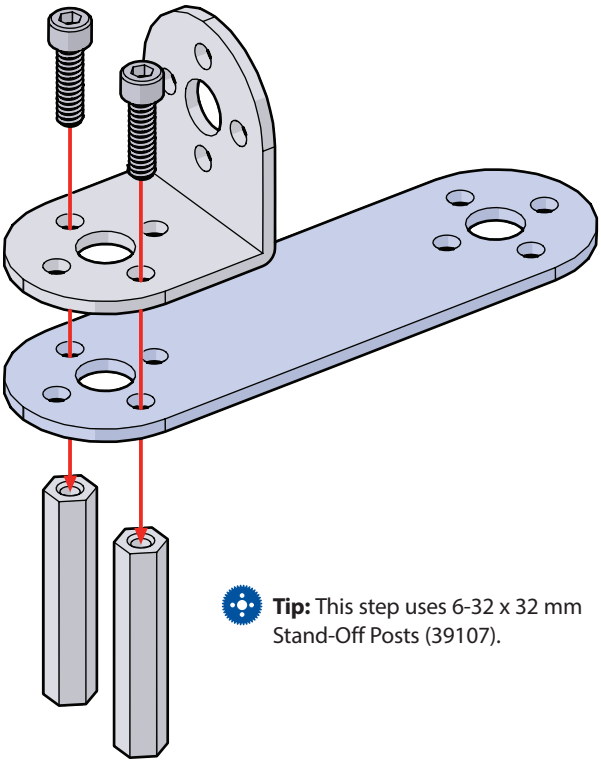
2x
Socket Head Cap Screw
6-32 x 5/16" 39098

2x
Kep Nut
39094

Finished assembly should look like this.

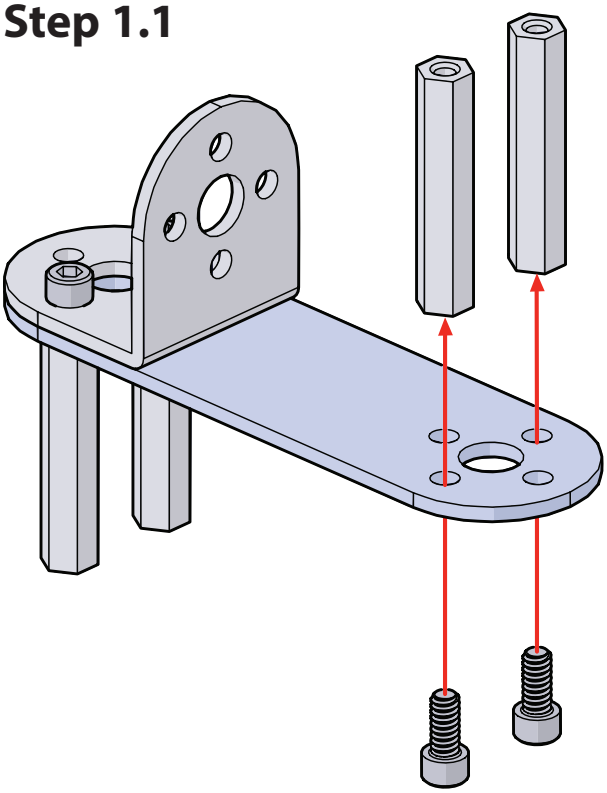


Step 1.0



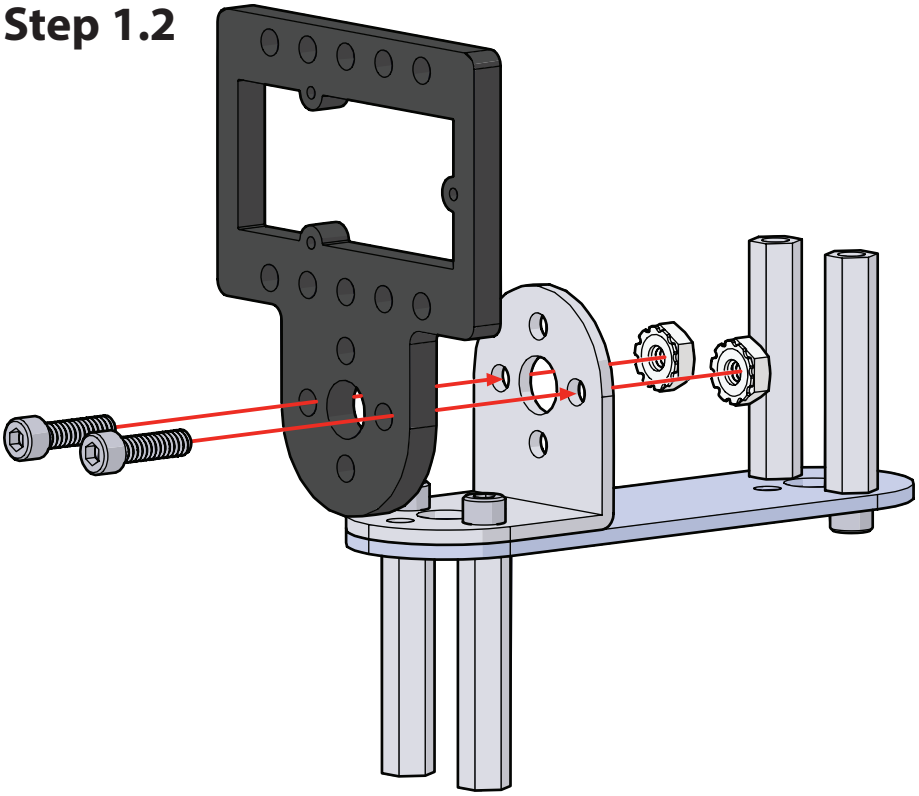
Tip: This step uses 6-32 x 32 mm Stand-Off Posts (39107).

Step 1.1

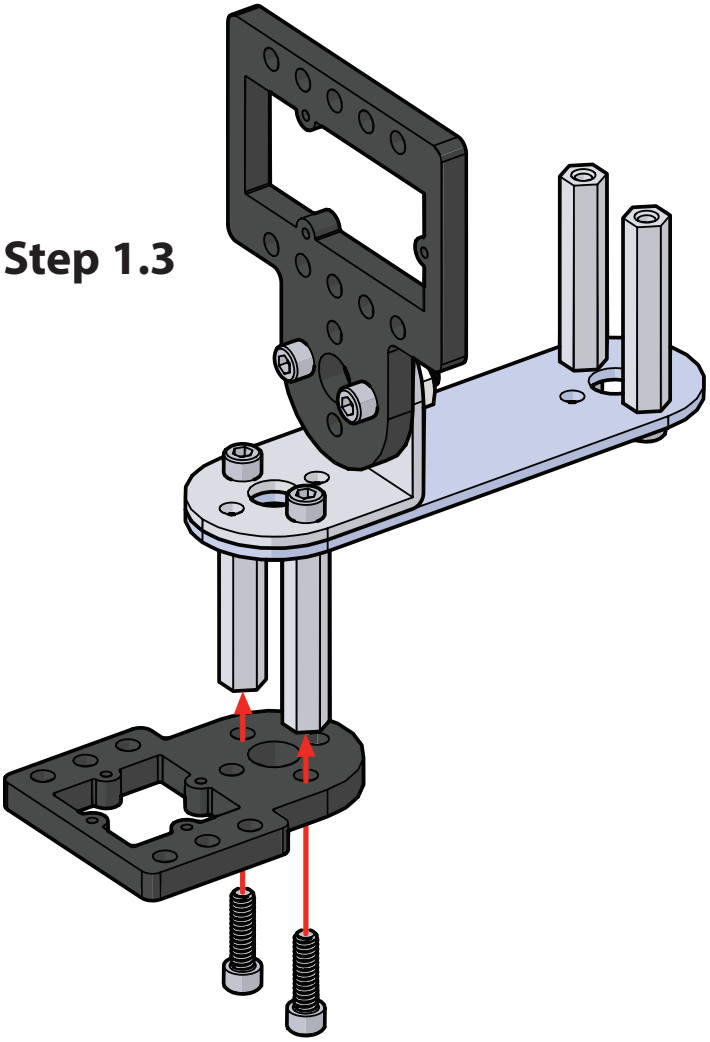


Tip: This step uses 6-32 x 32 mm Stand-Off Posts (39107).

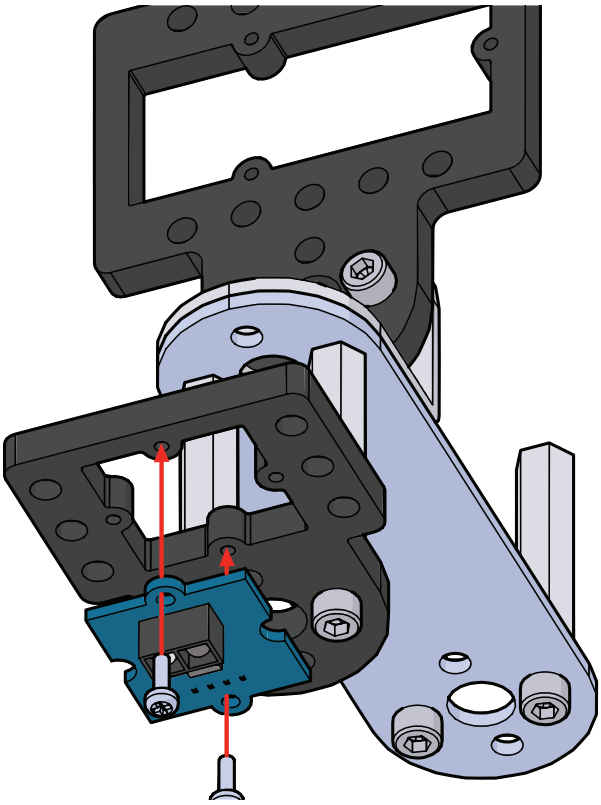
Step 1.2



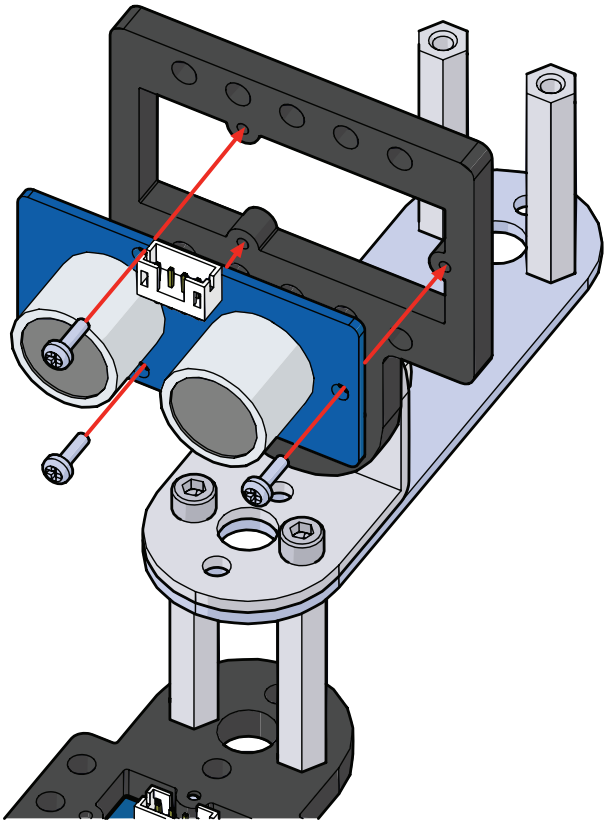
Step 1.3



Step 1.4



Step 1.5

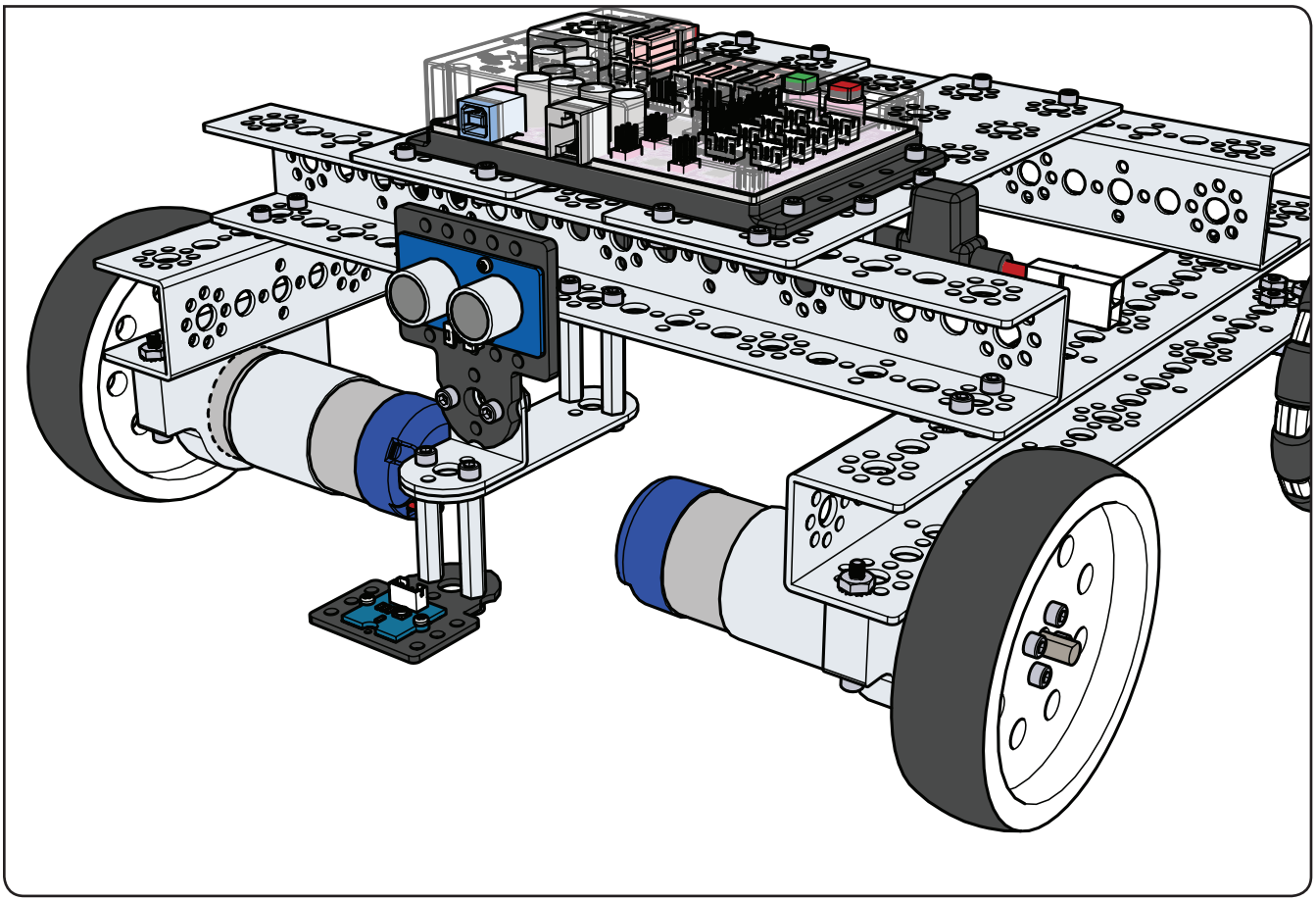


Step 2

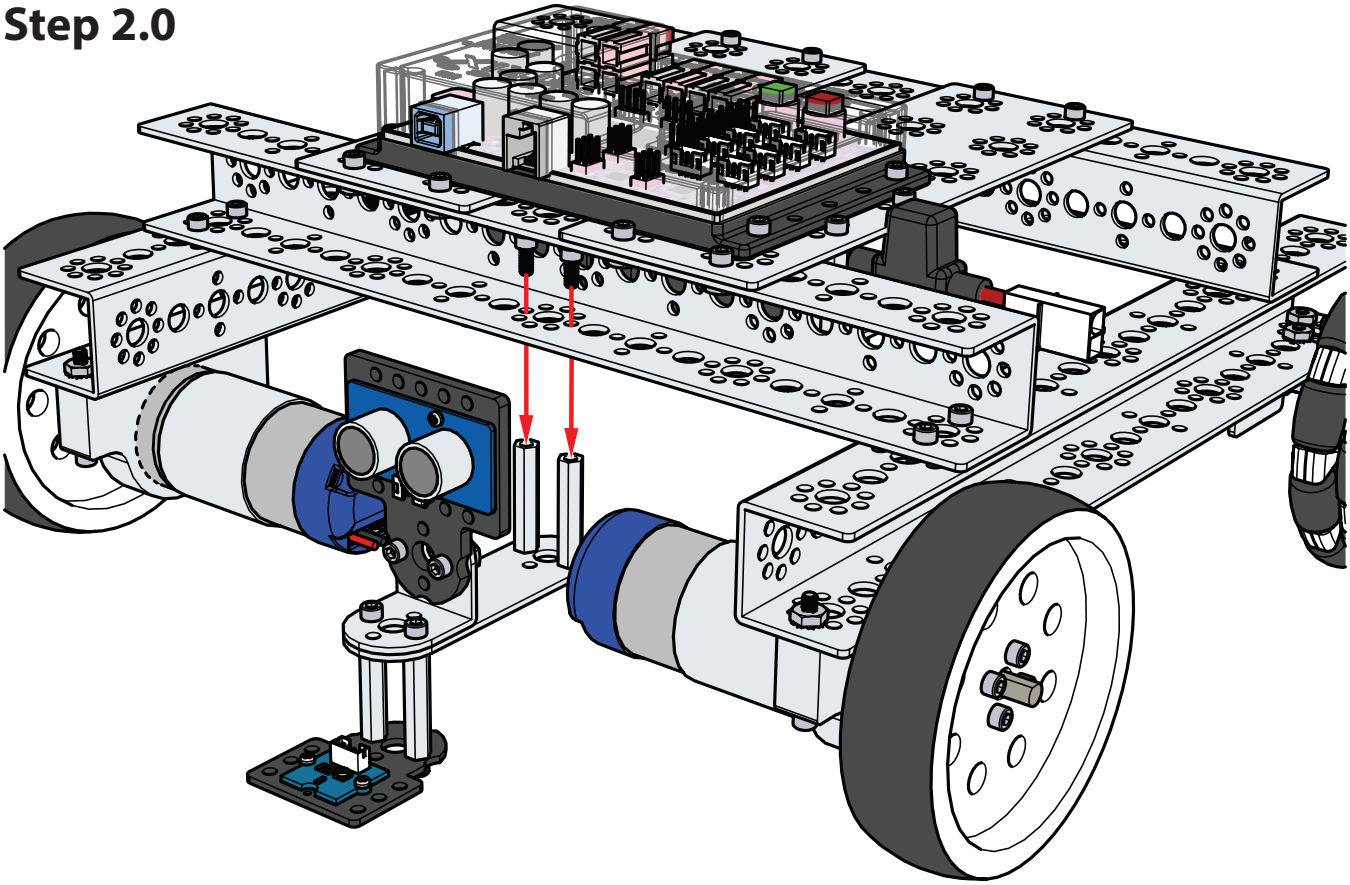
Parts Needed

2x
Socket Head Cap Screw
6-32 x 5/16" 39098

Finished assembly should look like this.

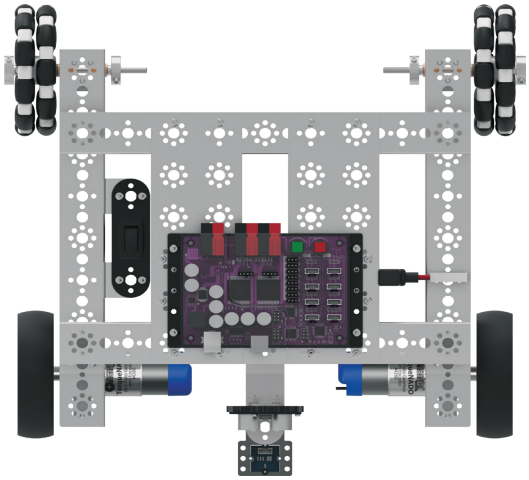


Step 2.0



Activity 11: Drive to a Line and Stop

Starting simple like we have in all of our activities, we will begin by adding the Line Finder Sensor to a basic move forward behavior. This will enable the TaskBot to stop based on the environment around it. In this activity the TaskBot will drive forward and stop at a line.



Parts Needed

- Contrasting light and dark surface
- Fully assembled PRIZM TaskBot complete with sensor module
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- Computer

Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > TaskBot_Act11_Drive_To_Line**. A new sketch window will open titled TaskBot_Act11_Drive_To_Line (Figure 24).

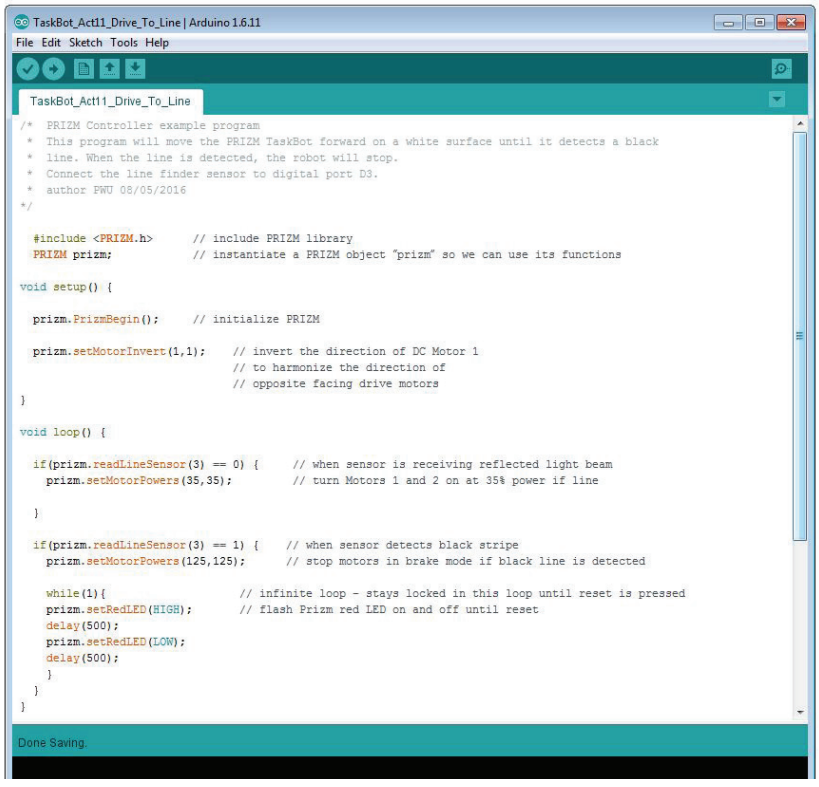


Figure 24

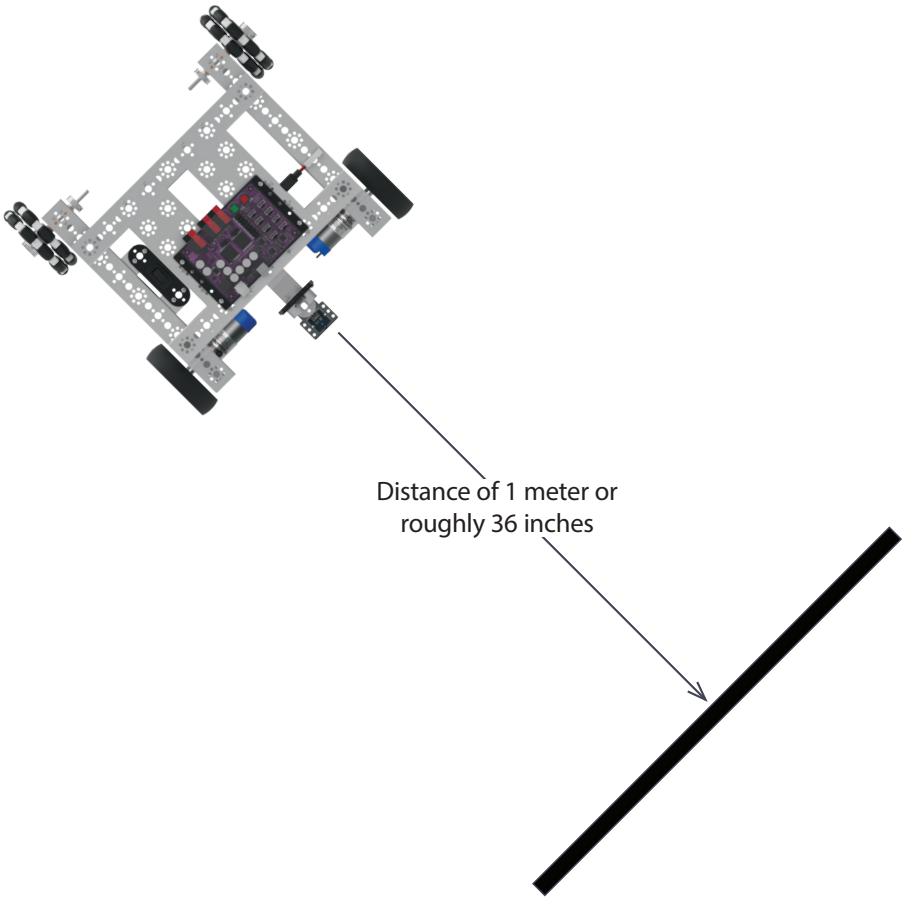
Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections. The Line Finder Sensor should be in digital sensor port 3.

Upload the sketch. The green LED will light up, indicating the code is ready to execute. When this has occurred, disconnect the USB cable and set the TaskBot on the floor.

The TaskBot should be on a white or reflective surface pointed toward a black line or non-reflective surface approximately 36 inches or one meter away.

Press the green Start button to execute the sketch. Observe the behavior of the robot. Press the red Reset button to end the sketch. Based on the sketch comments, did the behavior match expectations?



Tip: Be sure to check that the Line Finder Sensor is plugged into the correct sensor port and that it is adjusted properly to sense the line. Some height adjustment might be needed, or the small adjustment screw on the back side of the sensor module might need to be tweaked. To see if the sensor is working properly, manually move the robot back and forth over the black line and white surface. The red LED on the Line Finder Sensor should be on when the sensor is over the white surface and off when it is over the black line.

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

Moving Forward

This sketch uses two "if" statements, a program structure we used in Activity 4, and it introduces a "while" loop, which is a new program structure.

A "while" loop looks at a test condition inside parentheses and will loop continually until the test condition inside the parentheses becomes false.

In this sketch the "if" statements are looking at the condition of the Line Finder Sensor. The first "if" statement tells the TaskBot to move at 35% power when over a white or reflective surface. The second "if" statement contains two parts. The first part tells the TaskBot to brake when over a black line or non-reflective surface. The second part contains the "while" loop that locks the sketch into blinking the red LED until the sketch is reset.

For more detailed information about the sketch process and the PRIZM library functions involved in using the DC motors, refer to **www.arduino.cc** and the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 134-135 about sensor port functions
- Pages 137-142 about DC motor functions

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

Real-World Connection

For people who drive cars, pulling up to an intersection with a crosswalk is a common occurrence. For the safety of any pedestrians, you need to stop your car at or before that crosswalk. After some experience this becomes second nature to you as a driver. However, robots do not learn in the same manner as humans, so they have to have automated devices and sensors to know when and where to stop.

STEM Extensions

Science

- Velocity formula
- Constant acceleration in a straight line

Technology

- "While" loops
- Reading sensor values

Engineering

- Applying problem-solving strategies

Math

- Calculations for distance, time, and velocity
- Logic statements (for example, "if – then")

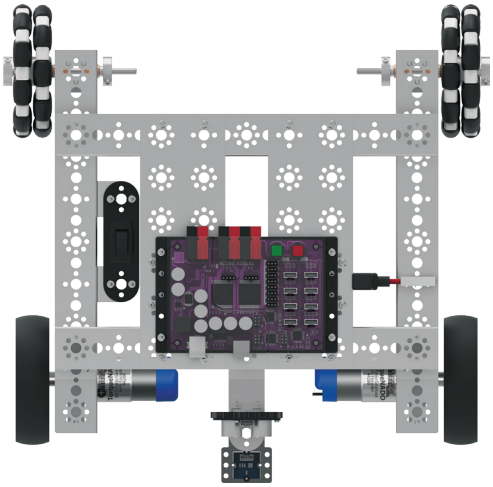
Hacking the Code Activity

With the example as a reference, try creating a new sketch to have the TaskBot drive to a black line and stop. Remember what we learned from our previous activities.

Challenge yourself to create different behaviors when the TaskBot meets a line such as backing up or turning a different direction.

Activity 12: Follow a Line

For this activity we can take what we learned in the previous activity and apply it in a slightly different way to create a new behavior. This will enable the TaskBot to follow a line.



Parts Needed

- Contrasting light and dark surface
- Fully assembled PRIZM TaskBot complete with sensor module
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- Computer

Black line minimum of 2 inches wide

Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > TaskBot_Act12_Follow_A_Line**. A new sketch window will open titled TaskBot_Act12_Follow_A_Line (Figure 25).

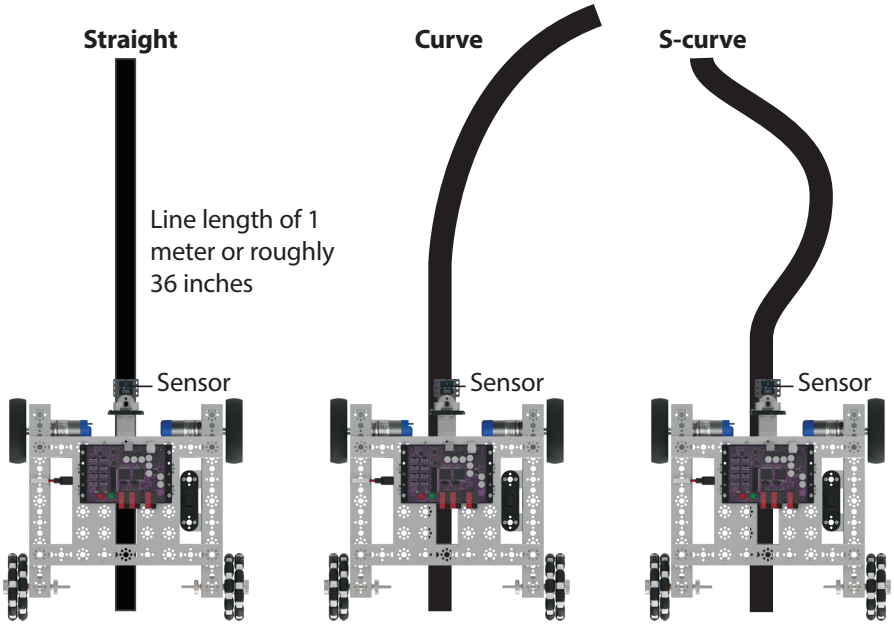
Figure 25

Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections. The Line Finder Sensor should be in digital sensor port 3.

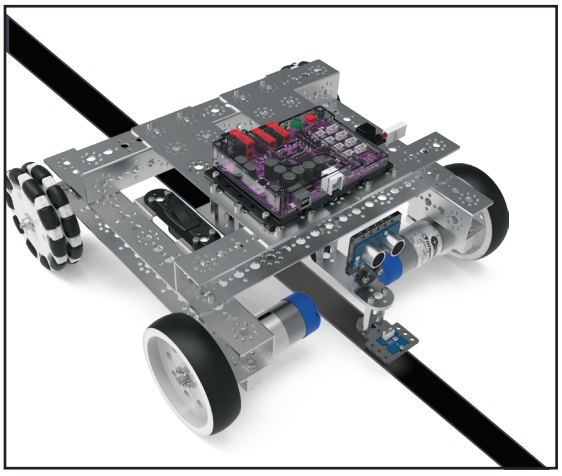
To run this code we will need a white or light-reflective surface on which to place a black stripe for the robot to traverse. White, glossy cardboard or several sheets of paper tiled together can work.

Our black stripe can be made using electrical tape stuck down to the cardboard or paper tiles. We can make a curvy path or just a straight line. There will be a limitation on how sharp a curve our robot will be able to follow, so do not make the curves too tight.



Upload the sketch. The green LED will light up, indicating the code is ready to execute. When this has occurred, disconnect the USB cable and set the TaskBot on the floor.

The TaskBot should be on a white or reflective surface with the Line Finder Sensor slightly to the side of a line to follow. Press the green Start button to execute the sketch. Observe the behavior of the robot.



Press the red Reset button to end the sketch. Based on the sketch comments, did the behavior match expectations?

Tip: Be sure to check that the Line Finder Sensor is plugged into the correct sensor port and that it is adjusted properly to sense the line. Some height adjustment might be needed, or the small adjustment screw on the back side of the sensor module might need to be tweaked. To see if the sensor is working properly, manually move the robot back and forth over the black line and white surface. The red LED on the Line Finder Sensor should be on when the sensor is over the white surface and off when it is over the black line.

Moving Forward

This sketch uses two "if" statements, which we are becoming increasingly familiar with. We have added a second action to execute based on the test condition for each "if" statement.

The first action of the "if" statements deals with the motors. The "if" statement tells the TaskBot to turn by sending power to one motor and braking the other based on the condition of the Line Finder Sensor. Each "if" statement is the opposite of the other. This creates a progressive series of turns that the TaskBot uses to follow the line.

The second action of the "if" statements deals with the red LED on the PRIZM. The "if" statement tells the red LED to turn on or off based on the condition of the Line Finder Sensor. This on-and-off action is synced with the turning action of the motors.

The actions in each "if" statement work together to make the TaskBot follow the line while producing visual cues in the form of the blinking red LED.

For more detailed information about the sketch process and the PRIZM library functions involved in using the DC motors, refer to **www.arduino.cc** and the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 134-135 about sensor port functions
- Pages 137-142 about DC motor functions

Real-World Connection

German brand PUMA is taking a new approach to physical fitness with the launch of its PUMA BeatBot. The BeatBot is a line-following robot that acts as a pacesetter that a runner can race against. The runner is able to give the robot information such as distance and pace through his or her smartphone. Then, it's off to the track, where the runner attempts to beat the BeatBot.

STEM Extensions

Science

- Light reflection and absorption
- Angle of incidence and reflection

Technology

- Threshold values
- Black and white (analog) acting as "on" and "off" (digital)

Engineering

- Designing machines to work close to edges

Math

- Applying logic statements to solve problems

Hacking the Code Activity

With the example as a reference, try creating a new sketch to have the TaskBot follow a black line. Remember what we learned from our previous activities.

Challenge yourself to make the TaskBot go faster and still follow the line accurately or change the code to make the TaskBot follow the line in the opposite direction.

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

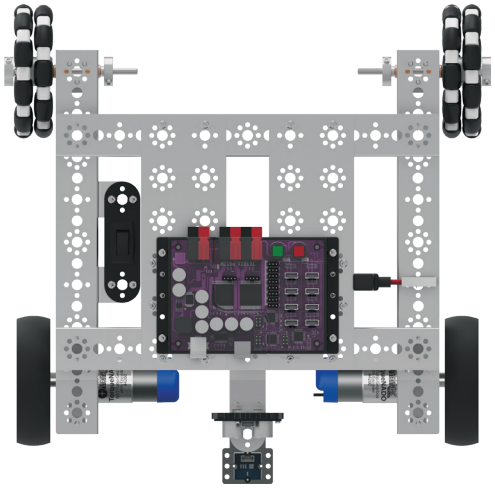
Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

Activity 13: Drive Toward a Wall and Stop

With this activity we will continue to build on what we have learned and apply it to the Ultrasonic Sensor. This will enable the TaskBot to drive toward a wall and stop a specified distance away.



Parts Needed

- Fully assembled PRIZM TaskBot complete with sensor module
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- Computer

Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > TaskBot_Act13_Drive_To_Wall**. A new sketch window will open titled TaskBot_Act13_Drive_To_Wall (Figure 26).

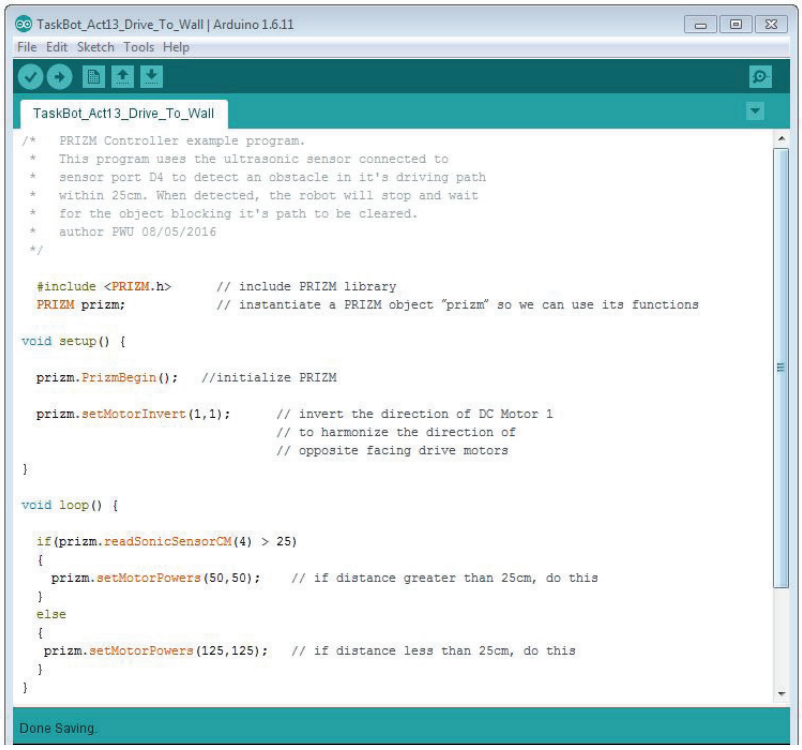


Figure 26

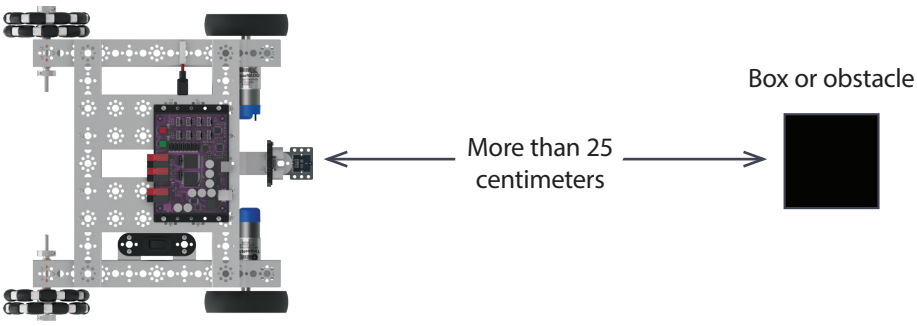
Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections. The Ultrasonic Sensor should be in digital sensor port 4.

Upload the sketch. The green LED will light up, indicating the code is ready to execute. When this has occurred, disconnect the USB cable and set the TaskBot on the floor.

The TaskBot should be pointed toward an object or wall at least 25 centimeters away. Press the green Start button to execute the sketch. Observe the behavior of the robot. What happens if the TaskBot or object is moved before the sketch is ended?

Press the red Reset button to end the sketch. Based on the sketch comments, did the behavior match expectations?



Tip: Be sure to check that the Ultrasonic Sensor is plugged into the correct sensor port and plugged in correctly. For this sketch, the correct sensor port is D4. Keep in mind that objects without adequate surface size or that have an irregular surface might not be detected or affect the distance the sensor reads. To see if the sensor is working properly, you can always go back to Activity 5; load and run that sketch to check the serial monitor and verify the output from the sensor. If you need to do this, remember to either physically change the sensor port the Ultrasonic Sensor is plugged into to match the sketch example or change the sketch example.

Moving Forward

This sketch uses an "if/else" statement, which is a new programming structure.

An "if/else" statement gives greater control over the flow of code than the basic "if" statement. It enables multiple tests to be grouped together so they can be run at the same time. In other words, the "if" part of the statement says if this condition is true, do this. The "else" part of the statement says if this condition is false, do this instead.

In this sketch the first part of the "if/else" statement tests the condition of the Ultrasonic Sensor, and if the sensor is greater than 25 centimeters away from an object, it sets the power for the motors at 50%. The second part of the "if/else" statement uses the same test, but if it is less than 25 centimeters away from an object, it gives an alternate action and tells the motors to brake.

For more detailed information about the sketch process and the PRIZM library functions involved in using the DC motors, refer to **www.arduino.cc** and the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 134-135 about sensor port functions
- Pages 137-142 about DC motor functions

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

Real-World Connection

In many newer cars, sensors are included to let drivers know when they are close to another car or a wall when they are pulling into a parking space. As the car gets closer to the wall, it will beep at the driver to alert him or her to slow and stop before hitting the wall.

STEM Extensions

Science

- Speed of sound
- Composition of sound waves

Technology

- Sensor calibration

Engineering

- Designing machines to detect obstacles

Math

- Distance computations based on the speed of sound

Hacking the Code Activity

With the example as a reference, try creating a new sketch to have the TaskBot drive toward a wall or object and stop. Remember what we learned from our previous activities.

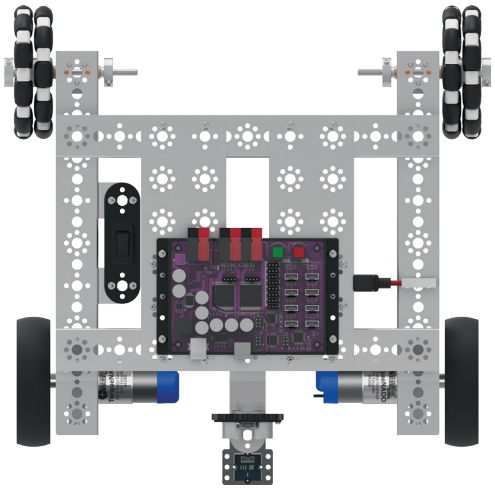
Challenge yourself by changing the direction, range, or speed. You can also test what size and shape of object the Ultrasonic Sensor is more likely to detect and at what range.

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

Activity 14: Avoiding Obstacles

For this activity we are going to expand on the "if/else" statement by adding actions. This will enable the TaskBot to avoid obstacles in its path.



Parts Needed

- Multiple objects to avoid
- Fully assembled PRIZM TaskBot complete with sensor module
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- Computer

Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > TaskBot_Act14_ Avoid_Obstacle**. A new sketch window will open titled TaskBot_Act14_Avoid_ Obstacle (Figure 27).

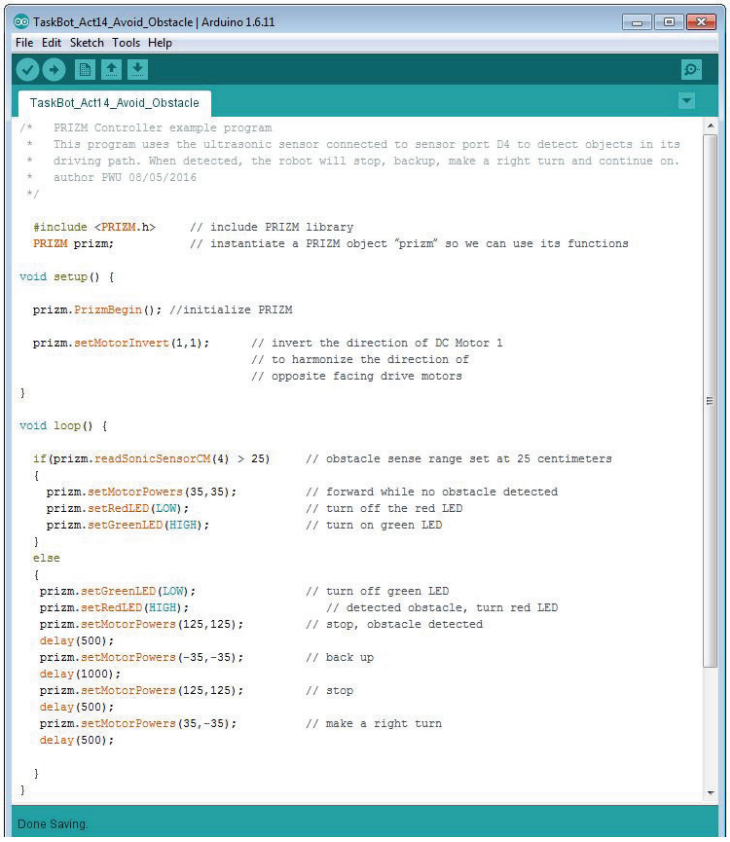


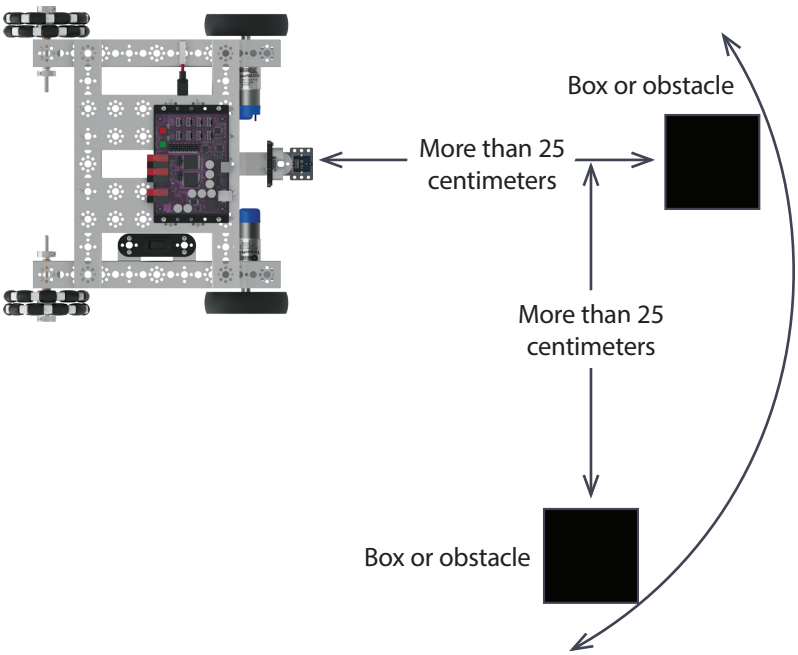
Figure 27

Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections. The Ultrasonic Sensor should be in digital sensor port 4.

Upload the sketch. The green LED will light up, indicating the code is ready to execute. When this has occurred, disconnect the USB cable and set the TaskBot on the floor.

When running this code, set an obstacle in front of our robot at a distance well outside the 25-centimeter detection range. Cardboard boxes work well for this purpose. We do not want anything too heavy that could cause damage to our robot just in case our robot crashes into it.



Tip: Be sure to check that the Ultrasonic Sensor is plugged into the correct sensor port and plugged in correctly. For this sketch, the correct sensor port is D4. Keep in mind that objects without adequate surface size or that have an irregular surface might not be detected or affect the distance the sensor reads. To see if the sensor is working properly, you can always go back to Activity 5; load and run that sketch to check the serial monitor and verify the output from the sensor. If you need to do this, remember to either physically change the sensor port the Ultrasonic Sensor is plugged into to match the sketch example or change the sketch example.

To begin, press the green Start button. Our robot will travel forward until it senses that it is within 25 centimeters of the object in its path.

When an object is detected, the robot will stop, back up, make a right turn, and continue. Observe the behavior of the robot.

Press the red Reset button to end the sketch. Based on the sketch comments, did the behavior match expectations?

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Moving Forward

This sketch continues using the "if/else" statement from the previous activity, but we add multiple actions to the statement.

Each part of the "if/else" statement here has multiple actions. In the "if" part of the statement, the TaskBot is moving forward at 35% power with the green LED on while the sensor watches for an object. In the "else" part of the statement when the TaskBot detects an object, the green LED turns off and the red LED turns on, and the TaskBot stops, backs up, and turns right. The loop then continues.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

For more detailed information about the sketch process and the PRIZM library functions involved in using the DC motors, refer to **www.arduino.cc** and the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 134-135 about sensor port functions
- Pages 137-142 about DC motor functions

Real-World Connection

Cars are beginning to include sensors to detect the world around them as they move. Some cars now have automatic braking systems to begin the braking process and slow the car down if they sense a situation in which a collision might occur. The electronics involved provide a quicker reaction time than is possible for a human. This extra amount of time can be very beneficial in minimizing the effects of a collision.

STEM Extensions

Science

- Physics of sound reflection from a flat surface
- Physics of sound reflection from a round surface
- Physics of sound reflection from a rough surface

Technology

- Interpreting data from the Ultrasonic Sensor

Engineering

- Designing machines to avoid obstacles

Math

- Data graphing
- Data averaging

Hacking the Code Activity

With the example as a reference, try creating a new sketch to have the TaskBot avoid obstacles. Remember what we learned from our previous activities.

Challenge yourself and experiment with different objects or make an obstacle course with multiple objects to detect. We can experiment and modify the example code as needed to change the reaction and behavior of our robot's obstacle avoidance skills.

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

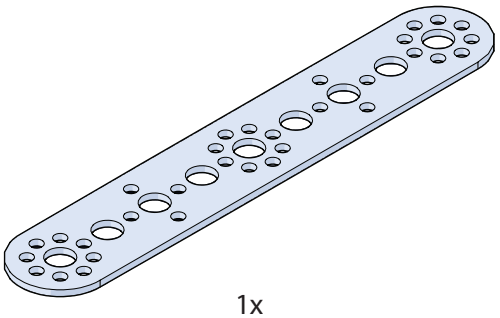
Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

Building Interlude: Give the TaskBot Attitude!

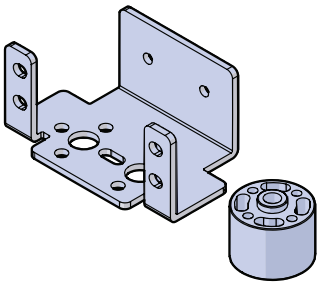
Before we create more code, we are going to add a servo motor to the PRIZM TaskBot. This will bring everything together that we used in the getting started activities. All the different kinds of motors and sensors will be used on the TaskBot to give our robot some attitude.

We will use the servo to signal when an object has been detected in the robot's path. The servo will raise a lever to signal that an object has been detected. Follow the steps below to add a servo and lever arm to the PRIZM TaskBot.

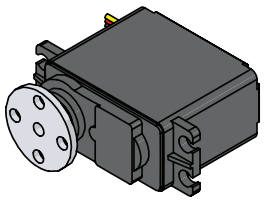
Parts Needed



1x
Flat 160 mm x 27 mm
39272



1x
Single Standard-Scale
Servo Motor Bracket
39060

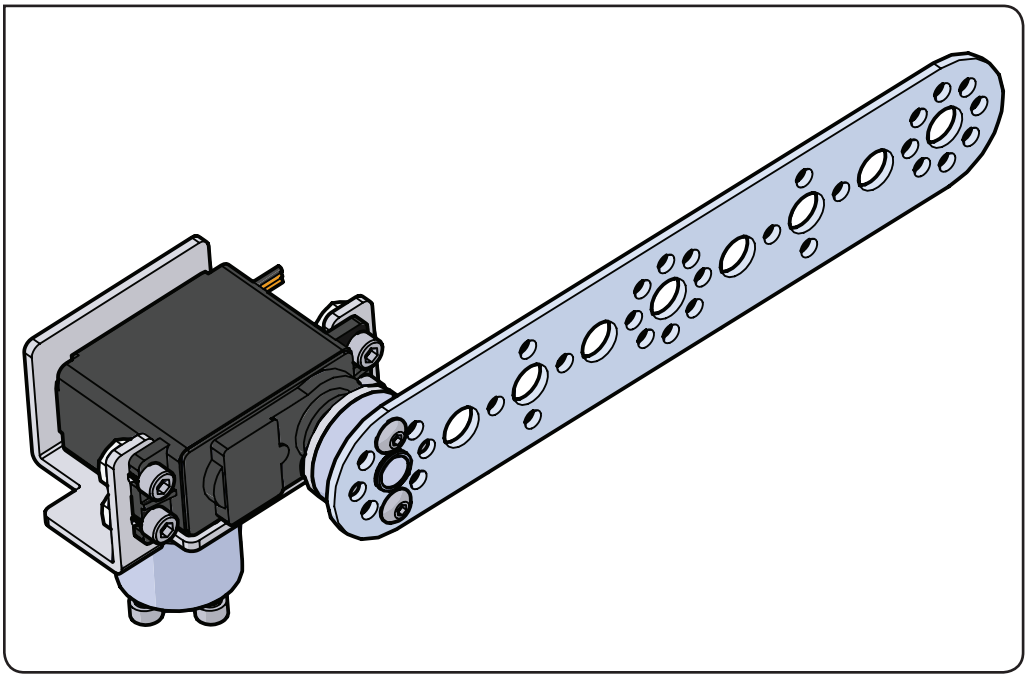


1x
Standard-Scale Servo
Motor 39197

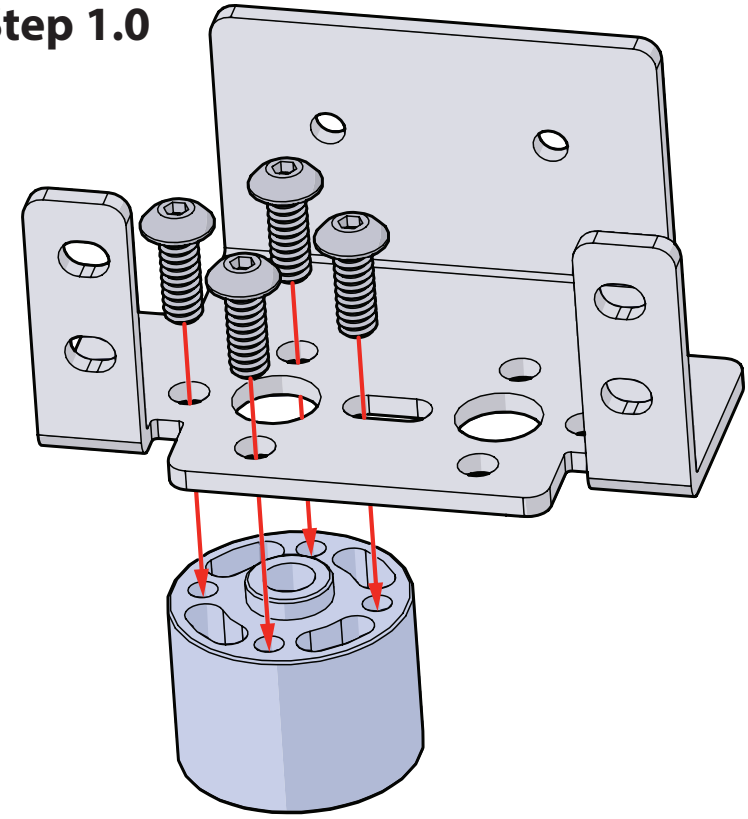
6x
Button Head Cap Screw
3/8" 39111

8x
Socket Head Cap Screw
6-32 x 5/16" 39098

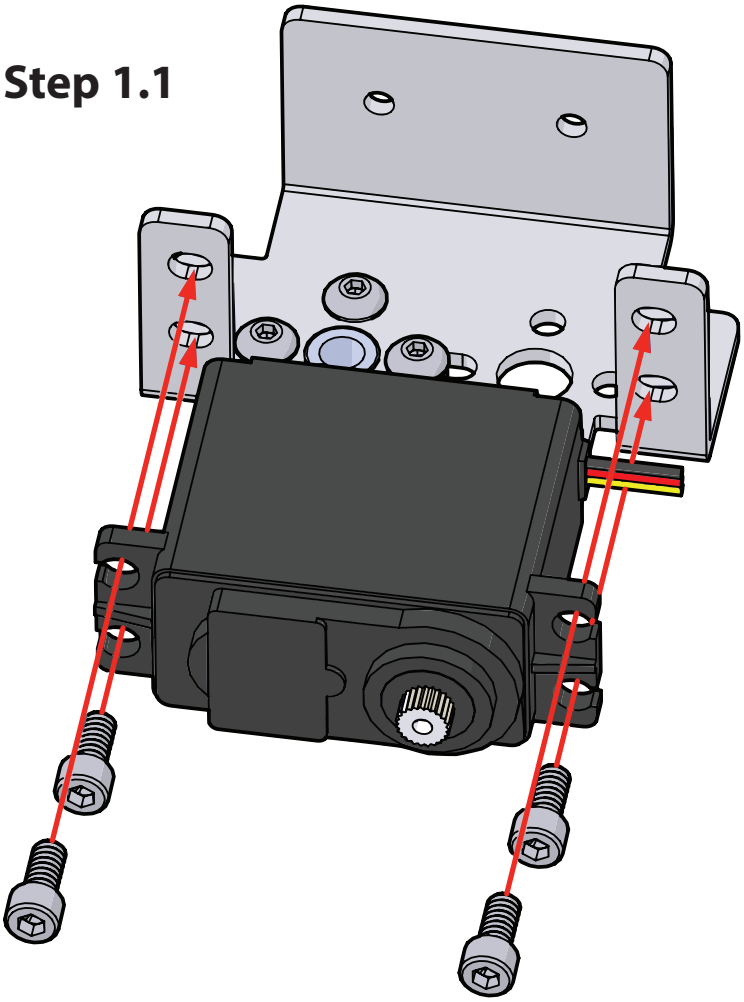
6x
Kep Nut
39094



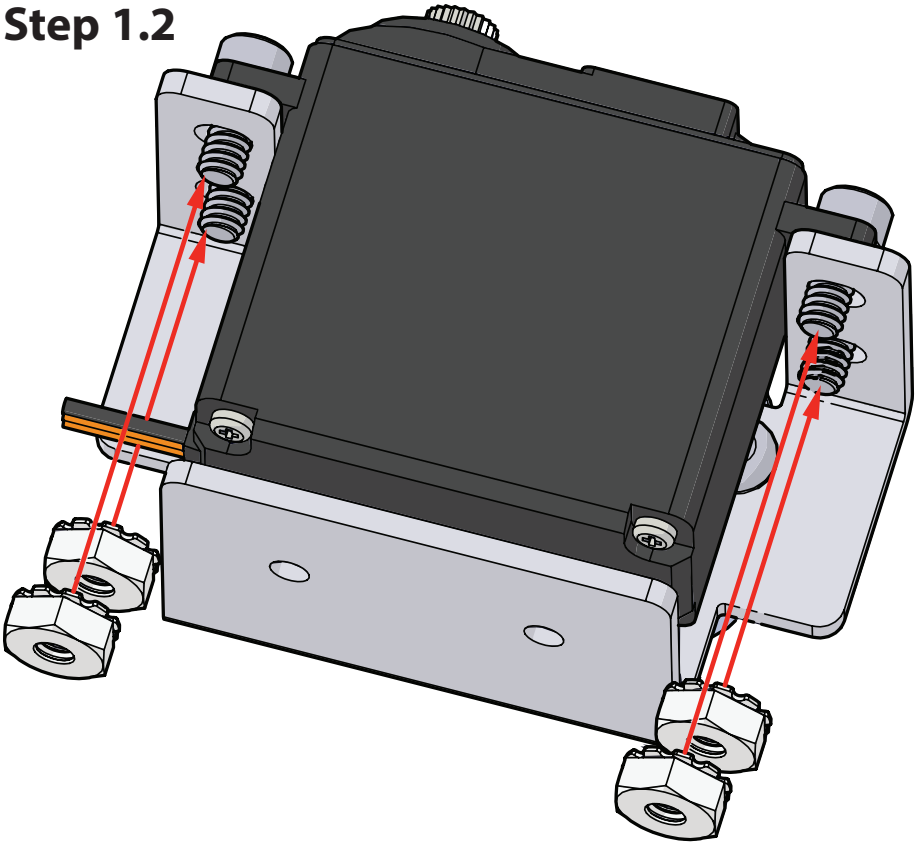
Step 1.0



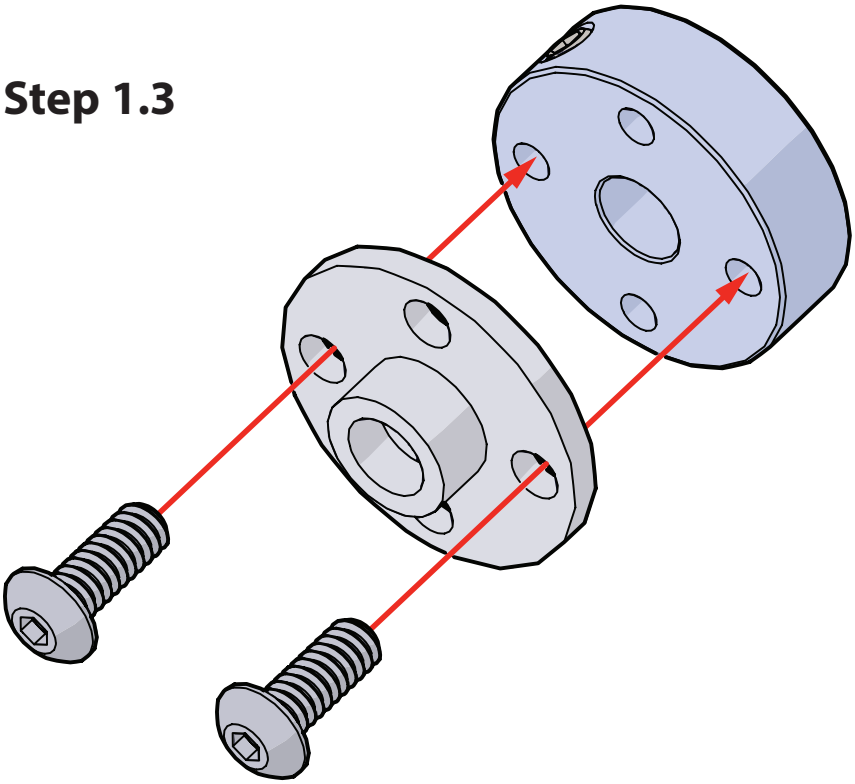
Step 1.1



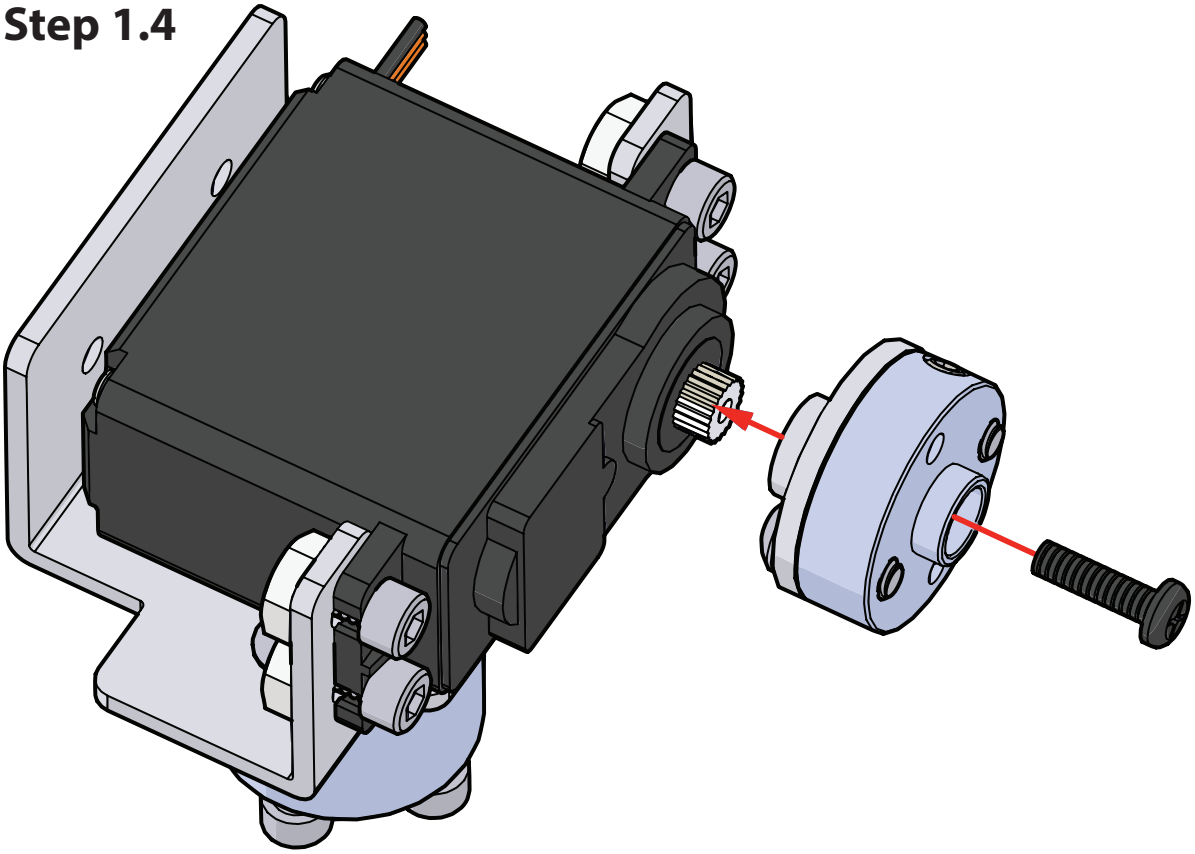
Step 1.2



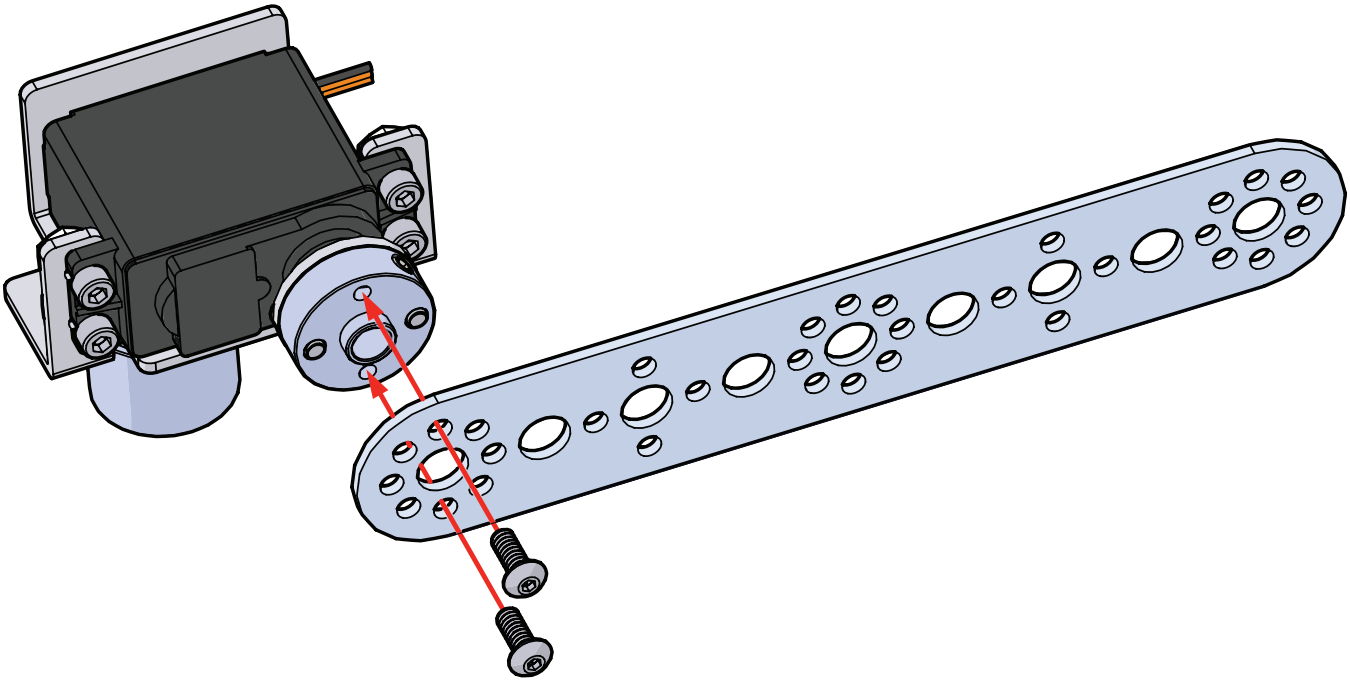
Step 1.3



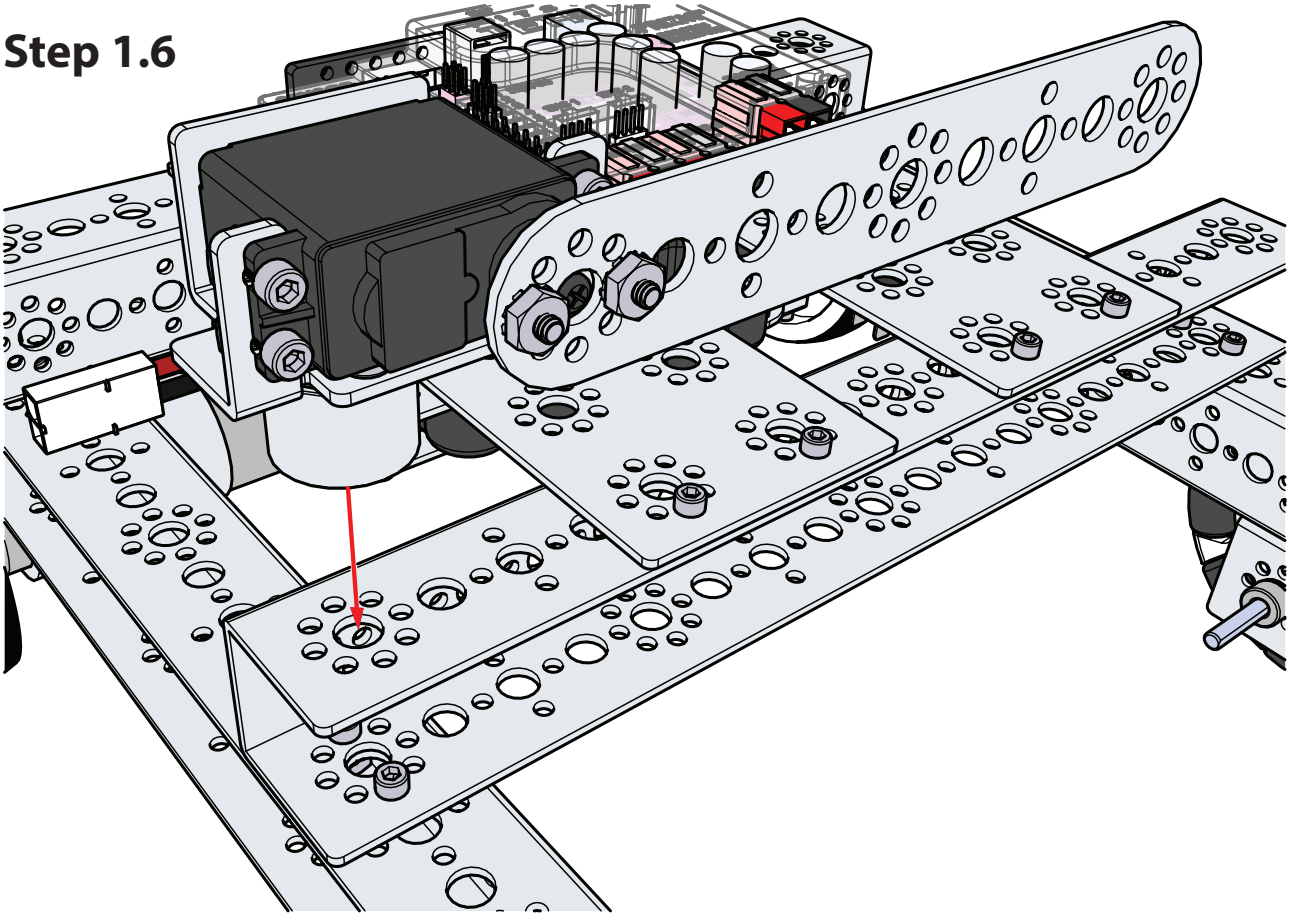
Step 1.4



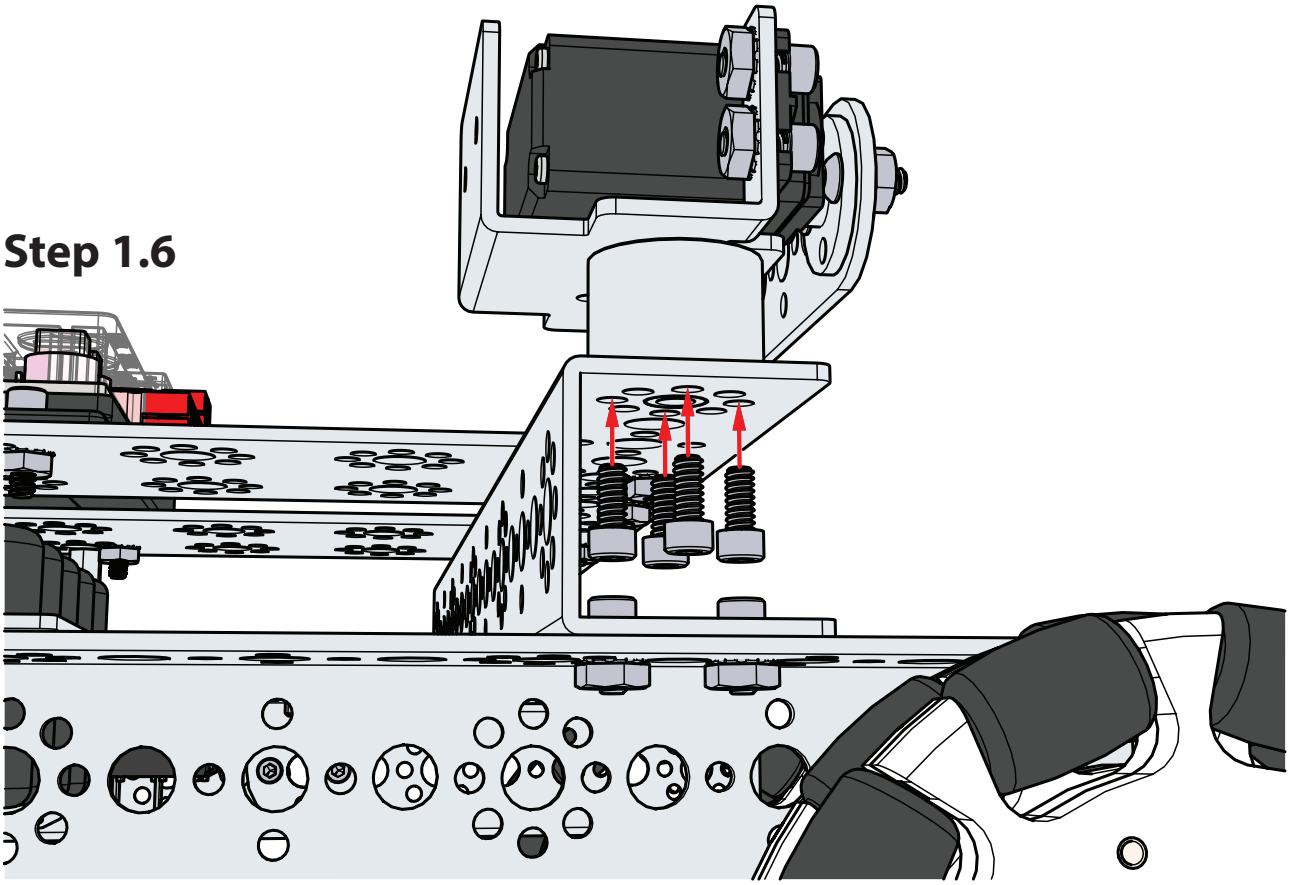
Step 1.5



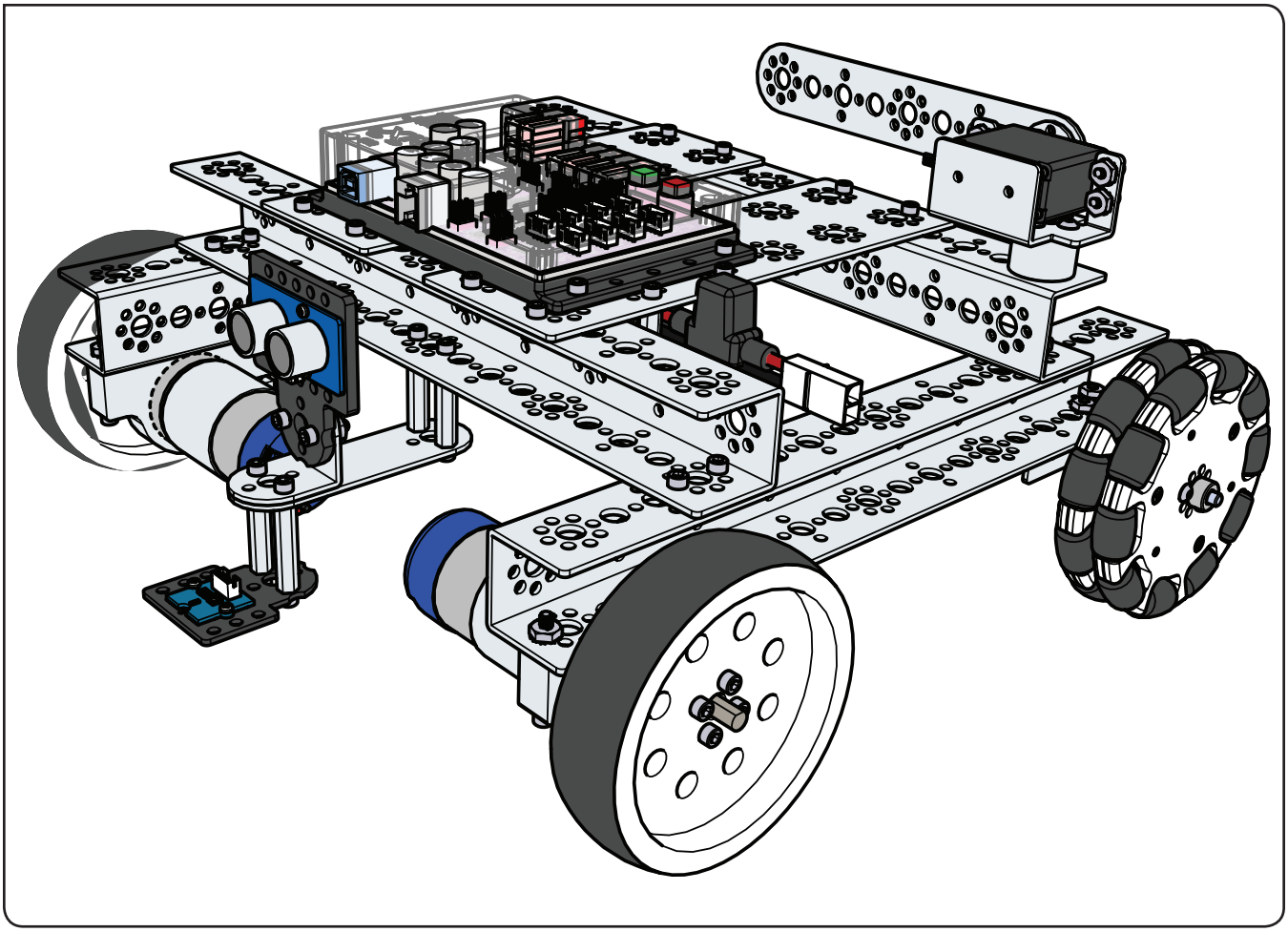
Step 1.6



Step 1.6



Finished assembly should look like this.

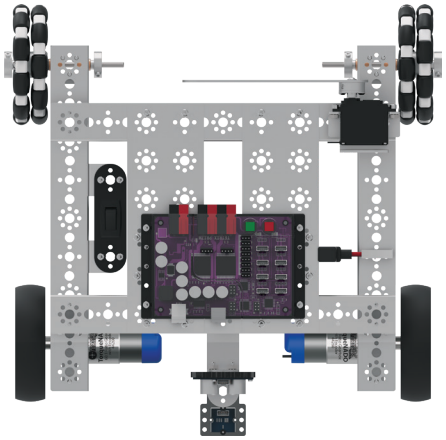


Activity 15: Combining the Sensors

This is a culmination of all the previous activities. The TaskBot will follow a line while watching for obstacles. If an obstacle is detected, then the TaskBot will stop, raise its arm, and wait for the obstacle to be moved before continuing.

Parts Needed

- Contrasting light and dark surface
- Obstacle
- Fully assembled PRIZM TaskBot complete with sensor module
- USB cable
- A charged TETRIX 12-Volt Rechargeable NiMH Battery Pack
- Computer



Opening the Sketch

Before we open our next example sketch, be sure to save any sketch you want to reference later.

Open the sketch by selecting **File > Examples > TETRIX_PRIZM > TaskBot_Act15_ Combining_Sensors**. A new sketch window will open titled TaskBot_Act15_ Combining_Sensors (Figure 28).

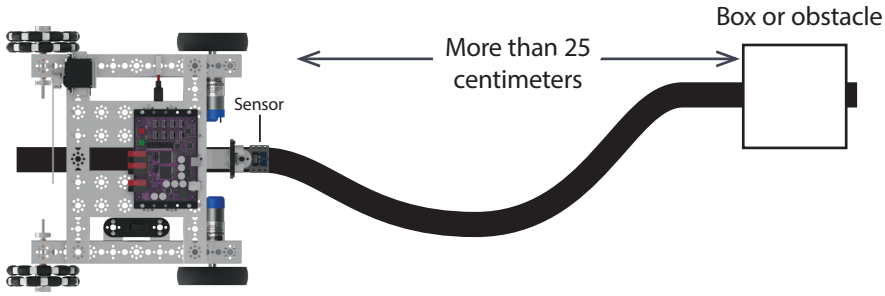
Figure 28

Executing the Code

Before we can upload the sketch to the PRIZM, remember to check our connections. The Line Finder Sensor will be in digital sensor port 3, and the Ultrasonic Sensor will be in digital sensor port 4.

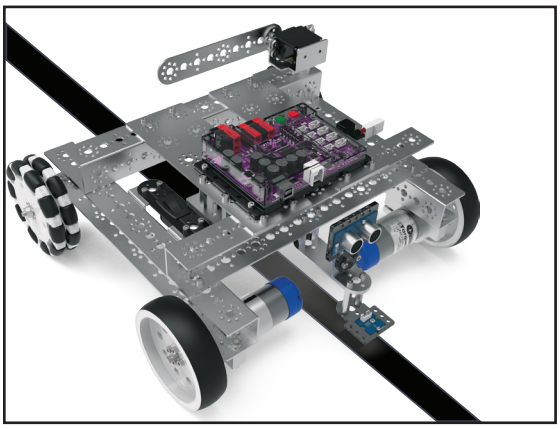
We will use the Line Finder Sensor for line following and the Ultrasonic Sensor for obstacle detection. This example sketch will have the robot follow the black line on a white surface and look for an object in its path.

When the object is detected at a distance of less than 25 centimeters, the robot will stop to avoid crashing into it. The robot will raise its arm and wait until the object is cleared and then continue.



Upload the sketch. The green LED will light up, indicating the code is ready to execute. When this has occurred, disconnect the USB cable and set the TaskBot on the floor.

The TaskBot should be on a white or reflective surface with the Line Finder Sensor slightly to the side of a line to follow.



When running this code, set an obstacle in front of our robot at a distance well outside the 25 centimeter detection range. Cardboard boxes work well for this purpose. We do not want anything too heavy that could cause damage to our robot just in case our robot crashes into it.

Press the green Start button to execute the sketch. Observe the behavior of the robot. You can also play around with moving your obstacles to see how your robot reacts.

Press the red Reset button to end the sketch. Based on the sketch comments, did the behavior match expectations?

Troubleshooting the Line Finder Sensor: Be sure to check that the Line Finder Sensor is plugged into the correct sensor port and that it is adjusted properly to sense the line. Some height adjustment might be needed, or the small adjustment screw on the back side of the sensor module might need to be tweaked. To see if the sensor is working properly, manually move the robot back and forth over the black line and white surface. The red LED on the Line Finder Sensor should be on when the sensor is over the white surface and off when it is over the black line.

Troubleshooting the Ultrasonic Sensor: Be sure to check that the Ultrasonic Sensor is plugged into the correct sensor port and plugged in correctly. For this sketch, the correct sensor port is D4. Keep in mind that objects without adequate surface size or that have an irregular surface might not be detected or affect the distance the sensor reads. To see if the sensor is working properly, you can always go back to Activity 5; load and run that sketch to check the serial monitor and verify the output from the sensor. If you need to do this, remember to either physically change the sensor port the Ultrasonic Sensor is plugged into to match the sketch example or change the sketch example.

Tip: If PRIZM does not execute the sketch after uploading, try cycling the robot power switch off and on.

Moving Forward

This sketch uses an "if/else" statement and a "while" loop, which we have covered before. This sketch is not about adding new elements. It is about combining the elements from the previous activities.

The "if/else" statement in this sketch does the same line-following behavior as in Activity 6 with two "if" statements but in a more controlled way. The "while" loop monitors the Ultrasonic Sensor and creates a behavior when an object is detected within 25 centimeters. This new behavior stops the TaskBot and uses the servo to raise the TaskBot's arm until the object is cleared from the TaskBot's path. When the object is cleared, the TaskBot will lower its arm and continue the original behavior of line following.

For more detailed information about the sketch process and the PRIZM library functions involved in using the DC motors, refer to **www.arduino.cc** and the appendix: TETRIX PRIZM Arduino Library Functions:

- Pages 134-135 about sensor port functions
- Pages 137-142 about DC motor functions
- Pages 144-145 about servo motor functions

Tip: Remember that you can print and use the TETRIX PRIZM Arduino Library Functions Cheat Sheet within the appendix on page 153 as a quick reference for all the functions available to you.

Tip: Want to see this in action? You can by watching our RoboBench video series for the *PRIZM Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

Real-World Connection

Electronic devices that are readily available these days (smartphones, health-monitoring devices, and so forth) collect sensory data that we can use to determine where we should go, what we should eat, how much we should exercise, when we should get up in the morning, and many more potential actions. Combining all these sensor inputs and determining what should be done happens within the coding of these devices. Coding for robotics has similar applications – it's all a matter of what you want your robot to do based on the sensors it has!

STEM Extensions

Science

- Human senses
- Human brain function based on sensory data

Technology

- Prioritizing sensor readings

Engineering

- Integration of sensory data to solve problems

Math

- Data distribution terms (mean, median, mode, range, cluster, outlier, skew)

Tip: In the sketch window the PRIZM library functions change color when they are typed or spelled correctly. Consequently, if spelled incorrectly they will not change. In Arduino, the PRIZM functions are recognized by the software as keywords and will turn orange when the syntax is correct.

Hacking the Code Activity

With the example as a reference, try creating a new sketch to have the TaskBot follow a black line and stop for obstacles. Remember what we learned from our previous activities.

Challenge yourself to create a sketch in which the TaskBot follows a line to an obstacle, goes around the object, and continues following the line instead of waiting for the object to be cleared.

Tip: An example library of code to help you get started with this challenge can be found in the appendix. If you have a digital copy of this guide, you can simply copy and paste the sample code for each activity into your sketch window. A digital download can be found at **Pitsco.com/TETRIX-PRIZM-Robotics-Controller#downloads**.

Build, Code, Test, Learn . . . Go!

You have built a robot, you have programmed PRIZM, and you have tested and tweaked your design, so where do you go from here?

This guide was intended to teach the essentials of building and coding a TETRIX MAX robot using the PRIZM controller and the *Arduino Software (IDE)*. However, this is only the beginning and there are plenty of additional resources available and opportunities to build larger robots that can complete more complex tasks. Now that you have learned the basics, you are limited only by your imagination. To help you get started, we have listed some of our favorite resources.

Please be sure to check out the appendix of this guide, which includes a vast array of technical documentation including detailed PRIZM technical specifications, component overviews and pinout diagrams, descriptions, function charts, and a cheat sheet for the PRIZM Arduino library functions. In the appendix you will also find support for integrating additional control components such as motor encoders.

For more on the *Arduino Software (IDE)* and programming tips and tricks:

PRIZM is built on Arduino architecture and is thus usable with programming languages that support Arduino. This enables PRIZM users to access the Arduino online community, which has produced a sea of support materials to get beginners coding quickly. Learn more at **www.arduino.cc**.

For more on sensor integration and support:

The PRIZM controller is designed with sensor ports that are compatible with the Grove system of modular sensors. Currently, the PRIZM Arduino library has integrated support for two of these sensors: the Line Finder and the Ultrasonic. For users who would like to integrate additional Grove sensors into their robotic builds, you can find Arduino example code for each at **wiki.seeedstudio.com/wiki/Category:Grove** or **wiki.seeedstudio.com/wiki/Grove_System**.

To view the sample Arduino code, simply click the sensor that is of interest.

Please note: TETRIX-compatible mounting adapters for the larger family of Grove sensors are sold separately and will be needed for securing sensors to the robot structure. Integrated support for additional Grove sensors will be added to the PRIZM Arduino library as further updates are released.

For more on the TETRIX building system:

The parts and pieces used in this guide are only a portion of the elements available in the TETRIX MAX system. Visit **Pitsco.com** to learn more about the additional structural, motion, and accessory components that can be added to your collection; to view support videos; and to download resources.

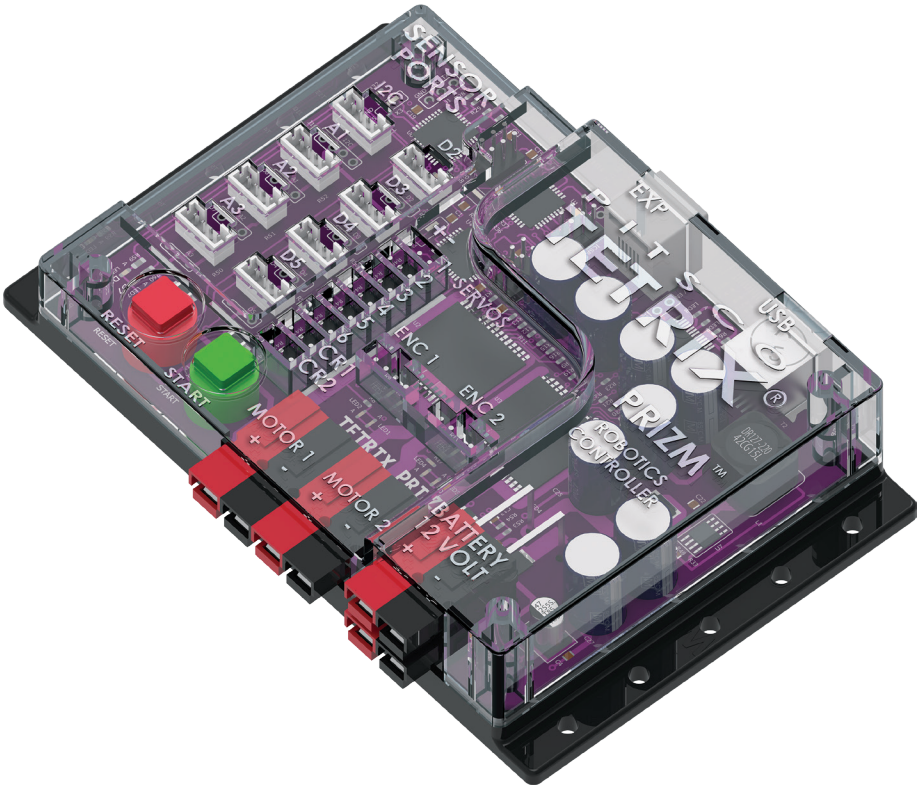
TETRIX PRIZM Robotics Controller Technical Specifications

Microcontroller:	ATmega328P with Arduino Optiboot bootloader installed
Memory:	32 KB flash programmable memory (ATmega328P)
Power:	9-18 volts DC
DC motor ports:	2 Powerpole connections; H-bridge PWM controlled; 10 amps continuous current each channel, 20-amp peak
Recommended motor:	12-Volt DC TETRIX MAX TorqueNADO Motor (44260)
DC motor control modes:	Constant power (-100% to 100%) PID constant speed (-720 to 720 degrees per second) PID constant speed to encoder target position and hold PID constant speed to encoder degrees position and hold Brake or Coast mode when stopping DC motor current monitoring (all modes)
Motor encoder ports:	2 quadrature, 5 volts DC, 50 mA max; Spec: 360 CPR, 1,440 PPR; ENC 1 and ENC 2
USB connector:	USB Type B
USB driver:	FTDI
Standard servo ports:	6 total: servo channels 1-6
Continuous rotation (CR) servo ports:	2 total: CR1 and CR2 channels
Total servo power limit:	6 volts DC, 6 amps max
Servo control modes:	Set servo speed (0% to 100%) Set servo position (0-180 degrees) Set CR servo state (Spin CW/Spin CCW)
Battery voltage monitoring:	0-18 volts range
Digital sensor ports (D2-D5):	Each can be configured as digital input or output. D2 can be configured as a serial communications port.
3 analog sensor ports (A1-A3):	Each can be configured as analog input or digital input/output ports.
1 I2C port (I2C):	100 Khz speed. This connection shares same I2C bus as internal DC motor and servo motor control chips. I2C addresses 0x01-0x06 reserved by the PRIZM controller.
Motor controller expansion port (EXP):	Additional TETRIX DC motor and servo control modules can be daisy-chained to this port.
Battery connection port:	Powerpole type; additional port for daisy-chaining battery power to motor controllers added to the expansion port
1 green Start button (START):	Programmable push button
1 red Stop/Reset button (RESET):	Non-programmable push button
1 red LED:	Programmable LED used as an indicator
1 green LED:	Programmable LED used as an indicator
1 blue LED:	Indicates the power is on when illuminated
2 yellow LEDs:	Indicates serial data activity on the USB port
1 red and 1 green DC motor LEDs:	Indicates DC motor rotation and direction for each DC motor channel

PRIZM Robotics Controller Functional Overview

The PRIZM controller is connected to a computer using a standard USB cable connection. Power is supplied from an external TETRIX 12-Volt Rechargeable NiMH Battery Pack. The controller has a dual high-current DC motor drive system, each motor having quadrature encoder support for implementing precise PID DC motor velocity and position control. In addition there are six standard control servo ports and two continuous rotation servo ports. There are four digital sensor ports, three analog sensor ports, and one I2C port for use with external sensors. Digital port D2 can also be configured with Arduino's software serial library for serial data communications. The digital sensor ports can be configured as input or output. The analog sensor ports can be configured for analog input or digital output mode. Also onboard are two LEDs – one Red and one Green – that can be used as visual indicators.

The PRIZM controller functions are driven by an ATmega328P chip using the Optiboot bootloader for uploading program code via a USB connection to the processor chip. There are two additional processor chips used to control motor functions: a DC motor control and encoder interface chip and a servo control chip. These chips communicate with the main processor chip via I2C communication. Each motor control chip contains firmware, which handles all of the complex DC motor, encoder, and servo control functions, thereby freeing up the main processor for running program code. The green Start button is used to begin running a program that has been uploaded to the controller. The red Reset button is used to stop a running program. Pressing the red Reset button resets the main processor chip and the DC and servo control chips and initializes all stored memory values to 0.



PRIZM Controller Component Overview and Pinout Diagrams

PRIZM Sensor Ports

The PRIZM controller uses Arduino UNO-compatible pin assignments. The sensors that are supported in the Arduino PRIZM coding library are set up automatically using the library functions. Support for different types of sensors will be added as they become available. However, the ports are all directly accessible using Arduino coding functions if you wish to work with them. With the exception of the I2C port, all others can be configured as inputs or outputs using the Arduino pinMode function. To learn more, visit the Language Reference section at **www.arduino.cc**.

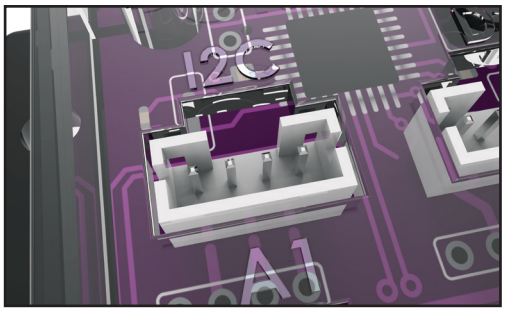


Figure 29: PRIZM Sensor Port (Pins are left to right: 1, 2, 3, 4.)

Table 1: I2C port pin assignments

Pin	Function	*Arduino Software (IDE)* pin assignment ( )
Pin 1	Ground	N/A
Pin 2	+5 volts, 100 mA	N/A
Pin 3	SDA (I2C serial data)	ADC4 input channel (A4)
Pin 4	SCL (I2C serial clock)	ADC5 input channel (A5)

Note: The PRIZM I2C port can be used only in I2C mode. It may not be configured for analog or digital mode.

Table 2: Analog Sensor Port (A1)

Pin	Function	*Arduino Software (IDE)* pin assignment ( )
Pin 1	Ground	N/A
Pin 2	+5 volts, 100 mA	N/A
Pin 3	No connect	N/A
Pin 4	Analog input or digital input/output	Analog input (A1) | digital I/O (15)

Table 3: Analog Sensor Port (A2)

Pin	Function	*Arduino Software (IDE)* pin assignment ( )
Pin 1	Ground	N/A
Pin 2	+5 volts, 100 mA	N/A
Pin 3	No connect	N/A
Pin 4	Analog input or digital input/output	Analog input (A2) | digital I/O (16)

Table 4 Analog Sensor Port (A3)

Pin	Function	*Arduino Software (IDE)* pin assignment ( )
Pin 1	Ground	N/A
Pin 2	+5 volts, 100 mA	N/A
Pin 3	No connect	N/A
Pin 4	Analog input or digital input/output	Analog input (A3) | digital I/O (17)

Note: The PRIZM I2C port can be used only in I2C mode. It may not be configured for analog or digital mode.

Table 5: Digital Sensor Port (D2)

Pin	Function	*Arduino Software (IDE)* pin assignment ( )
Pin 1	Ground	N/A
Pin 2	+5 volts, 100 mA	N/A
Pin 3	Digital input/output	Digital I/O (9)
Pin 4	Digital input/output	Digital I/O (2)

Note: Digital Port D2 can also be configured as a software-implemented serial port using the *Arduino Software (IDE) Serial Library*. Pins D9 and D2 can be set to RX/TX for serial port communications.

Table 6: Digital Sensor Port (D3)

Pin	Function	*Arduino Software (IDE)* pin assignment ( )
Pin 1	Ground	N/A
Pin 2	+5 volts, 100 mA	N/A
Pin 3	No connect	N/A
Pin 4	Digital input/output	Digital I/O (3)

Table 7: Digital Sensor Port (D4)

Pin	Function	*Arduino Software (IDE)* pin assignment ( )
Pin 1	Ground	N/A
Pin 2	+5 volts, 100 mA	N/A
Pin 3	No connect	N/A
Pin 4	Digital input/output	Digital I/O (4)

Table 8: Digital Sensor Port (D5)

Pin	Function	*Arduino Software (IDE)* pin assignment ( )
Pin 1	Ground	N/A
Pin 2	+5 volts, 100 mA	N/A
Pin 3	No connect	N/A
Pin 4	Digital input/output	Digital I/O (5)

PRIZM Servo Ports

The PRIZM controller has six position-controlled servo ports labeled 1 through 6. Each servo channel can power and control one standard hobby-type servo motor. The TETRIX PRIZM library for the *Arduino Software (IDE)* does all of the complex work of controlling the servo motors.

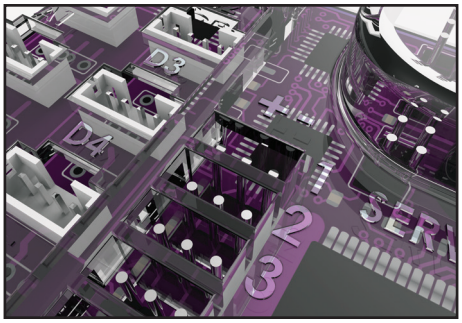


Figure 30: PRIZM Servo Motor Ports (Pins are left to right: 1, 2, 3.) (Labeled as SERVOS 1-6)

Pin 1: Servo PWM signal. It is usually the yellow or white wire.

Pin 2: Servo power supply wire. It is the red wire. PRIZM supplies +6 volt power.

Pin 3: Servo ground wire. It is the black wire.

Reversing the servo connection polarity will not damage the PRIZM controller. However, if not inserted properly, the servo motor will not function. The PRIZM controller can supply a total of 6 amps of DC power at 6 volts DC to the servo ports. The amount of current draw is limited by a thermal (self-resetting fuse).

PRIZM Continuous Rotation (CR) Servo Ports

The PRIZM controller has two additional specialized servo ports for connection of two continuous rotation servo motors. Continuous rotation servo motors are designed to rotate continuously either clockwise or counterclockwise and can be used as smaller, lightweight DC gearbox motors. The CR1 and CR2 ports are designed to be connected to these types of servo motors and can spin them continuously in either direction controlled by the commands in the PRIZM library.

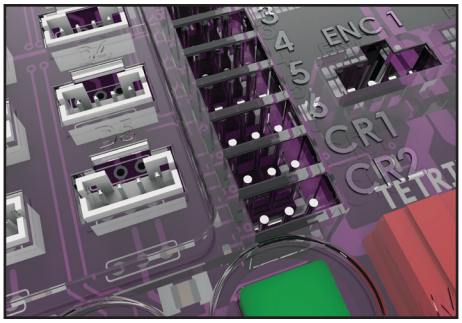


Figure 31: PRIZM Continuous Rotation (CR) Servo Motor Ports (Pins are left to right: 1, 2, 3.) (Labeled as CR1 and CR2)

Pin 1: Servo PWM signal. It is usually the yellow or white wire.

Pin 2: Servo power supply wire. It is the red wire. PRIZM supplies +6 volt power.

Pin 3: Servo ground wire. It is the black wire.

Reversing the servo connection polarity will not damage the PRIZM controller. However, if not inserted properly, the servo motor will not function. The PRIZM controller can supply a total of 6 amps of DC power at 6 volts DC to all the servo ports – this includes both CR1 and CR2. The amount of current draw is limited by a thermal (self-resetting fuse).

Note: We recommend the following servo motors, which are available at **Pitsco.com**: the TETRIX MAX Standard-Scale Servo Motor (39197), the Quarter-Scale HS-785HB Winch Servo Motor with Horn (39905), and the Quarter-Scale HS-755HB Servo Motor with Horn (39904).

PRIZM Encoder Ports

The PRIZM controller has two quadrature encoder inputs for precise control of the DC Motor 1 and Motor 2 channels. When utilized with DC motors, encoders provide position and velocity feedback that is used in programming code to implement powerful and precise DC motor speed and position control. The ENC1 port provides encoder feedback for DC Motor Channel 1, while ENC2 provides encoder feedback for DC Motor Channel 2. We have included code examples using encoders and all PRIZM Arduino library functions associated with encoders. These code examples can be found in the File > Examples > TETRIX_PRIZM drop-down menu.

Note: Encoders are included as part of the TETRIX MAX TorqueNADO Motors. To utilize the TorqueNADO motor encoder, plug the white end of the encoder cable into the TorqueNADO motor. Plug the black end of the encoder cable into the PRIZM ENC1 or ENC2 port.

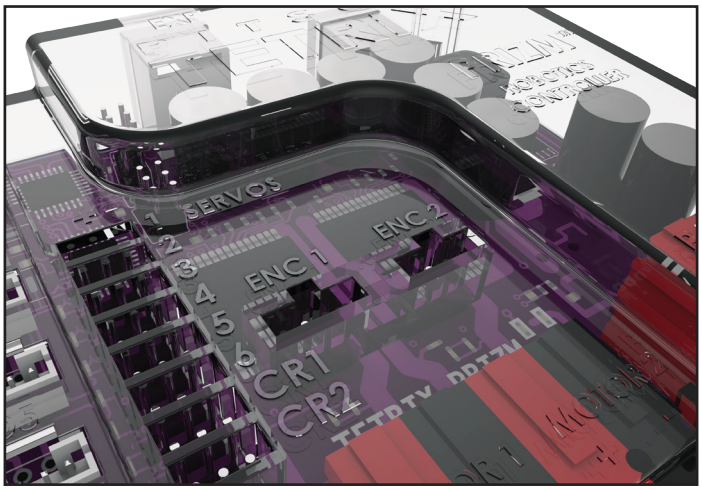


Figure 32: PRIZM Quadrature Encoder Port
(Pins are left to right: 1, 2, 3, 4.) (Labeled as ENC1 and ENC2)

Pin 1: +5 volts DC power supply

Pin 2: Encoder count signal (A)

Pin 3: Encoder ground

Pin 4: Encoder count signal (B)

Although the connector socket is "keyed" to prevent reverse polarity, take care to not force the connector into the socket in the opposite direction. Damage to the encoder or PRIZM controller could occur if the connector is forcibly inserted in the reverse polarity direction.

PRIZM Expansion Port

The PRIZM controller has a RJ-45 modular jack, labeled EXP, connected to the onboard I2C bus. This port can be used to connect additional TETRIX DC Motor Controllers and servo controllers in a daisy-chain arrangement. Up to four additional DC and/or servo motor controller boxes of any combination can be connected to the expansion port for added motor control channels. Each additional motor control box will dynamically set its own I2C address depending on its position in the daisy chain. The first box will use address "0x01", the second, "0x02", the third, "0x03", and the fourth "0x04".

When using additional I2C sensors or devices plugged into either the expansion port or the I2C sensor port, these devices may not use I2C addresses 0x01 through 0x06. These address locations are reserved by the PRIZM internal motor and servo control processor chips and any expansion motor or servo controller daisy-chained to PRIZM through the expansion port.

Note: This port is not compatible with LEGO® MINDSTORMS®-related devices. Do not plug in LEGO MINDSTORMS sensors, motors, the EV3 Brick, or any other LEGO device to this port.

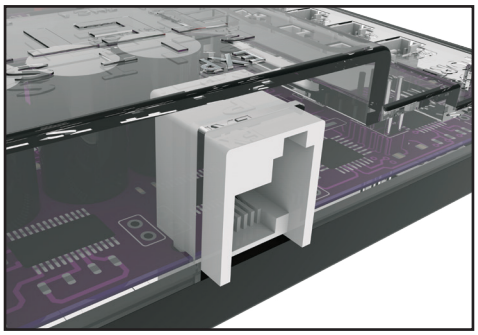


Figure 33: PRIZM Motor Controller Expansion Port
Offset tab RJ-45 modular jack

PRIZM USB Port

The PRIZM USB port is used for communication between PRIZM and a Windows, Mac, or Linux computer. Its main use is for downloading program code to the program memory processor. It can also be used to transfer data between PRIZM and a computer using serial communications protocol. For example, the *Arduino Software (IDE)* includes a built-in serial monitor display window used to display data values received from the PRIZM controller. This feature can be used to display sensor or encoder data or even other types of program data as needed.

Oftentimes, viewing data values can be very helpful when debugging program code. The serial monitor can also be used to send information to the PRIZM controller. An example of this would be to have the PRIZM controller moving DC motors or servos in response to keyboard entries. Users with advanced programming knowledge could create graphical interfaces using Java, Python, or another coding platform for monitoring data or controlling functions.

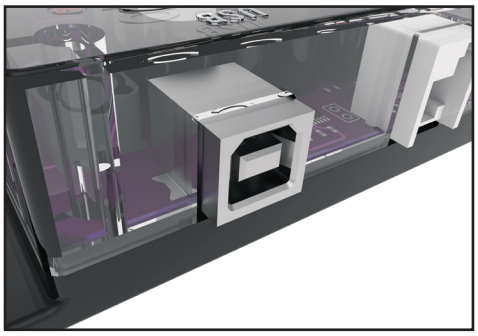


Figure 34: PRIZM USB Programming and Communication Port

PRIZM Reset Button

The red Reset button is used for two purposes. When pressed, it will terminate any program code that is being executed and reset all sensor and encoder data values to their initialized states. It effectively resets the entire system just as switching power off and back on would do.

Figure 35: PRIZM Reset button. Press down to activate.

PRIZM Start Button

The green Start button is used to begin the execution of program code that is downloaded to the PRIZM controller. When you are creating program code using the *Arduino Software (IDE)*, the PrizmBegin function executes all the instructions necessary to begin running a program. Calling this function initializes necessary parameters to their proper state and causes the program code to wait until the green Start button is pressed before execution.

Figure 36: PRIZM Start button. Press down to activate.

PRIZM DC Motor Ports

The PRIZM controller has two DC motor connection ports labeled Motor 1 and Motor 2. Each motor channel is used to control the speed and direction of DC motors via software commands defined in the PRIZM *Arduino Software (IDE)* Library. Each motor channel can provide 10 amps of continuous DC current at 12 volts. Each motor channel has a red and black connector, with red being the positive connection and black the negative connection. The TorqueNADO motor power cables come with the mating colored connectors and should be plugged red to red and black to black for proper operation.

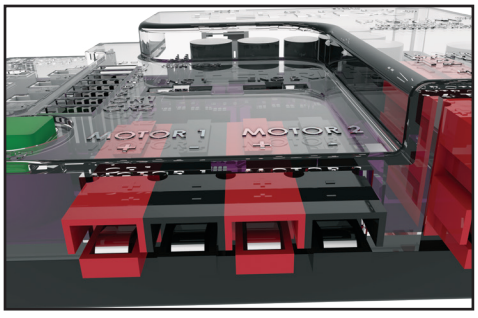


Figure 37: PRIZM DC Motor Ports

PRIZM Battery Connection Port

The PRIZM controller is powered by a TETRIX 12-Volt Rechargeable NiMH Battery Pack. Included with the PRIZM controller is a power on/off switch assembly designed to connect the battery pack connector to the PRIZM-style battery connection port. The battery pack can be plugged into either the top or bottom row of the port. The extra port is intended to be used to daisy-chain power to additional motor controllers that can be optionally added to PRIZM via the expansion port.

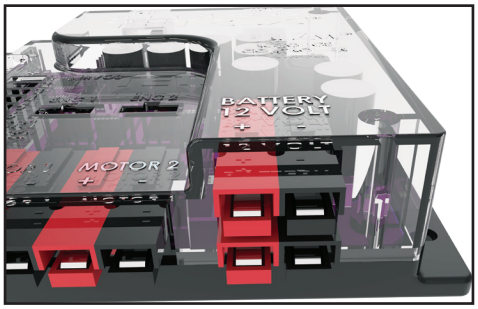


Figure 38: PRIZM Battery Power Inlet/Outlet Port

TETRIX PRIZM Arduino Library Functions

Include Statements

The include statement is used to import the functionality of the TETRIX PRIZM Arduino Library into an Arduino sketch. The PRIZM library is a collection of functions that simplifies programming the PRIZM controller. Think of a library as a container filled with mini programs, each with its own distinct function name that we can use in code to carry out complex tasks. To include a library in our Arduino sketch, use the #include statement at the beginning of our program code. For example, to add the PRIZM library to an Arduino sketch, we would insert this statement at the top of our program.

#include <PRIZM.h>

This statement tells the Arduino compiler to include all the program code contained in the PRIZM library when we compile our Arduino sketch.

Object Declarations

The object declaration is an important statement when using the PRIZM controller and the PRIZM library. In order to use the functions contained in the PRIZM library, we must first declare a library object name, which is then inserted as a prefix before each library function. We can do this by including the following statement just after the #include statement.

PRIZM *myname;*

The parameter *myname* is arbitrary – it can be anything that we choose or that makes sense to us depending on the code application that we are creating. For consistency in this programming guide, the parameter *myname* will be set to "prizm", meaning that the object declaration statement used in sketch code examples will look like this:

PRIZM prizm;

The statement above creates a library object named "prizm" used in code when calling a function from the PRIZM class in which the object was created. In our code, whenever we want to call a PRIZM library function, we will need to prefix each function with the defined object name. In the case of the statement above, that object would be "prizm". All the library functions below are prefixed by the parameter *myname*. Do not forget, when using these functions in our code, to change the parameter *myname* to whatever we have named our PRIZM library object.

Also, as we mentioned, the code examples in this guide use the object name "prizm". Also, for clarification, any statement preceded by "//" means that it is a comment or explanation only and is not included as program code when our sketch is compiled. It's considered good practice to be able to comment on and explain each step of code to make sure you can understand it.

Initialization Functions

The TETRIX PRIZM controller initialization functions are used to begin or, when needed, end a program.

***myname*.PrizmBegin();**

This is always called in the setup() section of an Arduino sketch. This function initializes critical parameters and invokes the use of the Start button on the PRIZM controller. It must always be used when creating sketches for the PRIZM controller.

Example:

```
prizm.PrizmBegin();                    // initialize the PRIZM controller
```

***myname*.PrizmEnd();**

This function ends or terminates a program, resets critical parameters, and sends the program back to start.

Example:

```
prizm.PrizmEnd();                      // when called, will terminate the program execution
```

The TETRIX PRIZM sensor ports are used for connecting various types of sensors for digital and analog communication. The digital sensors can be connected to PRIZM using ports D2-D5. Digital port D2 can also be configured as a software-implemented serial communications port using the Arduino's software serial library. All digital ports can be configured as either input or output. Analog sensors are connected to PRIZM using ports A1-A3. The analog sensor ports can be configured for analog input or digital outputs. There is also one I2C sensor port. The I2C port is connected to the PRIZM I2C communication bus. I2C addresses 0x01-0x06 are reserved internally by the PRIZM controller and may not be used for an external sensor device. Support for sensors will be added to PRIZM continually. Please check on support and availability of sensors at **Pitsco.com**.

***myname*.readLineSensor(port#);**

This function reads the state of the Line Finder Sensor and returns a value of "1" or "0." The parameter *port#* specifies which digital port the sensor is plugged into. The line sensor's most common application is to sense the difference between black and white surfaces. This is a very common way to program a robot to follow the edge of a dark line on a white surface or the edge of a white line on a dark surface. It accomplishes this by detecting a beam of reflected infrared light from the surface that it is facing. If the surface it is facing is a light color or is reflective, the light beam will bounce from it and be detected by the sensor. If the surface is dark, or if the sensor is too far away from the facing surface, a reflected light beam will not be detected. The Line Finder Sensor has a small adjustment potentiometer screw on the back side to adjust beam sensitivity. If the sensor detects a non-reflective surface (it does not receive the reflected beam; there is a dark area or line), the function will return "1", meaning "line detected." If the sensor is receiving the reflected beam (there is a white or reflective surface), the function will return "0", meaning "no line detected." See the example below or the PRIZM library for a full line follower example.

Example:

```
int x;                                 // set up an integer variable named x
x = prizm.readLineSensor(2);           // read the state of line sensor connected to digital
                                       // port (D2) into the variable x
```

or

```
Serial.print(prizm.readLineSensor(2)); // read the state of line sensor connected to digital
                                       // port (D2) and print it to the
                                       // Arduino serial monitor
```

***myname*.readSonicSensorCM(port#);**

This function reads and returns the distance of an object that is in front of the Ultrasonic Sensor. The parameter *port#* specifies which digital sensor port the sensor is plugged into. The Ultrasonic Sensor is a non-contact distance measurement module that can be used to give our robot vision so that it can know when objects are in its path and avoid them. The sensor's frequency is modulated at 42 kHz and has a measuring range of 3-400 centimeters. The sensor works by sending a sonic pulse burst and then waiting on its return to the sensor as it is reflected off an object in range. The reflected sonic pulse time period is measured to determine the distance to the object.

Example:

```
int x;                                  // set up an integer variable x
x = prizm.readSonicSensorCM(2);         // read and store in variable x the distance in
                                        // centimeters of an object that is detected to be
                                        // in front of the Ultrasonic Sensor connected to
                                        // digital port (D2)
```

or

```
Serial.print(prizm.readSonicSensorCM(2)); // read the distance in centimeters of an
                                          // object that is detected in front of the
                                          // Ultrasonic Sensor connected to digital
                                          // port (D2) and print it to the Arduino serial
                                          // monitor
```

***myname*.readSonicSensorIN(port#);**

This function reads and returns the distance of an object in inches that is in front of the Ultrasonic Sensor. The parameter *port#* specifies which digital sensor port the sensor is plugged into. The Ultrasonic Sensor is a non-contact distance measurement module that can be used to give our robot vision so that it can know when objects are in its path and avoid them. The sensor's frequency is modulated at 42 kHz and has a measuring range of 2-150 inches. The sensor works by sending a sonic pulse burst and then waiting on its return to the sensor as it is reflected off an object in range. The reflected sonic pulse time period is measured to determine the distance to the object.

Example:

```
int x;                                  // set up an integer variable x
x = prizm.readSonicSensorIN(port#);     // read and store in variable x the distance
                                        // in inches of an object that is detected to
                                        // be in front of the Ultrasonic Sensor
                                        // connected to digital sensor port (D2)
```

or

```
Serial.print(prizm.readSonicSensorIN(2)); // read the distance in inches of an
                                          // object that is detected in front of
                                          // the Ultrasonic Sensor connected
                                          // to digital port (D2) and print it to
                                          // the Arduino serial monitor
```

Battery Voltage

The TETRIX battery pack is rated at 12 volts DC at 3,000 mAh. When the battery pack is fully charged, the peak voltage could be as high as 15 volts. The battery pack is considered discharged when the voltage drops below 12.1 volts. This pack is also equipped with a 20-amp safety fuse that is replaceable. All TETRIX-powered devices must use the genuine TETRIX safety fuse-equipped battery pack for ensured safety. The battery voltage can be monitored in program code by calling the read battery voltage function.

***myname*.readBatteryVoltage();**

This function reads and returns the voltage of the TETRIX battery pack plugged into the PRIZM battery connection port. The value returned is an integer and can be divided by 100 to scale to actual battery voltage. For example, a value of 918 is equal to an actual battery voltage of 9.18 volts.

Example:

```
int x;                                  // set up an integer value x
x = prizm.readBatteryVoltage();         // read the battery voltage into variable x
```

or

```
Serial.print(prizm.readBatteryVoltage());   // read the battery pack voltage and print it
                                            // to the Arduino serial port monitor
```

Using the Start Button

***myname*.readStartButton();**

The green Start button can be used for purposes other than starting the execution of program code. After it is used to start a program, it can be read again at any point to execute other tasks, giving us more control over our program. One example would be to have two different robot behaviors sectioned into two parts of code and use the Start button to begin the first section, and then when pressed again, to jump to the second code section. The function will return "1" when the button is pressed and a "0" when the button is pressed again.

Example:

```
int x;                                  // set up a variable named x
x = prizm.readStartButton();            // read the state of the start button into variable x
```

or

```
while(prizm.readStartButton() == 0){    // wait here in a while loop until the start
                                        // button is pressed (returns a "1")
}
```

Onboard LEDs

The PRIZM controller has two onboard LEDs that can be controlled in code and used as indicators. The red and green LEDs are switched on and off by setting each HIGH or LOW.

***myname*.setGreenLED(HIGH);** // turns on the onboard green LED indicator

***myname*.setGreenLED(LOW);** // turns off the onboard green LED indicator

Example:

prizm.setGreenLED(HIGH); // turns the PRIZM green LED on

prizm.setGreenLED(LOW); // turns the PRIZM green LED off

***myname*.setRedLED(HIGH);** // turns on the onboard red LED indicator

***myname*.setRedLED(LOW);** // turns off the onboard red LED indicator

Example:

prizm.setRedLED(HIGH); // turns the PRIZM red LED on

prizm.setRedLED(LOW); // turns the PRIZM red LED off

DC Motor Functions

The TETRIX PRIZM DC motor functions are used to control DC motors connected to Motor 1 and Motor 2 output ports. There are three different types of control. Power control is used to set the amount of power and also set the direction. The speed and targeting functions require a quadrature encoder to be connected to the PRIZM as a feedback sensor to precisely control the speed rate and targeting position using a PID algorithm.

***myname*.setMotorPower(motor#, power);**

This function is used to control the power level and direction of a DC motor connected to the Motor 1 or Motor 2 channel ports. The parameter *motor#* selects the motor channel and the parameter *power* sets the power level. *Motor#* can be a number or variable of 1 or 2. *Power* is the amount of power to spin the motor, which can range between 0 and 100. This is a percentage of power, meaning "0" is fully off, and "100" is 100% power or fully on. The direction is set by the sign of the power parameter.

Example:

prizm.setMotorPower(1, 100); // spin Motor 1 clockwise at 100% power

prizm.setMotorPower(1, -100); // spin Motor 1 counterclockwise at 100% power

Also, when you are stopping a motor, there are two modes: coast stop and brake stop. Setting power to 0 will result in the motor coasting to a stop. Setting power to 125 results in the motor stopping with a braking action.

Example:

prizm.setMotorPower(2, 0); // stop Motor 2 with a coast to stop; 0 = coast

or

prizm.setMotorPower(2, 125); // stop Motor 2 with a brake stop; 125 = brake

***myname*.setMotorPowers(power1, power2);**

This function enables us to set the power level of Motor 1 and Motor 2 at the same time. Set parameters *power1* and *power2* to a power level percentage and both channels will respond at once. Remember, the sign (+/-) of the power level determines the direction of rotation.

Example:

```
prizm.setMotorPowers(50, -50);          // spin Motor 1 CW at 50% power and Motor 2
                                        // CCW at 50% power
```

***myname*.setMotorSpeed(motor#, speed);**

This function can precisely control the speed of a DC motor using a TETRIX quadrature encoder with PID control. The encoder cable must be plugged into the PRIZM encoder socket "ENC1" or "ENC2" that matches the motor being controlled in order for this function to work. The encoder sends counting data back to the PRIZM controller, where it is read and used to implement a complex algorithm called PID, short for *proportional integral derivative*. The PID algorithm is very powerful and will precisely regulate and attempt to spin the motor at a constant rate even when the load on the motor is changing. Using PID can enable precise control and positioning movement of the wheels and mechanisms being driven by the DC motors. The parameter *motor#* can be either "1" or "2". The parameter *speed* is the rate in degrees per second (DPS) that we wish to set our motor to spin. Values can range from a minimum of about 10 to a maximum of 720 DPS. The direction of the motor is set by the sign of the speed parameter. Values outside that range might not perform well.

Example:

```
prizm.setMotorSpeed(1, 360);            // spin Motor 1 clockwise at a rate of 360 DPS
```

or

```
prizm.setMotorSpeed(1, -360);           // spin Motor 1 counterclockwise at a rate of 360
                                        // DPS
```

***myname*.setMotorSpeeds(speed1, speed2);**

This function implements PID control and is identical to the previous function just explained with the difference being that it will set the *speed1* and *speed2* parameters of Motor 1 and Motor 2 channels at the same time. Both motors being controlled must have encoders installed and plugged into their matching encoder channel sockets ENC1 and ENC2 on the PRIZM controller.

Example:

```
prizm.setMotorSpeeds(360, -180);        // spin Motor 1 clockwise at 360 DPS and spin
                                        // Motor 2 counterclockwise at 180 DPS
```

***myname*.setMotorTarget(motor#, speed, target);**

In addition to speed control, this function adds a characteristic of control by using a PID algorithm to implement both velocity and position target PID control. The motor being controlled as set by parameter *motor#* must have its corresponding encoder plugged into the motor controller's encoder port. This function will spin the commanded DC motor to make the current encoder value become equal to a target value. The motor will rotate toward the commanded target at a rotation rate set in the *speed* parameter (degrees per second). When the motor has reached the encoder count location set in the *target* parameter, it will hold its position in a servo-like mode until a new command is received. This function ignores the positive/negative sign of the *speed* parameter. Instead, the direction of rotation is set by the commanded encoder target value of the *target* parameter. While this function is executing, the readMotorBusy() function can be read to indicate that the motor is busy moving to its target. When this function returns "HIGH", or "1", the command motor channel is busy (moving to the target). When the target has been reached, the readMotorBusy() function will return "LOW" or "0". The encoder resolution is 1,440 counts per revolution of the motor shaft, or 1/4 degree per count.

Example:

```
prizm.setMotorTarget(1, 360, 1440);      // spin Motor 1 clockwise at 360 DPS; stop
                                         // and hold position when the encoder
                                         // count equals 1440 (one revolution)

prizm.setMotorTarget(2, 180, -1440);     // spin Motor 1 counterclockwise at 180
                                         // DPS; stop and hold position when the
                                         // encoder count equals -1440
```

***myname*.setMotorTargets(speed1, target1, speed2, target2);**

This function implements velocity and position target PID control for both DC motor channels. This function sets the mode parameters of both Motor 1 and Motor 2 at the same time for an even start. Having our motors start as closely in sync as possible can be critical in some applications. The speed and stopping target of Motor 1 are set by parameters *speed1* and *target1*. Likewise, the speed and stopping target of Motor 2 are set by parameters *speed2* and *target2*. This function implements PID control for both DC motor channels. Encoders on both motors must be plugged into their matching encoder connection ports ENC1 and ENC2 on the PRIZM controller. This function will spin both DC motors to make their current encoder values become equal to their target values. The motors will rotate toward the commanded targets at rotation rates set in *speed1* and *speed2* (degrees per second). When each motor has reached its target set in *target1* and *target2*, it will hold its position in a servo-like mode until a new command is received. This function ignores the positive/negative sign value of speed1 and speed2. The direction of rotation is set by the commanded encoder target value of *target1* and *target2*. While this function is executing, the readMotorBusy() function can be read to indicate that the motor channel is busy moving to its target. When this function returns "HIGH" or "1", the command motor channel is busy (moving to the target). When the target has been reached, the readMotorBusy() function will return "LOW" or "0". The encoder resolution is 1,440 counts per revolution of the motor shaft, or 1/4 degree per count.

Example:

```
prizm.setMotorTargets(360, 1440, 180, -1440);  // this command will spin Motor 1
                                               // clockwise at a rate of 360 DPS,
                                               // stop, and hold position at encoder
                                               // count 1440. Motor 2 will spin
                                               // counterclockwise at a rate of 180
                                               // DPS, stop, and hold position
                                               // at encoder count -1440.
```

***myname*.setMotorDegree(motor#, speed, degrees);**

This function requires the DC motor to have an encoder connected to PRIZM in order to implement velocity and position target control using PID. This function is identical to the set "target" functions explained previously, except the target values are defined in degrees instead of encoder counts. Working in degrees might be a bit more intuitive than actual encoder counts. The parameter *motor#* sets the function to control either Motor 1 or Motor 2. The *speed* parameter is the rate in degrees per second for the commanded motor to maintain. The *degrees* parameter sets the position in degrees to which the motor will rotate, stop, and hold position. When called, this function will spin the commanded motor channel to make the current encoder degrees value to become equal to the degrees value. The motor will rotate toward the commanded target at a rotation rate set in *speed* (degrees per second). When the motor has reached the target, it will hold its position in a servo-like mode until a new command is received. This function ignores the positive/negative sign value of *speed*. The direction of rotation is set by the commanded encoder target value of *degrees*. While this function is executing, the readMotorBusy() function can be read to indicate that the motor is busy moving to its target. When this function returns "1", the command motor channel is busy (moving to the target). When the target has been reached, the readMotorBusy() function will return "0". The encoder degrees resolution is 1 degree.

Example:

```
prizm.setMotorDegree(1, 360, 180);          // spin DC Motor Channel 1 clockwise at
                                            // 360 degrees per second to target position
                                            // 180 degrees (1/2 revolution) and stop
                                            // when target is reached, stop motor and
                                            // hold position at the 180 degree mark

prizm.setMotorDegree(2, 180, -360);         // spin DC Motor Channel 1 counterclockwise
                                            // at 180 degrees per second to
                                            // target position -360 degrees or
                                            // (1 revolution) and stop; when target is
                                            // reached, stop motor and hold position at
                                            // the -360 degree mark
```

***myname*.setMotorDegrees(speed1, degrees1, speed2, degrees2);**

This function commands the speed and degrees target of both motor channels at the same time. This function requires both DC motors being controlled to have their encoders connected to PRIZM in order to implement velocity and position target PID control. This function is identical to the set "targets" functions explained previously, except the target values are defined in degrees instead of encoder counts. Working in degrees might be a bit more intuitive than actual encoder counts. The parameter *speed1* sets the rate of rotation for Motor 1 in degrees per second. The *degrees1* parameter is the position in degrees to which the Motor 1 will rotate, stop, and hold position. The *speed2* parameter sets the rate of rotation of Motor 2 in degrees per second. The parameter *degrees2* sets the position in degrees to which Motor 2 will rotate, stop, and hold position. When called, this function will spin the commanded motor channel to make the current encoder degrees value become equal to the degrees value. The motors will rotate toward their commanded targets at rotation rates set in *speed1* and *speed2* (degrees per second). When a motor has reached the target, it will hold its position in a servo-like mode until a new command is received. This function ignores the positive/negative sign values of *speed1* and *speed2*. The direction of rotation is set by the commanded encoder target value of *degrees1* and *degrees2*. While this function is executing, the readMotorBusy() function can be read to indicate that a motor is busy moving to its target. When this function returns "1," the command motor channel is busy (moving to the target). When the target has been reached, the readMotorBusy() function will return "0." The encoder degrees resolution is 1 degree.

Example:

```
prizm.setMotorDegrees(360, 360, 180, -360);   // spin Motor 1 clockwise at 360
                                              // degrees per second and stop at
                                              // encoder target position of 360
                                              // degrees; spin Motor 2 counterclockwise
                                              // at 180 degrees per
                                              // second and stop at encoder target
                                              // position of -360 degrees; each
                                              // motor will hold its position when it
                                              // has reached its target value
```

***myname*.setMotorInvert(motor#, invert);**

This function can be used to invert the forward/reverse direction mapping for both DC motor channels and their encoder inputs. This function is intended to harmonize the forward and reverse directions for motors on opposite sides of a skid-steer robot chassis. The *motor#* parameter sets the motor channel of 1 or 2. The value of the *invert* parameter sets the condition; *invert* = 0 means no invert, while *invert* = 1 inverts the motor and encoder set by channel. The default on controller power-up is no invert. This function needs to be called only once in the setup section of an Arduino sketch. However, it can be called again and changed at any time in program code.

Example:

```
prizm.setMotorInvert(1, 1);                   // invert the rotation direction of Motor 1
prizm.setMotorPower(1, 50);                   // normally, this function would cause Motor 1 to
                                              // spin at 50% power in the clockwise direction
                                              // but because we called the invert function, the
                                              // actual resulting rotation of Motor 1 will be
                                              // counterclockwise (-50); the same mapping of
                                              // direction applies to all the motor power,
                                              // speed, and targeting functions when invert is
                                              // called
```

***myname*.readMotorBusy(motor#);**

This function will return a value equal to "0" or "1" depending on the status of any DC motor channel that is in targeting mode. That is, any DC motor that is being controlled using an encoder for rotation to an encoder count or degree target will return its status when polled with this function. The parameter *motor#* sets the motor channel to 1 or 2. The function will return "1" when the motor is moving toward its target. The function will return "0" when the motor has reached its target. Being able to poll the status of a targeting motor can be very useful in code to wait for a motor to reach its target before continuing on to the next instruction.

Example:

```
int x;                                  // define an integer variable x
x = prizm.readMotorBusy(1);             // read the returned status of Motor 1 into x
or
x = prizm.readMotorBusy(2);             // read the returned status of Motor 2 into x
or
Serial.print(prizm.readMotorBusy(1));   // print the status of Motor 1 to the Arduino
                                        // serial monitor
```

***myname*.readMotorCurrent(motor#);**

This function reads the amount of DC current being drawn by a DC motor connected to either the Motor 1 or Motor 2 port. The *motor#* parameter sets the motor channel to 1 or 2. The amount of current that a motor draws will depend on the amount of load that the motor is moving. A heavier load on our motor will result in larger current draws. In order to keep from damaging a motor, it is important to not subject a motor to continuous high loads. High current draws will cause a DC motor to heat and eventually might result in failure. This function can be used to set a limit on how much current a motor can be allowed to draw. See the example program that came with the PRIZM library for a full code example of how to implement current limiting.

Example:

```
int x;                                      // create an integer variable x
x = prizm.readMotorCurrent(1);              // read the Motor 1 current and store it in x
or
Serial.print(prizm.readMotorCurrent(2));    // read Motor 2 current and print it to the
                                            // serial monitor; the value that this
                                            // function returns is an integer and is the
                                            // current in milliamps; for example, a value
                                            // of 1500 equals 1.5 amps
```

Encoder Functions

The TETRIX quadrature motor encoders enable precise speed and position control of TETRIX DC motors. The PRIZM controller will read the counting data of the encoder and implement an internal PID algorithm. The PRIZM controller's PID control algorithm has been tuned for TETRIX DC motors and quadrature encoders. TETRIX quadrature encoders produce 360 counts per revolution (CPR) and 1,440 pulses per revolution (PPR). This translates to 1/4 degree of DC motor shaft rotational resolution.

***myname*.readEncoderCount(enc#);**

This function reads and returns the encoder count of a TETRIX quadrature encoder attached to encoder port ENC1 or ENC2. The count value returned is a long-type value. The parameter *enc#* can be 1 or 2. A value of 1 will return the count value of ENC1. An *enc#* value of 2 will return the count value of ENC2. For reference, 1,440 counts = 1 motor shaft revolution.

Example:

```
long x;                                 // set up a long-type variable x
x = prizm.readEncoderCount(1);          // read and store the count of ENC1 into the long
                                        // variable x
```

or

```
Serial.print(prizm.readEncoderCount(2); // print the count value of ENC2 to the
                                        // Arduino serial monitor
```

***myname*.readEncoderDegrees(enc#);**

This function reads and returns the encoder count in degrees of a TETRIX quadrature encoder attached to encoder port ENC1 or ENC2. The value returned is a long-type value. The parameter *enc#* can be 1 or 2. A value of 1 will return the degree position of ENC1. A value of 2 will return the degree position of ENC2. Using the degrees function to track motor rotation might be more intuitive than using the raw encoder count values. Although the raw count value will yield 1/4-degree resolution, the degrees function resolution is limited to 1-degree resolution.

Example:

```
long x;                                 // set up a long-type variable x
x = prizm.readEncoderDegrees(1);        // set up a long-type variable x; read and
                                        // store the degree value of ENC1 into the
                                        // long variable x
```

or

```
Serial.print(prizm.readEncoderDegrees(2)); // print the degree value of ENC2 to
                                           // the Arduino serial monitor
```

***myname*.resetEncoder(enc#);**

When called, this function resets the encoder count set in parameter *enc#* (1 or 2) to 0.

Example:

```
prizm.resetEncoder(1);                  // reset Encoder count 1 (ENC1) to 0
prizm.resetEncoder(2);                  // reset Encoder count 2 (ENC2) to 0
```

***myname*.resetEncoders();**

When called, this function resets the encoder count of Encoders 1 and 2 to 0.

Example:

```
prizm.resetEncoders();                  // reset both encoders (1 and 2) to 0
```

Servo Motor Functions

TETRIX servo motors come in two types. A standard servo can be commanded to rotate to any position between 0 and 180 degrees. Once there, it will hold its position and resist being moved. The PRIZM controller has six servo ports marked 1-6. Each port can control one standard servo for six independent channels of servo control. A different type of servo is a continuous rotation (CR) servo. A CR servo is designed to rotate clockwise or counterclockwise in the same way that a DC motor can. The PRIZM controller has two specialized ports marked CR1 and CR2 that are designed to spin CR servo motors in either direction.

A CR motor can be helpful when we want a DC motor to fit into a small space or for light-duty loads. When you are plugging servos into a PRIZM servo port, the servo red wire matches the "+" mark, and the black wire matches the "-" mark.

***myname*.setServoSpeed(servo#, speed);**

This function sets the speed of a standard servo while it is rotating to its commanded position. The parameter *servo#* selects the servo channel. The parameter *speed* sets the speed of rotation. This function needs to be called only once at the beginning of a program. However, it can be called at any time in program code to change speeds. Speed range is between 0 and 100, equating to speeds of 0-100%. If this function is not called, servo speed will always default to 100.

Example:

```
prizm.setServoSpeed(1, 25);                    // set the speed of Servo channel 1 to 25%
```

***myname*.setServoSpeeds(speed1, speed2, speed3, speed4, speed5, speed6);**

This function enables setting all six servo channels within one function statement. The parameters *speed(#)* can range between 0 and 100. Each speed can be different. Please note that when using the setServoSpeeds functions, values must be set for all six ports, even if you are not using all six.

Example:

```
prizm.setServoSpeeds(25, 25, 25, 25, 25, 25);      // set all six servo channel speeds to 25%
```

***myname*.setServoPosition(servo#, position);**

This function rotates a servo to a specified target position. The parameter *servo#* refers to the servo that will change position. The *position* parameter specifies the target position to move toward. Position can be anywhere between 0 and 180. The *position* parameter represents the position rotational target of the servo in degrees. Be careful not to try to rotate a servo past its internal mechanical limit. This limit can vary slightly from servo to servo. When rotating toward an extreme value of 0 or 180, listen to the servo to be sure it is not making a buzzing noise, meaning it could be against a mechanical stop. This might cause damage to the servo and waste battery power.

Example:

```
prizm.setServoPosition(1, 180);                    // rotate Servo 1 to position 180 degrees
```

***myname*.setServoPositions(position1, position2, position3, position4, position5, position6);**

This function enables rotation of all six servo channels at once to target positions. The parameter *position(#)* sets the target position of each channel in order of 1-6. The target position can be any value of 0-180, which represents degrees. Each target position may be different. Please note that when using the setServoPositions functions, values must be set for all six ports, even if you are not using all six.

Example:

```
prizm.setServoPositions(90,90,90,90,90,90);        // rotate all six servos to 90 degrees
```

***myname*.readServoPosition(servo#);**

This function reads and returns the current position of a servo channel. The parameter *servo#* can be any channel 1-6. The value returned is an integer type of value 0-180.

Example:

```
int x;                                  // set up an integer variable named x
x = prizm.readServoPosition(1);         // read the current rotation position of Servo
                                        // channel 1 and store it in variable x
```

or

```
Serial.print(prizm.readServoPosition(2));   // read the position of Servo channel 2 and
                                            // print it to the Arduino serial monitor
```

***myname*.setCRServoState(CRservo#, state);**

This function controls the on/off state of a continuous rotation (CR) servo motor. The parameter *CRservo#* refers to the CR motor channel to be controlled. The *state* parameter sets the on/off state and the direction of rotation. The *state* parameter can be 100 for clockwise rotation, -100 for counterclockwise rotation, and 0 for off, or stop.

Example:

```
prizm.setCRServoState(1, 100);      // turn CR1 on to rotate continuous in a clockwise
                                    // direction

prizm.setCRServoState(1, 0);        // turn CR1 off and stop

prizm.setCRServoState(1, -100);     // turn CR1 on to rotate continuous in a
                                    // counterclockwise direction
```

TETRIX PRIZM Arduino Library Functions Chart

Description	Function	Coding Example
Prizm Begin Is called in the Arduino code setup() loop. Initializes the PRIZM controller.	**PrizmBegin();** Data Type: None	**PrizmBegin**(); *Reset and initialize PRIZM controller.*
Prizm End When called, immediately terminates a program and resets the PRIZM controller.	**PrizmEnd();** Data Type: None	**PrizmEnd**(); *Terminate a PRIZM program and reset controller.*
Set Red LED Sets the PRIZM red indicator LED to on or off.	**setRedLED**(state); Data Type: *state* = integer Data Range: *state* = 1 or 0 or *state* = HIGH or LOW	**setRedLED**(HIGH); *or* **setRedLED**(1); *Turn red LED on.* **setRedLED**(LOW); *or* **setRedLED**(0); *Turn red LED off.*
Set Green LED Sets the PRIZM green indicator LED to on or off.	**setGreenLED**(state); Data Type: *state* = integer Data Range: *state* = 1 or 0 or *state* = HIGH or LOW	**setGreenLED**(HIGH); *or* **setGreenLED**(1); *Turn green LED on.* **setGreenLED**(LOW); *or* **setGreenLED**(0); *Turn green LED off.*
Set DC Motor Power Sets the power level and direction of a TETRIX DC motor connected to the PRIZM DC motor ports. Power level range is 0 to 100. Direction is set by the sign (+/-) of the power level. Power level 0 = stop in coast mode. Power level 125 = stop in brake mode.	**setMotorPower**(motor#, power); Data Type: *motor#* = integer *power* = integer Data Range: *motor#* = 1 or 2 *power* = -100 to 100 or *power* = 125 (brake mode)	**setMotorPower**(1, 50); *Spin Motor 1 clockwise at 50% power.* **setMotorPower**(2, -50%); *Spin Motor 2 counterclockwise at 50% power.* **setMotorPower**(1, 0); *Turn off Motor 1 in coast mode.* **setMotorPower**(2, 125); *Turn off Motor 2 in brake mode.*
Set DC Motor Powers Simultaneously sets the power level and direction of **both** TETRIX DC motors connected to the PRIZM motor ports. Both PRIZM Motor 1 and Motor 2 channel parameters are set with a single statement. The power level range is 0 to 100. Direction is set by the sign (+/-) of the power level. Power level 0 = stop in coast mode. Power level 125 = stop in brake mode.	**setMotorPowers**(power1, power2); Data Type: *power1* = integer *power2* = integer Data Range: *power1* = -100 to 100 *power2* = -100 to 100 or *power1* = 125 (brake mode) *power2* = 125 (brake mode)	**setMotorPowers**(50, 50); *Spin Motor 1 and Motor 2 clockwise at 50% power.* **setMotorPowers**(-50, 50); *Spin Motor 1 counterclockwise and Motor 2 clockwise at 50% power.* **setMotorPowers**(0, 0); *Turn off Motor 1 and Motor 2 in coast mode.* **setMotorPowers**(125, 125); *Turn off Motor 1 and Motor 2 in brake mode.*
Set DC Motor Speed Uses velocity PID control to set the constant speed of a TETRIX DC motor with a TETRIX motor encoder connected. The *speed* parameter range is 0 to 720 degrees per second (DPS). The sign (+/-) of the *speed* parameter controls direction of rotation.	**setMotorSpeed**(motor#, speed); Data Type: *motor#* = integer *speed* = integer Data Range: *motor#* = 1 or 2 *speed* = -720 to 720	**setMotorSpeed**(1, 360); *Spin Motor 1 clockwise at a constant speed of 360 DPS.* **setMotorSpeed**(1, -360); *Spin Motor 1 counterclockwise at a constant speed of 360 DPS.*

Description	Function	Coding Example
Set DC Motor Speeds Uses velocity PID control to simultaneously set the constant speeds of **both** TETRIX DC motor channels with TETRIX motor encoders connected. Both PRIZM Motor 1 and Motor 2 channel parameters are set with a single statement. The speed parameter range is 0 to 720 degrees per second (DPS). The sign (+/-) of the speed parameter controls direction of rotation.	**setMotorSpeeds**(speed1, speed2); Data Type: *speed1* = integer *speed2* = integer Data Range: *speed1* = -720 to 720 *speed2* = -720 to 720	**setMotorSpeeds**(360, 360); *Spin Motor 1 and Motor 2 clockwise at a constant speed of 360 DPS.* **setMotorSpeeds**(360, -360); *Spin Motor 1 clockwise and Motor 2 counterclockwise at a constant speed of 360 DPS.* **setMotorSpeeds**(360, -180); *Spin Motor 1 clockwise and Motor 2 counterclockwise at a constant speed of 180 DPS.*
Set DC Motor Target Implements velocity and positional PID control to set the constant speed and the encoder count target holding position of a TETRIX DC motor with a TETRIX encoder connected. The *speed* parameter range is 0 to 720 degrees per second (DPS). The encoder count target position is a signed long integer from -2,147,483,648 to 2,147,483,647. Each encoder count = 1/4-degree resolution.	**setMotorTarget**(motor#, speed, target); Data Type: *motor#* = integer *speed* = integer *target* = long Data Range: *motor#* = 1 or 2 *speed* = 0 to 720 *target* = -2147483648 to 2147483647	**setMotorTarget**(1, 360, 1440); *Spin Motor 1 at a constant speed of 360 DPS until encoder 1 count equals 1,440. When at encoder target count, hold position in a servo-like mode.* **setMotorTarget**(2, 180, -1440); *Spin Motor 2 at a constant speed of 180 DPS until encoder 2 count equals -1,440 (1 revolution). When at encoder target count, hold position in a servo-like mode.*
Set DC Motor Targets Implements velocity and positional PID control to simultaneously set the constant speeds and the encoder count target holding positions of **both** TETRIX DC motor channels each with TETRIX encoders connected. Both PRIZM Motor 1 and Motor 2 channel parameters are set with a single statement. The speed parameter range is 0 to 720 degrees per second (DPS). The encoder count target position is a signed long integer from -2,147,483,648 to 2,147,483,647. Each encoder count = 1/4-degree resolution.	**setMotorTargets**(speed1, target1, speed2, target2); Data Type: *speed1* = integer *target1* = long *speed2* = integer *target2* = long Data Range: *speed1* = 0 to 720 *target1* = -2147483648 to 2147483647 *speed2* = 0 to 720 *target2* = -2147483648 to 2147483647	**setMotorTargets**(360, 1440, 360, 1440); *Spin Motor 1 and Motor 2 at a constant speed of 360 DPS until each motor encoder count equals 1,440. When a motor reaches its encoder target count, hold position in a servo-like mode.* **setMotorTargets**(360, 1440, 180, 2880); *Spin Motor 1 at a constant speed of 360 DPS until encoder 1 count equals 1,440. Spin Motor 2 at a constant speed of 180 DPS until encoder 2 equals 2,880. Each motor will hold its position in a servo-like mode when it reaches the encoder target.* ***Note:*** *One encoder count equals 1/4-degree resolution. For example, 1 motor revolution equals 1,440 encoder counts (1,440 / 4 = 360).*
Set Motor Degree Implements velocity and positional PID control to set the constant speed and the degree target holding position of a TETRIX DC motor with a TETRIX encoder connected. The *speed* parameter range is 0 to 720 degrees per second (DPS). The encoder degrees target position is a signed long integer from -536,870,912 to 536,870,911 with a 1-degree resolution.	**setMotorDegree**(motor#, speed, degrees); Data Type: *motor#* = integer *speed* = integer *degrees* = long Data Range: *motor#* = 1 or 2 *speed* = 0 to 720 *degrees* = -536870912 to 536870911	**setMotorDegree**(1, 180, 360); *Spin Motor 1 at a constant speed of 180 DPS until encoder 1 degree count equals 360. When at encoder target degree count, hold position in a servo-like mode.* **setMotorDegree**(2, 90, 180); *Spin Motor 2 at a constant speed of 90 DPS until encoder 2 degree count equals 180. When at encoder target degree count, hold position in a servo-like mode.*

Description	Function	Coding Example
Set Motor Degrees Implements velocity and positional PID control to set the constant speeds and the degree target holding positions of both TETRIX DC motor channels with TETRIX encoders connected. Both PRIZM Motor 1 and Motor 2 channel parameters are set with a single statement. The speed parameter range is 0 to 720 degrees per second (DPS). The encoder degree target position is a signed long integer from -536,870,912 to 536,870,911 with a 1-degree resolution.	**setMotorDegrees**(speed1, degrees1, speed2, degrees2); Data Type: *speed1* = integer *degrees1* = long *speed2* = integer *degrees2* = long Data Range: *speed1* = 0 to 720 *degrees1* = -536870912 to 536870911 *speed2* = 0 to 720 *degrees2* = -536870912 to 536870911	**setMotorDegrees**(180, 360, 180, 360); *Spin Motor 1 and Motor 2 at a constant speed of 180 DPS until each motor encoder degree count equals 360. When a motor reaches its degree target count, hold position in a servo-like mode.* **setMotorDegrees**(360, 720, 90, 360); *Spin Motor 1 at a constant speed of 360 DPS until encoder 1 degree count equals 720. Spin Motor 2 at a constant speed of 90 DPS until encoder 2 degree equals 360. Each motor will hold its position in a servo-like mode when it reaches the encoder target.*
Set Motor Direction Invert Inverts the forward/reverse direction mapping of a DC motor channel. This function is intended to harmonize the forward and reverse directions for motors on opposite sides of a skid-steer robot chassis. Inverting one motor channel can make coding opposite-facing DC motors working in tandem more intuitive. An *invert* parameter of 1 = invert. An *invert* parameter of 0 = no invert. The default is no invert.	**setMotorInvert**(motor#, invert); Data Type: *motor#* = integer *invert* = integer Data Range: *motor#* = 1 or 2 *invert* = 0 or 1	**setMotorInvert**(1, 1); *Invert the spin direction mapping of Motor 1.* **setMotorInvert**(2, 1); *Invert the spin direction mapping of Motor 2.* **setMotorInvert**(1, 0); *Do not invert the spin direction mapping of Motor 1.* **setMotorInvert**(2, 0); *Do not invert the spin direction mapping of Motor 2.* ***Note:*** *Non-inverting is the default on PRIZM power-up or reset.*
Read Motor Busy Status The busy flag can be read to check on the status of a DC motor that is operating in positional PID mode. The motor busy status will return "1" if it is moving toward a positional target (degrees or encoder count). When it has reached its target and is in hold mode, the busy status will return "0."	**readMotorBusy**(motor#); Data Type: *motor#* = integer Data Range: *motor#* = 1 or 2 Data Type Returned: *value* = integer	**readMotorBusy**(1); *Return the busy status of Motor 1.* **readMotorBusy**(2); *Return the busy status of Motor 2.*
Read DC Motor Current Reads the DC motor current of each TETRIX DC motor attached to PRIZM Motor 1 and Motor 2 ports. The integer value that is returned is motor load current in milliamps.	**readMotorCurrent**(motor#); Data Type: *motor#* = integer Data Range: *motor#* = 1 or 2 Data Type Returned: *value* = integer	**readMotorCurrent**(1); *Read the motor load current of the PRIZM Motor 1 channel.* **readMotorCurrent**(2); *Read the motor load current of the PRIZM Motor 2 channel.* *Example: 1500 = 1.5 amps*

Description	Function	Coding Example
Read Encoder Count Reads the encoder count value. The PRIZM controller uses encoder pulse data to implement PID control of a TETRIX DC motor connected to the motor ports. The PRIZM controller counts the number of pulses produced over a set time period to accurately control velocity and position. Each 1/4 degree equals one pulse, or count, or 1 degree of rotation equals 4 encoder counts. The current count can be read to determine a motor's shaft position. The total count accumulation can range from -2,147,483,648 to 2,147,483,647. A clockwise rotation adds to the count value, while a counterclockwise rotation subtracts from the count value. The encoder values are set to 0 at power-up and reset.	**readEncoderCount**(enc#); Data Type: *enc#* = integer Data Range: *enc#* = 1 or 2 Data Type Returned: *value* = long	**readEncoderCount**(1); *Read the current count value of encoder 1 (ENC 1 port).* **readEncoderCount**(2); *Read the current count value of encoder 2 (ENC 2 port).*
Read Encoder Degrees Reads the encoder degree value. The PRIZM controller uses encoder pulse data to implement PID control of a TETRIX DC motor connected to the motor ports. The PRIZM controller counts the number of pulses produced over a set time period to accurately control velocity and position. This function is similar to the encoder count function, but instead of returning the raw encoder count value, it returns the motor shaft position in degrees. The total degree count accumulation can range from -536,870,912 to 536,870,911. A clockwise rotation adds to the count value, while a counterclockwise rotation subtracts from the count value. The encoder values are set to 0 at power-up and reset.	**readEncoderDegrees**(enc#); Data Type: *enc#* = integer Data Range: *enc#* = 1 or 2 Data Type Returned: *value* = long	**readEncoderDegrees**(1); *Read the current degree count value of encoder 1 (ENC 1 port).* **readEncoderDegrees**(2); *Read the current degree count value of encoder 2 (ENC 2 port).*
Reset Each Encoder Resets the encoder count accumulator to 0.	**resetEncoder**(enc#); Data Type: *enc#* = integer Data Range: enc# = 1 or 2	**resetEncoder**(1); *Reset the encoder 1 count to 0.* **resetEncoder**(2); *Reset the encoder 2 count to 0.*
Reset Both Encoders (1 and 2) Resets the encoder 1 and encoder 2 count accumulators to 0.	**resetEncoders**(); Data Type: None	**resetEncoders**(); *Reset the encoder 1 count to 0 and encoder 2 count to 0.*

Description	Function	Coding Example
Read Line Sensor Output Reads the digital output of the Line Finder Sensor connected to a PRIZM sensor port. The value read is "0" when reflected light is received (detecting a light-colored surface) and "1" when light is not received (detecting a dark-colored surface, such as a line).	**readLineSensor**(port#); Data Type: *port#* = integer Data Range: *port#* (See note in adjacent column.) Data Type Returned: *value* = integer (0 or 1)	**readLineSensor**(2); *Read the digital value of a Line Finder Sensor on digital sensor port D2.* ***Note:*** *The Line Finder Sensor can be connected to any digital port D2-D5, or analog ports A1-A3 configured as digital input. See technical section on sensor ports for a more detailed explanation of sensor ports and pinouts.*
Read Ultrasonic Sensor in Centimeters Reads the distance in centimeters of an object placed in front of the Ultrasonic Sensor. The sensor is modulated at 42 kHz and has a range of 3 to 400 centimeters. The value read is an integer.	**readSonicSensorCM**(port#); Data Type: *port#* = integer Data Range: *port#* (See note in adjacent column.) Data Type Returned: *value* = integer (3 to 400) Min and max might slightly vary.	**readSonicSensorCM**(3); *Read the distance in centimeters of an object placed in front of the Ultrasonic Sensor connected to digital sensor port D3.* ***Note:*** *The Ultrasonic Sensor can be connected to any digital port D2-D5, or analog ports A1-A3 configured as digital input. See technical section on sensor ports for a more detailed explanation of sensor ports and pinouts.*
Read Ultrasonic Sensor in Inches Reads the distance in inches of an object placed in front of the Ultrasonic Sensor. The sensor is modulated at 42 kHz and has a range of 2 to 150 inches. The value read is an integer.	**readSonicSensorIN**(port#); Data Type: *port#* = integer Data Range: *port#* (See note in adjacent column.) Data Type Returned: *value* = integer (2 to 150) Min and max might slightly vary.	**readSonicSensorIN**(4); *Read the distance in inches of an object placed in front of the Ultrasonic Sensor connected to digital sensor port D4.* ***Note:*** *The Ultrasonic Sensor can be connected to any digital port D2-D5, or analog ports A1-A3 configured as digital input.*
Read Battery Pack Voltage Reads the voltage of the TETRIX battery pack powering the PRIZM controller. The value read is an integer.	**readBatteryVoltage**(); Data Type: None Data Type Returned: *value* = integer	**readBatteryVoltage**(); *Read the voltage of the TETRIX battery pack powering the PRIZM controller.* *Example: A value of 918 equals 9.18 volts.*
Read Start Button State Reads the state of the green PRIZM Start button. A returned value of "1" indicates a pressed state. A returned value of "0" indicates a not-pressed state.	**readStartButton**(); Data Type: None Data Type Returned: *value* = integer (0 or 1)	**readStartButton**(); *Read the Start button. A value of 1 means button is pressed. A value of 0 means button is not pressed.*

Description	Function	Coding Example
Set Speed of a Servo Motor Sets the speed of a servo motor connected to a PRIZM servo port 1-6. The *speed* parameter can be 0 to 100%. The servo motor channel parameter can be any number 1 to 6. If not specified, the speed of a servo defaults to 100 (maximum speed). When a servo speed has been set, it will always move at the set speed until changed. Unless we are changing speeds, it will need to be called only once at the beginning of a program.	**setServoSpeed**(servo#, speed); Data Type: *servo#* = integer *speed* = integer Data Range: *servo#* = 1 to 6 *speed* = 0 to 100	**setServoSpeed**(1, 25); *Set the speed of servo channel 1 to 25%.* **setServoSpeed**(2, 50); *Set the speed of servo channel 2 to 50%.*
Set Speeds of All Servo Motors Sets the speeds of all six servo channels simultaneously with a single command. All six speeds are in sequential order and can be 0 to 100%. All six servo speeds may be the same or different.	**setServoSpeeds**(speed1, speed2, speed3, speed4, speed5, speed6); Data Type: *speed1-speed6* = integer Data Range: *speed1-speed6* = 0 to 100	**setServoSpeeds**(25, 25, 25, 25, 25, 25); *Set the speeds of all six servo channels to 25%.* **setServoSpeeds**(25, 35, 45, 55, 65, 75); *Servo 1 speed = 25%* *Servo 2 speed = 35%* *Servo 3 speed = 45%* *Servo 4 speed = 55%* *Servo 5 speed = 65%* *Servo 6 speed = 75%*
Set Position of a Servo Motor Sets the angular position of a servo motor connected to a PRIZM servo motor port 1-6. The *position* parameter can be any value between 0 and 180 degrees. Any value outside this range is ignored. Not all servos are the same, so be careful when operating a servo motor at the extreme ranges. Listen closely; if a servo is buzzing, it is pressing against its mechanical stop, which might damage the motor. If this happens, limit the range to values slightly greater than 0 and slightly less than 180 to avoid damage to the servo motor.	**setServoPosition**(servo#, position); Data Type: *servo#* = integer *position* = integer Data Range: *servo#* = 1 to 6 *position* = 0 to 180	**setServoPosition**(1, 90); *Set the angular position of Servo Motor 1 to 90 degrees.* **setServoPosition**(2, 130); *Set the angular position of Servo Motor 2 to 130 degrees.*
Set Positions of All Servo Motors Sets the angular positions of all six servo motors connected to the PRIZM servo motor ports 1-6. The *position* parameter can be any value between 0 and 180 degrees. Any value outside this range is ignored. Not all servos are the same, so be careful when operating a servo motor at the extreme ranges. Listen closely; if a servo is buzzing, it is pressing against its mechanical stop, which might damage the motor. If this happens, limit the range to values slightly greater than 0 and slightly less than 180 to avoid damage to the servo motor.	**setServoPositions**(position1, position2, position3, position4, position5, position6); Data Type: *position1-position6* = integer Data Range: *position1-position6* = 0 to 180	**setServoPositions**(90, 90, 90, 90, 90, 90); *Set the angular positions of all six servo motors connected to PRIZM servo ports 1-6 to 90 degrees.* **setServoPositions**(10, 20, 30, 40, 50, 60); *Set the angular positions of all six servo motors connected to PRIZM servo ports 1-6.* *Servo 1 position = 10 degrees* *Servo 2 position = 20 degrees* *Servo 3 position = 30 degrees* *Servo 4 position = 40 degrees* *Servo 5 position = 50 degrees* *Servo 6 position = 60 degrees*

Description	Function	Coding Example
Read a Servo Position Reads the most recent commanded position of a servo motor connected to PRIZM servo ports 1-6. The value returned will be 0-180.	**readServoPosition**(servo#); Data Type: *servo#* = integer Data Range: *servo#* = 1 to 6 Data Type Returned: *value* = integer (0 to 180)	**readServoPosition**(1); *Read the most recent commanded position of Servo 1.* **readServoPosition**(2); *Read the most recent commanded position of Servo 2.*
Set Continuous Rotation (CR) State Sets the on/off and direction state of a CR servo connected to PRIZM CR1 and CR2 ports. A *state* parameter of -100 equals spin counterclockwise. A *state* parameter of 100 equals spin clockwise. A *state* parameter of 0 is off or stop. *CRservo#* parameter can be 1 or 2 commanding either CR1 or CR2 servo ports.	**setCRServoState**(CRservo#, state); Data Type: *CRservo#* = integer *state* = integer Data Range: *CRservo#* = 1 or 2 *state* = -100, 0, 100	**setCRServoState**(1, 100); *Spin CR1 servo continuously clockwise.* **setCRServoState**(1, 0); *Stop CR1 servo.* **setCRServoState**(1, -100); *Spin CR1 servo continuously counterclockwise.*

TETRIX PRIZM Arduino Library Functions Cheat Sheet

Below is a reference of each function statement in the TETRIX PRIZM Robotics Controller Library for Arduino.

Tip: When using these functions in our code, be sure to prefix each function with the PRIZM object name that we choose using the PRIZM *myname* statement.

PrizmBegin();

PrizmEnd();

setRedLED(HIGH/LOW);

setGreenLED(HIGH/LOW);

setMotorPower(motor#, power);

setMotorPowers(power1, power2);

setMotorSpeed(motor#, speed);

setMotorSpeeds(speed1, speed2);

setMotorTarget(motor#, speed, target);

setMotorTargets(speed1, target1, speed2, target2);

setMotorDegree(motor#, speed, degrees);

setMotorDegrees(speed1, degrees1, speed2, degrees2);

setMotorInvert(motor#, invert);

readMotorBusy(motor#);

readMotorCurrent(motor#);

readEncoderCount(enc#);

readEncoderDegrees(enc#);

resetEncoder(enc#);

resetEncoders();

readLineSensor(port#);

readSonicSensorCM(port#);

readSonicSensorIN(port#);

readBatteryVoltage();

readStartButton();

setServoSpeed(servo#, speed);

setServoSpeeds(speed1, speed2, speed3, speed4, speed5, speed6);

setServoPosition(servo#, position);

setServoPositions(position1, position2, position3, position4, position5, position6);

readServoPosition(servo#);

setCRServoState(CRservo#, state);

TETRIX PRIZM Sample Code Library

Example 1: GettingStarted_Act1_Blink_RedLED

```
/*  PRIZM controller example program
 *   Blink the PRIZM red LED at a 1-second flash rate.
 *   author PWU on 08/05/2016
*/

  #include <PRIZM.h>                  // include the PRIZM library

  PRIZM prizm;                        // instantiate a PRIZM object “prizm” so we can use its functions

void setup() {

   prizm.PrizmBegin();                // initialize the PRIZM controller

}

void loop() {                         // repeat this code in a loop

  prizm.setRedLED(HIGH);              // turn the red LED on
  delay(1000);                        // wait here for 1,000 ms (1 second)
  prizm.setRedLED(LOW);               // turn the red LED off
  delay(1000);                        // wait here for 1,000 ms (1 second)

}
```

Example 2: GettingStarted_Act2_Move_DCMotor

```
/*  PRIZM controller example program
 *  Spin DC Motor Channel 1 for 5 seconds and then coast to a stop.
 *  After stopping, wait for 2 seconds and then spin in opposite direction.
 *  Continue to repeat until red Reset button is pressed.
 *  author PWU on 08/05/2016
*/

  #include <PRIZM.h>                  // include the PRIZM library in the sketch
  PRIZM prizm;                        // instantiate a PRIZM object “prizm” so we can use its functions

void setup() {

  prizm.PrizmBegin();                 // initialize the PRIZM controller
}

void loop() {                         // repeat in a loop

  prizm.setMotorPower(1,25);          // spin Motor 1 CW at 25% power
  delay(5000);                        // wait while the motor runs for 5 seconds
  prizm.setMotorPower(1,0);           // stop motor (coast to stop)
  delay(2000);                        // after stopping, wait here for 2 seconds
  prizm.setMotorPower(1,-25);         // spin Motor 1 CCW at 25% power
  delay(5000);                        // wait while the motor runs for 5 seconds
  prizm.setMotorPower(1,0);           // stop motor (coast to stop)
  delay(2000);                        // after stopping, wait here for 2 seconds and then repeat

}
```

Example 3: GettingStarted_Act3_Move_Servo

```
/*  PRIZM controller example program
 *  This program sets the speed of Servo 1 to 25%.
 *  Servo 1 is then rotated back and forth between 0- and 180-degree range.
 *  author PWU on 08/05/2016
*/

#include <PRIZM.h>                  // include the PRIZM library in the sketch
PRIZM prizm;                        // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {

 prizm.PrizmBegin();                // initialize the PRIZM controller
 prizm.setServoSpeed(1,25);         // set Servo 1 speed to 25%

}

void loop() {                       // repeat in a loop

 prizm.setServoPosition(1,180);     // rotate Servo 1 to 180 degrees
 delay(3000);                       // wait for 3 seconds to give Servo 1 time
                                    // to get to position 180
 prizm.setServoPosition(1,0);       // rotate Servo 1 to 0 degrees
 delay(3000);                       // wait for 3 seconds to give Servo 1 time
                                    // to get to position 0

}
```

Example 4: GettingStarted_Act4_Intro_LineFinder

```
/*  PRIZM controller example program
 *  This program will read the digital signal of the Line Finder Sensor attached to digital port D3.
 *  If the sensor is facing a reflective surface and receiving a reflected IR beam, the PRIZM red LED
 *  will switch on. If the sensor is facing a dark surface or is too far away from a reflective surface,
 *  the red LED will be switched off.
*/

 #include <PRIZM.h>                 // include the PRIZM library
 PRIZM prizm;                       // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {                      // this code runs once

 prizm.PrizmBegin();                // initialize PRIZM

}

void loop() {                       // this code repeats in a loop

 if(prizm.readLineSensor(3) == HIGH) {prizm.setRedLED(LOW);}        // LED off

 if(prizm.readLineSensor(3) == LOW) {prizm.setRedLED(HIGH);}        // LED on

delay(50);                          // slow the loop down

}
```

Example 5: GettingStarted_Act5_Intro_UltraSonic

```
/*   PRIZM controller example program
 *   This example program will read the digital signal of the
 *   Ultrasonic Sensor attached to the digital sensor port D3.
 *   The distance in centimeters of an object placed in front
 *   of the sensor will be sent to the serial monitor window.
 */

  #include <PRIZM.h>                  // include the PRIZM library
  PRIZM prizm;                        // instantiate a PRIZM object “prizm” so we can use its functions

void setup() {                        // this code runs once

  prizm.PrizmBegin();                 // initialize PRIZM

  Serial.begin(9600);                 // configure the serial monitor for 9600 baud rate

}

void loop() {                         // this code repeats in a loop

  Serial.print(prizm.readSonicSensorCM(3));   // print the cm distance to the serial monitor

  Serial.println(" Centimeters");     // print " Centimeters"

  delay(200);                         // slow down the loop. wait for 200 ms to keep from printing to serial monitor too
fast
}
```

Example 6: TaskBot_Act7_Drive_Forward

```
/* PRIZM controller example program
 *  This program will move the PRIZM TaskBot forward for 3 seconds, stop, and end program.
 *  author PWU 08/05/2016
 */

  #include <PRIZM.h>                  // include PRIZM library
  PRIZM prizm;                        // instantiate a PRIZM object “prizm” so we can use its functions

void setup() {

  prizm.PrizmBegin();                 // initialize PRIZM

  prizm.setMotorInvert(1,1);          // invert the direction of DC Motor 1
                                      // to harmonize the direction of opposite-facing drive motors
}

void loop() {

  prizm.setMotorPowers(50,50);        // turn Motors 1 and 2 on at 50% power
  delay(3000);                        // wait here for 3 seconds while motors are spinning
  prizm.PrizmEnd();                   // end program and reset PRIZM

}
```

Example 7: TaskBot_Act8_Drive_Circle

```
/*  PRIZM controller example program
 *  This program will cause the PRIZM TaskBot to drive in a continuous circle.
 *  Press the red Reset button to stop the program.
 *  author PWU 08/05/2016
 */

  #include <PRIZM.h>                  // include PRIZM library
  PRIZM prizm;                        // instantiate a PRIZM object “prizm” so we can use its functions

void setup() {

  prizm.PrizmBegin();                 // initialize PRIZM
  prizm.setMotorInvert(1,1);          // invert the direction of DC Motor 1 to harmonize direction

}

void loop() {

  prizm.setMotorPowers(50,25);        // spin Motor 1 at 50% power and Motor 2 at 25% power,
                                      // resulting in the robot driving in a clockwise circle
}
```

Example 8: TaskBot_Act9_Drive_Square_1

```
/*  PRIZM controller example program
 *  This program will move the PRIZM TaskBot in a square driving pattern.
 *  author PWU 08/05/2016
*/
 #include <PRIZM.h>                 // include PRIZM library
 PRIZM prizm;                       // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {
 prizm.PrizmBegin();                // initialize PRIZM
 prizm.setMotorInvert(1,1);         // invert the direction of DC Motor 1 to harmonize the direction
}
void loop() {
 prizm.setMotorPowers(50,50);       // go forward at 50% power
 delay(3000);                       // wait here for 3 seconds while motors spin
 prizm.setMotorPowers(125,125);     // stop both motors in brake mode
 delay(1000);                       // wait here for 1 second
 prizm.setMotorPowers(50,-50);      // make a right turn
 delay(600);                        // wait here for 0.6 seconds while motors spin
 prizm.setMotorPowers(125,125);     // stop both motors in brake mode
 delay(1000);                       // wait here for 1 second
 prizm.setMotorPowers(50,50);       // go forward at 50% power
 delay(3000);                       // wait here for 3 seconds while motors spin
 prizm.setMotorPowers(125,125);     // stop both motors in brake mode
 delay(1000);                       // wait here for 1 second
 prizm.setMotorPowers(50,-50);      // make a right turn
 delay(600);                        // wait here for 0.6 seconds while motors spin
 prizm.setMotorPowers(125,125);     // stop both motors in brake mode
 delay(1000);                       // wait here for 1 second
 prizm.setMotorPowers(50,50);       // go forward at 50% power
 delay(3000);                       // wait here for 3 seconds while motors spin
 prizm.setMotorPowers(125,125);     // stop both motors in brake mode
 delay(1000);                       // wait here for 1 second
 prizm.setMotorPowers(50,-50);      // make a right turn
 delay(600);                        // wait here for 0.6 seconds while motors spin
 prizm.setMotorPowers(125,125);     // stop both motors in brake mode
 delay(1000);                       // wait here for 1 second
 prizm.setMotorPowers(50,50);       // go forward at 50% power
 delay(3000);                       // wait here for 3 seconds while motors spin
 prizm.setMotorPowers(125,125);     // stop both motors in brake mode
 delay(1000);                       // wait here for 1 second
 prizm.PrizmEnd();                  // end program and reset PRIZM
}
```

Example 9: TaskBot_Act10_Drive_Square_2

```
/*  PRIZM controller example program
 *   This program will move the TaskBot in a square driving pattern using a forward and right turn function.
 *  author PWU 08/05/2016
 */
 #include <PRIZM.h>                  // include PRIZM library
 PRIZM prizm;                        // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {

 prizm.PrizmBegin();                 // initialize PRIZM
 prizm.setMotorInvert(1,1);          // invert the direction of DC Motor 1 to harmonize direction

}

void loop() {

for(int x=0; x<=3; x++) {            // do this four times, increment x by +1
 forward();
 rightTurn();
 }
 prizm.PrizmEnd();
}

void forward() {                     // function to go forward
 prizm.setMotorPowers(50,50);        // go forward at 50% power
 delay(3000);                        // wait here for 3 seconds while motors spin
 prizm.setMotorPowers(125,125);      // stop both motors in brake mode
 delay(1000);                        // wait here for 1 second

 }

void rightTurn() {                   // function for a right turn
 prizm.setMotorPowers(50,-50);       // make a right turn
 delay(600);                         // wait here for 0.6 seconds while motors spin
 prizm.setMotorPowers(125,125);      // stop both motors in brake mode
 delay(1000);                        // wait here for 1 second

 }
```

Example 10: TaskBot_Act11_Drive_To_Line

```
/*  PRIZM controller example program
 *  This program will move the PRIZM TaskBot forward on a white surface until it detects a black
 *  line. When the line is detected, the robot will stop.
 *  Connect the Line Finder Sensor to digital port D3.
 *  author PWU 08/05/2016
*/

  #include <PRIZM.h>                    // include PRIZM library
  PRIZM prizm;                          // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {

  prizm.PrizmBegin();                   // initialize PRIZM

  prizm.setMotorInvert(1,1);            // invert the direction of DC Motor 1
                                        // to harmonize the direction of
                                        // opposite-facing drive motors
}

void loop() {

  if(prizm.readLineSensor(3) == 0) {    // when sensor is receiving reflected light beam
    prizm.setMotorPowers(35,35);        // turn Motors 1 and 2 on at 35% power if light beam is detected

  }

  if(prizm.readLineSensor(3) == 1) {    // when sensor detects black stripe
    prizm.setMotorPowers(125,125);      // stop motors in brake mode if black line is detected

    while(1) {                          // infinite loop – stays locked in this loop until Reset is pressed
    prizm.setRedLED(HIGH);              // flash PRIZM red LED on and off until reset
    delay(500);
    prizm.setRedLED(LOW);
    delay(500);
    }
  }
}
```

Example 11: TaskBot_Act12_Follow_A_Line

```
/* PRIZM controller example program
 * This program implements line following with the PRIZM TaskBot. Using the Line Finder Sensor
 * on sensor port D3, the robot will follow the edge of a black stripe on a white surface.
 * The program will power one motor at a time, resulting in back-and-forth forward motion to
 *  keep the robot traversing the line edge.
 *  author PWU 08/05/2016
 */

 #include <PRIZM.h>                    // include PRIZM library
 PRIZM prizm;                          // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {

 prizm.PrizmBegin();                   // initialize PRIZM

 prizm.setMotorInvert(1,1);            // invert the direction of DC Motor 1
                                       // to harmonize the direction of
                                       // opposite-facing drive motors
}

void loop() {

 // beam reflected, no line detected
 if(prizm.readLineSensor(3) == 0) {prizm.setMotorPowers(125,30); prizm.setRedLED(LOW);}

 // no reflected beam, line detected
 if(prizm.readLineSensor(3) == 1) {prizm.setMotorPowers(30,125); prizm.setRedLED(HIGH);}

}
```

Example 12: TaskBot_Act13_Drive_To_Wall

```
/*   PRIZM controller example program
 *   This program uses the Ultrasonic Sensor connected to
 *   sensor port D4 to detect an obstacle in its driving path
 *   within 25 cm. When detected, the robot will stop and wait
 *   for the object blocking its path to be cleared.
 *   author PWU 08/05/2016
 */

  #include <PRIZM.h>                    // include PRIZM library
  PRIZM prizm;                          // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {

  prizm.PrizmBegin();                   // initialize PRIZM

  prizm.setMotorInvert(1,1);            // invert the direction of DC Motor 1
                                        // to harmonize the direction of
                                        // opposite-facing drive motors
}

void loop() {

  if(prizm.readSonicSensorCM(4) > 25)
  {
    prizm.setMotorPowers(50,50);        // if distance greater than 25 cm, do this
  }
  else
  {
    prizm.setMotorPowers(125,125);      // if distance less than 25 cm, do this
  }
}
```

Example 13: TaskBot_Act14_Avoid_Obstacle

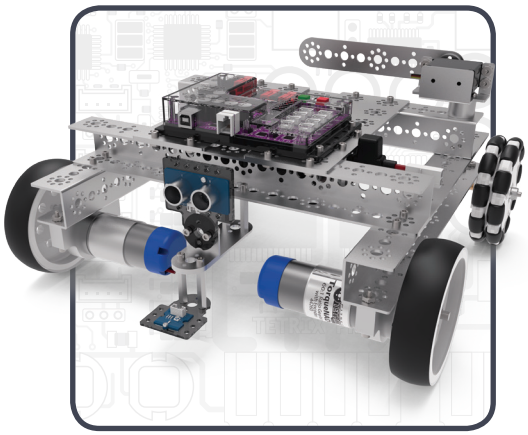
```
/*   PRIZM controller example program
 *  This program uses the Ultrasonic Sensor connected to sensor port D4 to detect objects in its
 *   driving path. When an object is detected, the robot will stop, back up, make a right turn, and continue.
 *   author PWU 08/05/2016
 */

  #include <PRIZM.h>                    // include PRIZM library
  PRIZM prizm;                          // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {

  prizm.PrizmBegin();                   // initialize PRIZM

  prizm.setMotorInvert(1,1);            // invert the direction of DC Motor 1
                                        // to harmonize the direction of
                                        // opposite-facing drive motors
}

void loop() {

  if(prizm.readSonicSensorCM(4) > 25)   // obstacle sense range set at 25 centimeters
  {
    prizm.setMotorPowers(35,35);        // forward while no obstacle detected
    prizm.setRedLED(LOW);               // turn off red LED
    prizm.setGreenLED(HIGH);            // turn on green LED
  }
  else
  {
   prizm.setGreenLED(LOW);              // turn off green LED
   prizm.setRedLED(HIGH);               // detected obstacle, turn on red LED
   prizm.setMotorPowers(125,125);       // stop, obstacle detected
   delay(500);
   prizm.setMotorPowers(-35,-35);       // back up
   delay(1000);
   prizm.setMotorPowers(125,125);       // stop
   delay(500);
   prizm.setMotorPowers(35,-35);        // make a right turn
   delay(500);

  }
}
```

Example 14: TaskBot_Act15_Combining_Sensors

```
/*  PRIZM controller example program
 *  This program uses the Line Finder and the Ultrasonic Sensors at the same time.
 *  The robot will follow the edge of a black line (stripe) on a white surface and scan for an object
 *   in its path. When an obstacle is detected, it will wait for the obstacle to be cleared and then continue.
 *   The line sensor is connected to digital port D4. The Ultrasonic Sensor is connected to
 *   digital port D4. The red LED shows the line sensor status. The green LED shows the
 *   sonic Sensor status.
 *   author PWU 08/05/2016
 */

 #include <PRIZM.h>                          // include PRIZM library
 PRIZM prizm;                                // instantiate a PRIZM object "prizm" so we can use its functions

void setup() {

 prizm.PrizmBegin();                         // initialize PRIZM
 prizm.setMotorInvert(1,1);                  // invert the direction of DC Motor 1
                                             // to harmonize the direction of
                                             // opposite-facing drive motors
 prizm.setServoSpeed(1,50);                  // set Servo 1 speed to 50
}

void loop() {
   if(prizm.readLineSensor(3) == 1) {
      prizm.setMotorPowers(30,125);          // line detected
      prizm.setRedLED(HIGH);
    }
    else
    {
      prizm.setMotorPowers(125,30);          // no line detected
      prizm.setRedLED(LOW);
    }

   while(prizm.readSonicSensorCM(4) < 25) {  // object is in path, loop here until cleared
       prizm.setGreenLED(HIGH);              // turn on green LED
       prizm.setMotorPowers(125,125);        // stop, obstacle detected
       prizm.setServoPosition(1,0);          // raise detection flag!
    }

 prizm.setGreenLED(LOW);                     // turn off green LED
 prizm.setServoPosition(1,90);               // lower flag position
}
```

PITSCO TETRIX MAX

TETRIX® PRIZM® Robotics Controller Programming Guide



Call Toll-Free
800•835•0686

Visit Us Online at
Pitsco.com

PITSCO EDUCATION