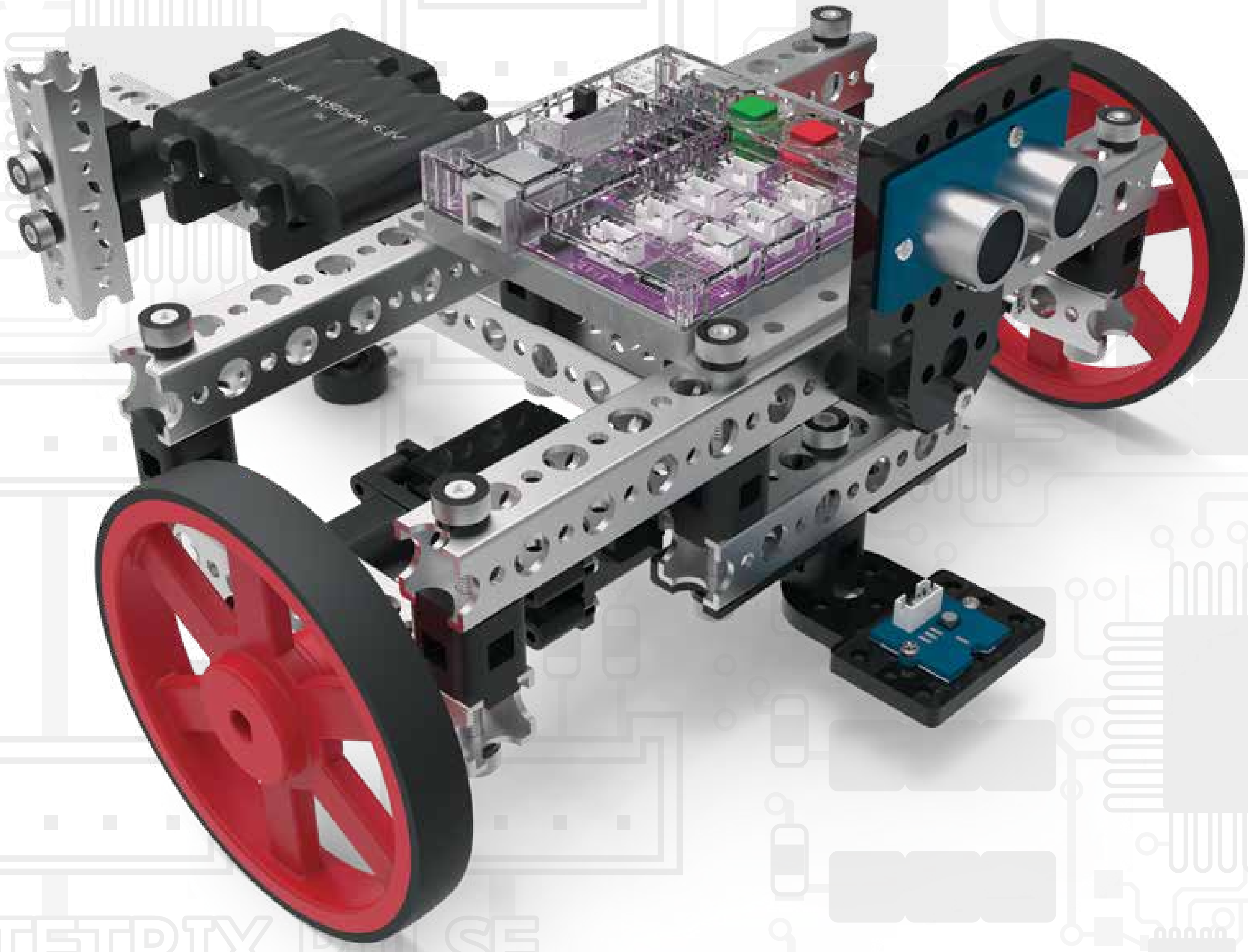
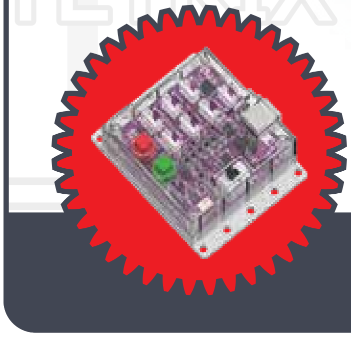
PITSCO
TETRIX®
PRIME





TETRIX® PULSE™ Robotics Controller Programming Guide

PITSCO
EDUCATION

44301

Content advising by Tammy Pankey, Paul Uttley, and Tim Lankford.

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

©2019 Pitsco, Inc., 915 E. Jefferson, Pittsburg, KS 66762

Check Pitsco.com/TETRIX-PULSE-Robotics-Controller#resources for PDF updates of this guide.

V1.2
6/19
44301

Table of Contents

Appendix

Preface

This programming guide has been developed to provide students with positive experiences in robotics engineering and programming. With the assistance of this guide, students will learn to use the TETRIX® PRIME parts to construct different robots while learning about programming with the PULSE controller. After using this guide, students should be able to use the TETRIX PRIME parts to construct a robot of their own design.

Grade Level Appropriateness

The activities used in this guide are targeted toward middle school students. With some additional instruction, upper-elementary students should be able to successfully complete the activities. Additionally, secondary teachers could use these parts to provide an exploratory experience in engineering.

Using This Guide

The activities in this guide build upon each other. The activities should be completed in the order in which they are presented. Concepts explored in one activity might not be repeated in later activities, but students could be required to understand the concepts in order to be most successful.

Safety Information

Mechanical

- Keep fingers, hair, and loose articles of clothing clear of gears and moving parts.
- Never pick up the robot while it is moving or the servo motors are running.

Electrical

- Make sure the power is turned off when the robot is not in operation.
- Do not operate the robot in a wet environment.
- Always power down the robot before making any changes.
- Use caution when working with bare wires to avoid creating a short circuit situation.
- Route wires carefully, and secure them if necessary to avoid damage to the wire or its insulation.
- Mount the battery pack securely.

Welcome to Coding with PULSE™ and Building with TETRIX PRIME!

PULSE Controller Introduction

Pitsco Education is pleased to bring you the *TETRIX® PULSE™ Robotics Controller Programming Guide* – an exciting and progressive series of activities that teaches the essentials of learning to program your TETRIX PRIME creations using the PULSE controller and the graphic-based TETRIX Ardublockly software.

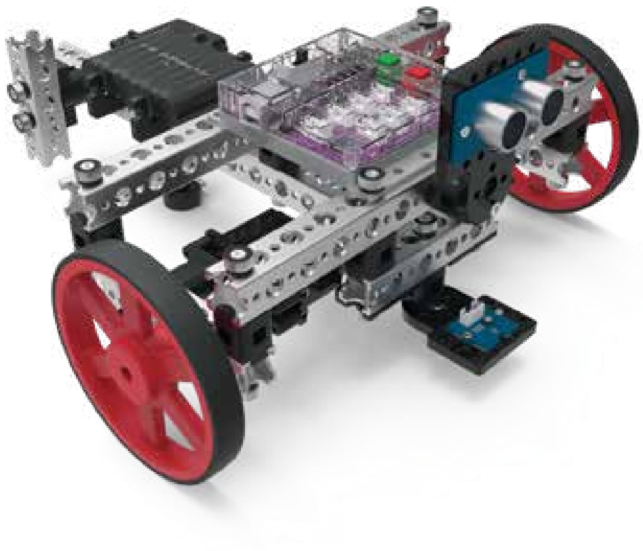
This programming guide offers a valuable tool for teaching students and teachers how to use the PULSE controller (the brain) and the TETRIX PRIME system to build and code smart, precise robots that are as real world as it gets. The guide comes with five getting started activities, step-by-step building instructions for creating a Codee Bot, 10 complete Codee Bot-oriented lessons, and extension activities. This is a great tool for exploring the functionality of the PULSE controller, TETRIX hardware components, and software. Your students are offered a great foundation to build on.

By combining the plug-and-play PULSE controller with the intuitive PRIME building system and an easy-to-use, graphic-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, PULSE 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 connections (concepts beyond the scope of this guide) that can be covered in each lesson if you choose to do so. These connections can be incorporated if you have content knowledge of these concepts or if you can work with other teachers to integrate these concepts.

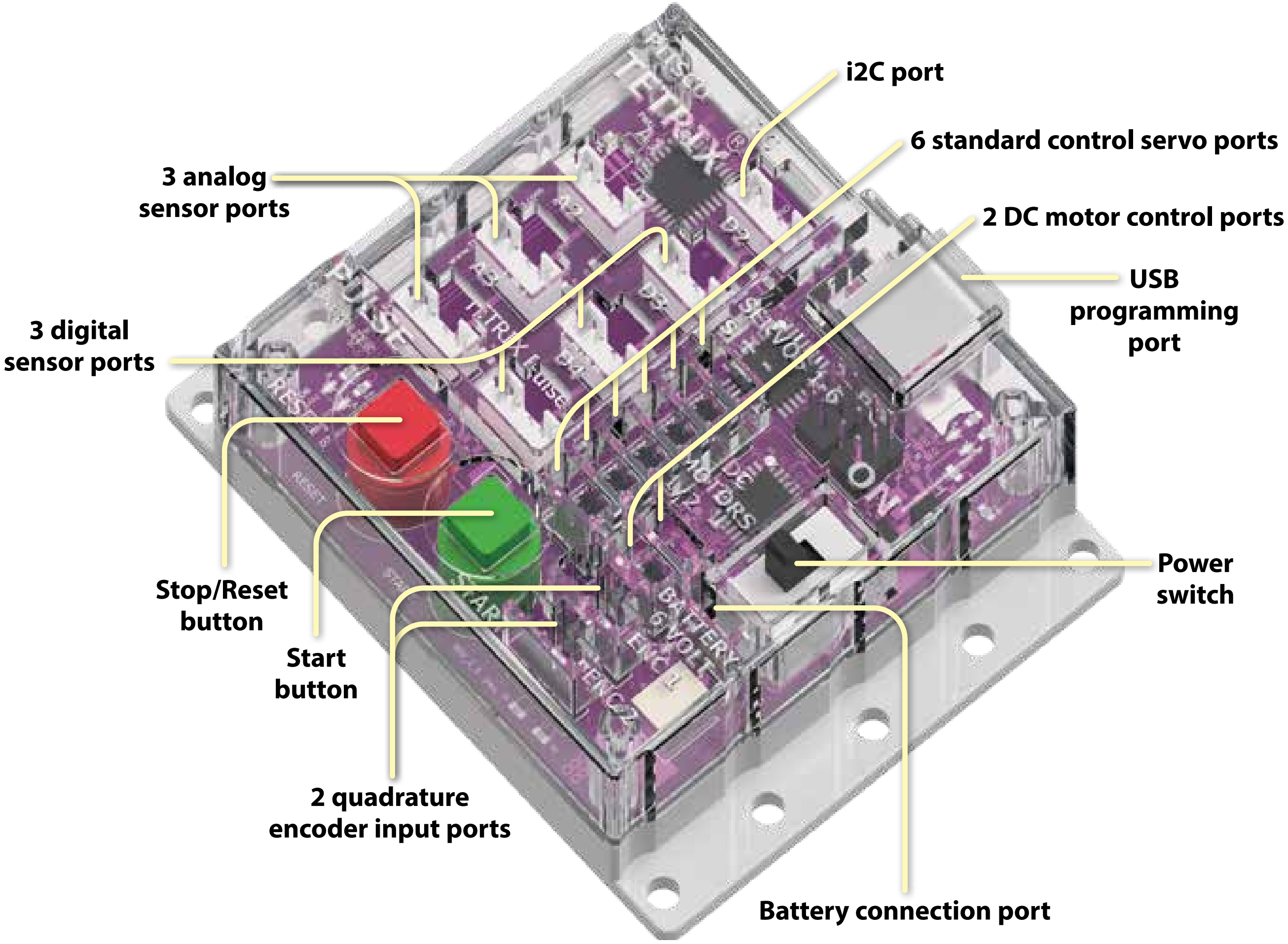
We hope this guide offers a great jumping-off point to learning with PULSE. We cannot wait to see the innovative projects and robotic creations that result.



My name is Codee! I'll follow your every command.

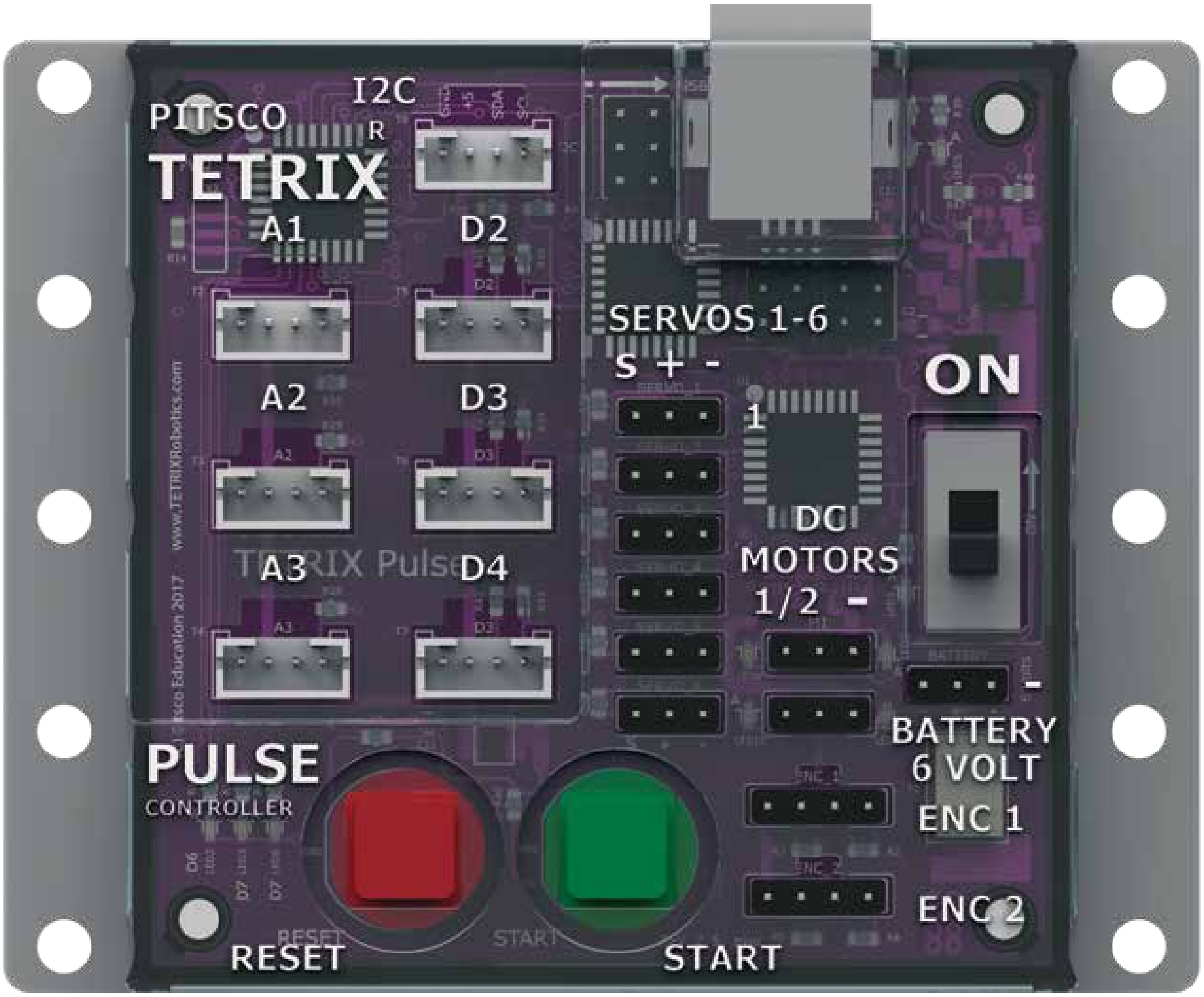
PULSE Controller Technology Overview

PULSE Robotics Controller:
A programmable device that is the brain of the TETRIX PRIME robot



Tip: For complete, detailed specifications, please refer to the TETRIX PULSE Robotics Controller Specifications located in the appendix on page 131.

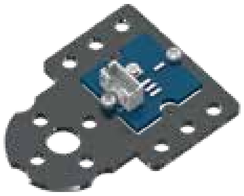
PULSE Controller Technology Overview



Sensor:

A device that detects surrounding environmental factors for the controller

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

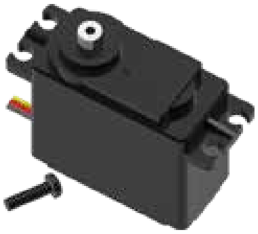


Line Finder Sensor:
Enables a robot to follow a black line on a white background or vice versa

Motor:

A machine that produces motion or power for completing work

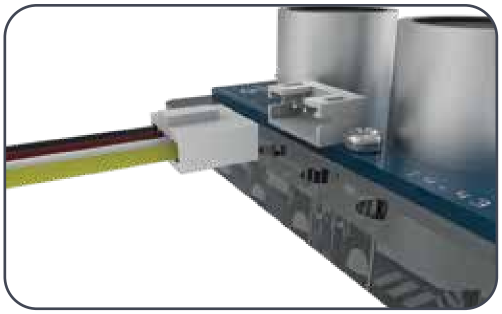
DC Motor:
Allows for speed and torque

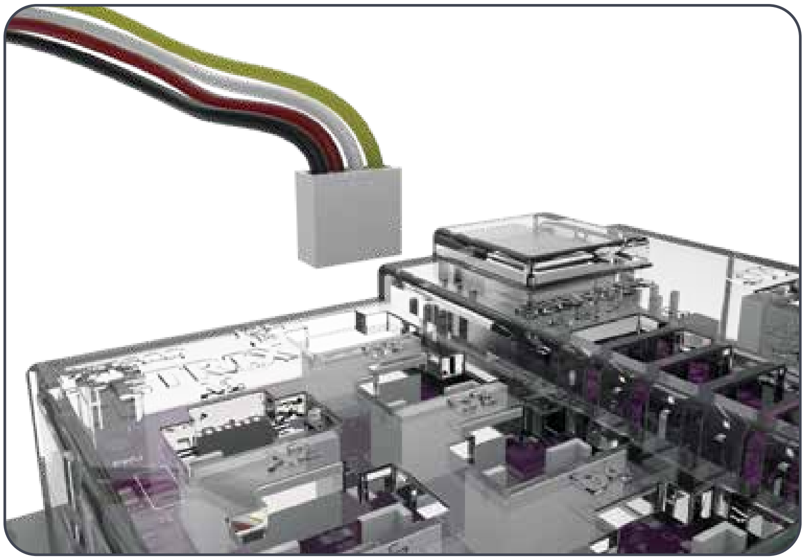


Standard Servo Motor:
Allows for exact positioning within a 180-degree range of motion

PULSE Setup

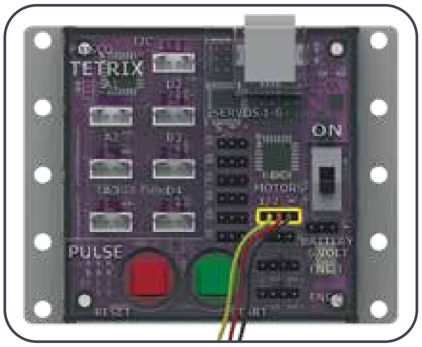
Attaching the Sensors:
To connect a sensor to the PULSE, plug the end of the sensor wire into ports labeled D2-D4 for digital sensors, A1-A3 for analog sensors, or I2C for I2C components.

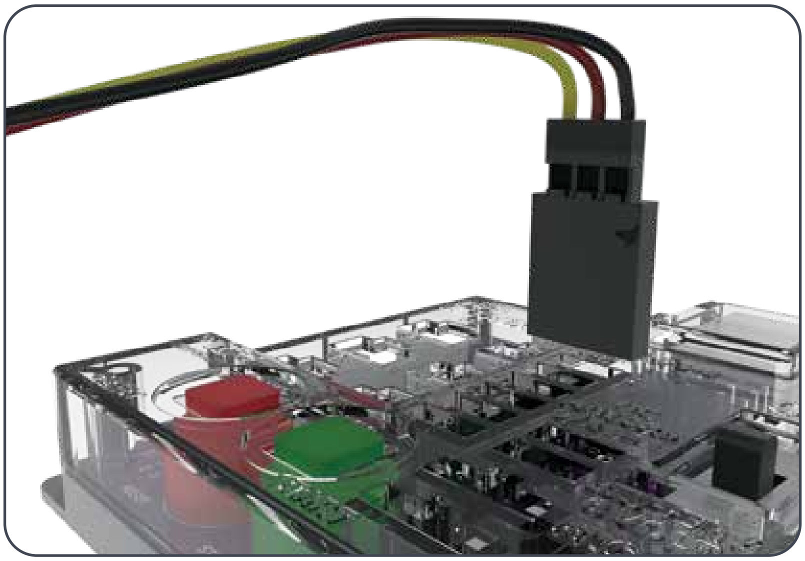




Attaching the DC Motors:
To connect a DC motor to the PULSE, plug the end of the DC motor wire into one of the two DC motor ports.

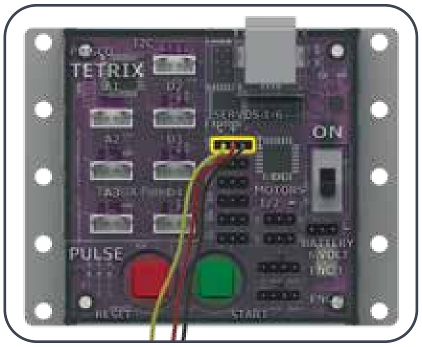
Important: The black wire end must be closest to the minus sign on the controller.

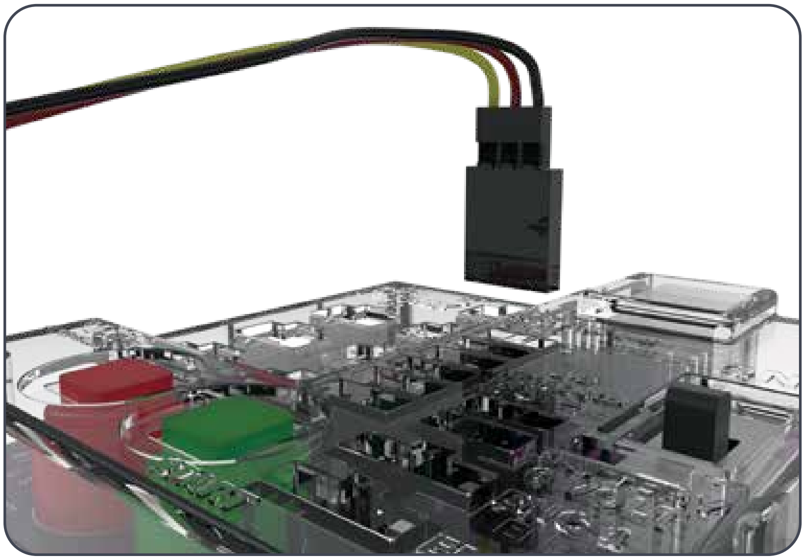




Attaching the Servo Motors:
To connect a standard servo motor to the PULSE, plug the end of the servo motor wire into one of the servo ports

Important: The black wire end must be closest to the minus sign on the controller.

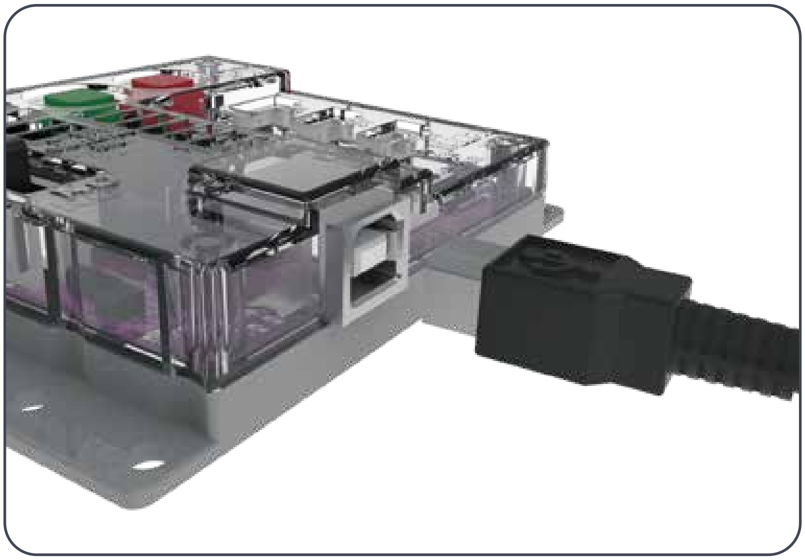




Downloading and Uploading:
The PULSE USB port is used for communication between PULSE and a Windows or Macintosh device.

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

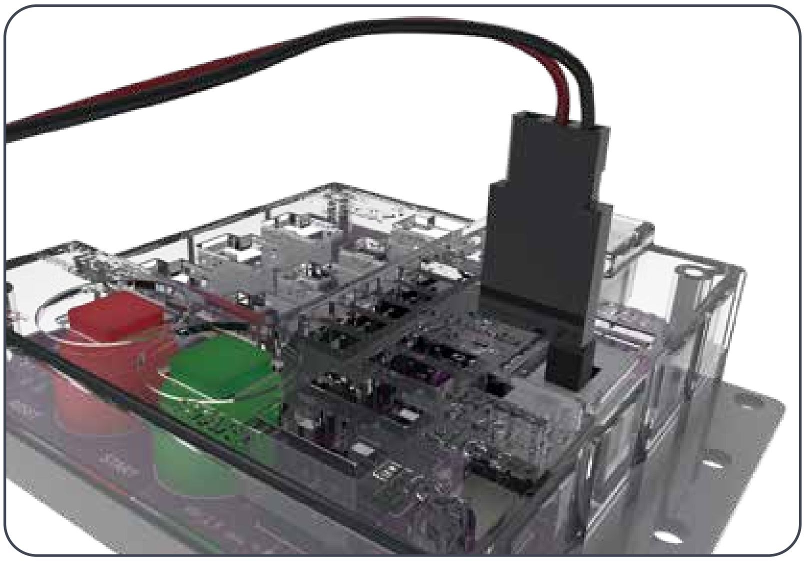
To upload a program to the PULSE, plug one end of the USB cable into the controller's USB port and plug the other end into a USB port on your device.

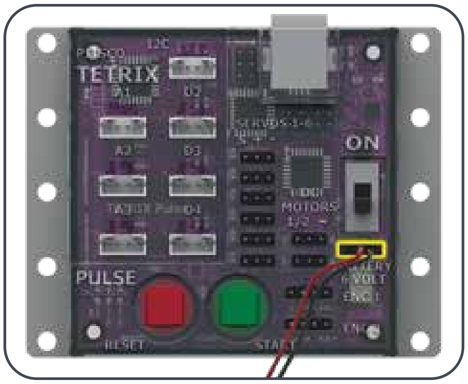


Attaching the Battery to the PULSE:
The PULSE controller is powered by a TETRIX 6-Volt Rechargeable NiMH Battery Pack.

To connect the battery pack to the PULSE, plug the end of the battery wire into the battery port located on the controller.

Important: The black wire end must be closest to the minus sign on the controller.





Warning: Do not attempt to use third-party battery packs with the PULSE 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.

Software Overview

- The TETRIX Ardublockly software is a special programming interface created by Pitsco for exclusive use with the PULSE controller. The TETRIX Ardublockly software was developed using the Google interface called Blockly.
- It can be used on a variety of Windows and Macintosh devices. The software is placed on the hard drive of a device. The device used must have a USB port for connection to the PULSE controller. It is not supported on tablets or Chromebooks.
- The Arduino Software (IDE) will be used to communicate with the PULSE in the background. You don’t need to open the Arduino Software (IDE) to create your programs. It only needs to be installed on the computer. The PULSE controller can be programmed through the Arduino Software (IDE), but for this guide, all programming will occur within the TETRIX Ardublockly software.
- Within the TETRIX Ardublockly software, a 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 TETRIX Ardublockly software as it applies to the PULSE controller. Working through examples and hands-on application of code using a small Codee Bot constructed with the TETRIX PRIME robotics building system will show you how easy it is to use TETRIX Ardublockly with PULSE.

Note: This is not meant to be a 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**.

Arduino Software (IDE) 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 and Macintosh operating systems. From the Arduino homepage, click the Software tab. On the Software page, select the download for your operating system and follow any additional instructions.

Installing the PULSE Controller 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 must add the Arduino PULSE controller library. The PULSE controller library contains special programs written for the TETRIX PULSE controller.

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

After the library has been downloaded, there are two ways to install the PULSE 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).

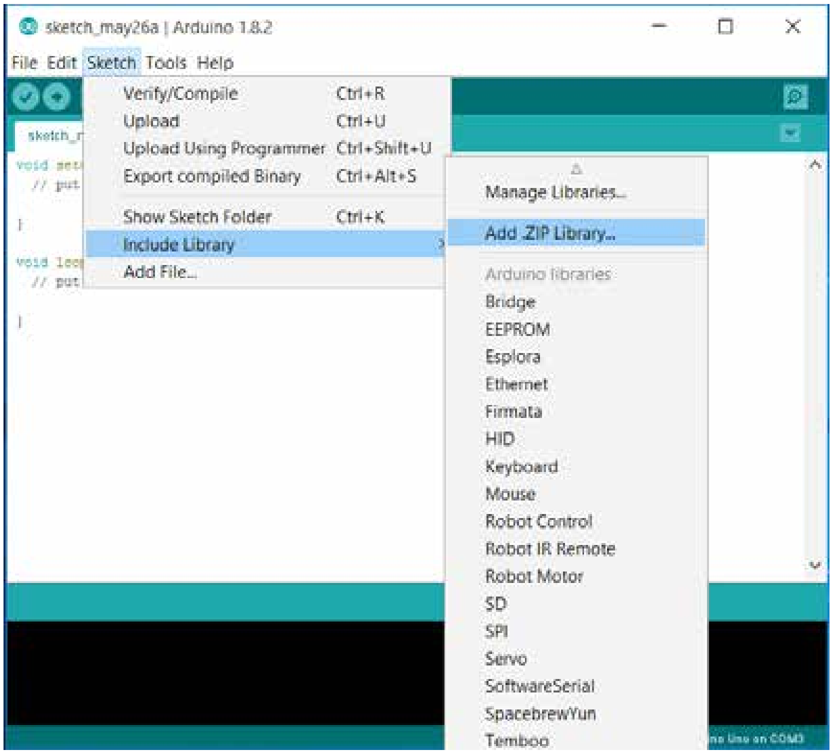


Figure 1

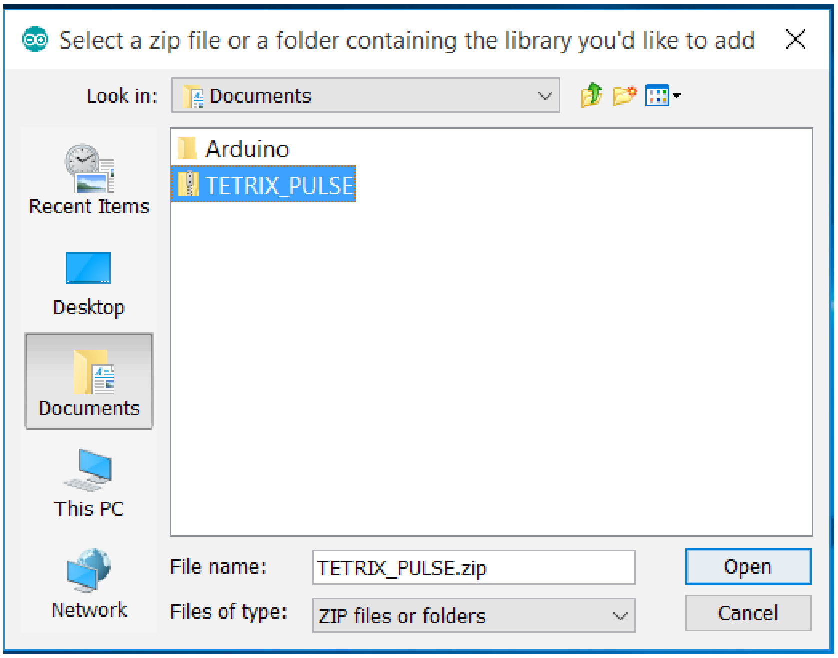
Note: All instructions and screen shots used throughout this guide are based on the 1.8.2 version of the Arduino Software (IDE). Instructions and views might slightly vary based on the platform and version you are using.

Teacher note:

- Depending on your classroom and IT situation, you might want to download and install the Arduino Software (IDE) and the TETRIX PULSE Arduino Library on the computers you and your students will be using.
- It is recommended that you organize the class into teams of two. 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.

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

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


Select a zip file or a folder containing the library you'd like to add

Look in: Documents

Arduino

TETRIX_PULSE

File name: TETRIX_PULSE.zip

Files of type: ZIP files or folders

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 PULSE library manually, first close the Arduino Software (IDE) application. Then, extract the .zip file TETRIX_PULSE.zip containing the library. After the folder is extracted, drag or copy the TETRIX_PULSE 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**.

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

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

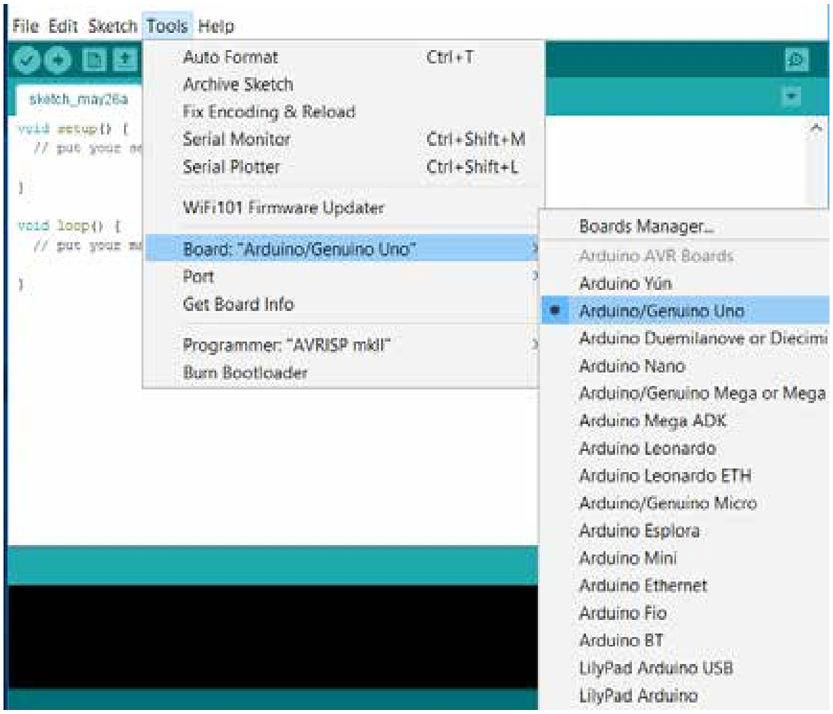
That's it! We have successfully installed the PULSE Arduino library.

Configuring USB Communication

PULSE 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 PULSE 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 PULSE controller uses the same processor chip as a genuine Arduino UNO, so this is the board you will select.



```
void setup() {
  // put your se

}

void loop() {
  // put your ma

}
```

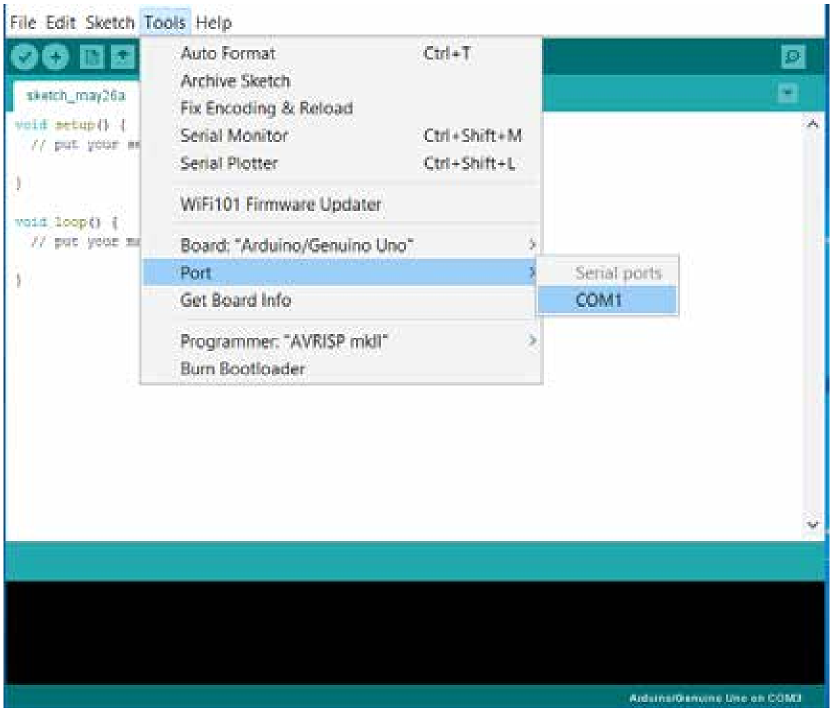
Figure 3

Next, **without the PULSE connected**, navigate to **Tools > Port** 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.

Next, plug the PULSE 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 PULSE controller time to complete the first-time connect installation. This could take 5-10 seconds. After the PULSE 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 PULSE. By selecting the new COM port, you are telling the Arduino Software (IDE) to use this port for communications. The COM port you use could be different from the one in Figure 4.


File Edit Sketch Tools Help
sketch_may26a
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino/Genuino Uno"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM1
Arduino/Genuino Uno on COM3

```
void setup() {
  // put your m

}

void loop() {
  // put your m

}
```

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

The new port that appears in this example is COM1. Select the new port item to tell the Arduino Software (IDE) to use this port for communications. Your port will likely be different, and that is OK. When the communications port has been set up, communications with the PULSE controller have been enabled.

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

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

Note: Other devices connected to the computer, such as a cell phone, might show up as a COM port as well. You might have to go back into the Arduino Software (IDE) if you have another device connected at the same time to ensure the right COM port is selected.

TETRIX Ardublockly Software Setup

The TETRIX Ardublockly software is distributed as a .zip file: TETRIX_Ardublockly.zip. There is a version for the Windows operating system and a version for the Mac operating system. The first step is to download the appropriate version for your device from the TETRIX website. You can find the library at **Pitsco.com/TETRIX-PULSE-Robotics-Controller#downloads**.

Note: This software will not work on a 32-bit Windows machine.

Place the downloaded file on the hard drive of your machine. Choose to extract the zipped folder (Figure 5).

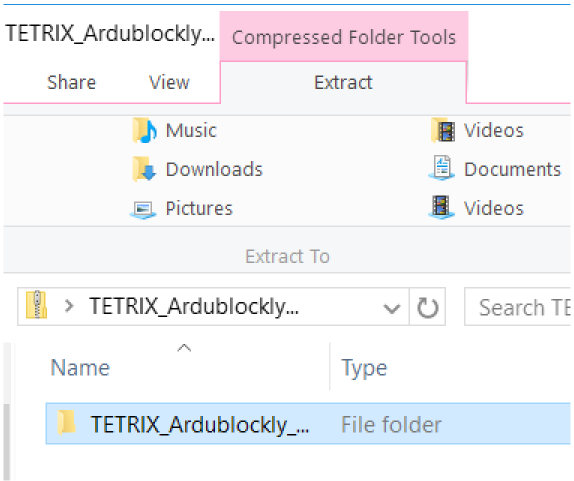
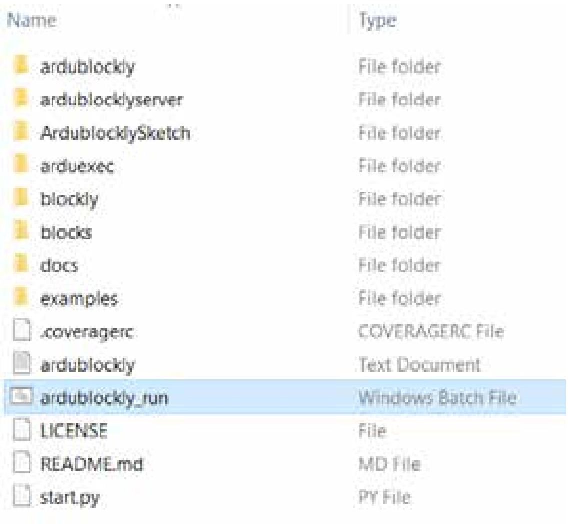


Figure 5

Locate the ardublockly_run file (Figure 6).



Name	Type
ardublockly	File folder
ardublocklyserver	File folder
ArdublocklySketch	File folder
arduexec	File folder
blockly	File folder
blocks	File folder
docs	File folder
examples	File folder
.coveragerc	COVERAGERC File
ardublockly	Text Document
ardublockly_run	Windows Batch File
LICENSE	File
README.md	MD File
start.py	PY File

Figure 6

Create a shortcut and place it on your desktop (Figure 7). Do not delete the extracted folder. It must remain on your machine to run the software.

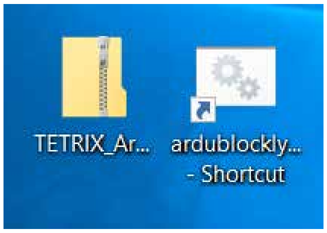


Figure 7

You will see the TETRIX Ardublockly icon show up on your desktop while it's loading the software (Figure 8).

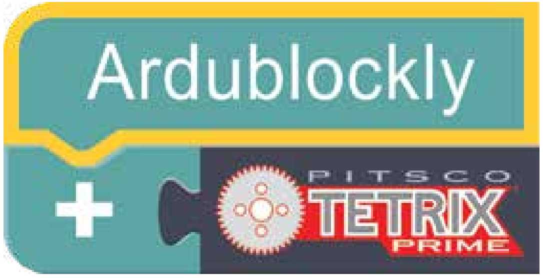


Figure 8

When the software is loaded, you should see this screen (Figure 9).

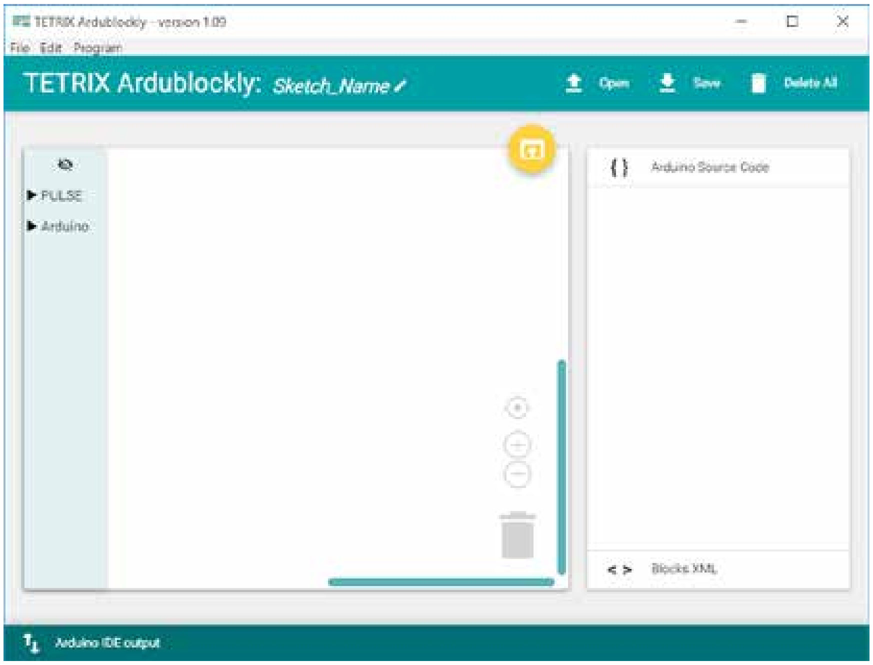


Figure 9

Navigate to **Edit** > **Preferences** (Figure 10).

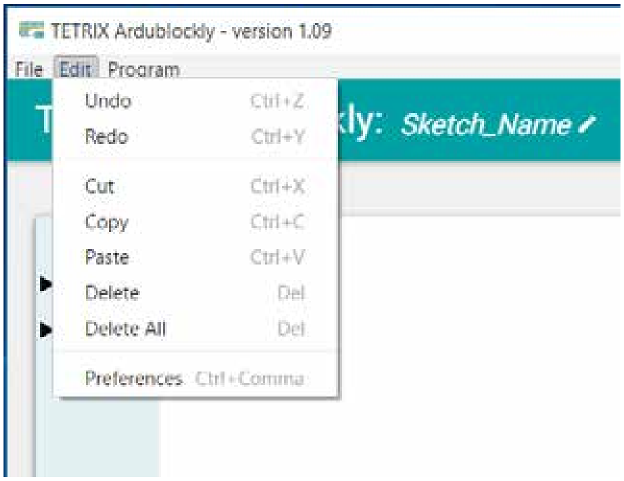
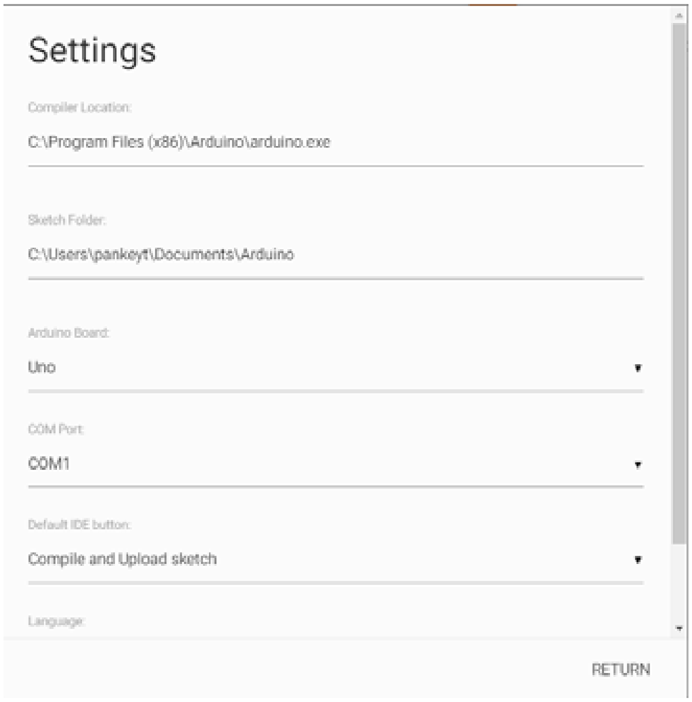


Figure 10

You will need to adjust the settings in the software. Make sure the compiler location links to where the Arduino Software (IDE) is located on your device (Figure 11).

Click below **Compiler Location**.



Settings

Compiler Location: C:\Program Files (x86)\Arduino\arduino.exe

Sketch Folder: C:\Users\pankeyt\Documents\Arduino

Arduino Board: Uno

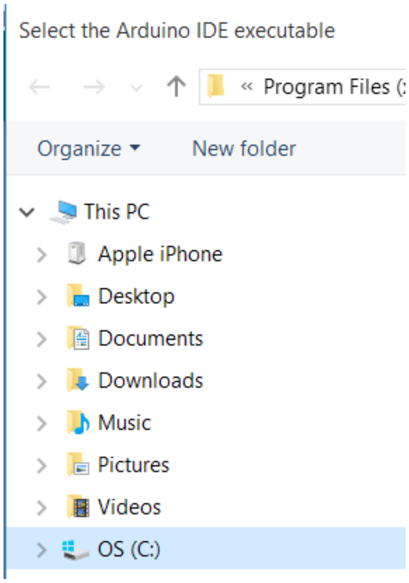
COM Port: COM1

Default IDE button: Compile and Upload sketch

Language:

Figure 11

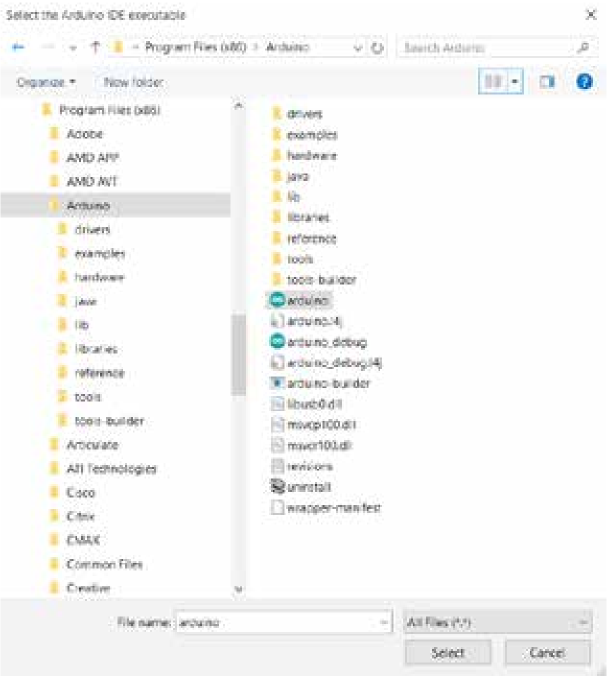
Choose your operating system folder (Figure 12).



Select the Arduino IDE executable

Figure 12

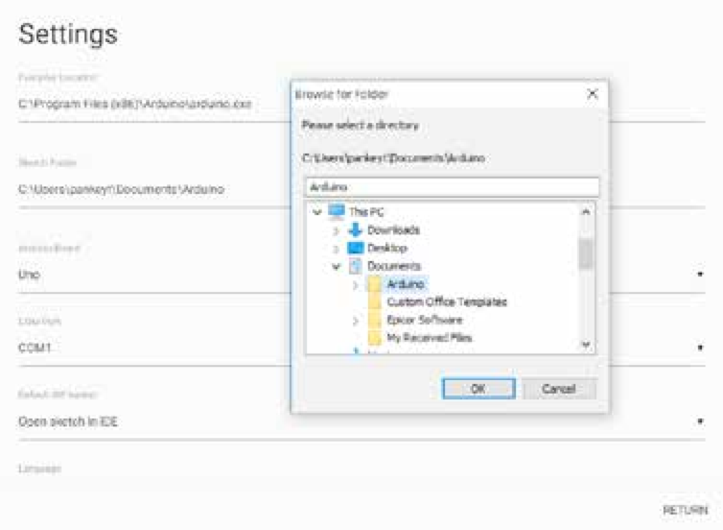
Find the Arduino program files. Select the Arduino application (Figure 13).



File name: arduino

Figure 13

Click below **Sketch Folder**. Choose which location the sketch folder will save to. You can save to your Documents folder or another location on your device (Figure 14).



C:\Program Files (x86)\Arduino\arduino.exe

C:\Users\pankeyt\Documents\Arduino

Uno

COM1

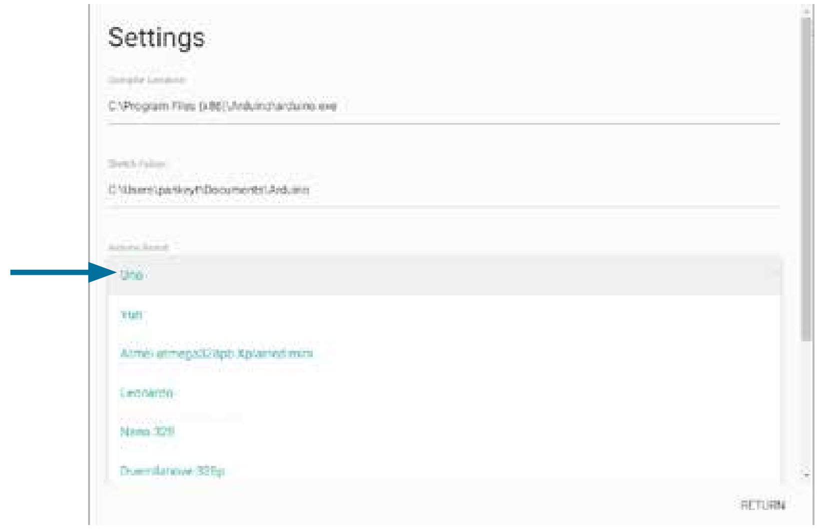
Open sketch in IDE

C:\Users\pankeyt\Documents\Arduino

Arduino

Figure 14

The Arduino board should be Uno (Figure 15).



Settings

Compiler Location

C:\Program Files (x86)\Arduino\arduino.exe

Sketch Folder

C:\Users\parkeyt\Documents\Arduino

Arduino Board

Uno

Yun

Atmel atmega328pb Xplained mini

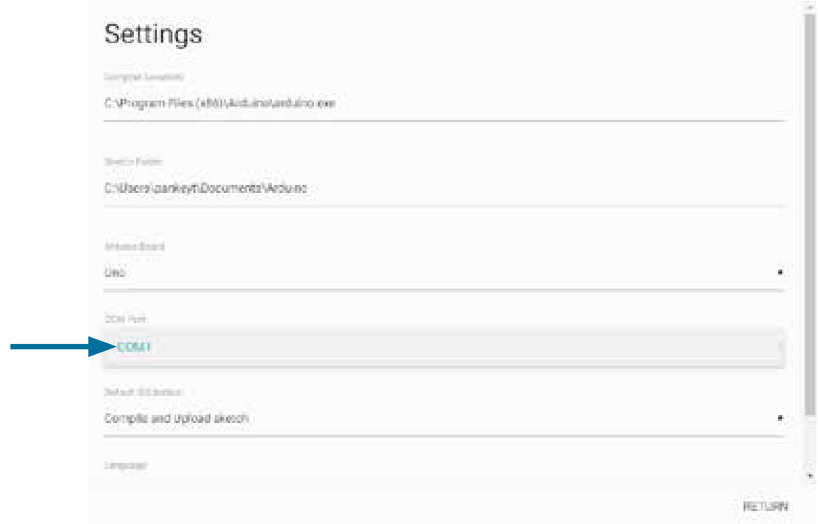
Leonardo

Nano 328

Duemilanove 328p

Figure 15

Adjust the COM port to whichever port the PULSE controller is associated with (Figure 16).



Settings

Compiler Location

C:\Program Files (x86)\Arduino\arduino.exe

Sketch Folder

C:\Users\parkeyt\Documents\Arduino

Arduino Board

Uno

COM Port

COM1

Default IDE button

Compile and Upload sketch

Language

Figure 16

When this step is complete, your PULSE controller is connected to the TETRIX Ardublockly software and coding can begin.

TETRIX Ardublockly Introduction

Software Basics:

This is the main interface you will use when you open the software. On the far-left side is the tool palette. To hide the tool palette, click the eye icon on the top of the bar. This tool palette contains two different types of blocks. These are blocks that were specifically created for use with the PULSE controller. The Arduino blocks can also be used for additional functions and programming.

In the middle is the programming space. This is where you will place your blocks to create your program. On the bottom right of the programming space, there is a trash can where you can place blocks to delete them. Also, the plus and minus signs allow you to zoom in and out of the programming space. The bull's-eye will center the programming space to where your blocks are placed.

On the top right of the programming space, there are three buttons. One button is Open Sketch, which allows you to open your sketch in the Arduino Software (IDE). Your program will be opened in syntax format. You can edit the syntax code within Arduino Software (IDE) if desired. This process will not be covered in this guide. The Verify button allows you to check your sketch to ensure that your program has no errors. The final button is Upload. This is the button that transfers your program to the PULSE controller (Figure 17).

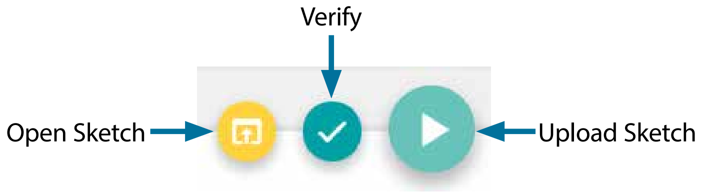


Figure 17

Note: These buttons will change order to show the last button action chosen.

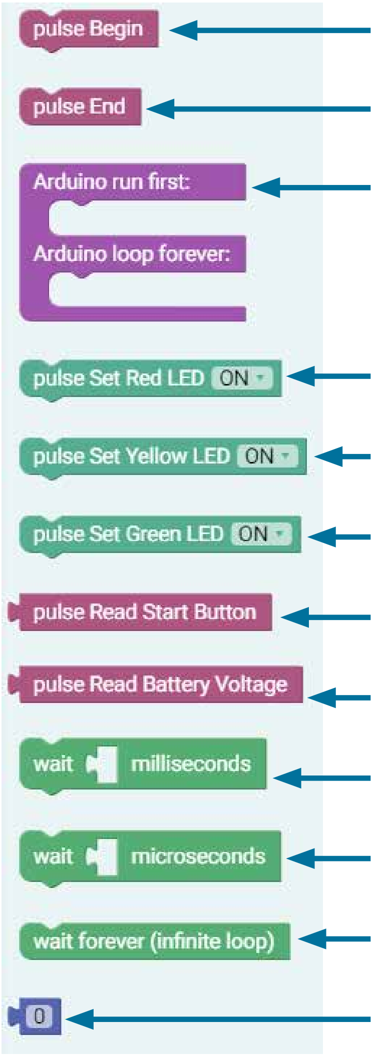
The far-right side shows the text-based Arduino source code. You can't edit the source code. You can hide the source code bar if desired.

On the very bottom of the screen, there is the Arduino IDE output command bar. It defaults to being hidden. If you click anywhere along the bottom bar, it will open the status bar. This provides real-time information on the status of an upload.

To name your program, click where it says **Sketch_Name** and type in your file name. You must click **Save** to keep a copy of the program you create. If you are opening an existing file, you can click **Open**.

PULSE Blocks:

Control



Initializes the PULSE Robotics Controller

Immediately terminates the program

Defines the Arduino setup() and loop() functions

Sets the state of the PULSE red LED

Sets the state of the PULSE yellow LED

Sets the state of the PULSE green LED

Reads the state of the PULSE Start button

Reads the battery pack voltage

Waits for a specific time in milliseconds

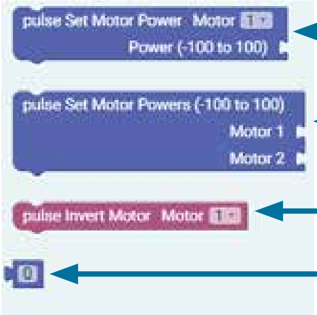
Waits for a specific time in microseconds

Waits indefinitely, infinite loop

Time element (mini block that is attached to other blocks and edited)

Figure 18

Motors


pulse Set Motor Power Motor 1
Power (-100 to 100)
pulse Set Motor Powers (-100 to 100)
Motor 1
Motor 2
pulse Invert Motor Motor 1
0

Sets the power and direction of the DC motor; the power range is -100 to 100

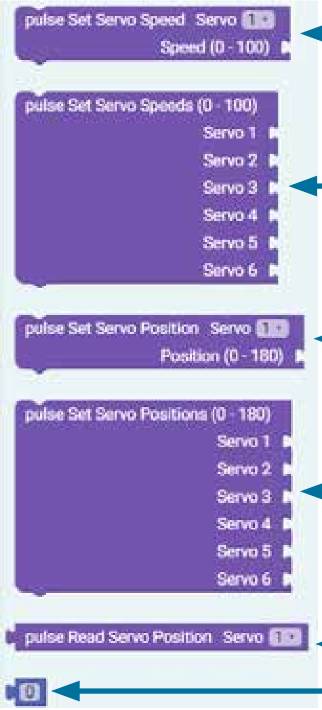
Sets the power and direction of DC Motors 1 and 2

Inverts the rotational direction of the selected motor channel

Time element

Figure 19

Servos


pulse Set Servo Speed Servo 1
Speed (0 - 100)
pulse Set Servo Speeds (0 - 100)
Servo 1
Servo 2
Servo 3
Servo 4
Servo 5
Servo 6
pulse Set Servo Position Servo 1
Position (0 - 180)
pulse Set Servo Positions (0 - 180)
Servo 1
Servo 2
Servo 3
Servo 4
Servo 5
Servo 6
pulse Read Servo Position Servo 1
0

Sets the speed of the selected servo motor channel

Sets the speeds of all six servo motor channels

Sets the position of the selected servo to a position between 0 and 180 degrees

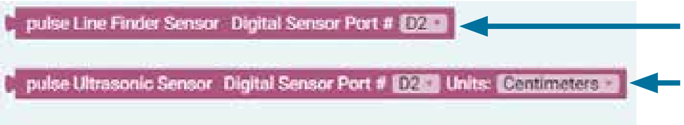
Sets the positions of all six servo motor channels

Reads the position of the selected servo and stores it in a variable

Time element

Figure 20

Sensors


pulse Line Finder Sensor Digital Sensor Port # D2
pulse Ultrasonic Sensor Digital Sensor Port # D2 Units: Centimeters

Reads the output value of the Line Finder Sensor

Reads the output value of the Ultrasonic Sensor in centimeters or inches

Figure 21

Tip: There are a variety of additional blocks that perform specific functions in the Arduino drop-down menu.

Troubleshooting Tips:

- Your sketch always has to start with the pulse Begin block. If you want to end a program, you can use the pulse End block or press the red Stop/Reset button on the controller.
- If your sketch won't load onto the PULSE controller, try disconnecting the USB cable and reconnecting it. You can also try closing the software and opening it again.
- Open your sketch in the Arduino IDE output window to see if you have coding errors.
- Remember that you can use the resources in the appendix if you need additional information.
- Want to see it in action? You can by watching the RoboBench video series for the *PULSE Programming Guide*. You can find the entire series at **video.pitsco.com/TETRIX** or on the Pitsco YouTube channel.

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 TETRIX Ardublockly software and how it works with the PULSE and select basic hardware. Success with these first five activities will demonstrate how easy it is to use TETRIX Ardublockly software with the PULSE and prepare you for coding the PULSE Codee Bot.

The TETRIX Ardublockly Sketch

As you begin creating your 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 block within the sketch.

Each block also represents lines of text known as code, which is why programming is sometimes called coding. Text-based programming is also known as syntax programming. TETRIX Ardublockly uses visual programming, also known as graphic programming.

Many times, the best way to learn how to code is by following an example. In the following activities, you will work through several coding examples to better learn how to create sketches and upload them to the PULSE controller.

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 132-138. More library functions might be added in the future as the library is updated to newer versions.

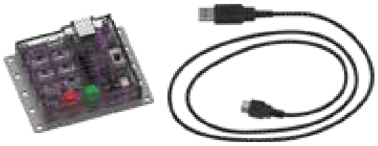
Note: In addition to the PULSE library, there is an entire collection of Arduino language commands that are necessary to understand before we can create functional programs. The Arduino language reference as well as numerous language learning tutorials can be found by visiting the Arduino homepage at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

Activity 1: Hello World!

Introduction

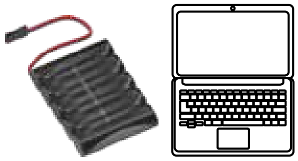
You will create a simple program, or sketch, that will blink the red LED on the PULSE controller. Think of the controller as if it's winking at you! This activity is the equivalent of a Hello World! program, which is usually the intro activity for any new programmer. The sketch you will create is a simple and basic PULSE code. All you need is the PULSE controller, a power source, and a USB connection to the computer.

Activity 1 Parts Needed



Electronics & Control

Part No.	Part Name	Quantity
44268	TETRIX PULSE Robotics Controller with USB cable	1



Batteries & Hardware

Part No.	Part Name	Quantity
40235	TETRIX PRIME 6 V NiMH Battery Pack	1
	Computer	1

Open the Program

Let's start by looking at the first example sketch. Open the sketch by selecting **Examples > GS_Activity_1**. A new sketch window will open titled GS_Activity_1 (Figure 22).

Tip: Can't find the program? Check the Arduino Software (IDE) Setup section on pages 10-11.

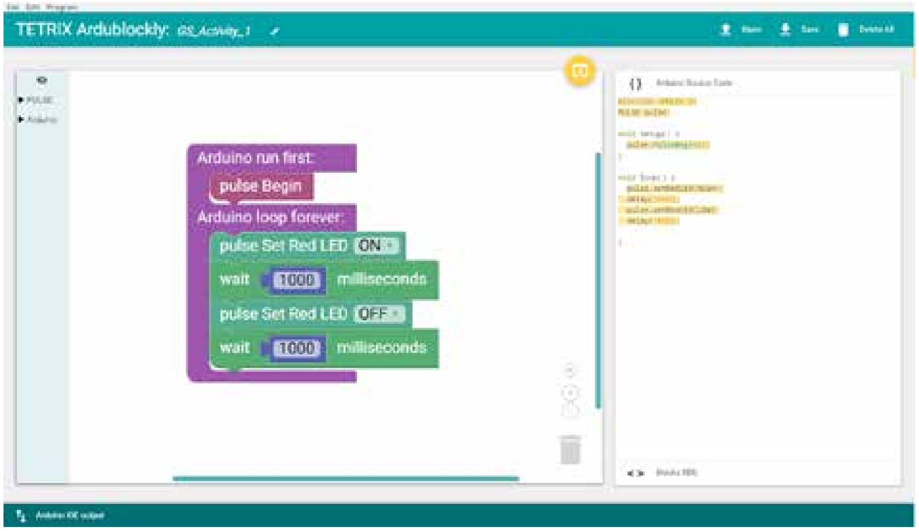


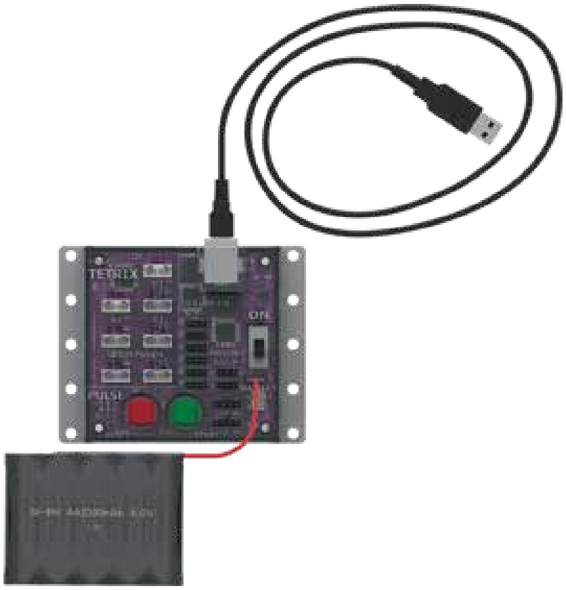
Figure 22

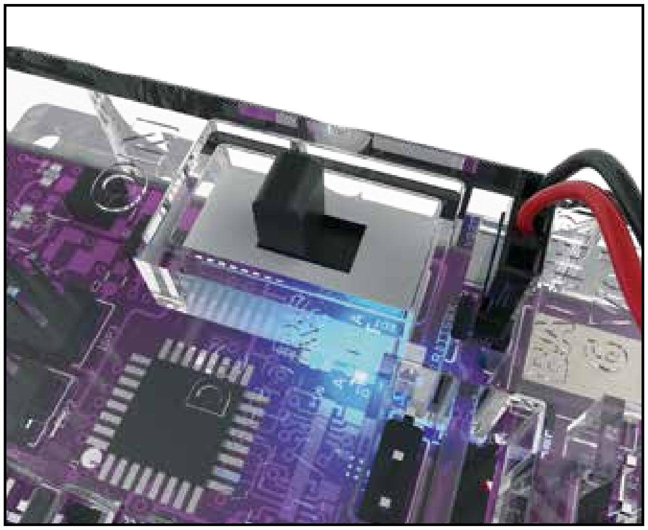
Background

Before you can upload the sketch to the PULSE, you need to make sure the PULSE has power, is connected to the computer, and is detected by the computer.

When the PULSE is connected as shown, turn on the PULSE with the on/off switch. You will know the PULSE has power by the glowing blue light.

Tip: To see if the PULSE is detected by the computer, check the port as you did in the Configuring USB Communication section on pages 12-14.





Execute the Code

To upload the sketch to the PULSE, click **Upload Sketch** (Figure 23). To check the status of the upload, click **Arduino IDE output** on the bottom of the program (Figure 24).

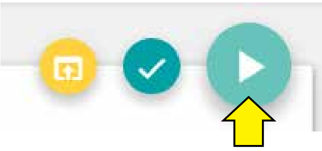


Figure 23

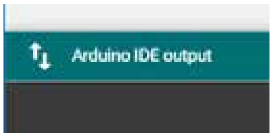


Figure 24

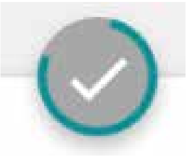


Figure 25

Watch the Upload Sketch button. A colored line will spin around the circle while the upload is in progress (Figure 25). Be patient!

Tip: Wait, why is the time in milliseconds? Programmers use milliseconds for more precise timing. 1,000 milliseconds equal 1 second.

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

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

Further Investigate

Look at the right side of the program. You see text with different punctuation. This is called syntax, or text-based programming. Each block in the sketch is typically represented by one line of text. Lines of text within a sketch are also known as code, which is why programming is sometimes called coding.

You can change some parameters in the sketch to see how they affect the behavior of the red LED. The wait block determines how long the LED will be on and off (Figure 26). This is a parameter you can change in your sketch. Experiment with changing those values to create new blinking behaviors for the LED. Try making the LED blink faster or slower.

Tip: The time block is located under the Motors category in the software.

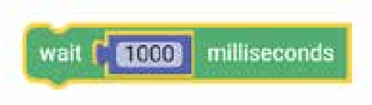


Figure 26

Every program starts with a setup-loop block. Under the setup portion of the block, the pulse Begin block always comes first. Any other blocks that need to be set up only once are placed here. Blocks that will occur in a continuous loop in the program are placed in the loop portion of the block.

In this example, the turning on and off of the red LED occurs over and over because it is in a loop. Only terminating the program stops the program loop (Figure 27).

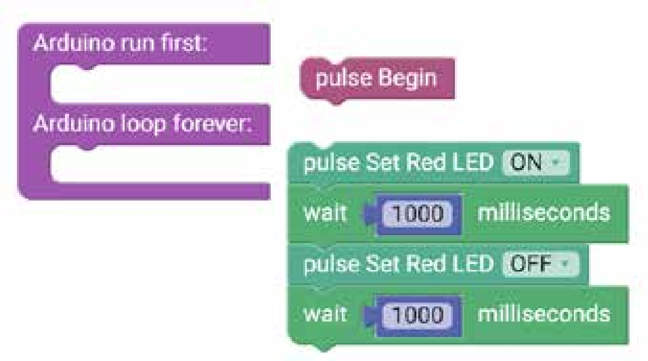


Figure 27

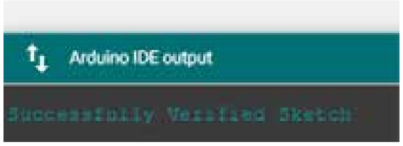
Extension Activity

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 can be used for signaling or long-distance communication. Challenge yourself to create a sequence of blinking LED lights like a stoplight.

To start a new sketch, select **File > New**. Place the appropriate blocks into the sketch. When you create your own sketch, there is a built-in software tool to help ensure your code is free of errors. You can check your program by clicking **Verify the Sketch** (Figure 28).

This will cause the code to compile but not upload. If there are errors in the code, they will be displayed in the compiler error window at the bottom of the sketch window (Figure 29). Errors will need to be corrected before code can be uploaded to the PULSE controller.



```
Successfully Verified Sketch
```

Figure 28

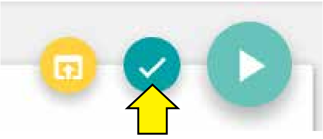


Figure 29

If there are no errors, the compiler will complete and indicate that it is done compiling, and you can upload your code. Click the **Upload the Sketch** button on the sketch and see if you were able to simulate a stoplight.

Tip: To solve this extension activity, you can find a sample sketch in the appendix titled **GS_Activity_1_Extension_Example**.

Real-World Link

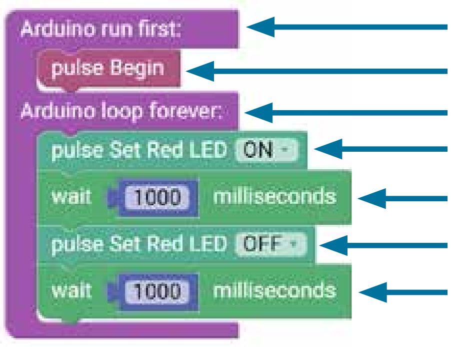
Think about the world around you. What things use blinking or flashing lights? There are many examples such as traffic lights, holiday lights, and police lights. They can serve as a warning or draw attention.

Careers: traffic technician, traffic light engineer, electronic engineer

STEM Connections

- Science
    - o Electricity
    - o LED lighting
- Technology
    - o Electronic design
    - o Traffic light programming
- Engineering
    - o Lighting types
    - o Optics
- Math
    - o Frequency
    - o Pattern

Block-Text Correlation



```
void setup() {
pulse.PulseBegin();
void loop() {
pulse.setRedLED(HIGH);
delay(1000);
pulse.setRedLED(LOW);
delay(1000);
```

Arduino Source Code

```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();
}

void loop() {
pulse.setRedLED(HIGH);
delay(1000);
pulse.setRedLED(LOW);
delay(1000);

}
```

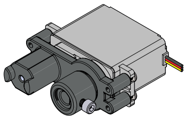
Note: Notice that On means High and Off means Low.

Activity 2: Moving Your DC Motors

Introduction

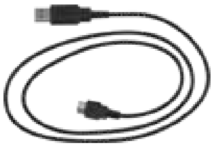
For your second activity, you will add an element of motion. You will create a sketch that will rotate a DC motor.

Activity 2 Parts Needed



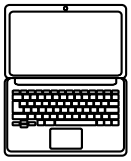
Wheels, Gears, & Servos

Part No.	Part Name	Quantity
40379	DC Motor 44298 with Servo Mounting Bracket 40232 (Assembly instructions on pages 54-55)	1



Electronics & Control

Part No.	Part Name	Quantity
44268	TETRIX PULSE Robotics Controller with USB cable	1

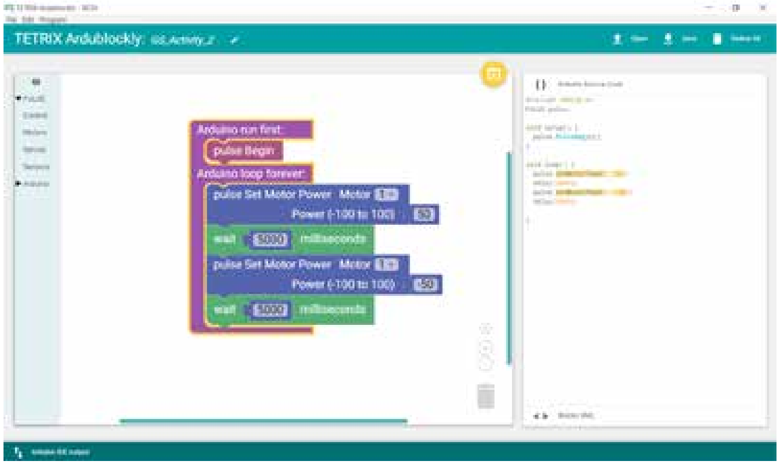


Batteries & Hardware

Part No.	Part Name	Quantity
40235	TETRIX PRIME 6 V NiMH Battery Pack	1
	Computer	1

Open the Program

Before you open your next example sketch, be sure to save any sketch you want to reference later. Let's start by looking at the example sketch. Open the sketch by selecting **Examples > GS_Activity_2**. A new sketch window will open titled GS_ Activity_2 (Figure 30).



```
Arduino run first:
  pulse Begin
Arduino loop forever:
  pulse Set Motor Power  Motor 1
                         Power (-100 to 100) 50
  wait 5000 milliseconds
  pulse Set Motor Power  Motor 1
                         Power (-100 to 100) -50
  wait 5000 milliseconds
```

Figure 30

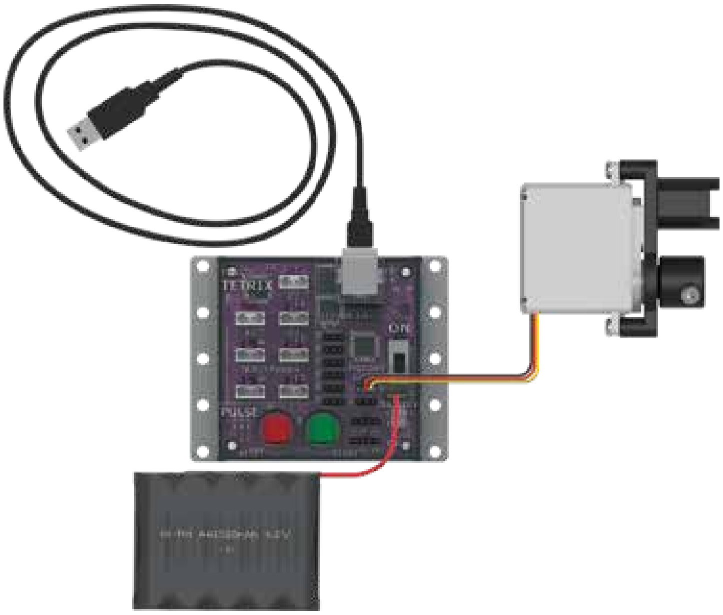
Background

The intent of this sketch is to spin a DC motor for five seconds and then stop. Then, the motor will spin in the opposite direction for five seconds. The motor will operate at half power. This behavior will continue until the Stop/Reset button is pressed.

Tip: What's the difference between a DC motor and a servo motor? A DC motor has two wires and can rotate continuously. A TETRIX servo motor has three wires and can be placed into different positions but can't rotate beyond 180 degrees.

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your connections. Keep in mind that you added a new connection with the motor. Plug the end of the wire that extends from the DC motor into the DC Motor 1 port.

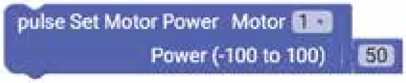


Upload the sketch. The green LED will light up, indicating that the code is ready to execute. When this has happened, press the green Start button on the PULSE controller.

Observe the direction and duration of the motor rotation. Did the motor's behavior match the intended program? Press the Stop/Reset button when you are ready to stop the motor.

Further Investigate

The pulse Set Motor Power blocks allow you to adjust the percentage of power the motor has. It can range from 0 (no power) to 100 (full power). If you wanted half power, you would set the power to 50 (Figure 31).



```
pulse Set Motor Power  Motor 1 ▾
              Power (-100 to 100)  50
```

Figure 31

When the program is running, you will see a red light by the motor cord when the motor is going forward, or clockwise. Within the sketch, this motor direction is represented by a positive value (Figure 32).

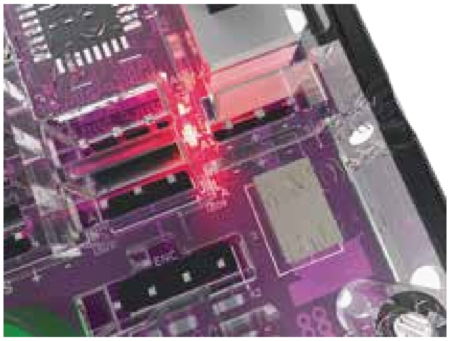


Figure 32

When the program is running, you will see a green light by the motor cord when the motor is going backward, or counterclockwise. Within the sketch, this motor direction is represented by a negative value (Figure 33).

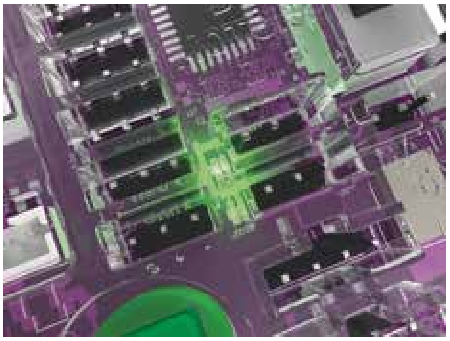


Figure 33

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

Extension Activity

With the example as a reference, try creating a new sketch using your DC motor and LEDs on the controller. Remember what you learned from your first activity and come up with a creative way to include blinking LEDs with your rotating motor.

Tip: To solve this extension activity, you can find a sample sketch in the appendix titled GS_Activity_2_Extension_Example.

Real-World Link

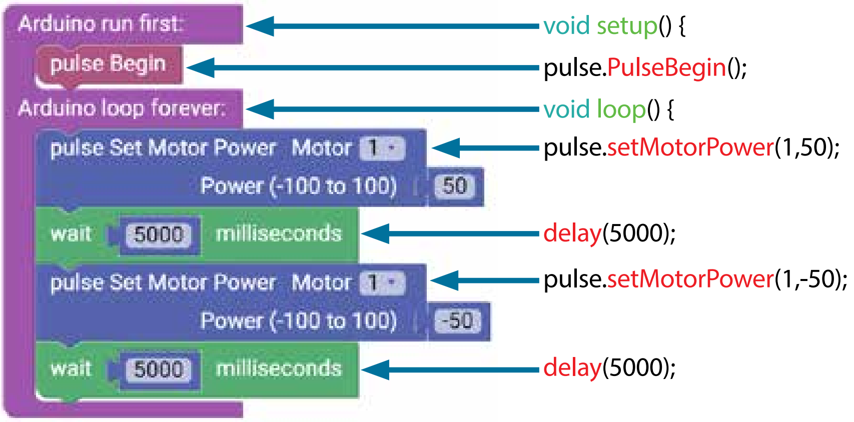
You can find DC motors in many places. They are in elevators, trains, machinery, power tools, cars, and fans. They are often used to power different electronics.

Careers: small-engine mechanic, mechanical engineer, machinery maintenance worker

STEM Connections

- Science
  - o Direct current
  - o Power
- Technology
  - o DC motor use
  - o Torque
- Engineering
  - o Motor wiring
  - o Ground
- Math
  - o Revolutions per minute (rpm)
  - o Degrees

Block-Text Correlation



```
void setup() {
pulse.PulseBegin();
void loop() {
pulse.setMotorPower(1,50);
delay(5000);
pulse.setMotorPower(1,-50);
delay(5000);
```

Note: In the parentheses, (1,50) means Motor 1 will spin clockwise at a power of 50.

Arduino Source Code

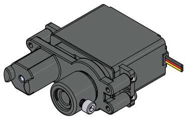
```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();
}

void loop() {
pulse.setMotorPower(1,50);
delay(5000);
pulse.setMotorPower(1,-50);
delay(5000);

}
```

Activity 3: Moving Your Servo Motors

Introduction

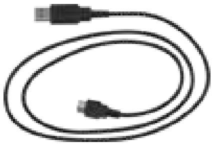
In the third activity, you will create a sketch to rotate a servo motor. Servo motors allow movement to a set position regardless of the start position. The TETRIX servo motors have a limited range of motion from 0 to 180°. For example, you can tell a servo to go to position 45° regardless of where it starts. If it starts at 0°, it will move clockwise to 45°. If it starts at 120°, it will move counterclockwise to 45°.

Activity 3 Parts Needed



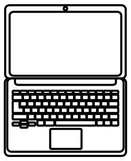
Wheels, Gears, & Servos

Part No.	Part Name	Quantity
40379	Standard-Scale Servo Motor 40538 with Servo Mounting Bracket 40232 (Assembly instructions on pages 54-55)	1



Electronics & Control

Part No.	Part Name	Quantity
44268	TETRIX PULSE Robotics Controller with USB cable	1

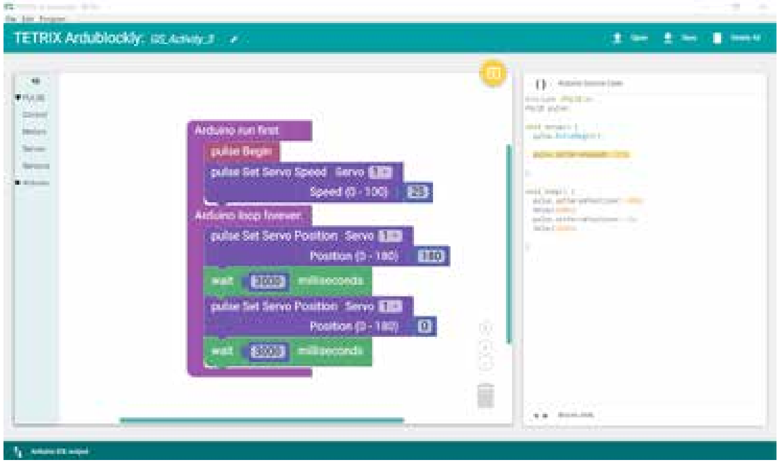


Batteries & Hardware

Part No.	Part Name	Quantity
40235	TETRIX PRIME 6 V NiMH Battery Pack	1
	Computer	1

Open the Program

Before you open your next example sketch, be sure to save any sketch you want to reference later. Let's start by looking at the example sketch. Open the sketch by selecting **Examples > GS_Activity_3**. A new sketch window will open titled GS_Activity_3 (Figure 34).



```
Arduino run first
  pulse Begin
  pulse Set Servo Speed  Servo 1
                         Speed (0 - 100) 25
Arduino loop forever
  pulse Set Servo Position  Servo 1
                            Position (0 - 180) 180
  wait 3000 milliseconds
  pulse Set Servo Position  Servo 1
                            Position (0 - 180) 0
  wait 3000 milliseconds
```

Figure 34

Background

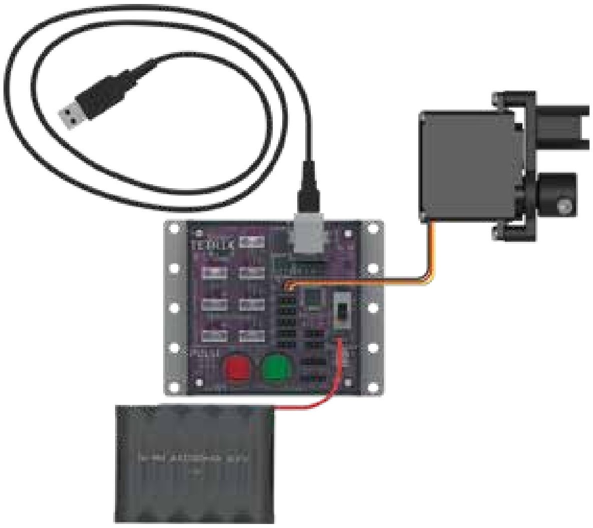
In this third sketch, you will spin a servo motor back and forth between two different positions at a set speed. The motor will operate at 25% speed. This behavior will continue until the Stop/Reset button is pressed.

Servo motors allow for a lot more precision in movement than DC motors do.

Tip: What's the difference between a DC motor and a servo motor? A DC motor has two wires and can rotate continuously. A TETRIX servo motor has three wires and can be placed into different positions but can't rotate beyond 180 degrees.

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your connections. Keep in mind that you added a new connection with the motor. Plug the end of the wire that extends from a servo motor into the Servo 1 port.



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 PULSE controller.

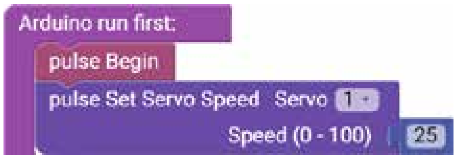
Observe the direction and duration of the servo motor rotation. Did the motor's behavior match the intended program? Press the Stop/Reset button when you are ready to stop the motor.

Further Investigate

In this sketch, you will use two new PULSE blocks: pulse Set Servo Speed and pulse Set Servo Position. Both blocks have two parameters, but they are different.

The two parameters of the pulse Set Servo Speed block are servo channel and servo speed.

In the example, Servo 1 will spin at 25% power while it rotates to the position commanded by the pulse Set Servo Position block. This block is in the setup portion of the setup-loop block because it needs to be listed once at the beginning of the program (Figure 35).

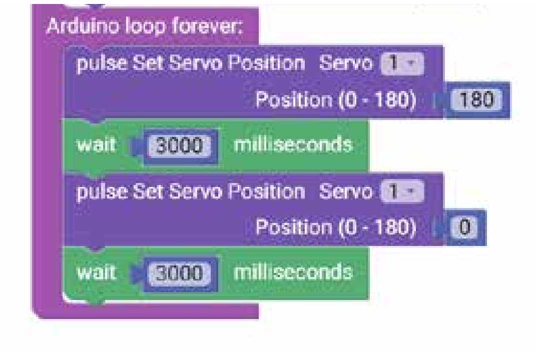


```
Arduino run first:
  pulse Begin
  pulse Set Servo Speed  Servo 1
                   Speed (0 - 100)  25
```

Figure 35

The two parameters of the pulse Set Servo Position block are servo channel and target position. In the example, pulse Set Servo Position means Servo 1 will rotate to the target position of 180°. It then changes position to 0° but continues at the same speed.

This servo will continue to change position because the program will continue to execute the loop until the program is ended (Figure 36).



```
Arduino loop forever:
    pulse Set Servo Position  Servo 1
                    Position (0 - 180) 180
    wait 3000 milliseconds
    pulse Set Servo Position  Servo 1
                    Position (0 - 180) 0
    wait 3000 milliseconds
```

Figure 36

In the sketch, both blocks work together to tell the servo motor not only the target position but also the speed to use while moving to the target position.

You can alter the position and speed of the servo by changing the values of both functions. Practice changing the parameters in the sketch. Observe the effect these changes have on the servo motor.

Extension Activity

With the example as a reference, try creating a new sketch to move your servo motor. You could also incorporate the use of your DC motor and LEDs on the controller. Remember what you learned from your previous activities and think of creative ways to combine the functions you have learned.

Tip: To solve this extension activity, you can find a sample sketch in the appendix titled **GS_Activity_3_Extension_Example**.

Real-World Link

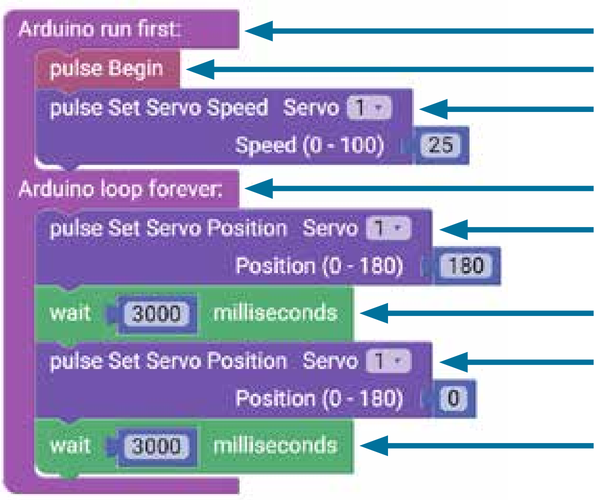
You can find servo motors in R/C model cars for steering and R/C model airplanes for controlling flaps and rudders. Servos are used where precise movement is needed such as in the operation of robotic arms, grippers, and rotating camera mounts.

Careers: fabricator, industrial engineer, machinist

STEM Connections

- Science
  - o Current
  - o Fleming's rules
- Technology
  - o Motor assembly
  - o Feedback system
- Engineering
  - o Energy conversion
  - o Motor gearing
- Math
  - o Accuracy
  - o Precision

Block-Text Correlation



```
Arduino run first:
  pulse Begin
  pulse Set Servo Speed  Servo 1
                Speed (0 - 100) 25
Arduino loop forever:
  pulse Set Servo Position  Servo 1
                Position (0 - 180) 180
  wait 3000 milliseconds
  pulse Set Servo Position  Servo 1
                Position (0 - 180) 0
  wait 3000 milliseconds
```

```
void setup() {
pulse.PulseBegin();
pulse.setServoSpeed(1,25);
void loop() {
pulse.setServoPosition(1,180);
delay(3000);
pulse.setServoPosition(1,0);
delay(3000);
```

Arduino Source Code

```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();
pulse.setServoSpeed(1,25);
}

void loop() {
pulse.setServoPosition(1,180);
delay(3000);
pulse.setServoPosition(1,0);
delay(3000);

}
```

Note: The servo motor changes position from 0 degrees to 180 degrees.

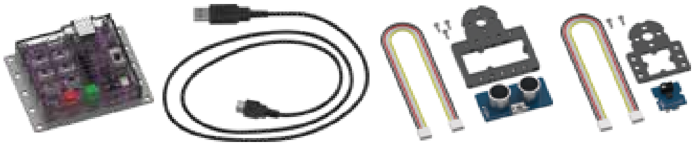
Activity 4: Introduction to the Line Finder Sensor

Introduction

For the fourth activity, you will use a line-finding sensor. In this example, you will connect a Line Finder Sensor to digital sensor port D2. You will create a sketch to read digital input from the Line Finder Sensor.

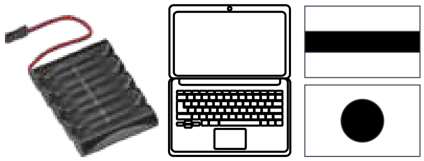
Sensors enable us to gather information from the world around us. The type of information depends on the type of sensor. The Line Finder Sensor uses reflected infrared light to distinguish between light and dark surfaces.

Activity 4 Parts Needed



Electronics & Control

Part No.	Part Name	Quantity
44268	TETRIX PULSE Robotics Controller with USB cable	1
43055	Ultrasonic Sensor Pack	1
43056	Line Finder Sensor Pack	1



Batteries & Hardware

Part No.	Part Name	Quantity
40235	TETRIX PRIME 6 V NiMH Battery Pack	1
	Computer	1
	Contrasting light and dark surface	1

Open the Program

Before you open your next example sketch, be sure to save any sketch you want to reference later. Let's start by looking at the example sketch. Open the sketch by selecting **Examples > GS_Activity_4**. A new sketch window will open titled GS_Activity_4 (Figure 37).

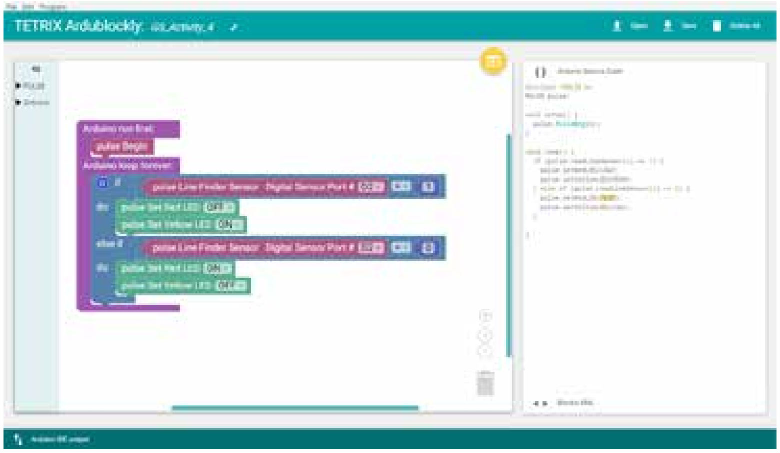
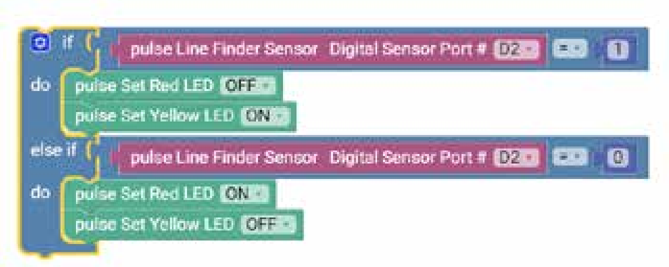


Figure 37

Background

For the fourth sketch, you will take a closer look at programming a sensor and using a logic block. You will use the Line Finder Sensor to determine whether a surface is light or dark.

The if-do logic block does exactly what its name suggests. Everything contained within the loop will repeat consecutively until the sketch is ended with a command or the Stop/Reset button (Figure 38).



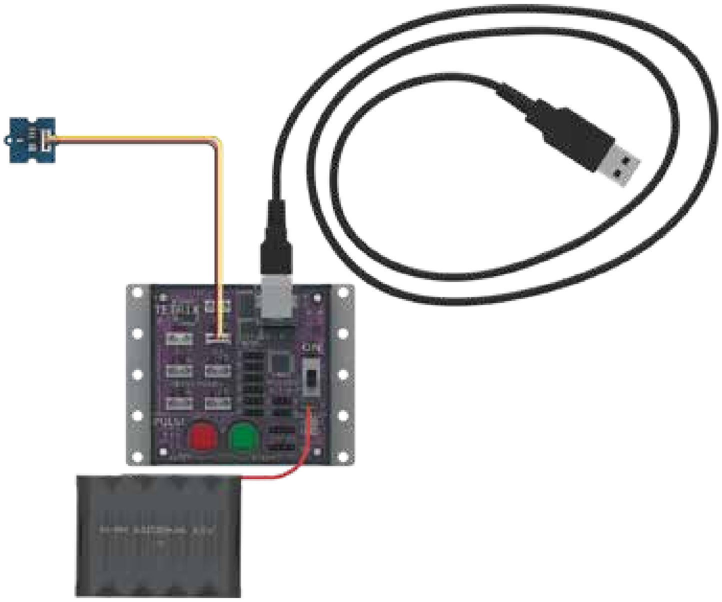
```
if      pulse Line Finder Sensor  Digital Sensor Port # D2  =  1
do      pulse Set Red LED OFF
        pulse Set Yellow LED ON
else if pulse Line Finder Sensor  Digital Sensor Port # D2  =  0
do      pulse Set Red LED ON
        pulse Set Yellow LED OFF
```

Figure 38

Depending on the surface, a red or yellow LED will light up on the PULSE controller.

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your connections. Keep in mind that you added a new connection in digital sensor port D2 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 PULSE controller.

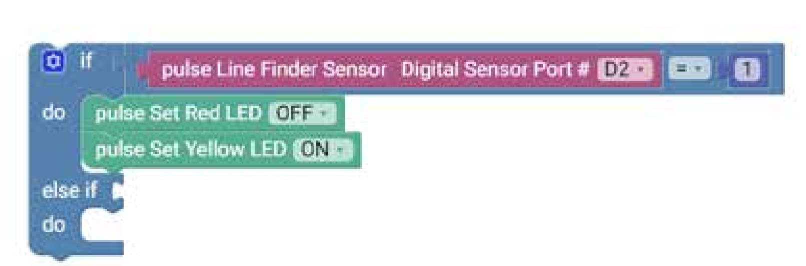
Hold the sensor over a contrasting surface. As the sensor moves from light to dark, observe the red LED on the PULSE. When the sensor is over a nonreflective or dark surface, the yellow LED will be on. When the sensor is over a white or reflective surface, the red LED will be on.

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

Further Investigate

This sketch introduces a program structure, new blocks, and a comparison statement. The program structure is an "if" statement, the reading of the sensor, and the comparison statement is "=" (equal to). The comparison statement "=" (equal to) defines a type of test. In this sketch, the input of the Line Finder Sensor will turn different LEDs on or off.

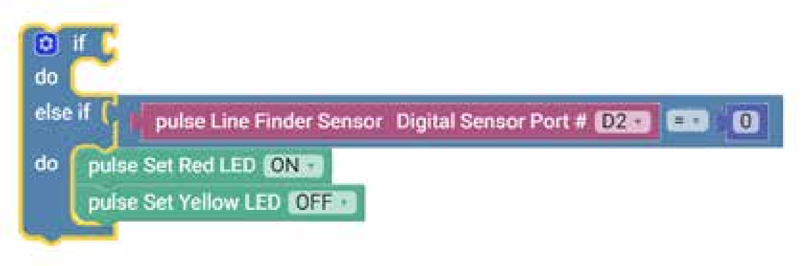
The basic "if" statement enables us to test for a certain condition. If this condition is met, then the program can perform an action. If the variable in the if section of the loop block is true, then the blocks within the do section are run. In this instance, the output of the Line Finder Sensor must equal 1, or HIGH, for the do section to run (Figure 39).



```
if    pulse Line Finder Sensor  Digital Sensor Port # D2 = 1
do      pulse Set Red LED OFF
        pulse Set Yellow LED ON
else if
do
```

Figure 39

If the if portion of the loop block isn't true, then the else if portion enables you to test for a different condition. If this condition is met, then the blocks within the do section under the else if section will run. In the else if portion, the output of the Line Finder Sensor must equal 0, or LOW (Figure 40).



```
if
do
else if    pulse Line Finder Sensor  Digital Sensor Port # D2 = 0
do           pulse Set Red LED ON
             pulse Set Yellow LED OFF
```

Figure 40

The Line Finder Sensor can return 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 nonreflective surface; a value of "0" is returned when the Line Finder Sensor detects a white or reflective surface.

If the Line Finder Sensor detects a line or a nonreflective surface, then it turns the red LED on. Think of it as an object placed in its path. The red LED on the Line Finder Sensor also lights up. In the program, this occurs when the line variable is set to LOW.

If the Line Finder Sensor detects a white or reflective surface, it turns the yellow LED on. In the program, this occurs when the line variable is set to HIGH (Figure 41).

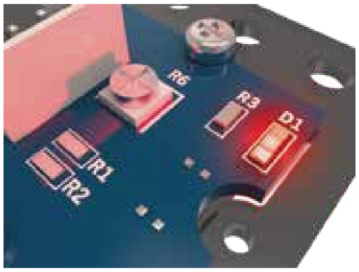


Figure 41

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

Extension Activity

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

Real-World Link

There are many uses for a robot that can follow a line. Automated robots are being developed to follow lines to deliver materials within hospitals. In the future, they could also move materials around warehouses or transport goods.

Careers: materials engineer, electromechanical technician, software developer

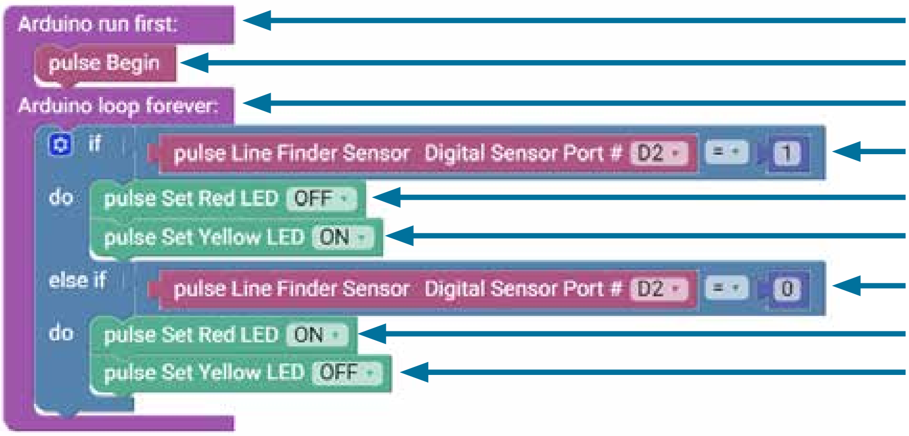
STEM Connections

- Science
  - o Light
  - o Electromagnetic spectrum
- Technology
  - o Guidance system
  - o Automation
- Engineering
  - o Microcontroller
  - o Embedded system
- Math
  - o Angles
  - o Lines

Arduino Source Code

```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

}

void loop() {
if (pulse.readLineSensor(2) == 1) {
pulse.setRedLED(LOW);
pulse.setYellowLED(HIGH);
} else if (pulse.readLineSensor(2) == 0) {
pulse.setRedLED(HIGH);
pulse.setYellowLED(LOW);
}
}
```

Block-Text Correlation



```
void setup() {
pulse.PulseBegin();
void loop() {
if (pulse.readLineSensor(2) == 1) {
pulse.setRedLED(LOW);
pulse.setYellowLED(HIGH);
} else if (pulse.readLineSensor(2) == 0) {
pulse.setRedLED(HIGH);
pulse.setYellowLED(LOW);
```

Note: All the actions completed in the if () or if/else () loops are contained within braces.

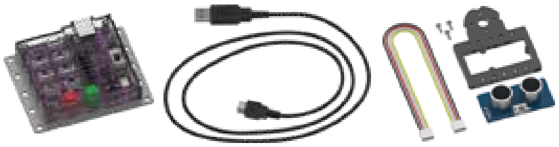
Activity 5: Introduction to the Ultrasonic Sensor

Introduction

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

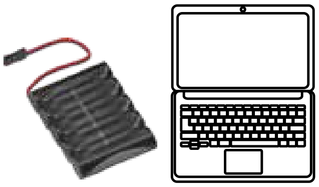
Like all sensors, the Ultrasonic Sensor enables us to gather information. The Ultrasonic Sensor gathers information to communicate distance. The sensor works by sending a sonic pulse burst and then waiting on its return as it is reflected off an object in range. The reflected sonic pulse time period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

Activity 5 Parts Needed



Electronics & Control

Part No.	Part Name	Quantity
44268	TETRIX PULSE Robotics Controller with USB cable	1
43055	Ultrasonic Sensor Pack	1



Batteries & Hardware

Part No.	Part Name	Quantity
40235	TETRIX PRIME 6 V NiMH Battery Pack	1
	Computer	1

Open the Program

Before you open your next example sketch, be sure to save any sketch you want to reference later. Let's start by looking at the example sketch. Open the sketch by selecting **Examples > GS_Activity_5**. A new sketch window will open titled GS_Activity_5 (Figure 42).

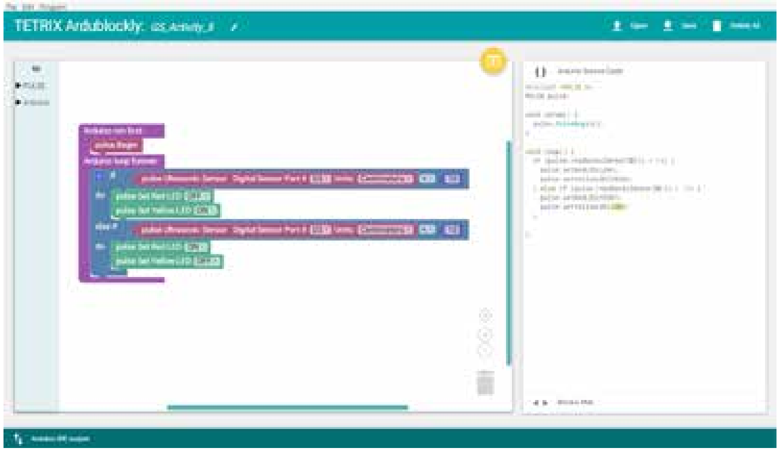


Figure 42

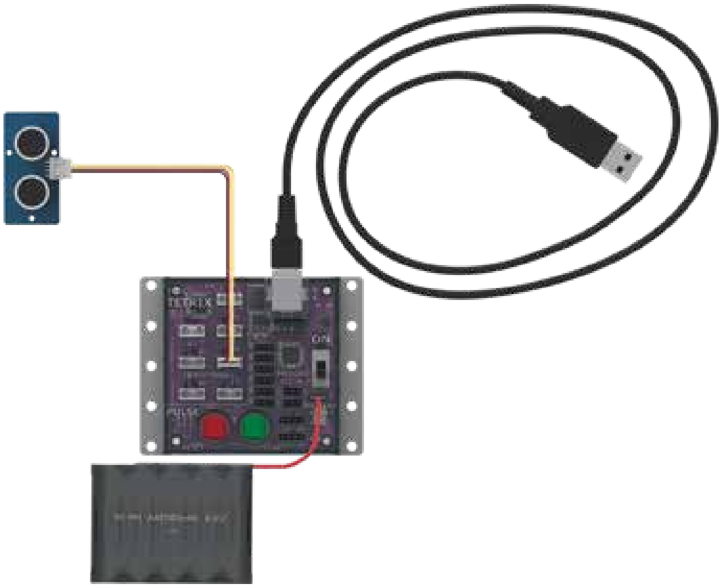
Background

For your fifth sketch, you will use the Ultrasonic Sensor, which sends out pulses that measure the distance from the sensor to the object. You will use the same logic block from the previous activity.

Instead of measuring light, this sensor will measure distance. Depending on the distance of the object from the sensor, a red or yellow LED will light up on the PULSE controller.

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your connections. Keep in mind that you added a new connection in digital sensor port D3 with the Ultrasonic Sensor. Upload the sketch.



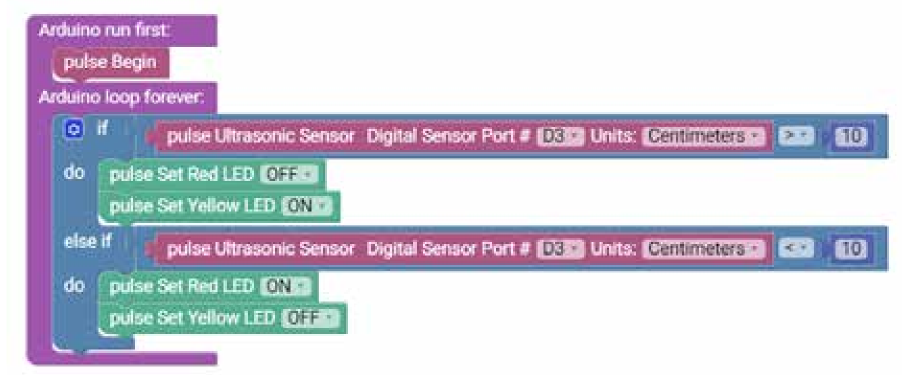
With the sensor lying flat on the desk pointed up, press the green Start button to execute the code.

Hold your hand above the sensor. Move it up and down. Watch what happens to the LEDs on the PULSE controller. Press the Stop/Reset button when you are ready to stop the sensor.

Further Investigate

In this sketch, the input of the Ultrasonic Sensor will turn different LEDs on or off.

If the variable in the if section of the loop block is true, then the blocks within the do section of the loop block are run. In this instance, if there isn't an object closer than 10 cm to the Ultrasonic Sensor, the yellow LED on the PULSE controller will be lit (Figure 43).



```
Arduino run first:
    pulse Begin
Arduino loop forever:
    if      pulse Ultrasonic Sensor  Digital Sensor Port # D3  Units: Centimeters  >  10
    do      pulse Set Red LED OFF
            pulse Set Yellow LED ON
    else if pulse Ultrasonic Sensor  Digital Sensor Port # D3  Units: Centimeters  <  10
    do      pulse Set Red LED ON
            pulse Set Yellow LED OFF
```

Figure 43

If the if portion of the loop block isn't true, then the else if portion enables you to test for a different condition. If this condition is met, then the blocks within the do section under the else if section will run. In this instance, if there is an object closer than 10 cm to the Ultrasonic Sensor, the red LED on the PULSE controller will be lit.

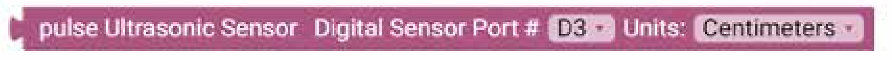
Experiment with the Ultrasonic Sensor with different objects and different heights to see how the sensor reacts.

Extension Activity

With the example as a reference, try creating a new sketch to use your Ultrasonic Sensor. Remember what you learned from your previous activities and experiment with different objects in front of the Ultrasonic Sensor to see if they are detectable. You can also program the motors to stop when an object reaches a certain distance from the Ultrasonic Sensor.

Try changing the distance units on the pulse Ultrasonic Sensor block from centimeters to inches. Understanding how to use the Ultrasonic Sensor will give your robot vision so that it can steer around objects and obstacles (Figure 44).

Tip: To solve this extension activity, you can find a sample sketch in the appendix titled **GS_Activity_5_Extension_Example**.



```
pulse Ultrasonic Sensor  Digital Sensor Port # D3  Units: Centimeters
```

Figure 44

Real-World Link

Modern vehicles are smart. They have backup cameras and assisted parallel parking and will even beep at you if an object is too close to the car. This is how an ultrasonic sensor works!

Careers: car designer, sound engineering technician, sonar technician

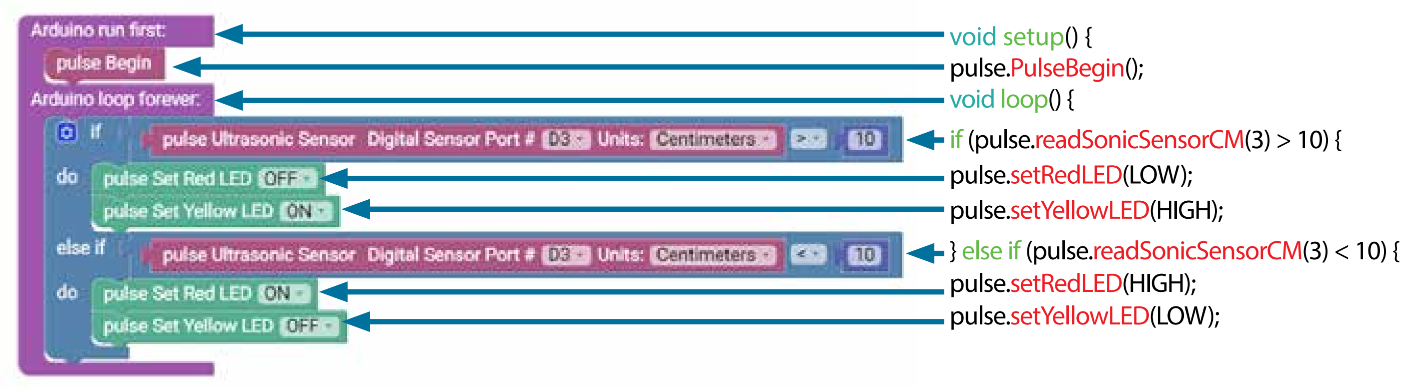
STEM Connections

- Science
    - o Sound waves
    - o Reflection of sound waves
- Technology
    - o Frequency
    - o Sound digitization
- Engineering
    - o Sonic measurements
    - o Sonar
- Math
    - o Distance
    - o Comparison symbols

Arduino Source Code

```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

}

void loop() {
if (pulse.readSonicSensorCM(3) > 10) {
pulse.setRedLED(LOW);
pulse.setYellowLED(HIGH);
} else if (pulse.readSonicSensorCM(3) < 10) {
pulse.setRedLED(HIGH);
pulse.setYellowLED(LOW);
}
}
```

Block-Text Correlation

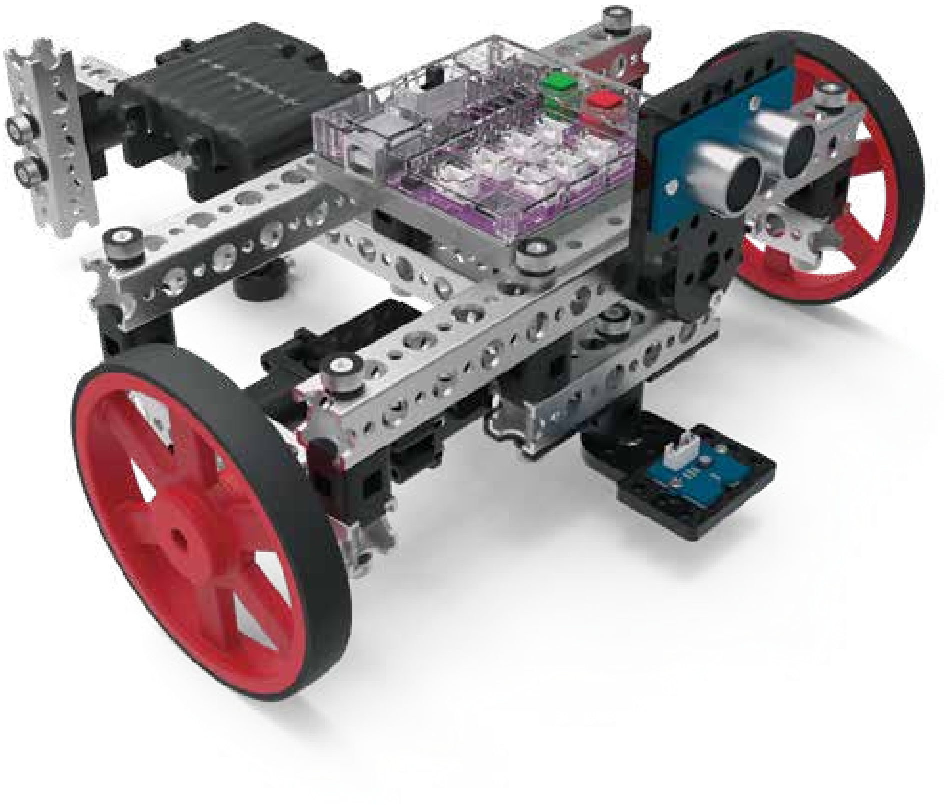


```
void setup() {
pulse.PulseBegin();
void loop() {
if (pulse.readSonicSensorCM(3) > 10) {
pulse.setRedLED(LOW);
pulse.setYellowLED(HIGH);
} else if (pulse.readSonicSensorCM(3) < 10) {
pulse.setRedLED(HIGH);
pulse.setYellowLED(LOW);
```

Note: This program uses the math values of greater than (>) and less than (<). A specific LED will be lit depending on whether the output value is greater than or less than 10.

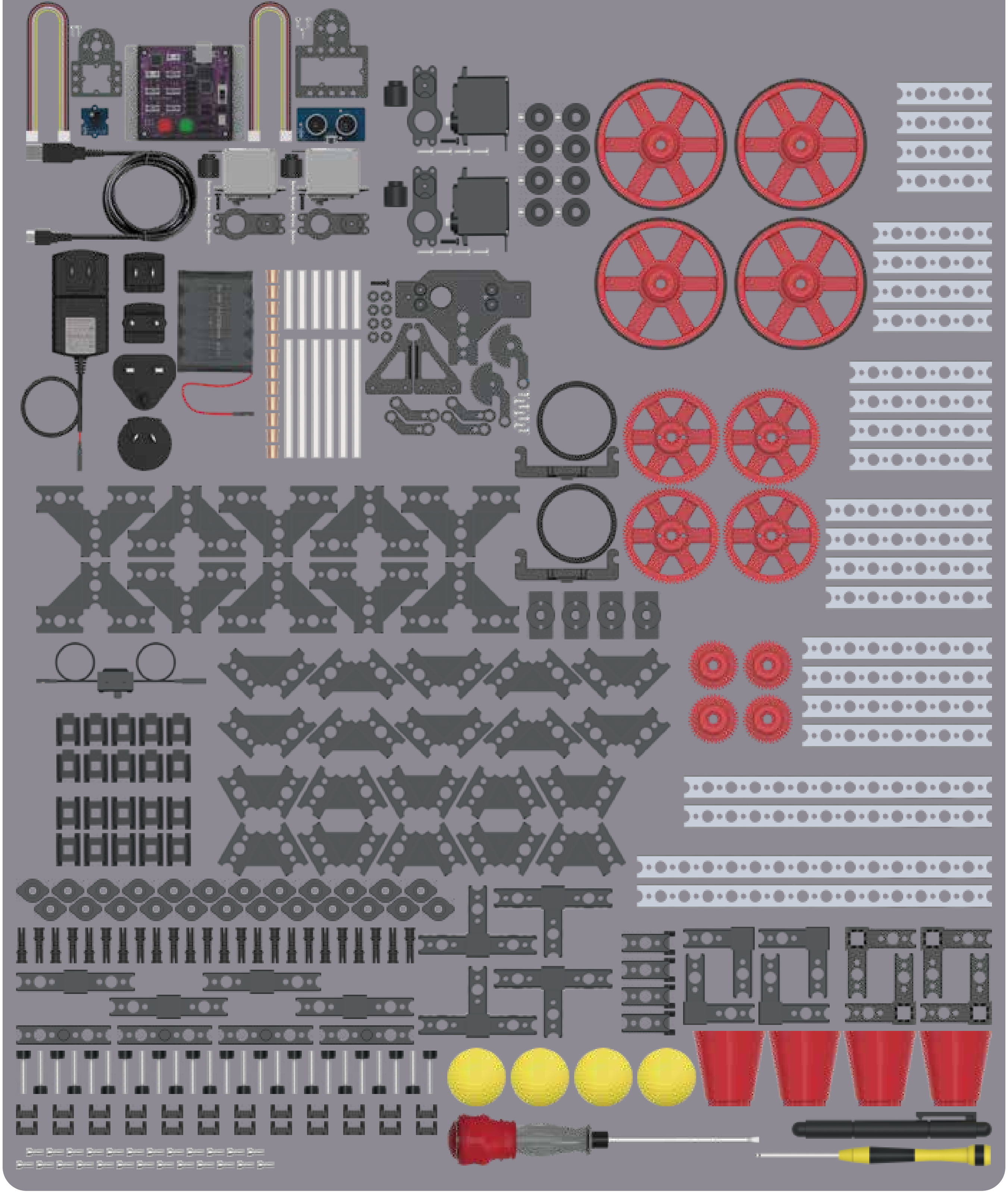
Building and Coding the PULSE Codee Bot

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 you through building, moving, navigating, and adding sensors to a robot, and more, culminating in an activity that combines everything. It's about to get exciting!

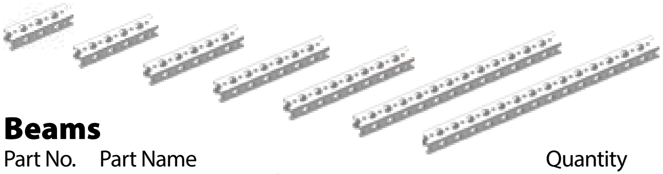


Programmable Robotics Set

Note: In order to complete the build shown in this book, you must have the TETRIX PRIME Programmable Set.

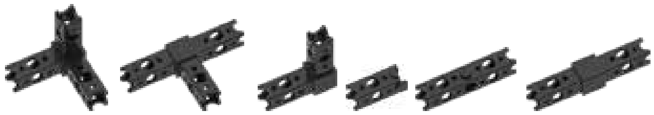


TETRIX PRIME Programmable Robotics Set Parts Index



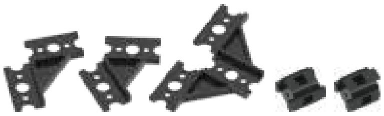
Beams

Part No.	Part Name	Quantity
40201	TETRIX PRIME 4-Hole Square Beam	4
40202	TETRIX PRIME 5-Hole Square Beam	4
40203	TETRIX PRIME 6-Hole Square Beam	4
40204	TETRIX PRIME 7-Hole Square Beam	4
40205	TETRIX PRIME 8-Hole Square Beam	4
40206	TETRIX PRIME 13-Hole Square Beam	2
40207	TETRIX PRIME 15-Hole Square Beam	2



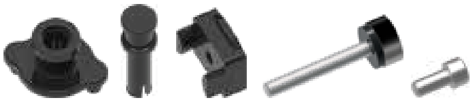
Internal Connectors

Part No.	Part Name	Quantity
40212	TETRIX PRIME 3-Way Beam Connector	4
40213	TETRIX PRIME Tee Beam Connector	4
40211	TETRIX PRIME 90-Degree Beam Connector	4
40214	TETRIX PRIME Beam End Connector	4
40322	TETRIX PRIME Beam Extension Connector	4
40215	TETRIX PRIME Beam Straight Connector	4



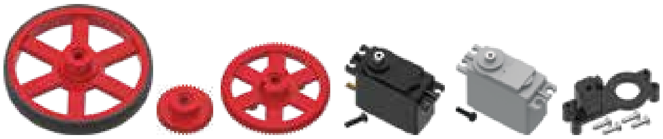
External Connectors

Part No.	Part Name	Quantity
40208	TETRIX PRIME 90-Degree Beam Bracket	10
40209	TETRIX PRIME 60-Degree Beam Bracket	10
40210	TETRIX PRIME Tee Beam Bracket	10
40216	TETRIX PRIME Straight Block Beam Connector	10
40217	TETRIX PRIME 90-Degree Cross Block Connector	10



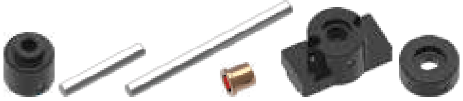
Connecting Hardware

Part No.	Part Name	Quantity
40219	TETRIX PRIME Quick Rivet Connector	24
40220	TETRIX PRIME Quick Rivet Peg	24
40221	TETRIX PRIME Wing Nut	24
40323	TETRIX PRIME Thumbscrew	24
40516	Socket Head Cap Screw	25



Wheels, Gears, & Servos

Part No.	Part Name	Quantity
40222	TETRIX PRIME Wheel with Tire	4
40223	TETRIX PRIME 40-Tooth Plastic Gear	4
40224	TETRIX PRIME 80-Tooth Plastic Gear	4
40538	TETRIX Standard-Scale Servo Motor	2
44298	TETRIX PRIME DC Motor	2
40232	TETRIX PRIME Servo Mounting Bracket	6

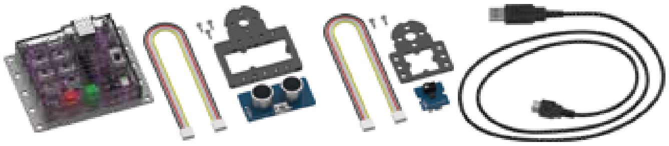


Wheel, Gear, & Servo Hardware

Part No.	Part Name	Quantity
40230	TETRIX PRIME Shaft Servo Hub	6
40225	TETRIX PRIME 80 mm Steel Axle	6
40226	TETRIX PRIME 40 mm Steel Axle	6
40227	TETRIX PRIME 8 mm x 6 mm Bronze Bushing	12
40228	TETRIX PRIME Beam Attachment Hub	4
40229	TETRIX PRIME D-Shaft Set Collar	8

Gripper Assembly

Part No.	Part Name	Quantity
40234	TETRIX PRIME Gripper Kit	1



Electronics & Control

Part No.	Part Name	Quantity
44268	TETRIX PULSE Robotics Controller with USB cable	1
43055	Ultrasonic Sensor Pack	1
43056	Line Finder Sensor Pack	1

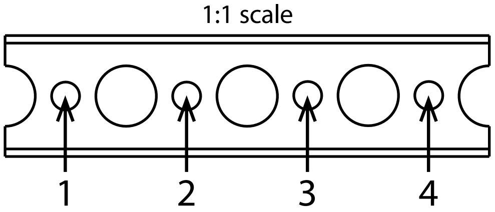
Batteries & Hardware

Part No.	Part Name	Quantity
40236	TETRIX PRIME Battery Mount Bracket	2
40235	TETRIX PRIME 6 V NiMH Battery Pack	1
44498	TETRIX PRIME 6 V NiMH Global Battery Pack Charger	1
36404	4-in-1 Screwdriver	1
40341	Miniature Ball-Point Hex Driver	1
42991	2-in-1 Screwdriver	1
41769	Plastic 2 oz Cups	4
14041	Practice Golf Balls	4
44301	TETRIX PULSE Robotics Controller Programming Guide	1

TETRIX PRIME Hardware Components

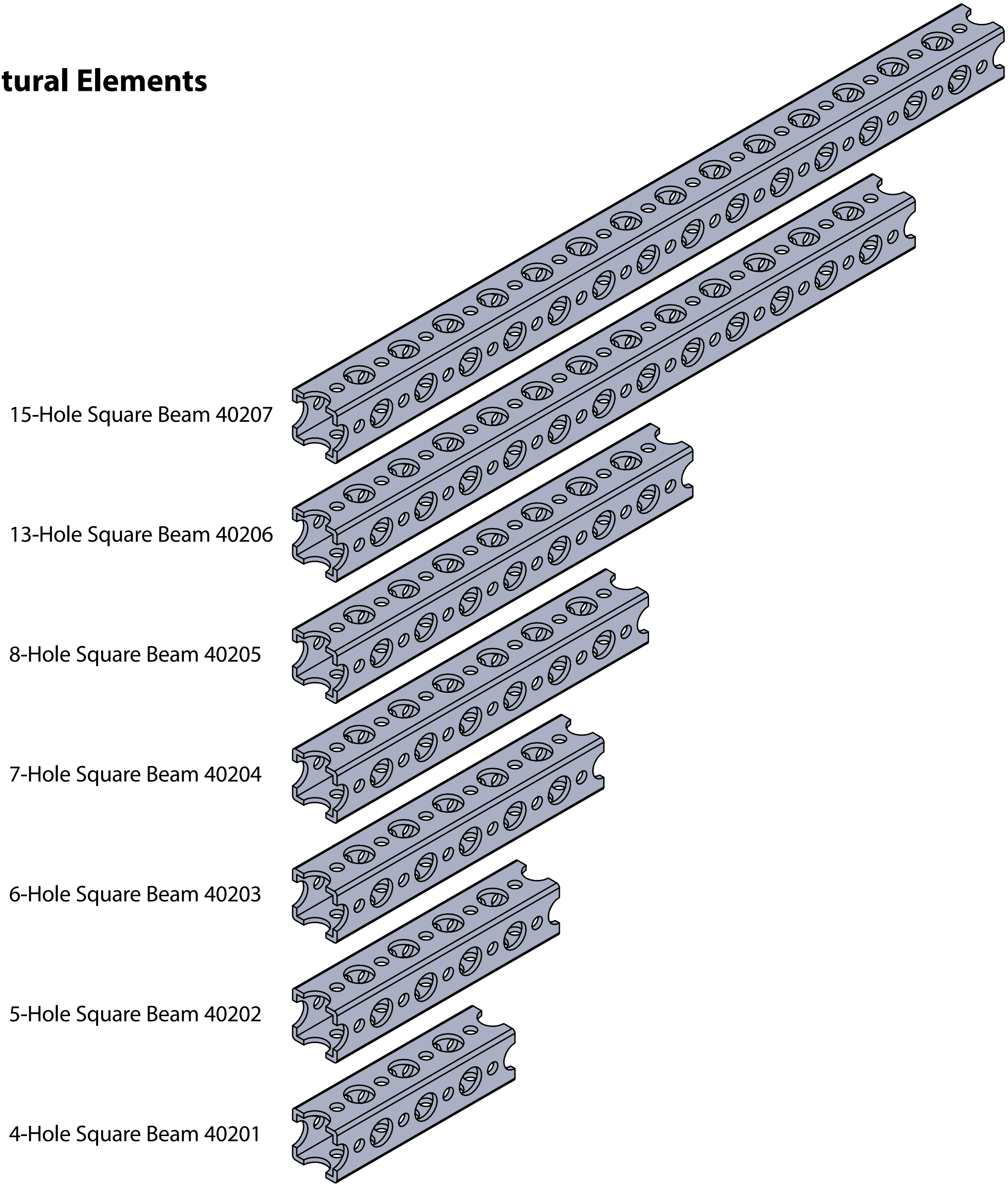
Beams

The beams are named by the number of small holes on one side of the beam. Do not select beams by counting the larger holes (see right).



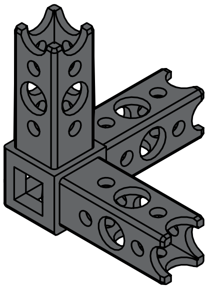
To identify TETRIX PRIME Square Beams, count the small holes. The example above is a 4-hole square beam.

Structural Elements

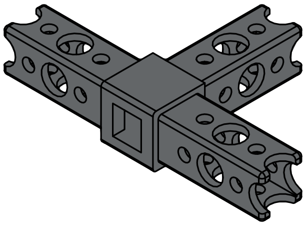


Structural Elements

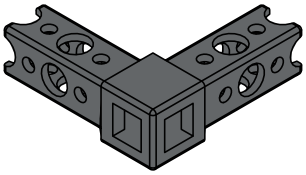
3-Way Beam Connector 40212



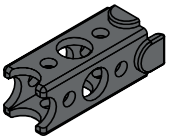
Tee Beam Connector 40213



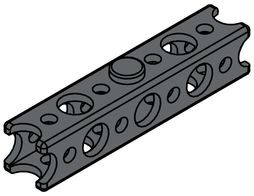
90-Degree Beam Connector 40211



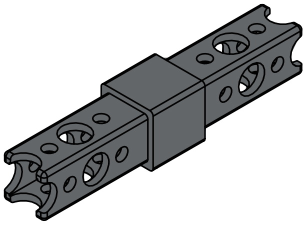
Beam End Connector 40214



Beam Extension Connector 40322

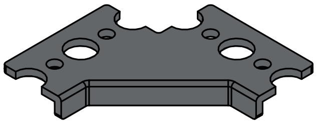


Beam Straight Connector 40215

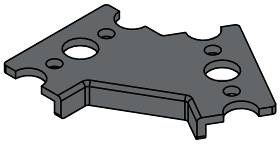


Structural Elements

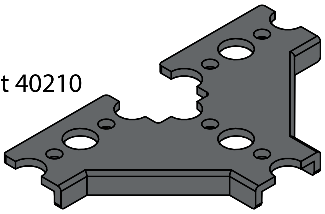
90-Degree Beam Bracket 40208



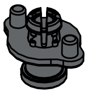
60-Degree Beam Bracket 40209



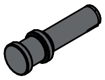
Tee Beam Bracket 40210



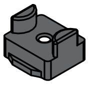
Quick Rivet Connector 40219



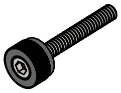
Quick Rivet Peg 40220



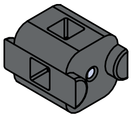
Wing Nut 40221



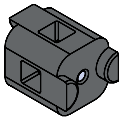
Thumbscrew 40323



Straight Block Beam Connector 40216

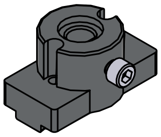


90-Degree Cross Block Connector 40217



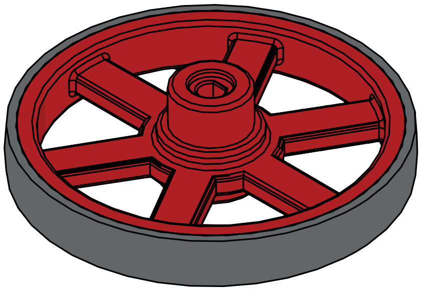
Tip: These two parts, the straight block beam connector and the 90-degree cross block connector, are very close in appearance and can be easily confused. Please double-check that you are using the one the directions call for.

Beam Attachment Hub 40228

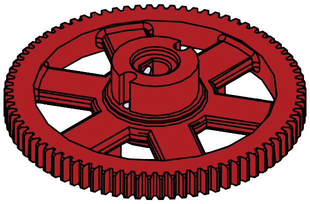


Motion Elements

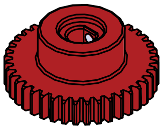
Wheel with Tire 40222



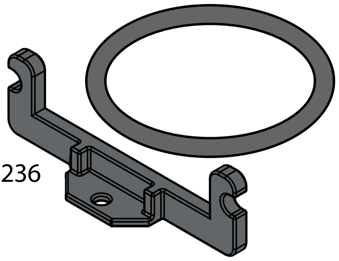
80-Tooth Plastic Gear 40224



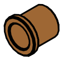
40-Tooth Plastic Gear 40223



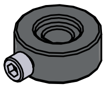
Battery Mount Bracket 40236



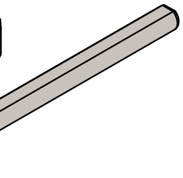
8 mm x 6 mm Bronze Bushing 40227



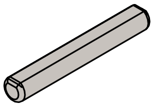
D-Shaft Set Collar 40229



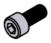
80 mm Steel Axle 40225



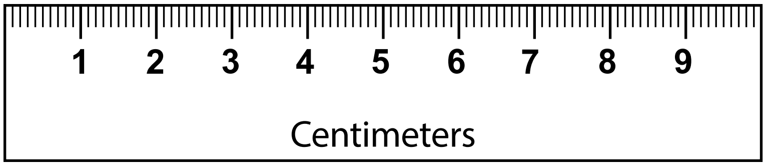
40 mm Steel Axle 40226



Socket Head Cap Screw 40516



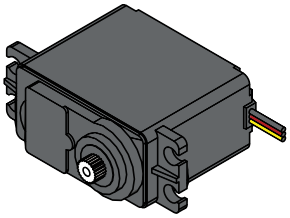
Measure 80 mm and 40 mm steel axles here.



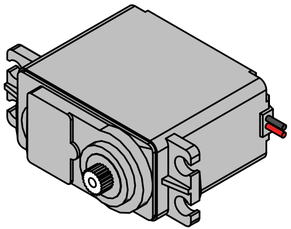
Note: 10 mm = 1 cm

Motion Elements

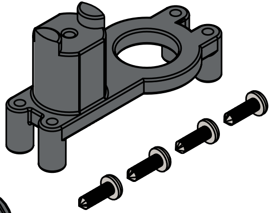
Standard-Scale Servo Motor 40538



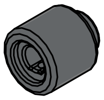
DC Motor 44298



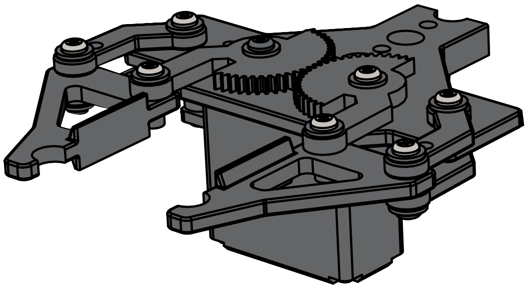
Servo Mounting Bracket 40232



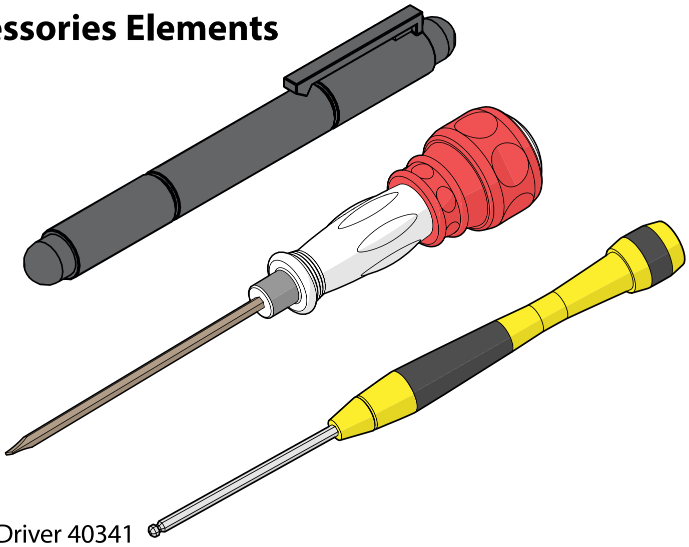
Shaft Servo Hub 40230



Gripper Kit 40234



Power, Tools, and Accessories Elements

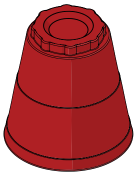


4-in-1 Screwdriver 36404

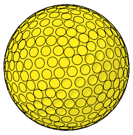
2-in-1 Screwdriver 42991

Miniature Ball-Point Hex Driver 40341

Plastic 2 oz Cups 41769

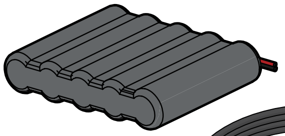


Practice Golf Balls 14041

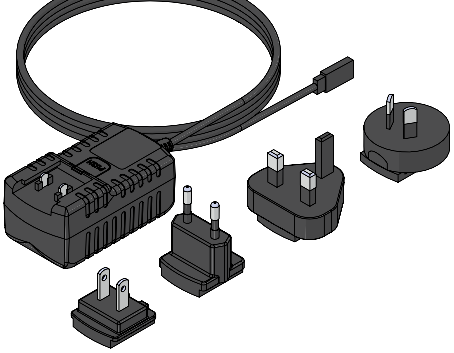


Power, Tools, and Accessories Elements

6 V NiMH Battery Pack 40235

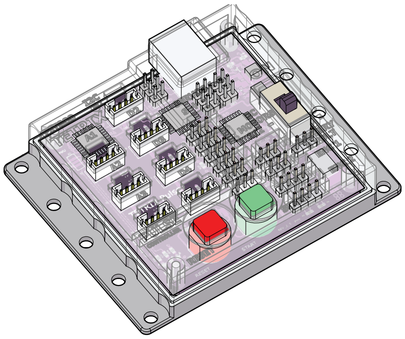


6 V NiMH Global Battery Pack Charger 44498

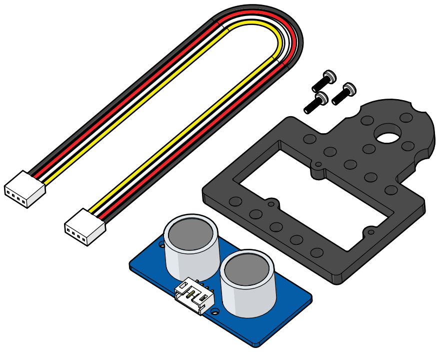


Control Elements

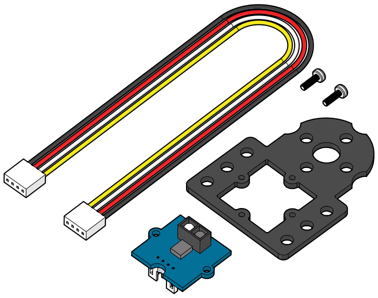
TETRIX PULSE Robotics Controller 44268



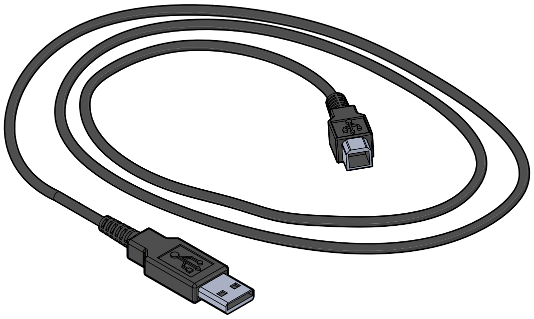
Ultrasonic Sensor Pack 43055



Line Finder Sensor Pack 43056



3-Foot Type A-B USB Cable 40967



Standard Servo Assembly

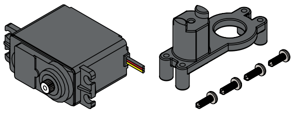
You will need the standard-scale servo motor with screw, the servo mounting bracket with screws, a shaft servo hub, and a socket head cap screw. You will also need the 4-in-1 screwdriver and the miniature ball-point hex driver. remove the white plastic servo horn attached to the servo. Retain the screw for future use, but discard the white plastic servo horn. Attach the Standard Servo label to the servo. Assemble one of the standard servos as indicated in the illustrations. The other standard servo will be used for the gripper assembly.

Standard servo motors are used for proportional rotation and for grippers, steering, and positioning.

Servo Setup Parts Needed

Connecting Hardware

Part No.	Part Name	Quantity
40516	Socket Head Cap Screw	1

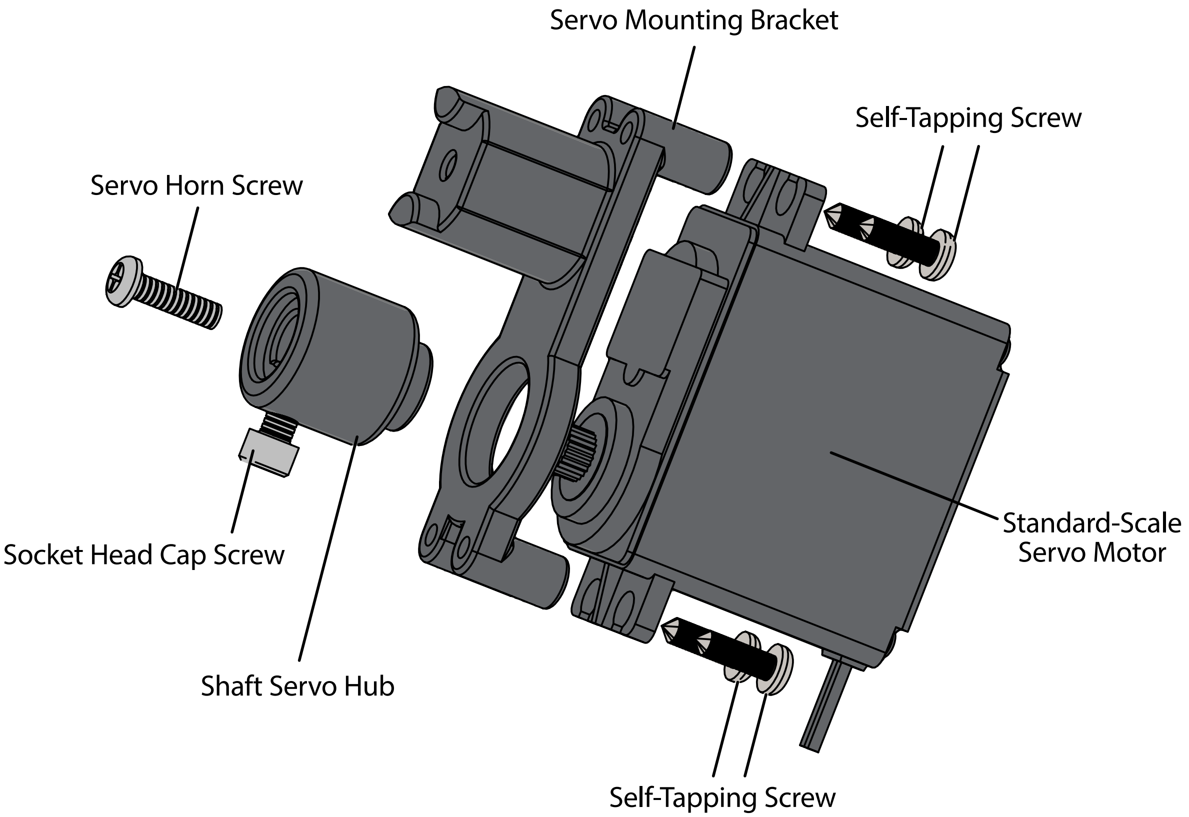


Wheels, Gears, & Servos

Part No.	Part Name	Quantity
40538	TETRIX Standard-Scale Servo Motor	1
40232	TETRIX PRIME Servo Mounting Bracket	1

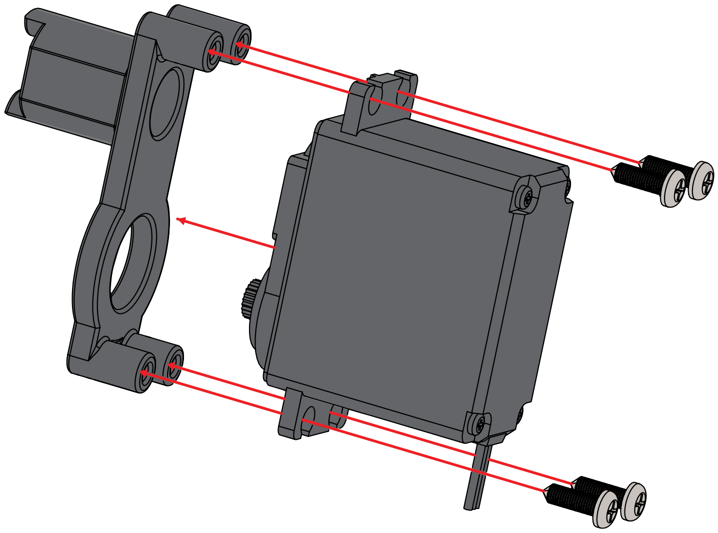
Wheel, Gear, & Servo Hardware

Part No.	Part Name	Quantity
40230	TETRIX PRIME Shaft Servo Hub	1

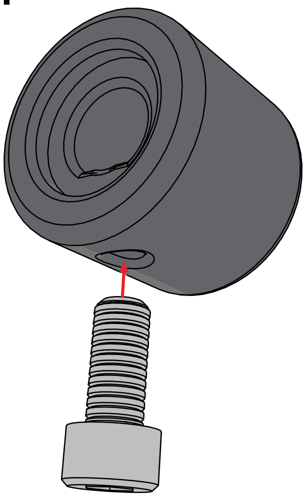


Tip: Keep in mind that the continuous rotation servo mounts in the servo mounting bracket the same way as the standard servo, but there is no need to center the servo horn.

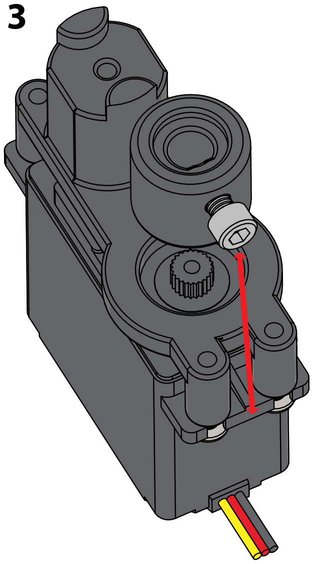
Step 1



Step 2

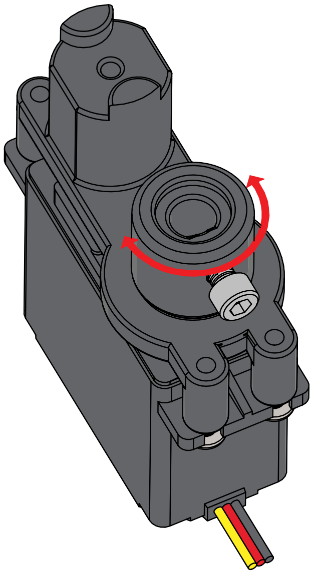


Step 3

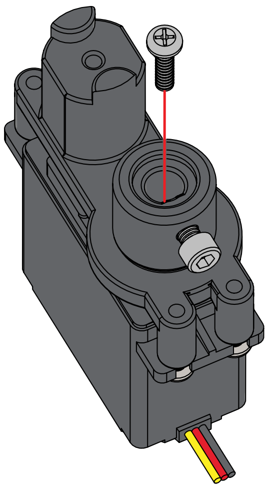


Step 4

Before attaching the servo hub, you must make sure the servo is in the neutral position. To do this, connect both the servo and the battery to the receiver and turn on the power to the remote transmitter. Make sure both joysticks and trimmers are in the center, or neutral, position. The servo motor will move to its neutral position.



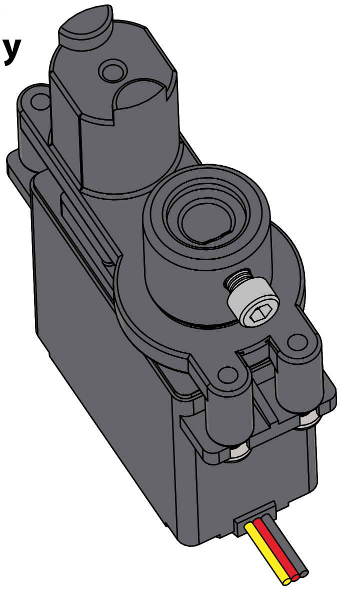
Step 5



Line up the splines on the hub with the splines on the servo and press the two parts together. The set screw should line up as close as possible with the center of the servo case. Tighten the screw holding the hub in place.

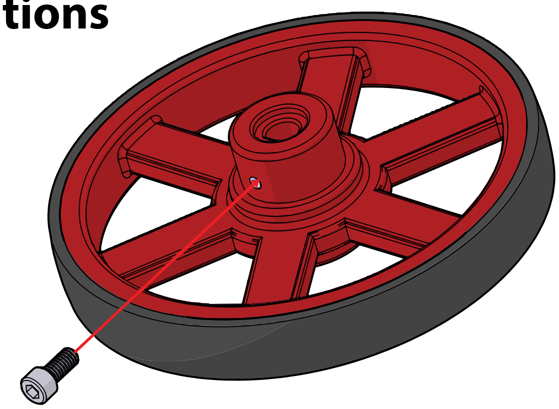
Using the remote transmitter, verify the operation of the servo. If the servo operates properly, disconnect the battery from the receiver and the servo. Your servos are ready for use.

Finished Assembly

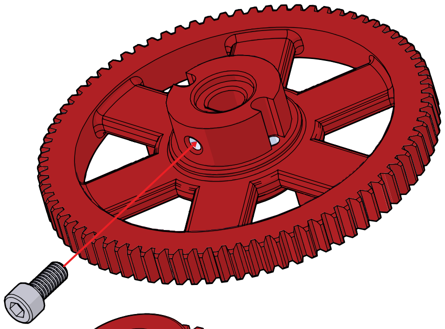


Set Screw Applications

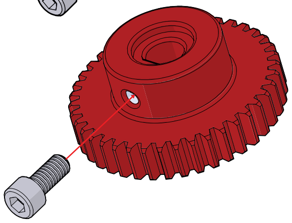
Wheel with Tire 40222



80-Tooth Plastic Gear 40224

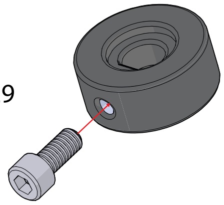


40-Tooth Plastic Gear 40223



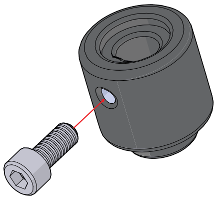
Tip: Please keep in mind that if your parts already have socket head cap screws installed and the directions call for that operation, you can skip to the next step.

D-Shaft Set Collar 40229

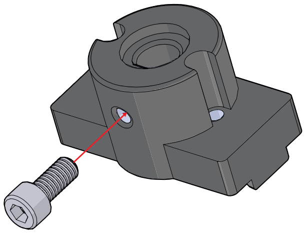


Tip: Directions will show these types of parts in proper position. After the parts are in place, don't forget to tighten the socket head cap screws before moving on to the next step.

Shaft Servo Hub 40230



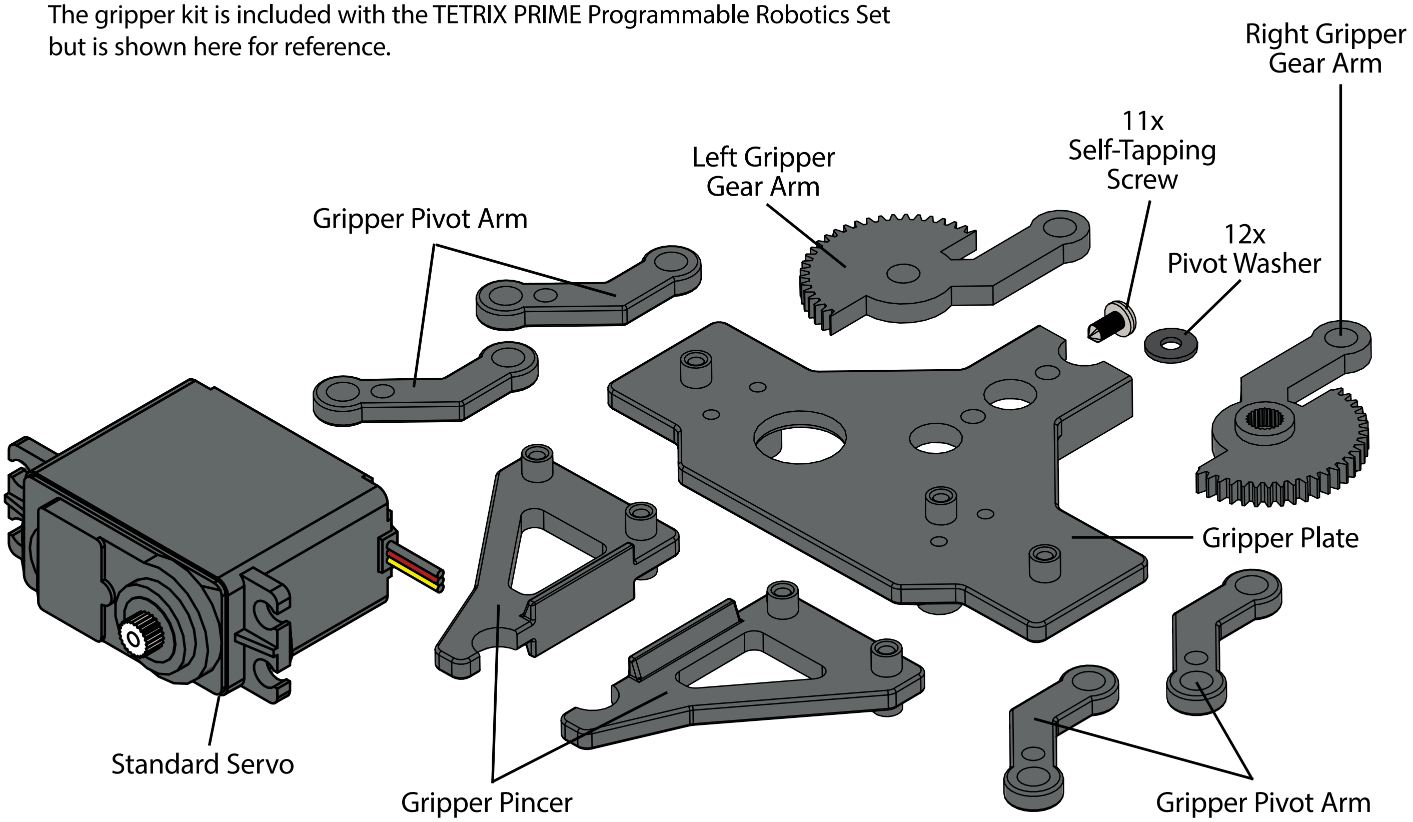
Beam Attachment Hub 40228



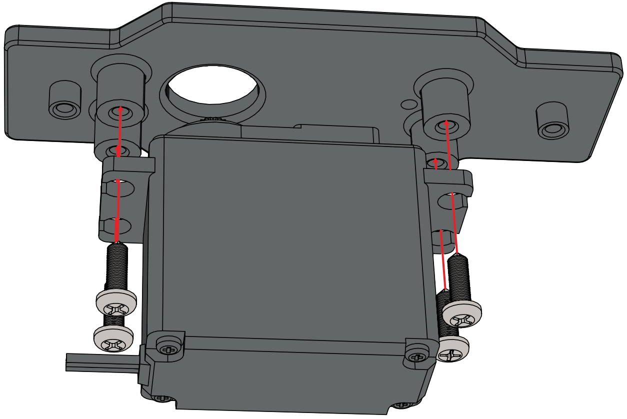
Tip: Socket head cap screws should be snug but not overtightened. Overtightening socket head cap screws can damage components.

Gripper Assembly

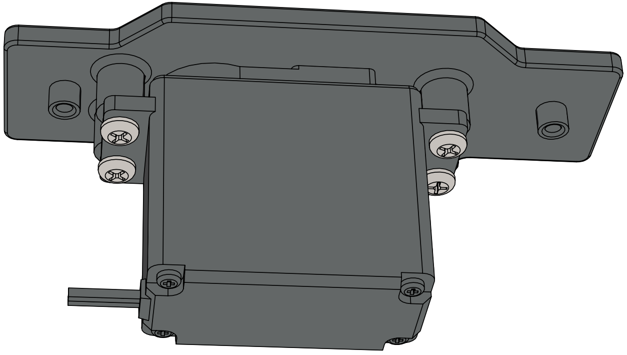
The gripper kit is included with the TETRIX PRIME Programmable Robotics Set but is shown here for reference.



Step 1

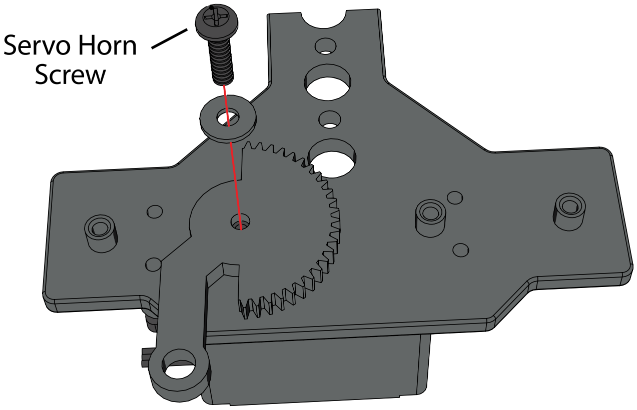


Step 2

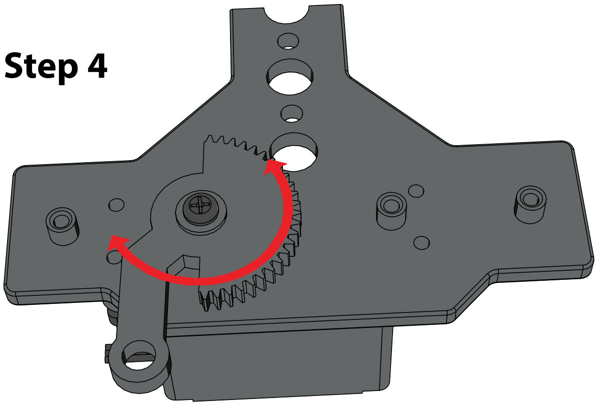


Important: Before attaching the right gripper gear arm to the standard servo, attach the standard servo to the gripper plate. Then, connect the standard servo to the wireless joystick gamepad system and position the servo motor to the neutral position (center joystick) for proper gear position alignment as shown in the following steps. Instructions on assembly of TETRIX PRIME parts can also be found at **Pitsco.com/TETRIX**.

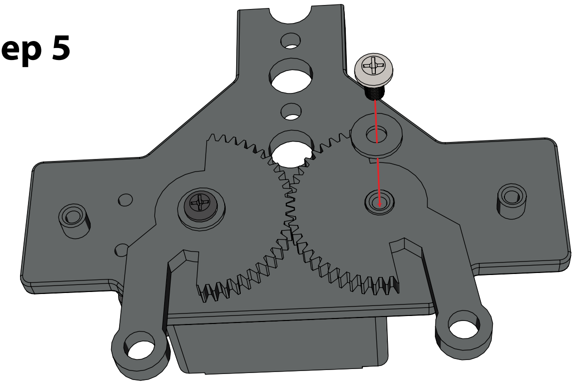
Step 3



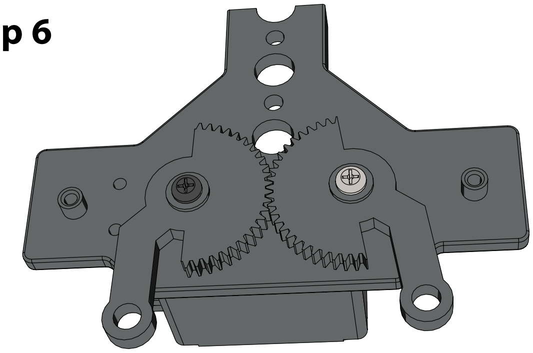
Step 4



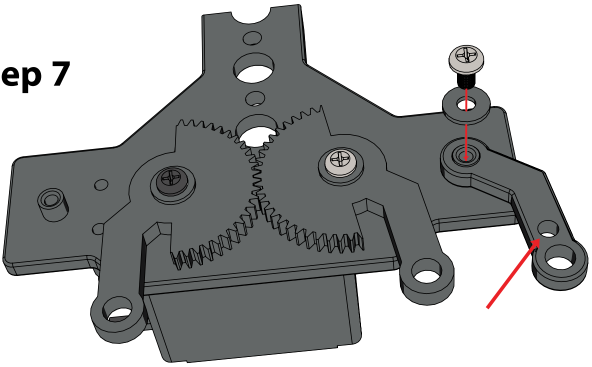
Step 5



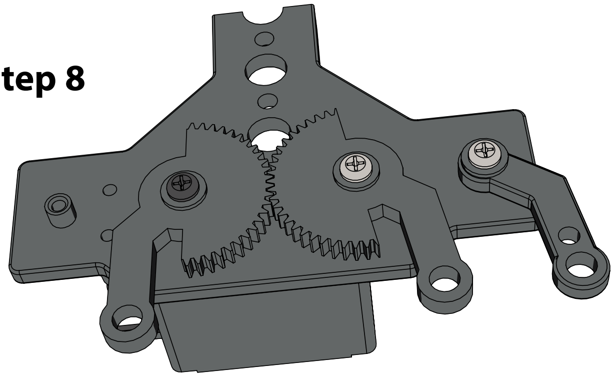
Step 6



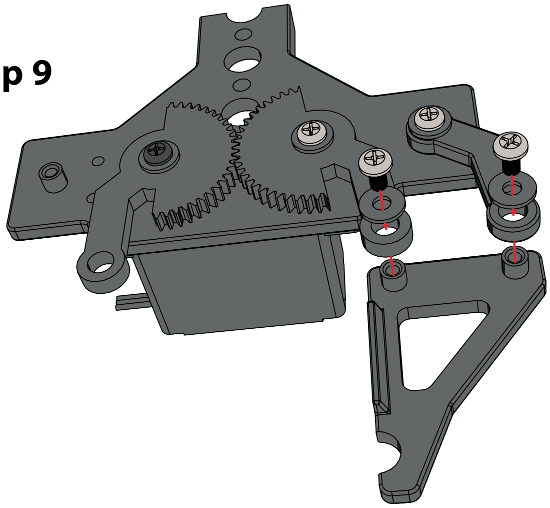
Step 7



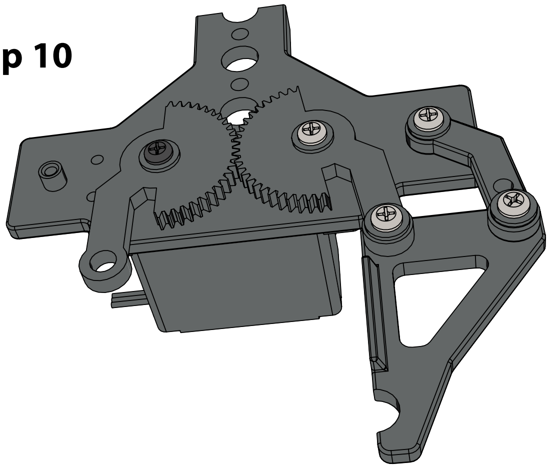
Step 8



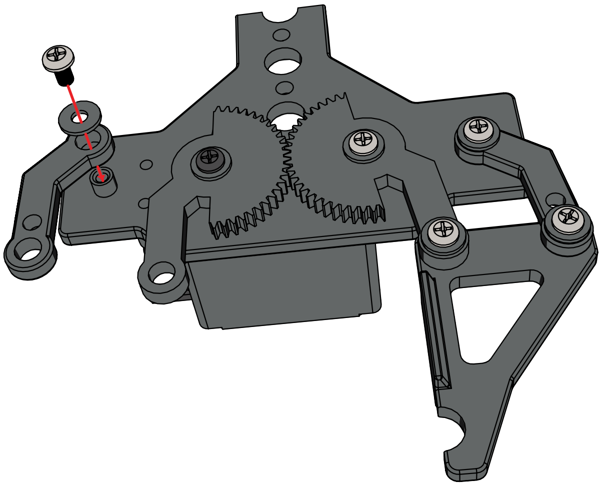
Step 9



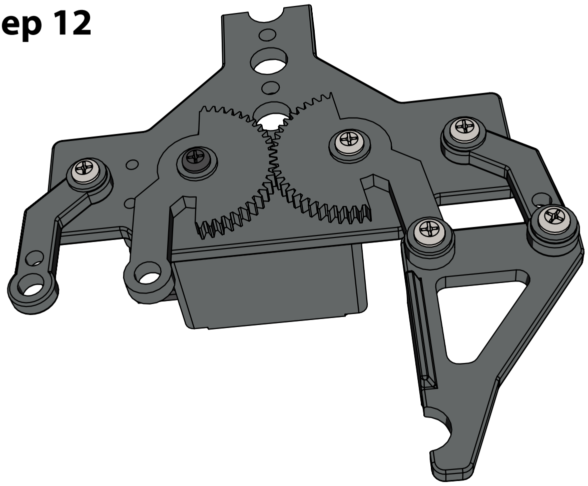
Step 10



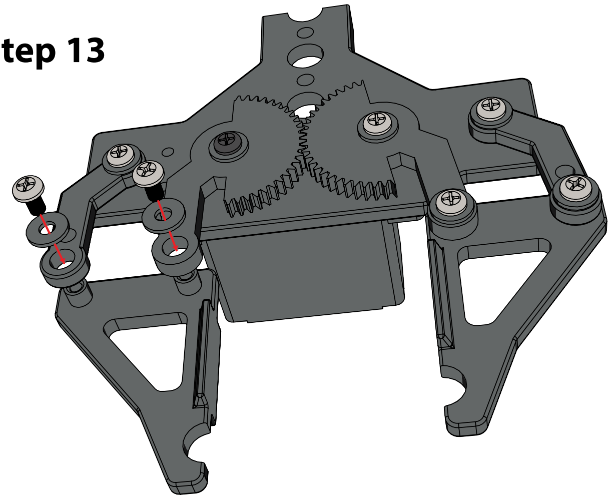
Step 11



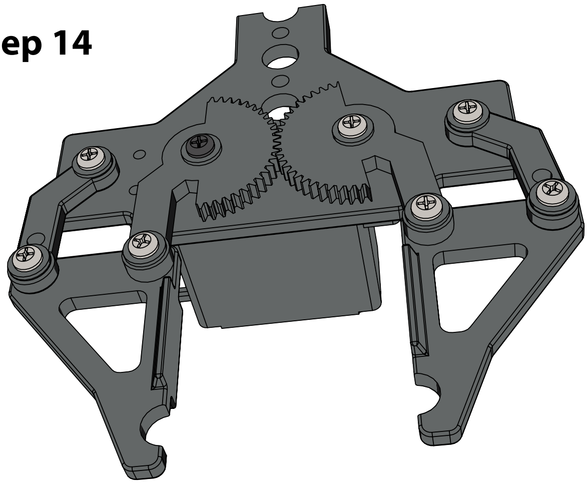
Step 12



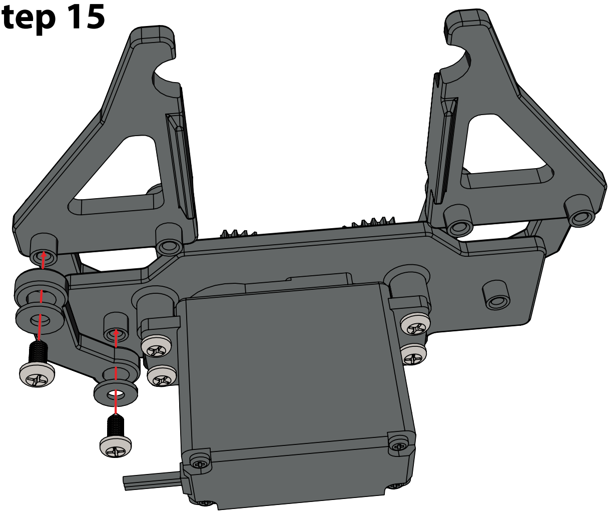
Step 13



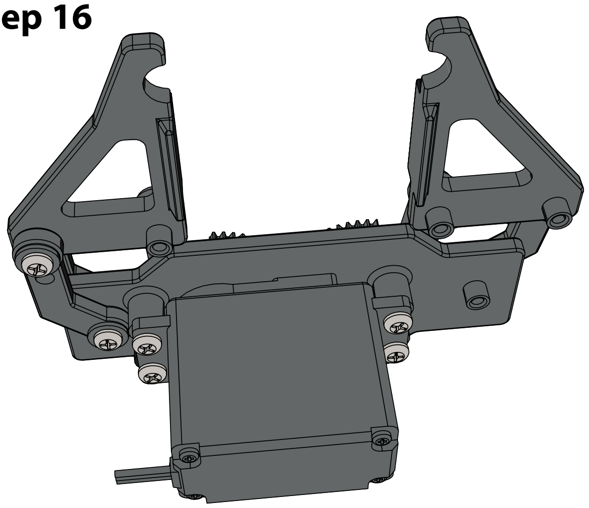
Step 14



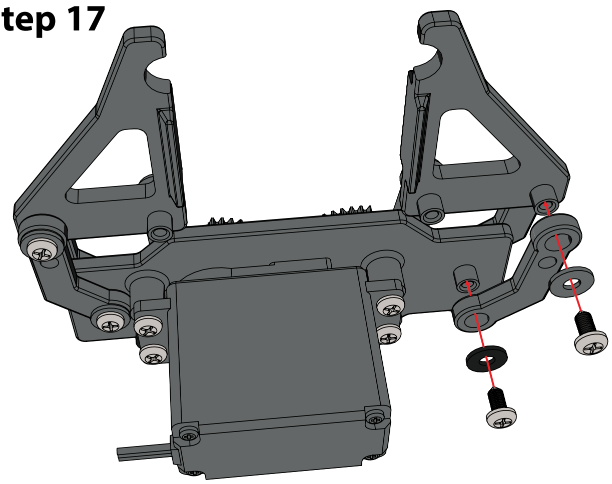
Step 15



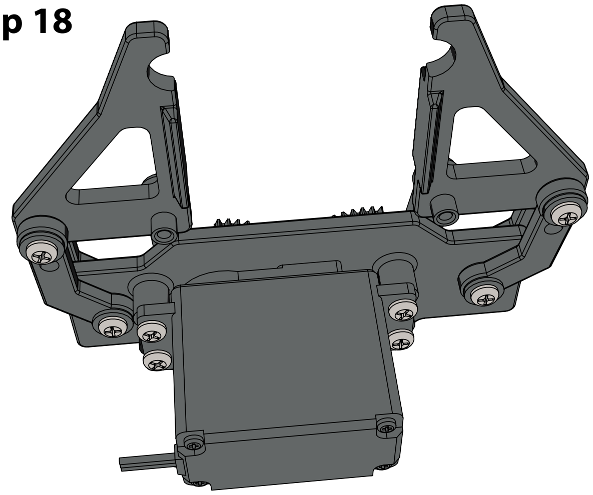
Step 16



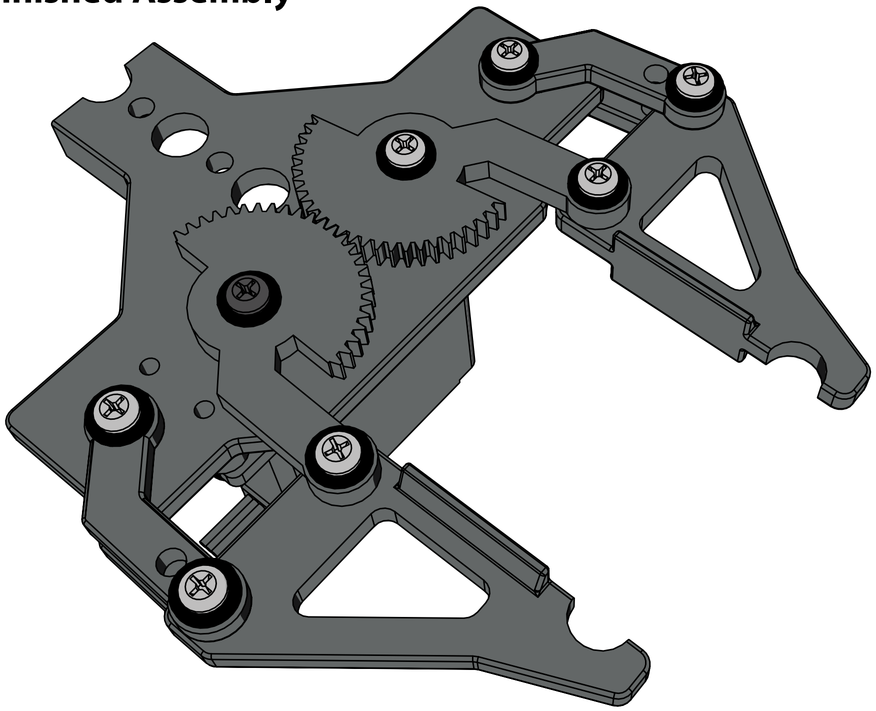
Step 17



Step 18

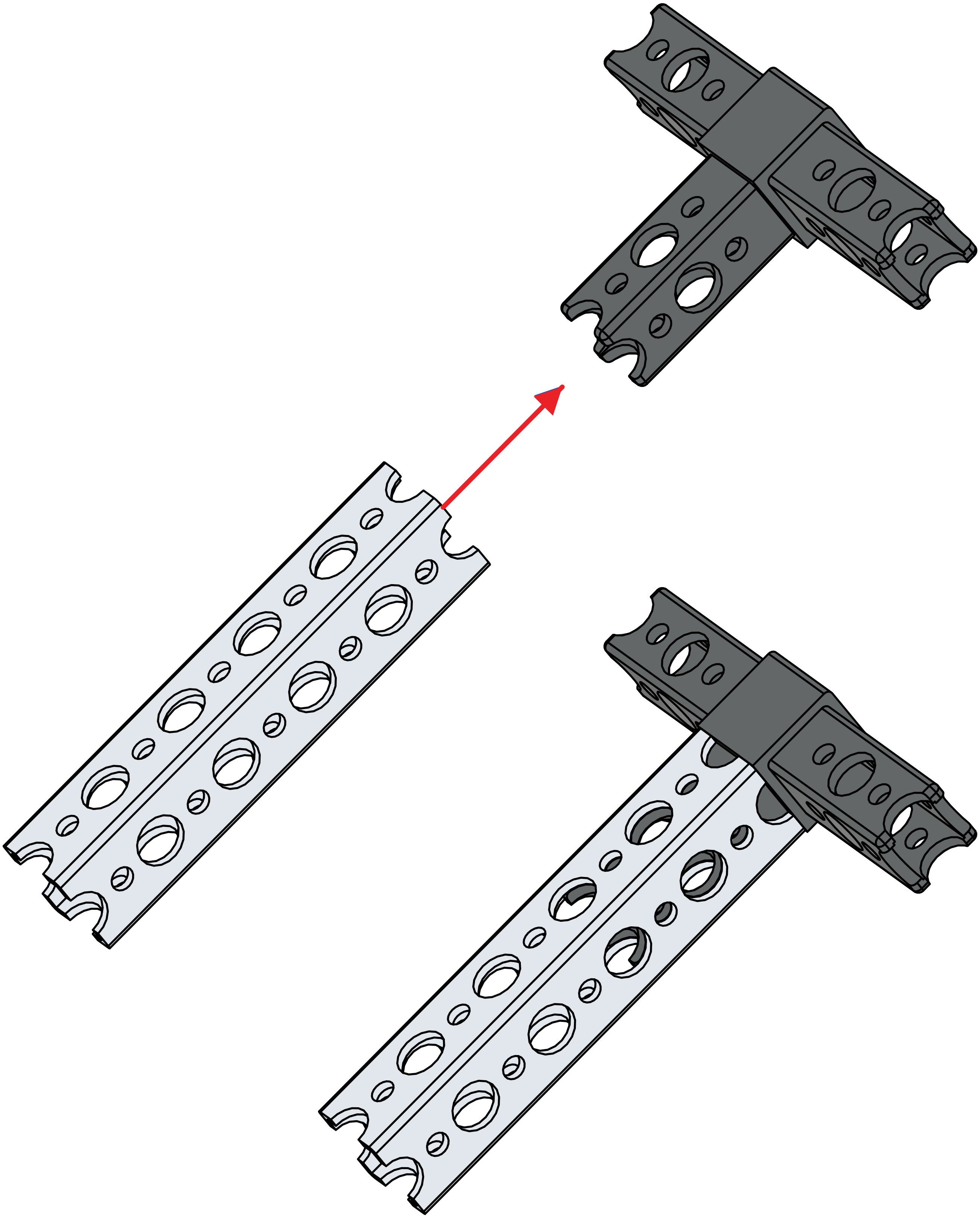


Finished Assembly

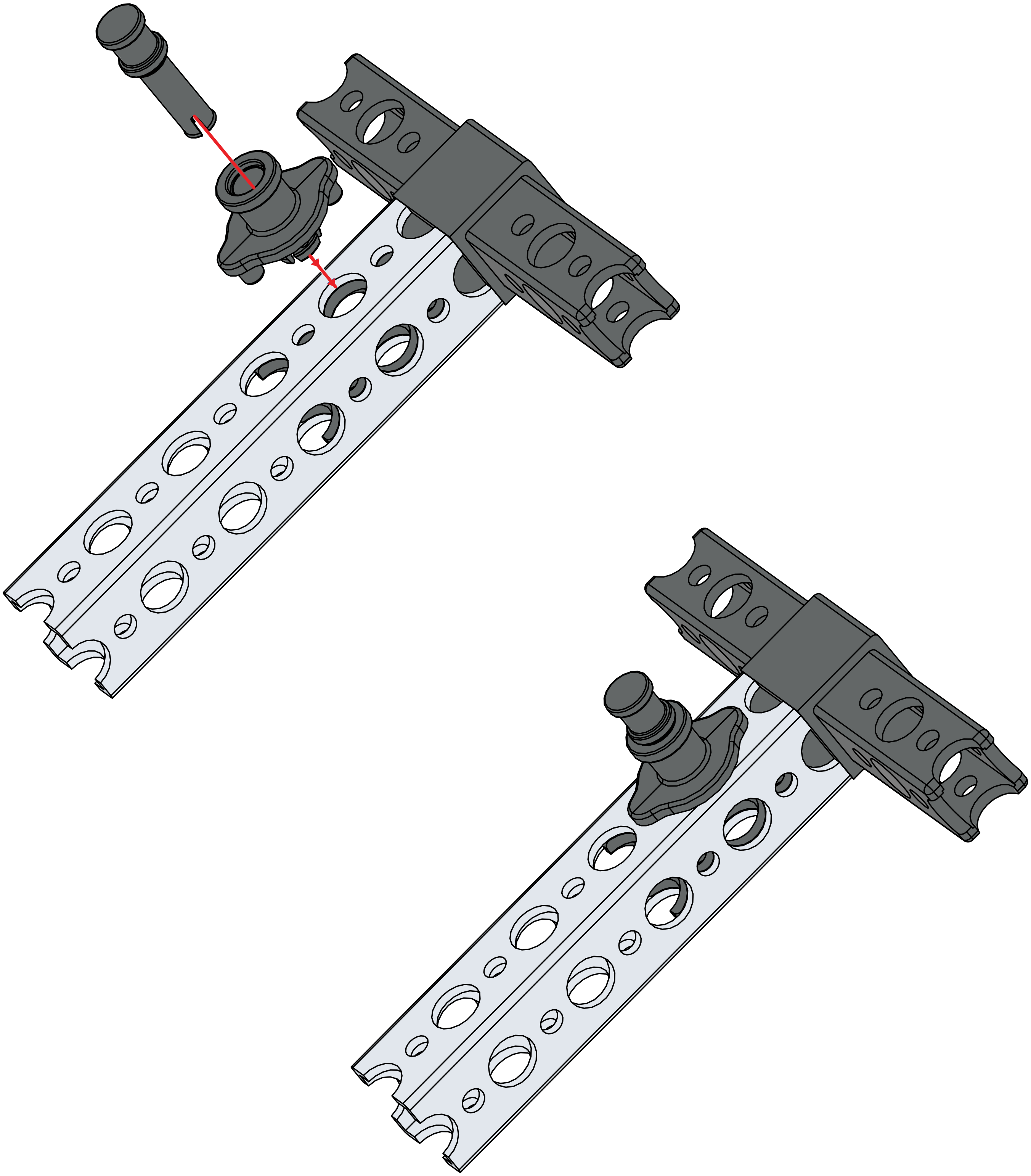


Construction Tips

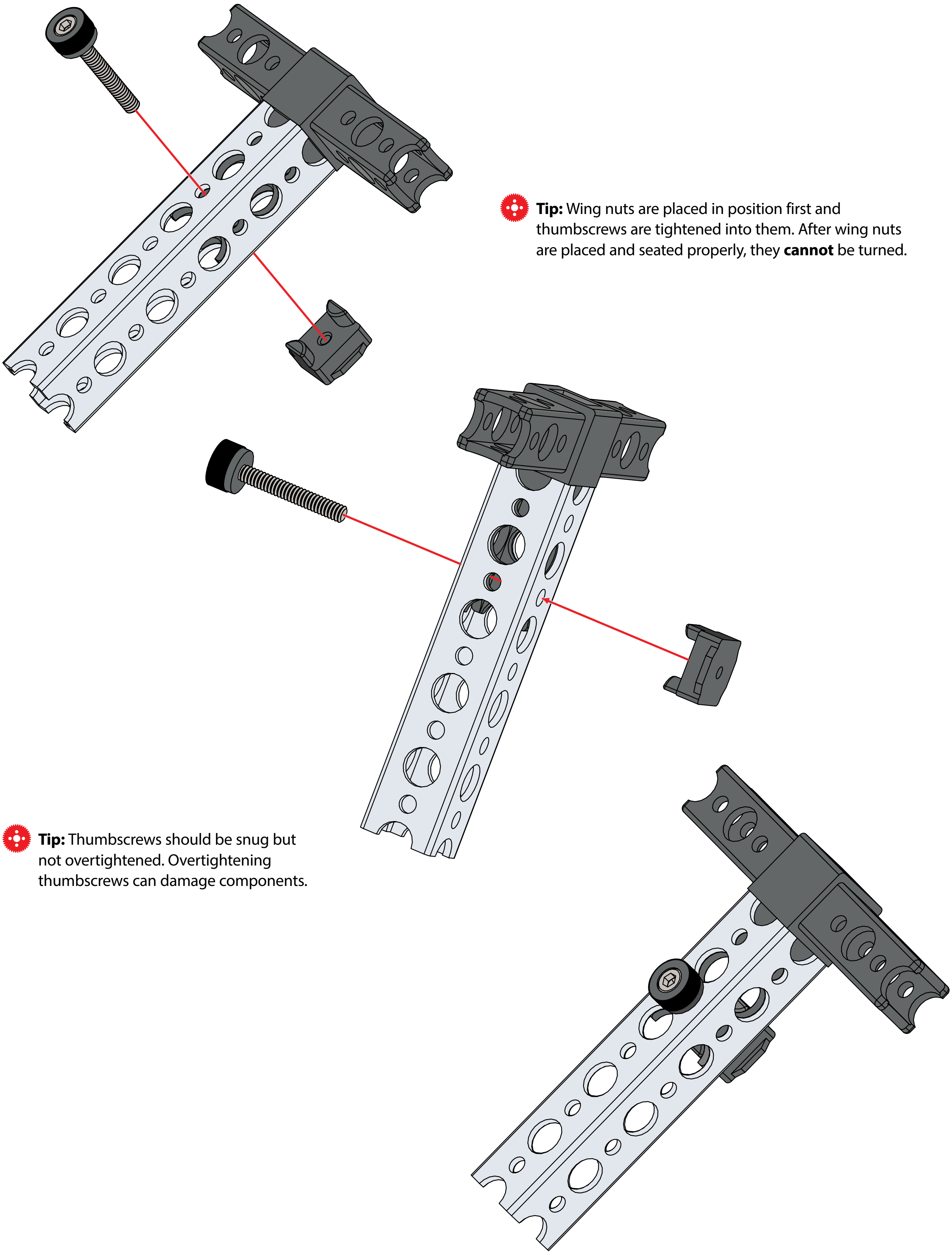
Connectors fit inside beams and come in 3-way, tee, 90-degree, end, extension, and straight beam block connector designs.



Quick rivet connectors and pegs are a quick option for securing connectors. Press the rivet in place on the beam and use the peg to spread the rivet to secure the connection. Using rivets on two sides of the connection will make it more stable.



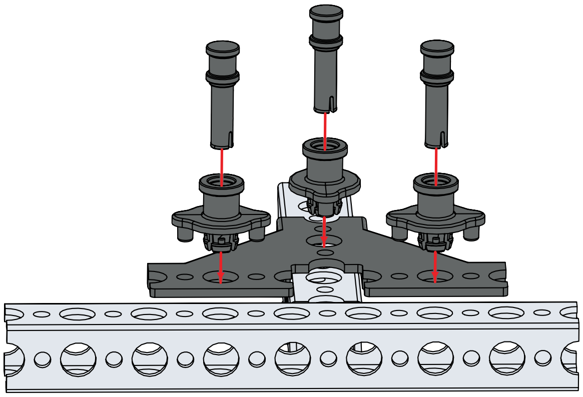
Joints can be made more permanent by using a thumbscrew and wing nut to secure the beams and connectors.

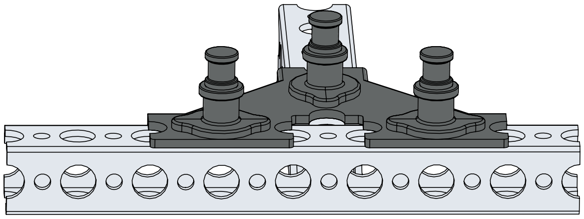


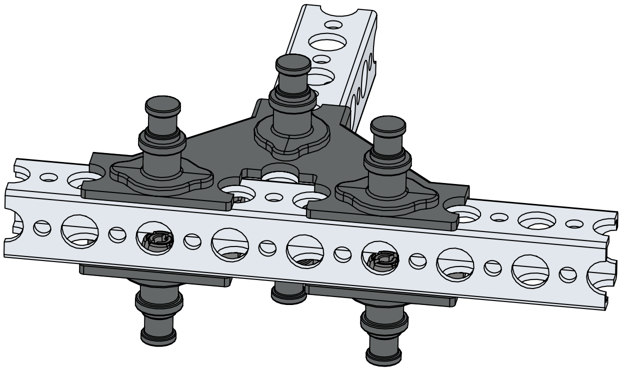
Tip: Wing nuts are placed in position first and thumbscrews are tightened into them. After wing nuts are placed and seated properly, they **cannot** be turned.

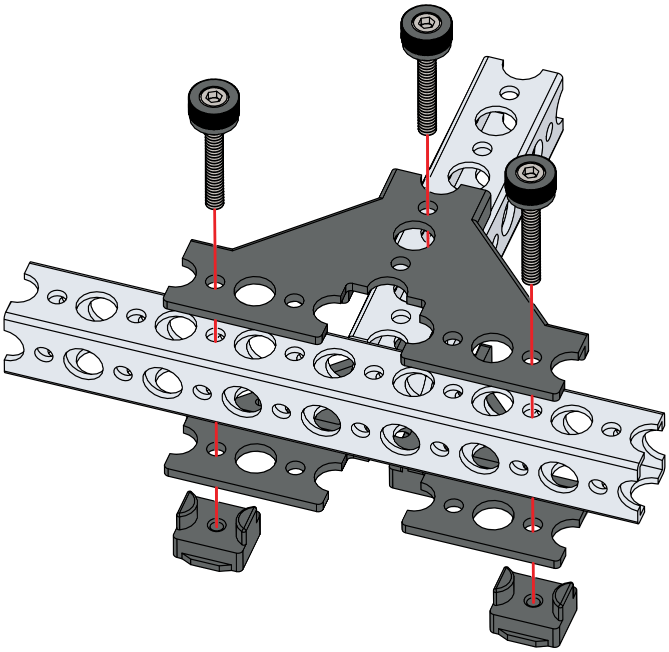
Tip: Thumbscrews should be snug but not overtightened. Overtightening thumbscrews can damage components.

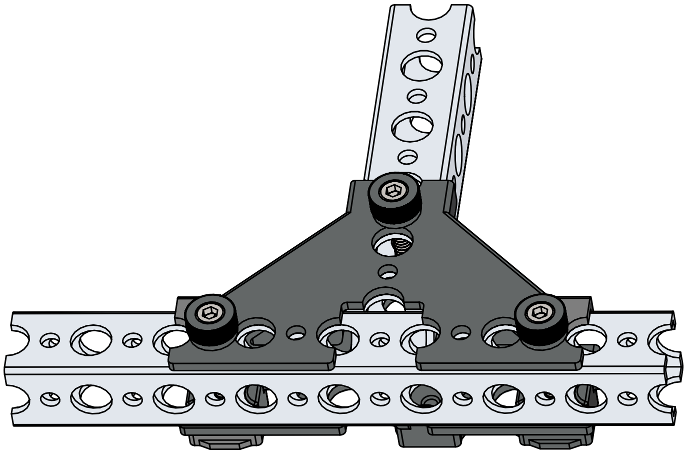
Brackets can also be used to connect beams. Brackets are available for a tee connection, 60-degree connection, or 90-degree connection. Brackets should be used in pairs, with two brackets on opposite sides of a beam. Brackets are secured using quick rivets and pegs or thumbscrews and wing nuts.



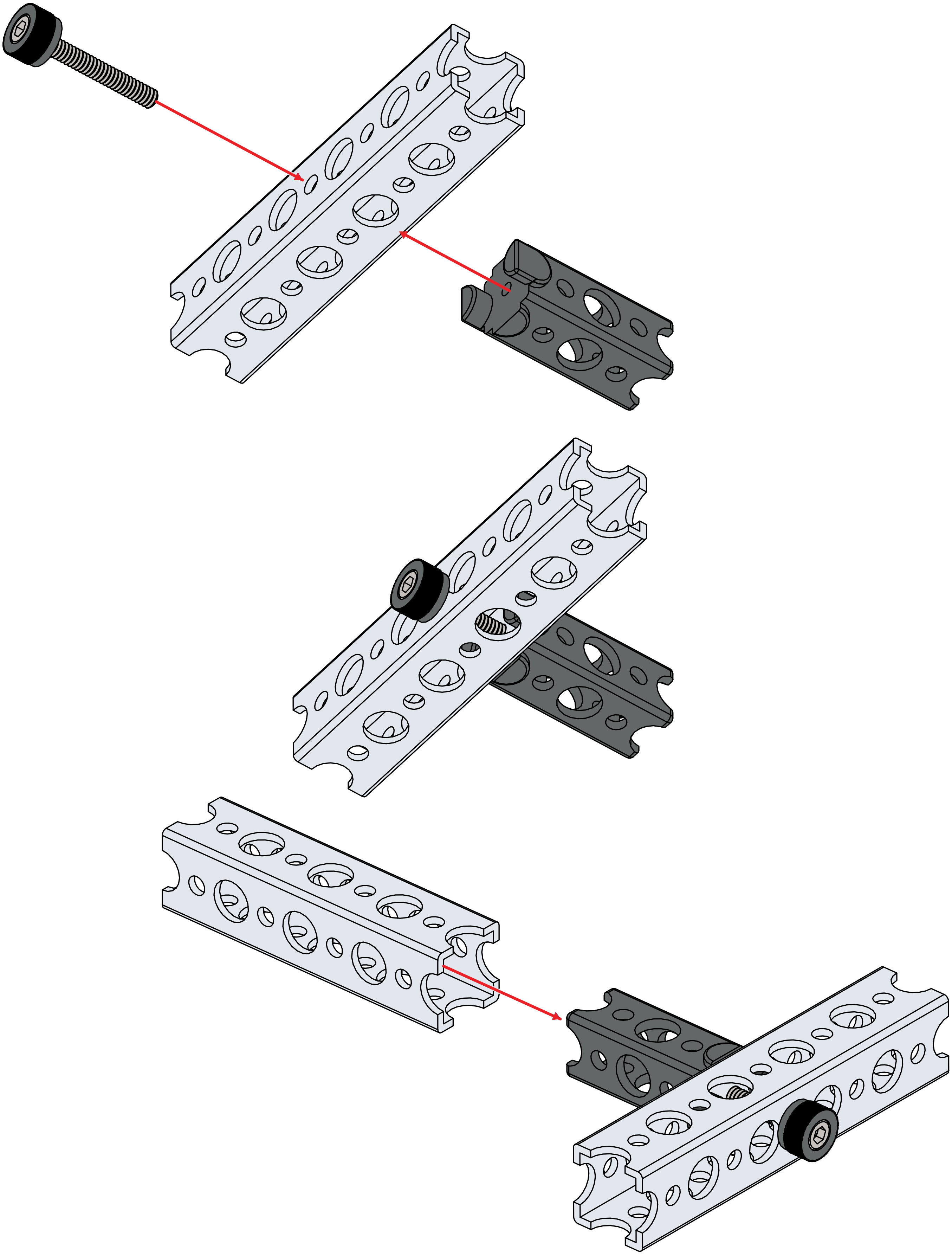




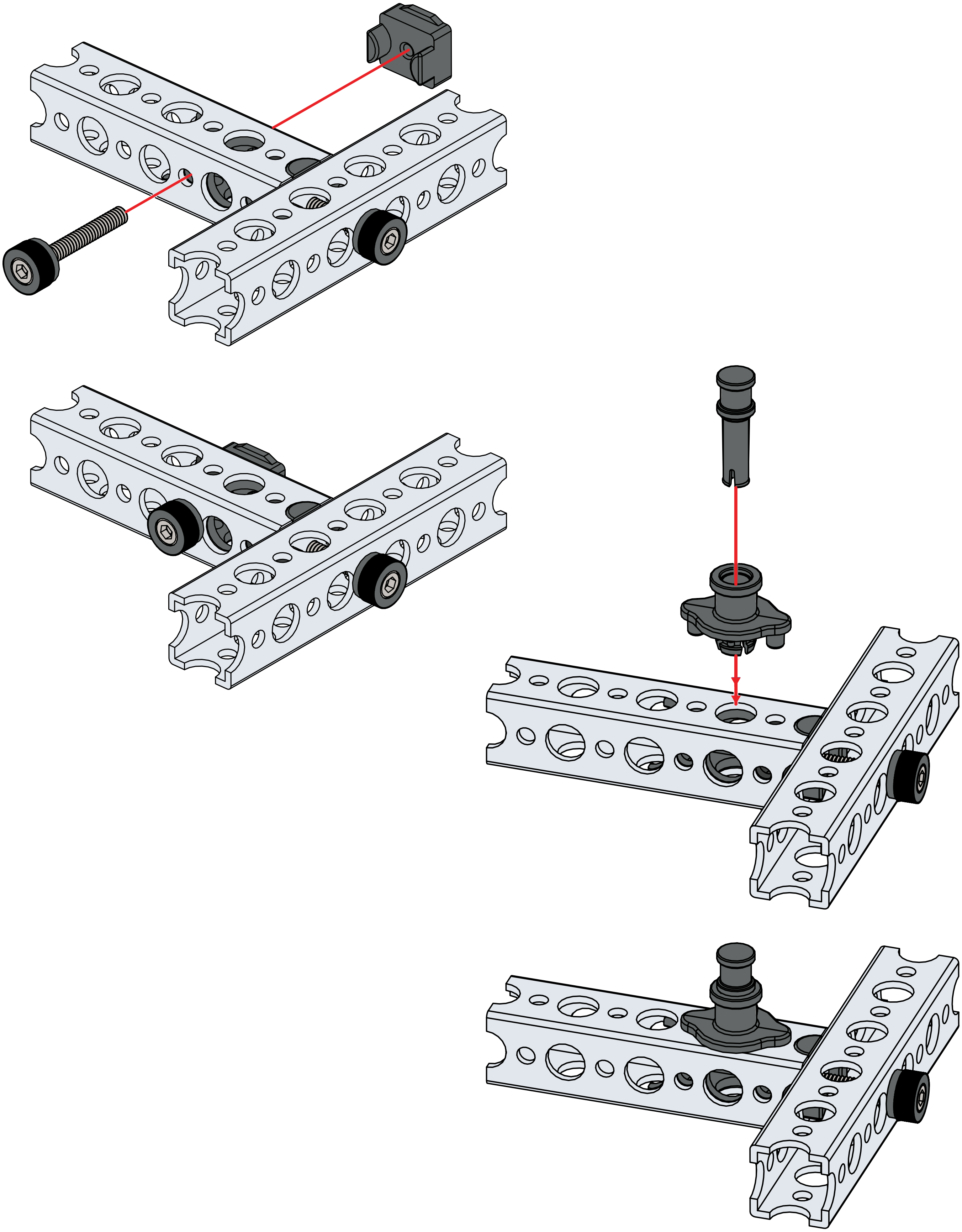




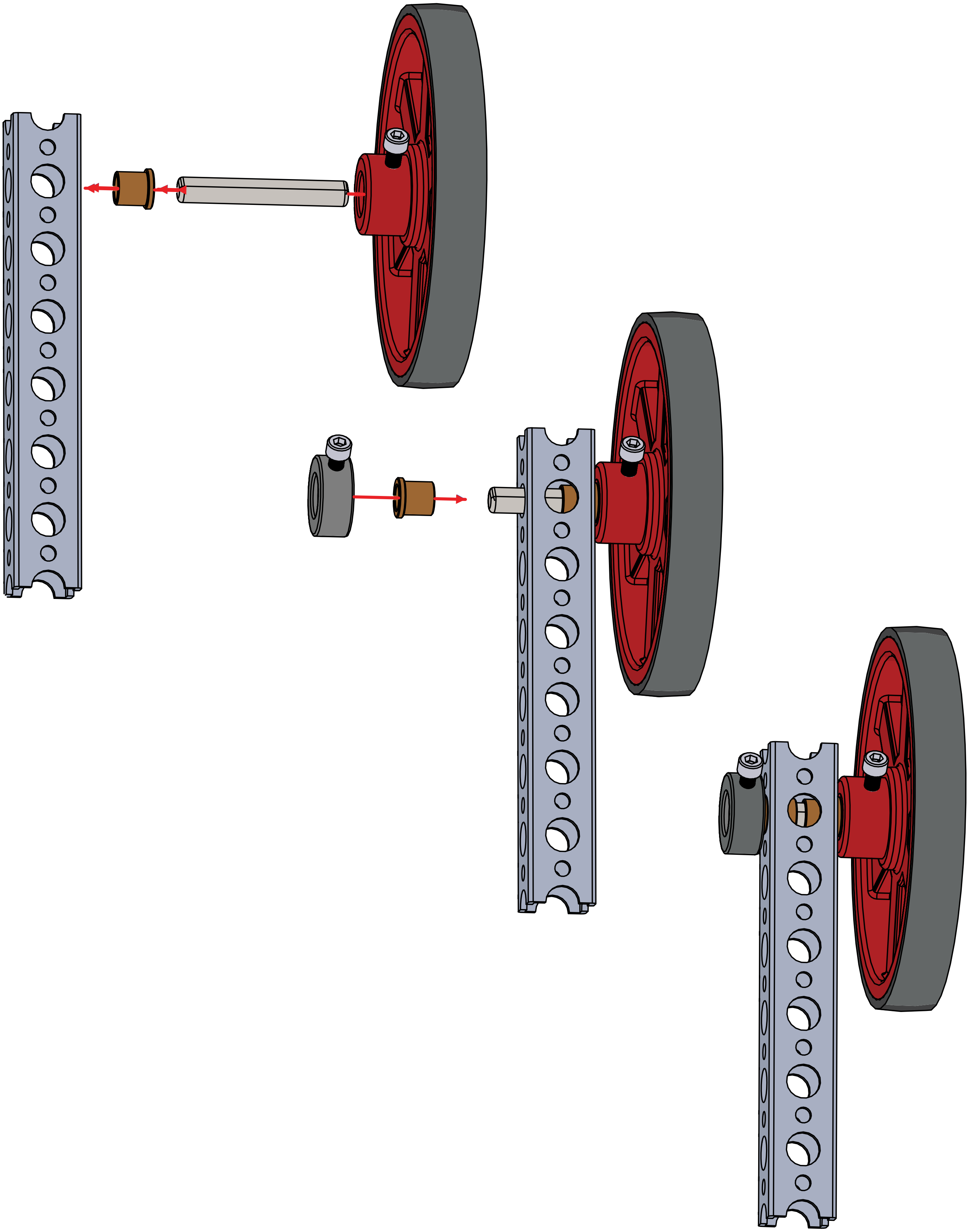
Beam end connectors, straight block beam connectors, and 90-degree cross block connectors are secured using a thumbscrew through the beam and into the connector.



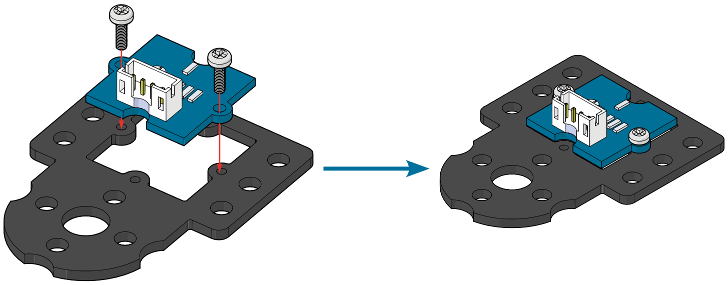
After the thumbscrew is used to secure the end of the connector, a quick rivet and peg or a thumbscrew and wing nut are used to secure the intersecting beam.



Anytime an axle is used, it should be supported at two points. Place a bronze bushing on opposite sides of a beam and place the axle through the bushings. Secure the axle to a D-shaft set collar, wheel, gear, or hub.

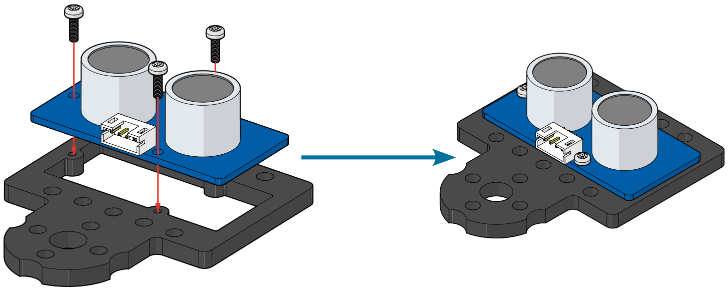


Line Finder Sensor Pack Assembly



The screws in this pack will self-thread into the plastic mounts. The 4-in-1 screwdriver that comes as part of the TETRIX robotics sets can be used to install the screws. Take care not to overtighten the screws upon installation or the plastic mount might be damaged.

Ultrasonic Sensor Pack Assembly



The screws in this pack will self-thread into the plastic mounts. The 4-in-1 screwdriver that comes as part of the TETRIX robotics sets can be used to install the screws. Take care not to overtighten the screws upon installation or the plastic mount might be damaged.

Activity 6: Build the PULSE Codee Bot

Introduction

You need a robot. Because the focus of this guide is on working with PULSE and the TETRIX Ardublockly software, you do not need a complicated robot. With that in mind, you will create the PULSE Codee Bot. The Codee Bot is meant to be simple, easy to build, and exactly what you need for the purposes of this guide without any unnecessary parts.

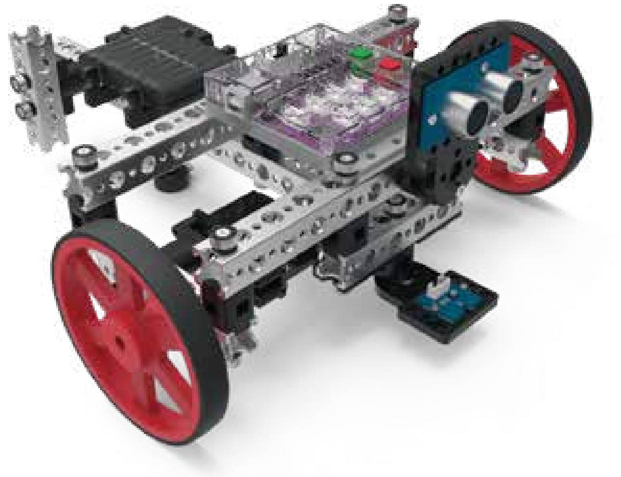
That being said, everything you learn with this basic bot can be transferred to a more complicated bot as you continue on your robotics journey. The Codee Bot uses two DC motors and PRIME wheels mounted on either side to make a perfect test vehicle for your work with PULSE.

You will start with a basic drive frame and add more to it in later activities. You can complete this build even if you have little to no experience with metal-based building systems.

Teacher Note: Other factors can affect building time, including matters such as set organization and whether the builder has a partner. The building time is only an estimate. Actual time might vary.

Building Time Expectations

30 minutes



Real-World Link

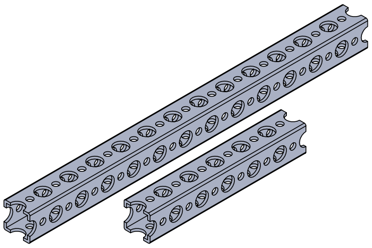
In manufacturing and assembly, it's important to be able to follow instructions exactly. Try making a piece of furniture without following the instructions and it will likely have to be redone. Being able to follow a detailed procedure of steps is an important skill that is used in a multitude of real-life applications.

Careers: machine operator, roboticist, engineer

STEM Connections

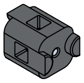
- Science
  - o Wheel and axle
  - o Screw
- Technology
  - o Materials
  - o Fasteners
- Engineering
  - o Machine design
  - o Construction
- Math
  - o Length
  - o Diameter

Step 1 Parts Needed



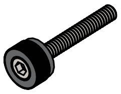
Beams

Part No.	Part Name	Quantity
40206	TETRIX PRIME 13-Hole Square Beam	2
40203	TETRIX PRIME 6-Hole Square Beam	2



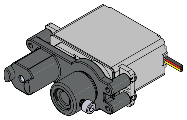
External Connectors

Part No.	Part Name	Quantity
40217	TETRIX PRIME 90-Degree Cross Block Connector	5



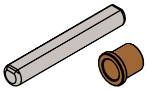
Connecting Hardware

Part No.	Part Name	Quantity
40323	TETRIX PRIME Thumbscrew	11



Wheels, Gears, & Servos

Part No.	Part Name	Quantity
40379	DC Motor 44298 with Servo Mounting Bracket 40232 (Assembly instructions on pages 54-55)	2

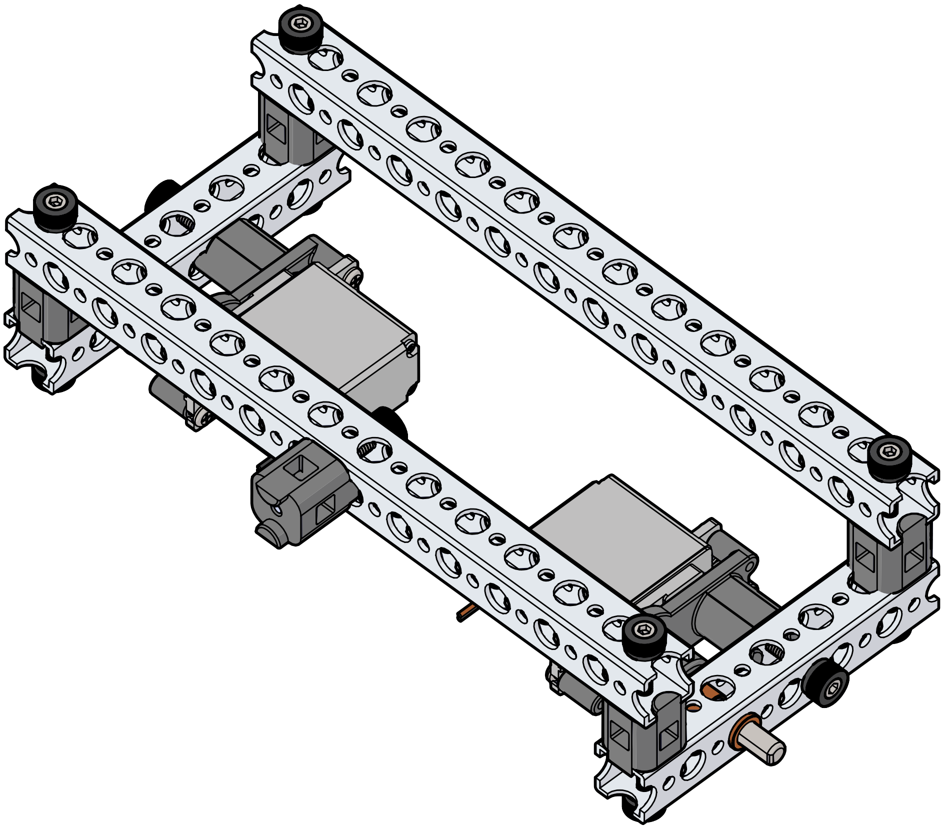


Wheel, Gear, & Servo Hardware

Part No.	Part Name	Quantity
40226	TETRIX PRIME 40 mm Steel Axle	2
40227	TETRIX PRIME 8 mm x 6 mm Bronze Bushing	4

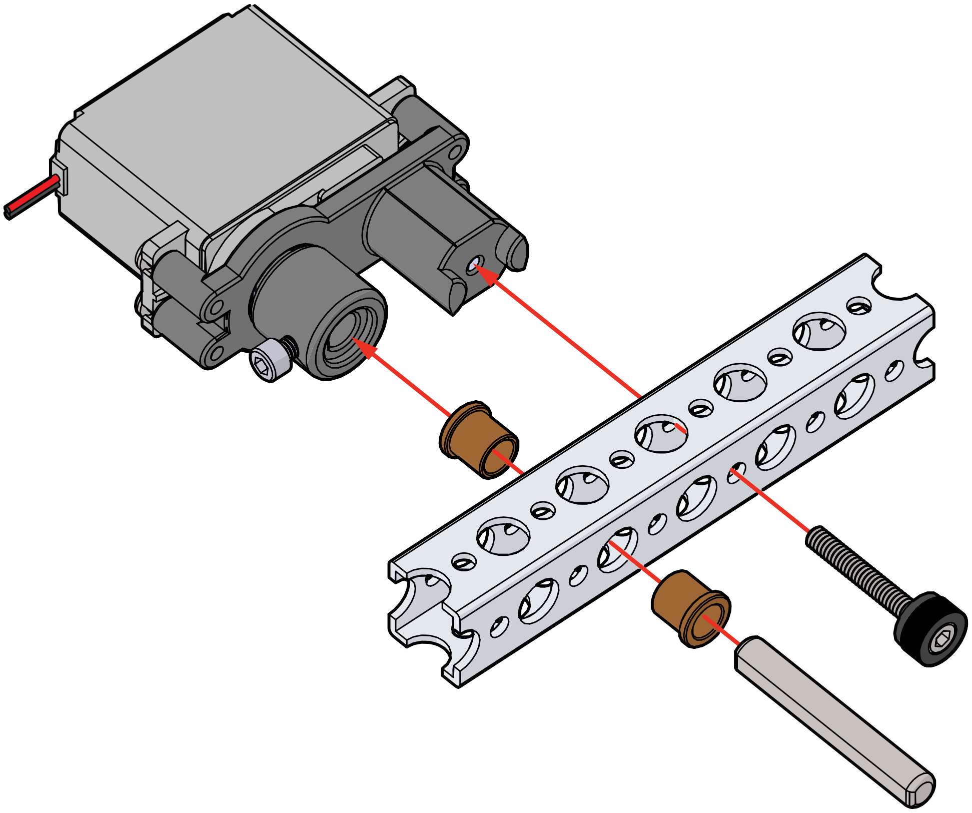
Tip: See page 48 for help with identifying beam elements. To identify TETRIX PRIME square beams, count the small holes.

Partial assembly should look like this.

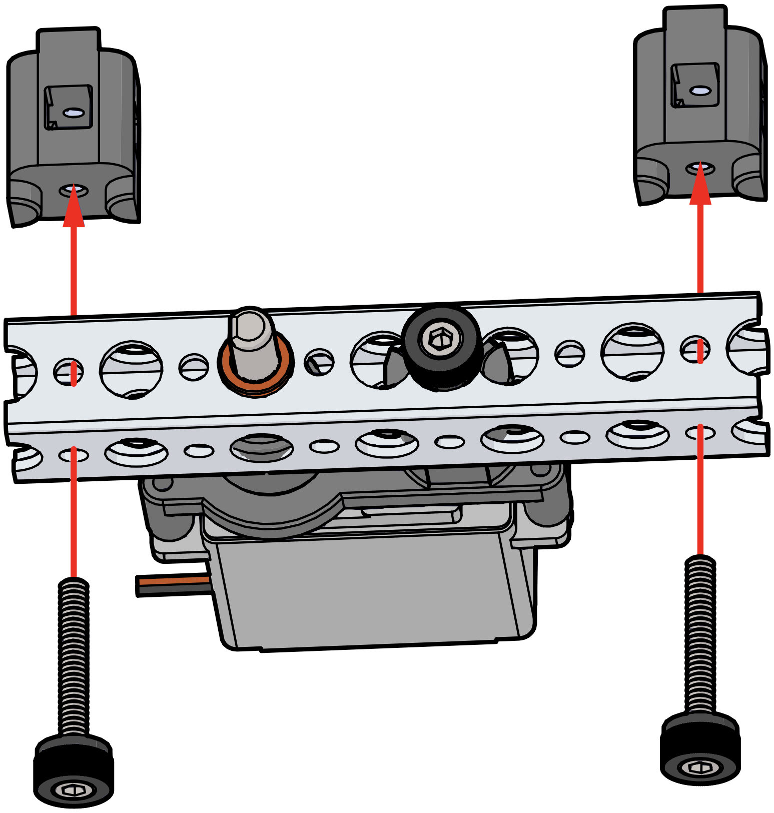


Step 1.0

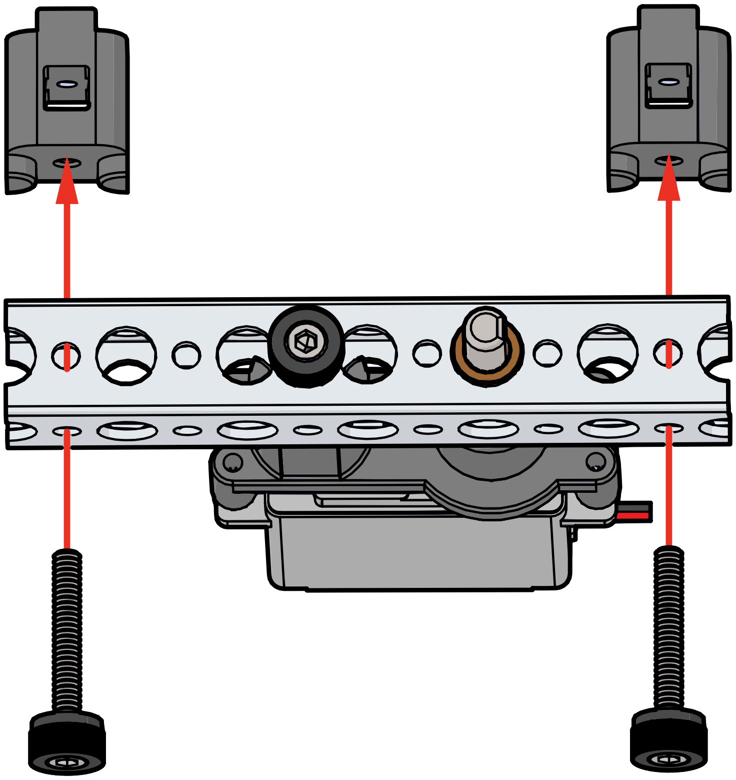
Build two like this.



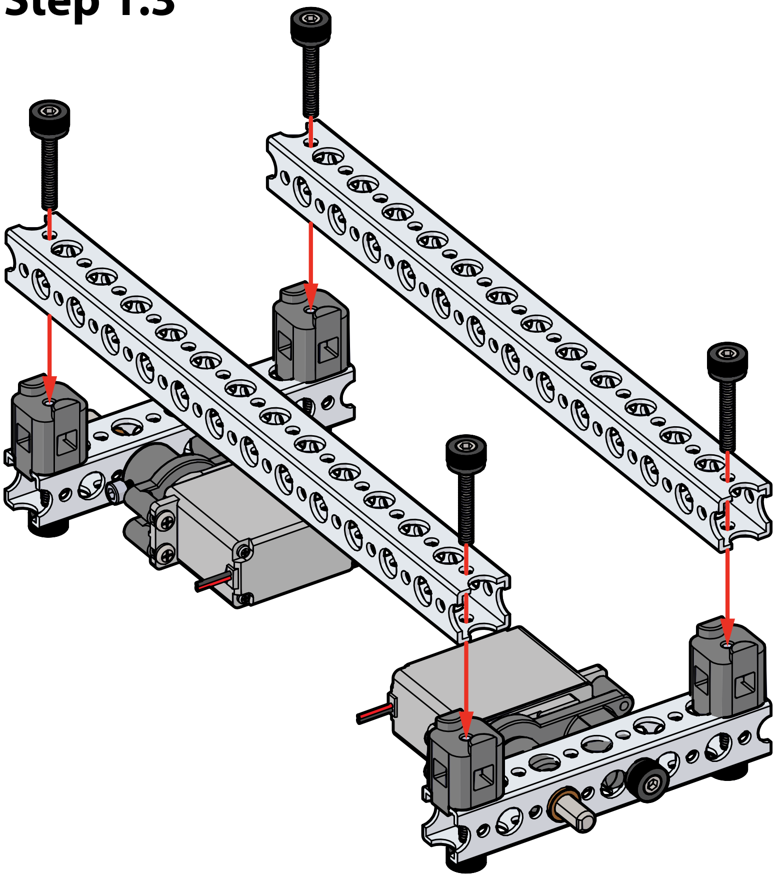
Step 1.1



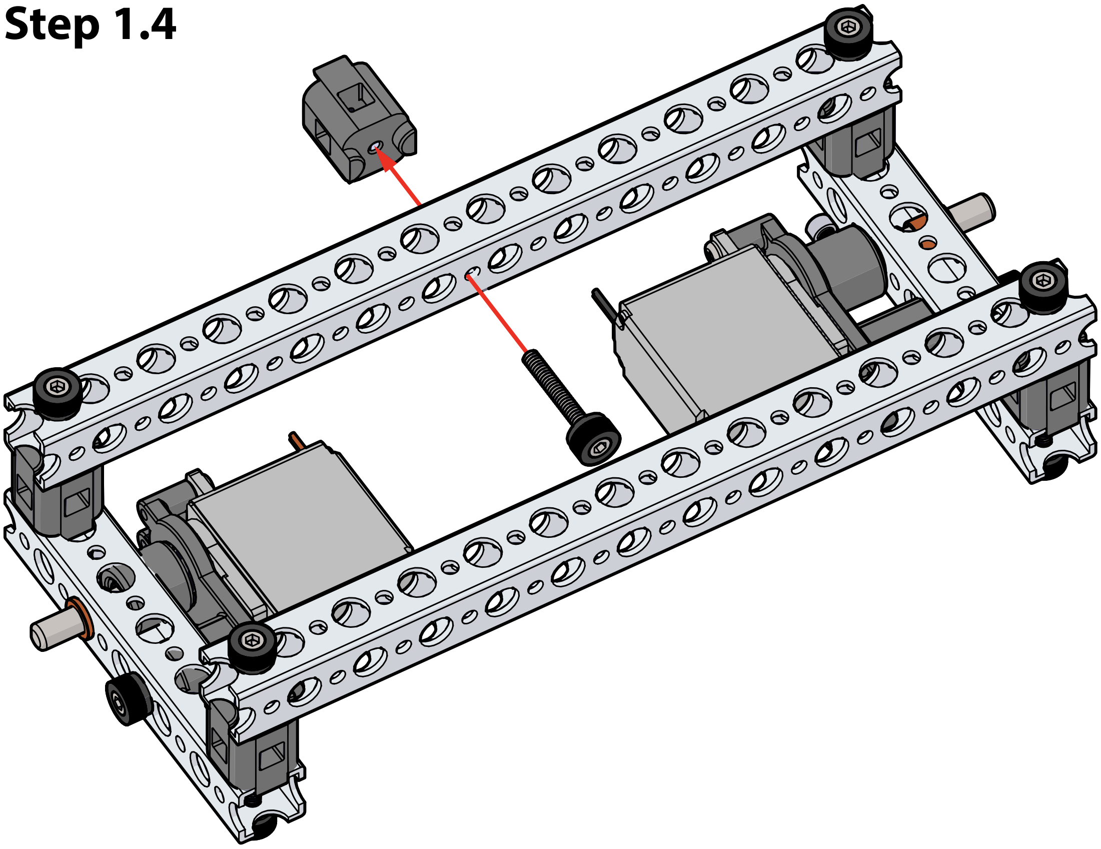
Step 1.2



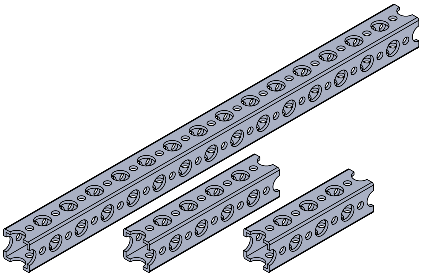
Step 1.3



Step 1.4

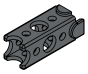


Step 2 Parts Needed



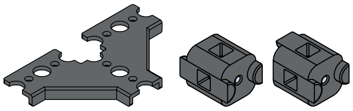
Beams

Part No.	Part Name	Quantity
40207	TETRIX PRIME 15-Hole Square Beam	1
40202	TETRIX PRIME 5-Hole Square Beam	1
40201	TETRIX PRIME 4-Hole Square Beam	1



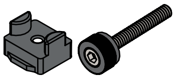
Internal Connectors

Part No.	Part Name	Quantity
40214	TETRIX PRIME Beam End Connector	1



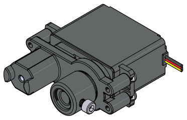
External Connectors

Part No.	Part Name	Quantity
40210	TETRIX PRIME Tee Beam Bracket	1
40216	TETRIX PRIME Straight Block Beam Connector	2
40217	TETRIX PRIME 90-Degree Cross Block Connector	2



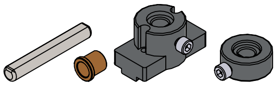
Connecting Hardware

Part No.	Part Name	Quantity
40221	TETRIX PRIME Wing Nut	1
40323	TETRIX PRIME Thumbscrew	10



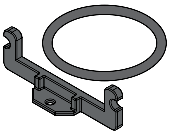
Wheels, Gears, & Servos

Part No.	Part Name	Quantity
40538	Standard-Scale Servo Motor 40538 with Servo Mounting Bracket 40232 (Assembly instructions on pages 54-55)	1



Wheel, Gear, & Servo Hardware

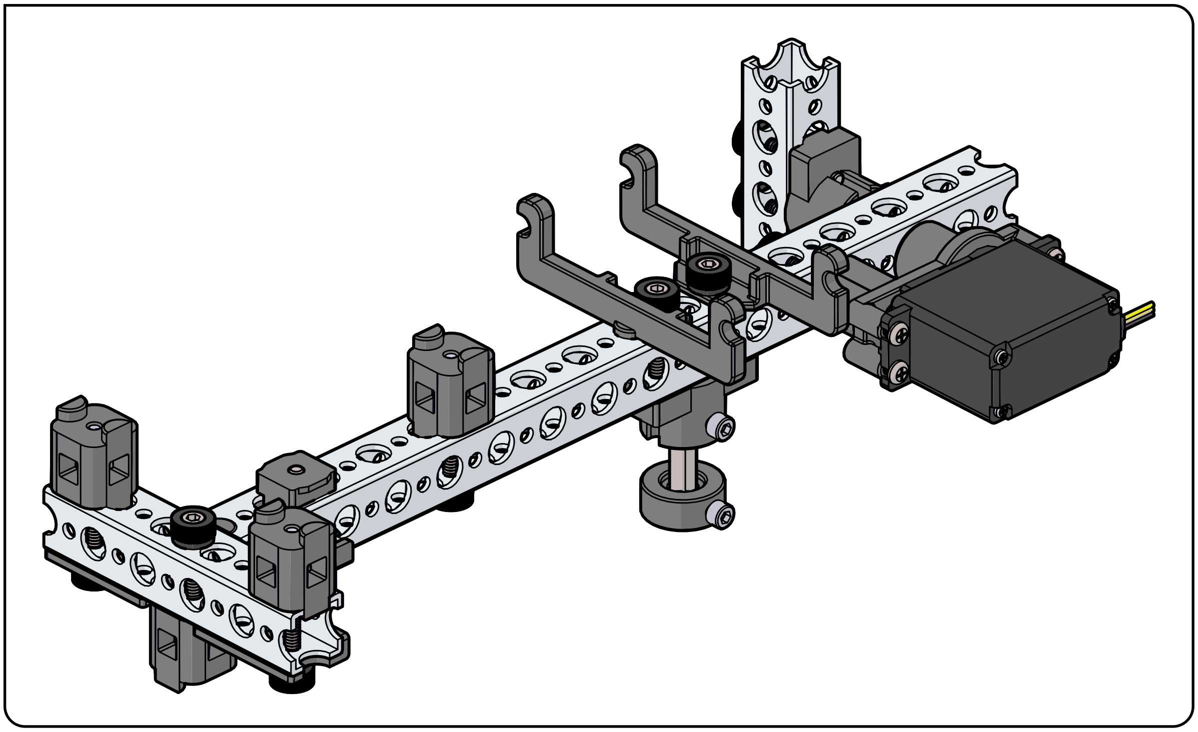
Part No.	Part Name	Quantity
40226	TETRIX PRIME 40 mm Steel Axle	2
40227	TETRIX PRIME 8 mm x 6 mm Bronze Bushing	2
40228	TETRIX PRIME Beam Attachment Hub	2
40229	TETRIX PRIME D-Shaft Set Collar	1

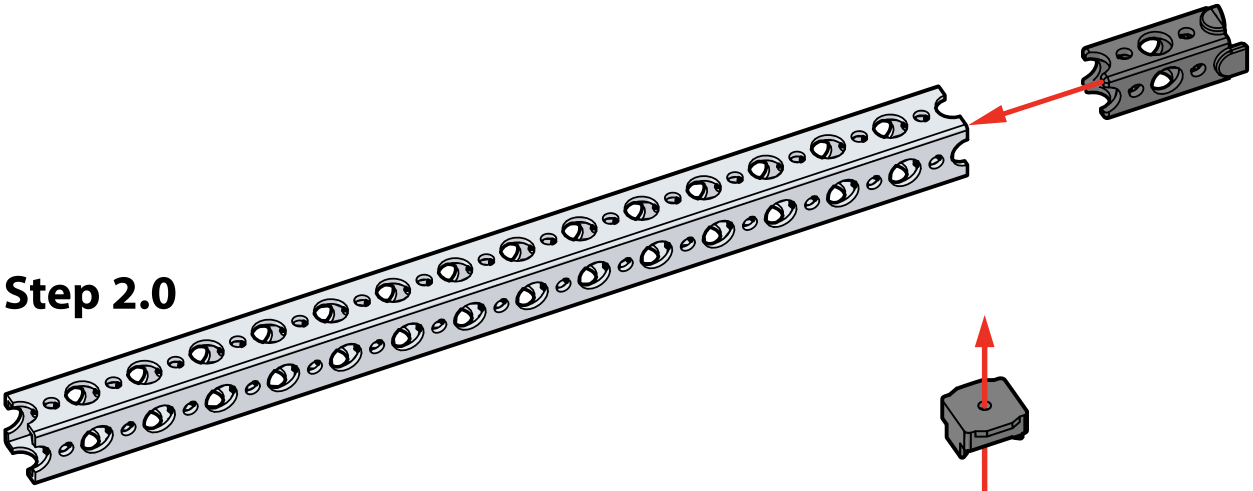


Batteries & Hardware

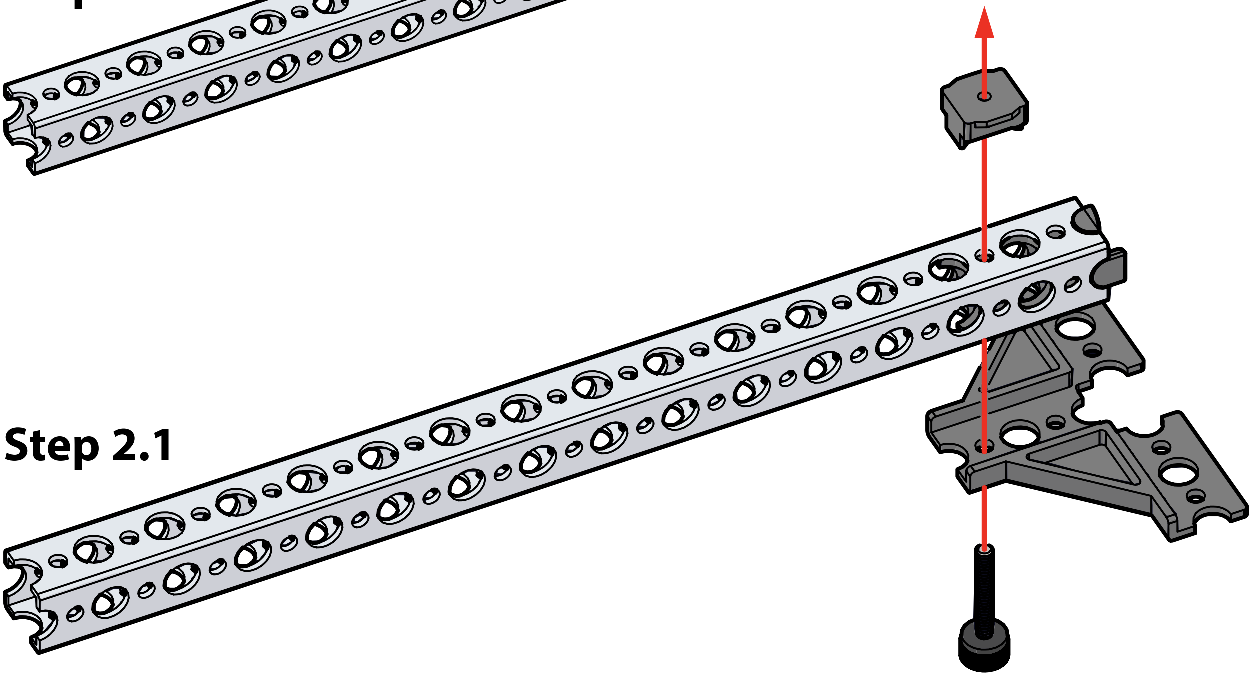
Part No.	Part Name	Quantity
40236	TETRIX PRIME Battery Mount Bracket	2

Partial assembly should look like this.



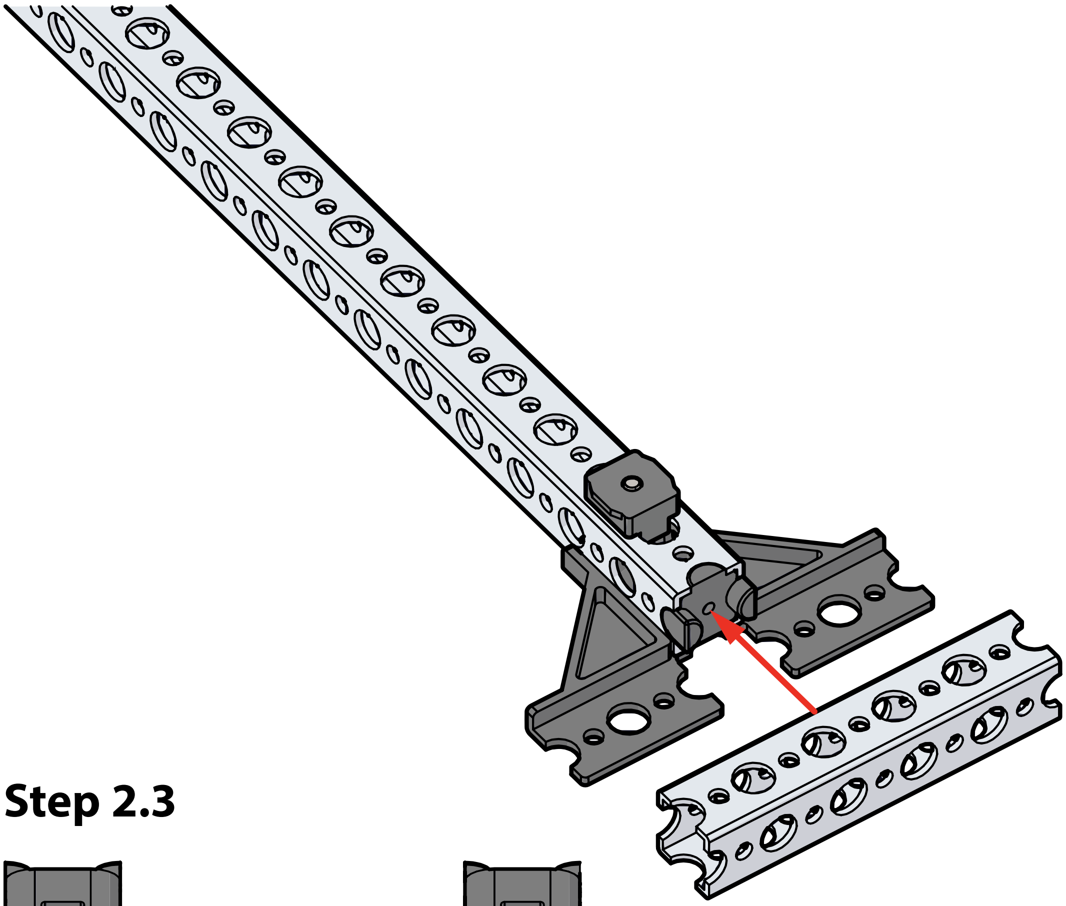


Step 2.0

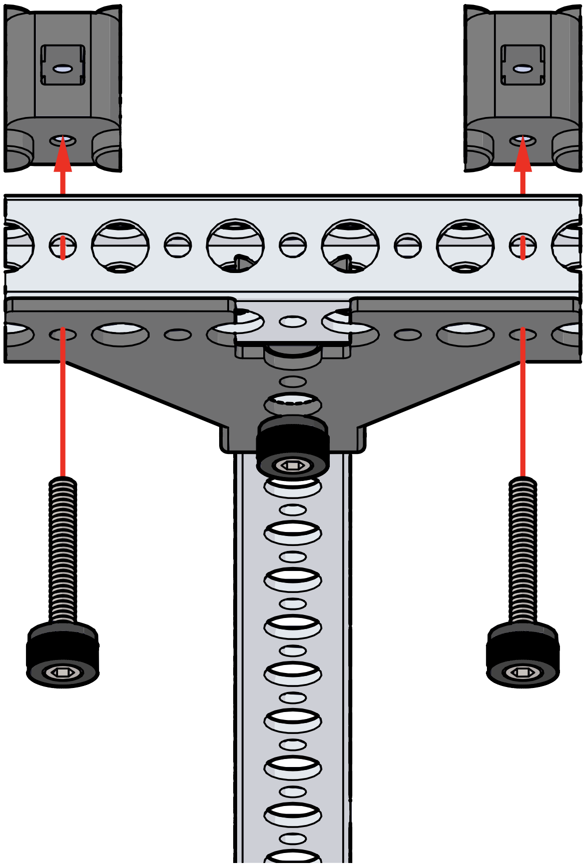


Step 2.1

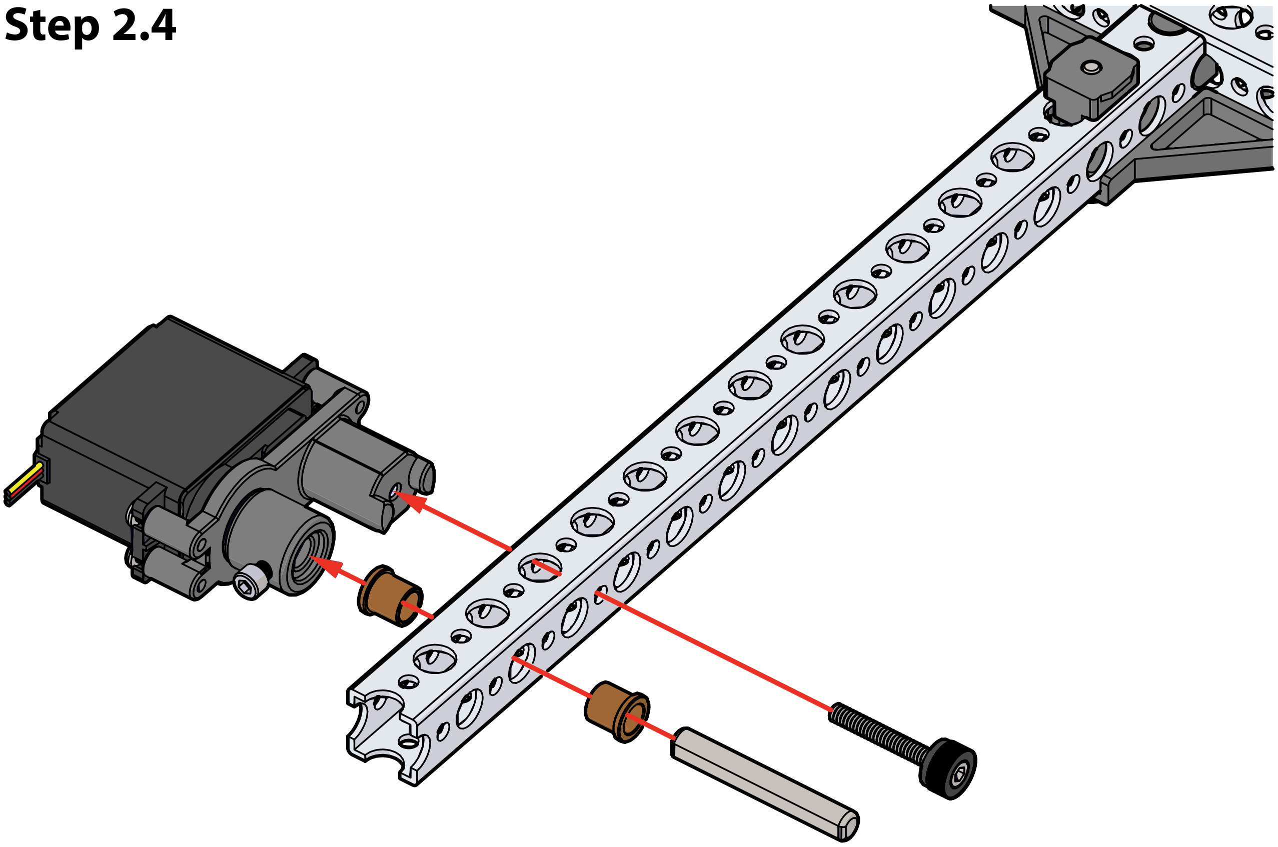
Step 2.2



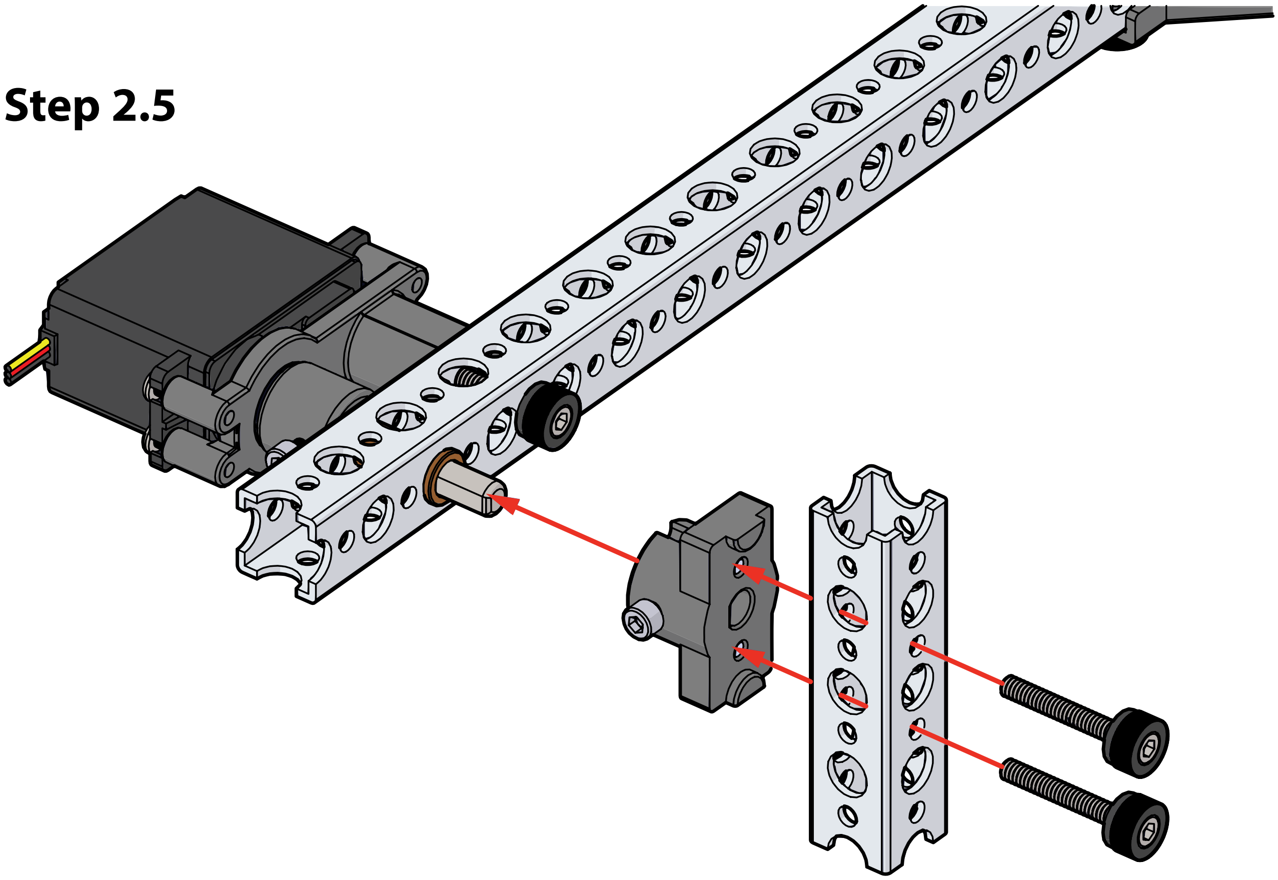
Step 2.3



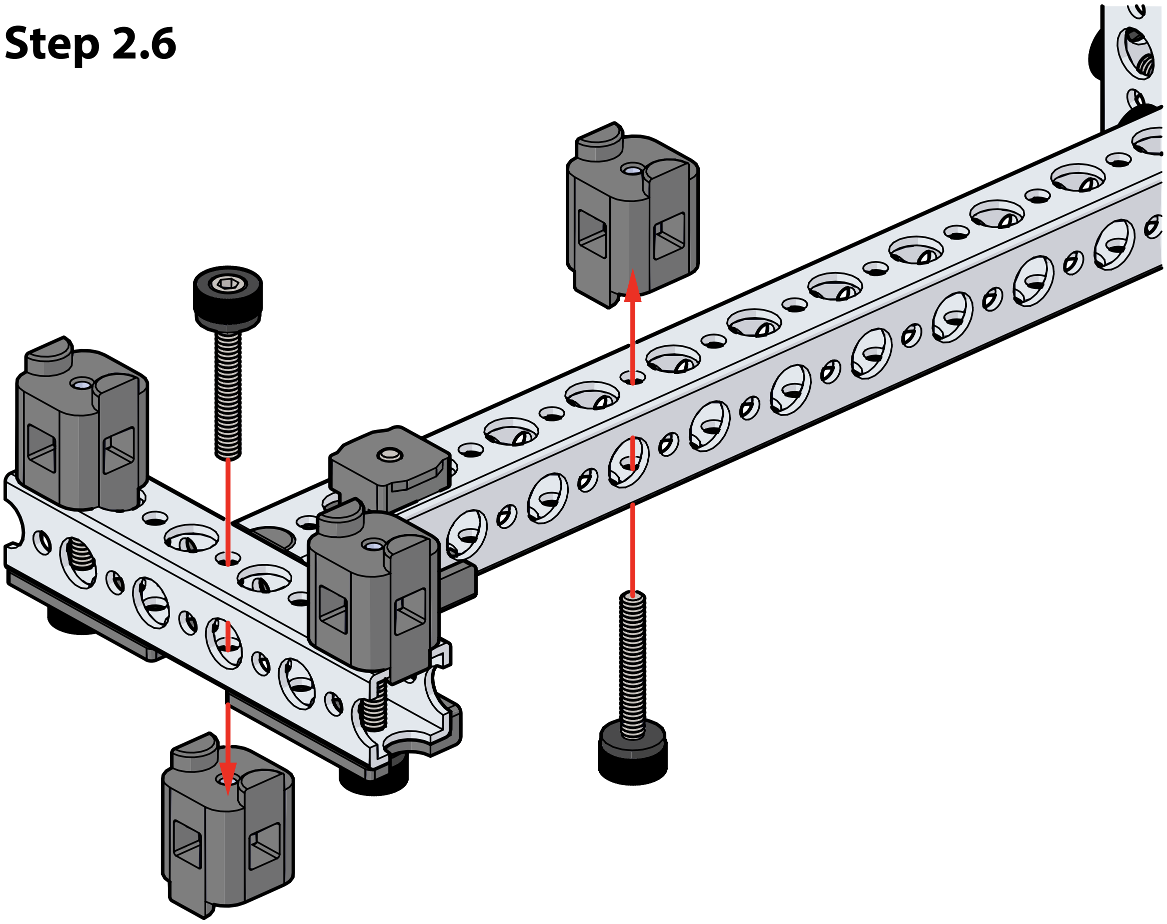
Step 2.4



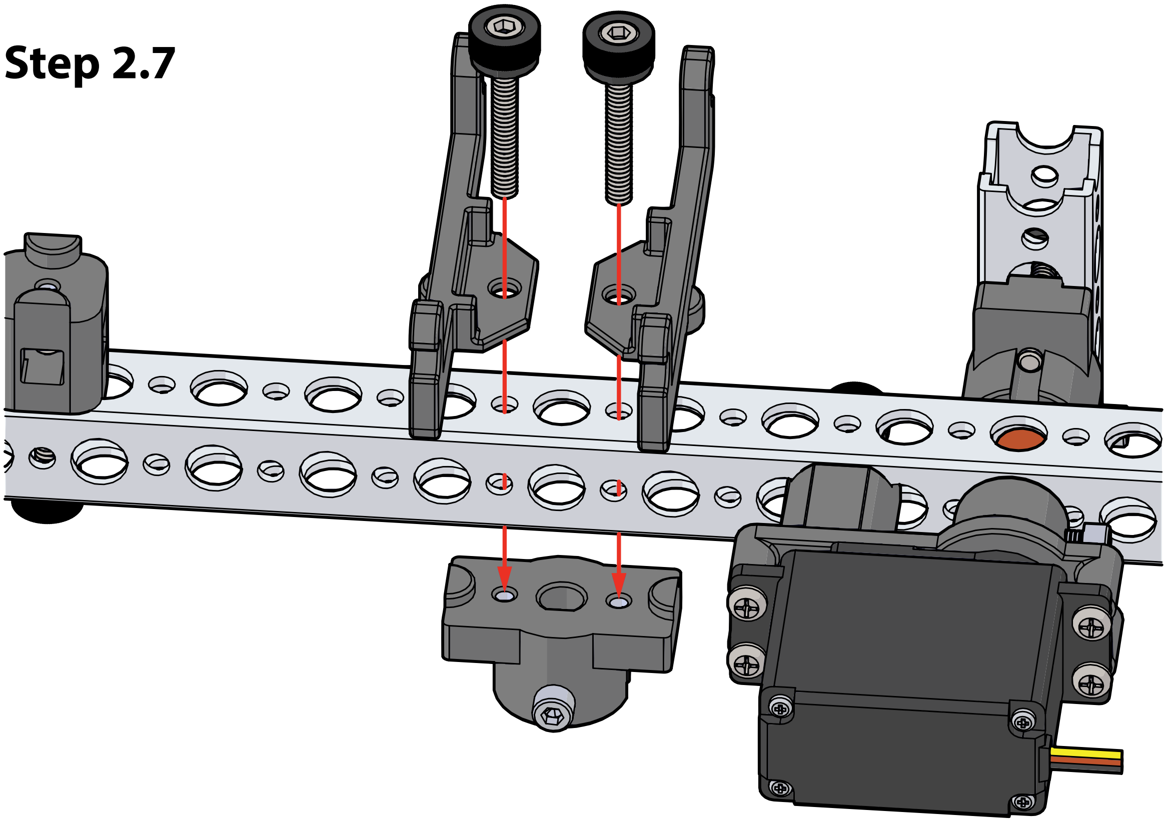
Step 2.5



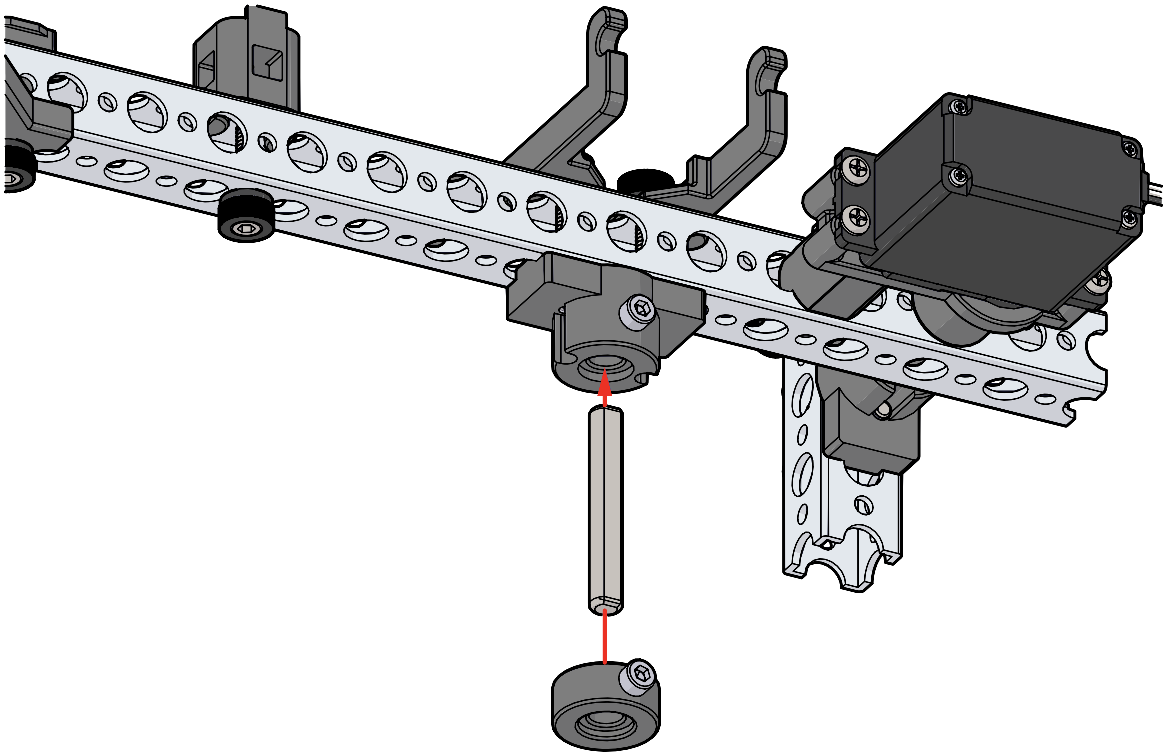
Step 2.6



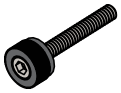
Step 2.7



Step 2.8

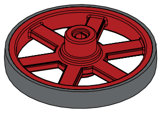


Step 3 Parts Needed



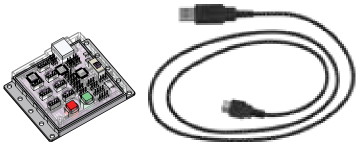
Connecting Hardware

Part No.	Part Name	Quantity
40323	TETRIX PRIME Thumbscrew	3



Wheels, Gears, & Servos

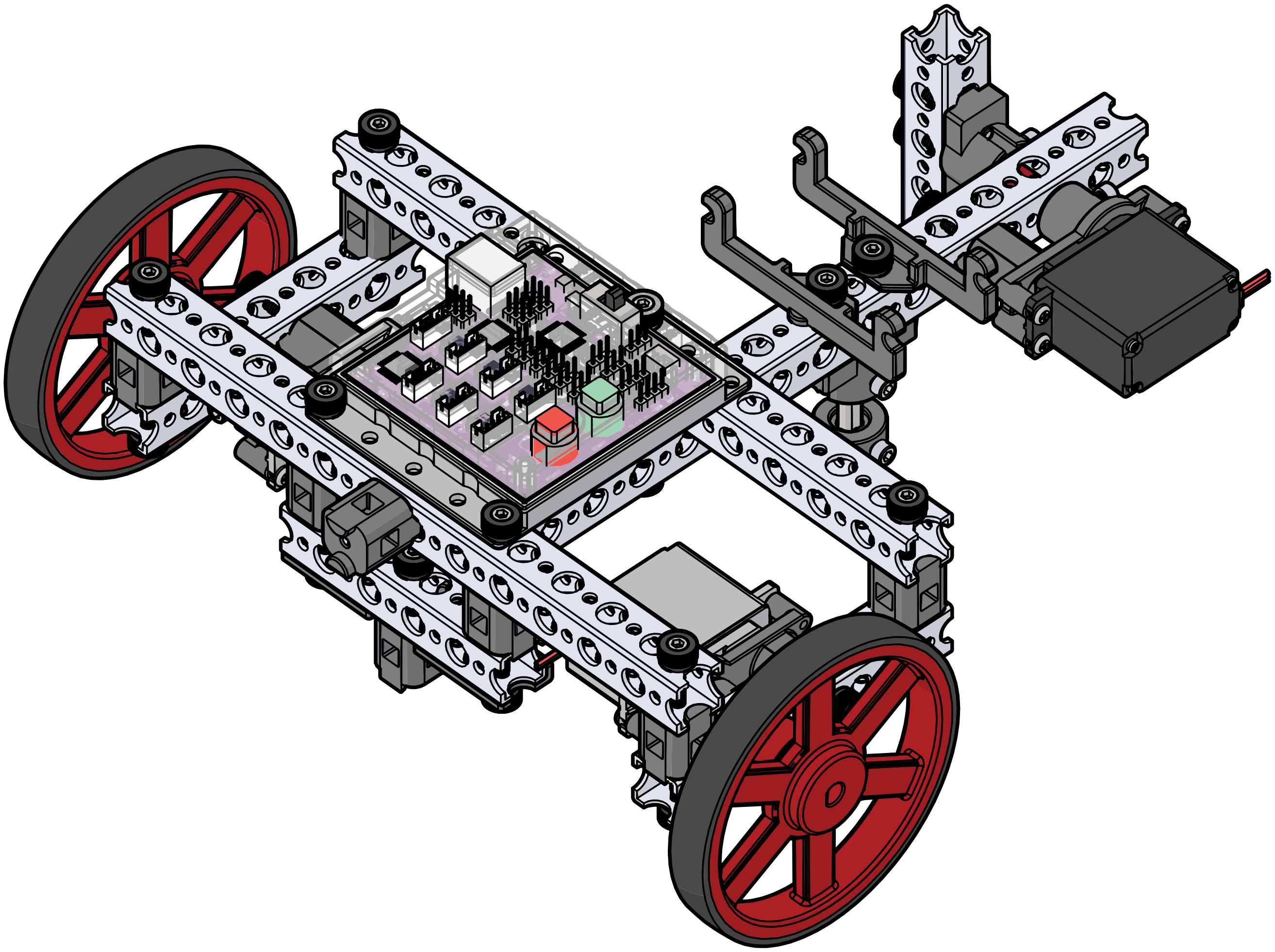
Part No.	Part Name	Quantity
40222	TETRIX PRIME Wheel with Tire	2



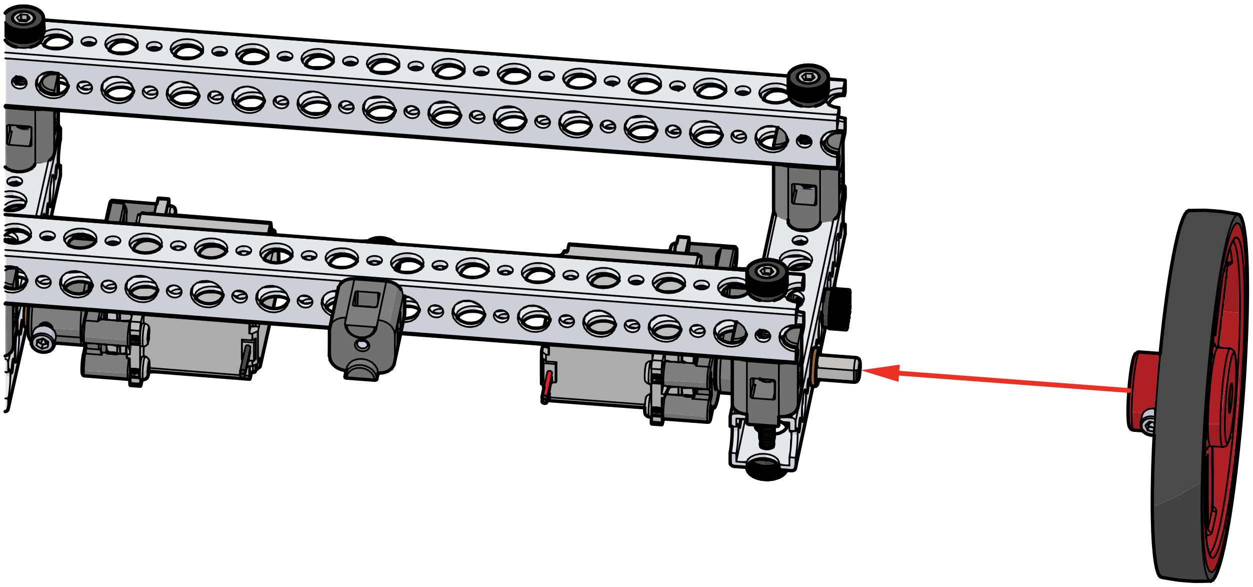
Electronics & Control

Part No.	Part Name	Quantity
44268	TETRIX PULSE Robotics Controller with USB cable	1

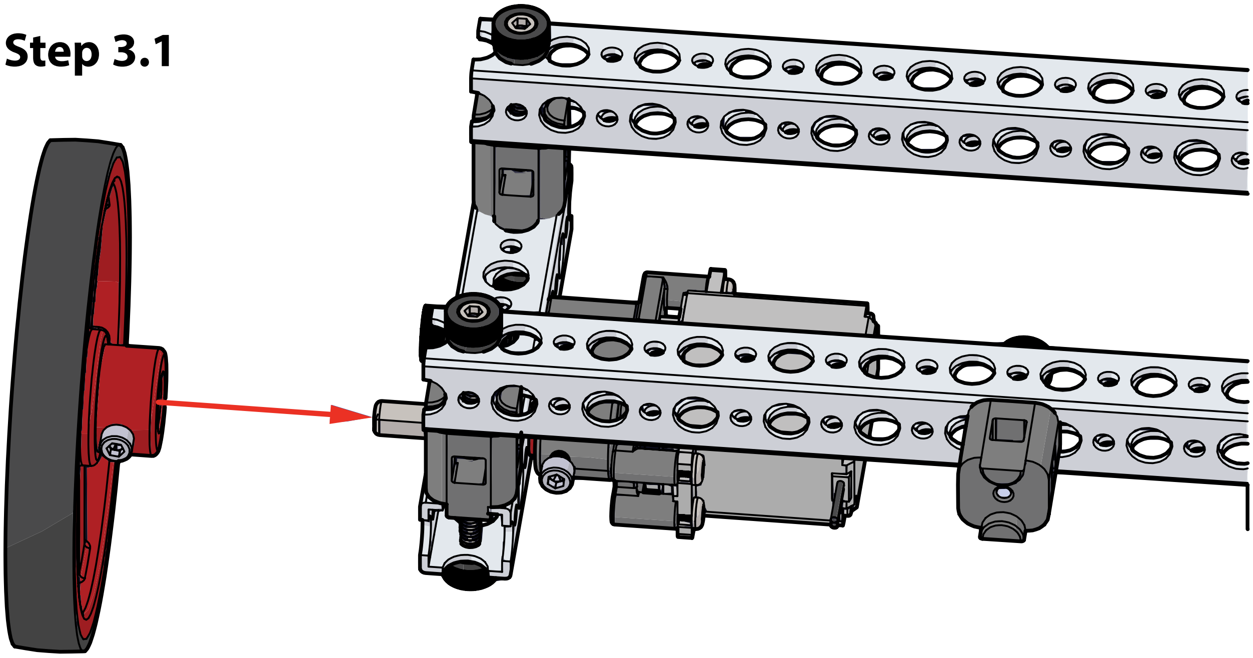
Partial assembly should look like this.



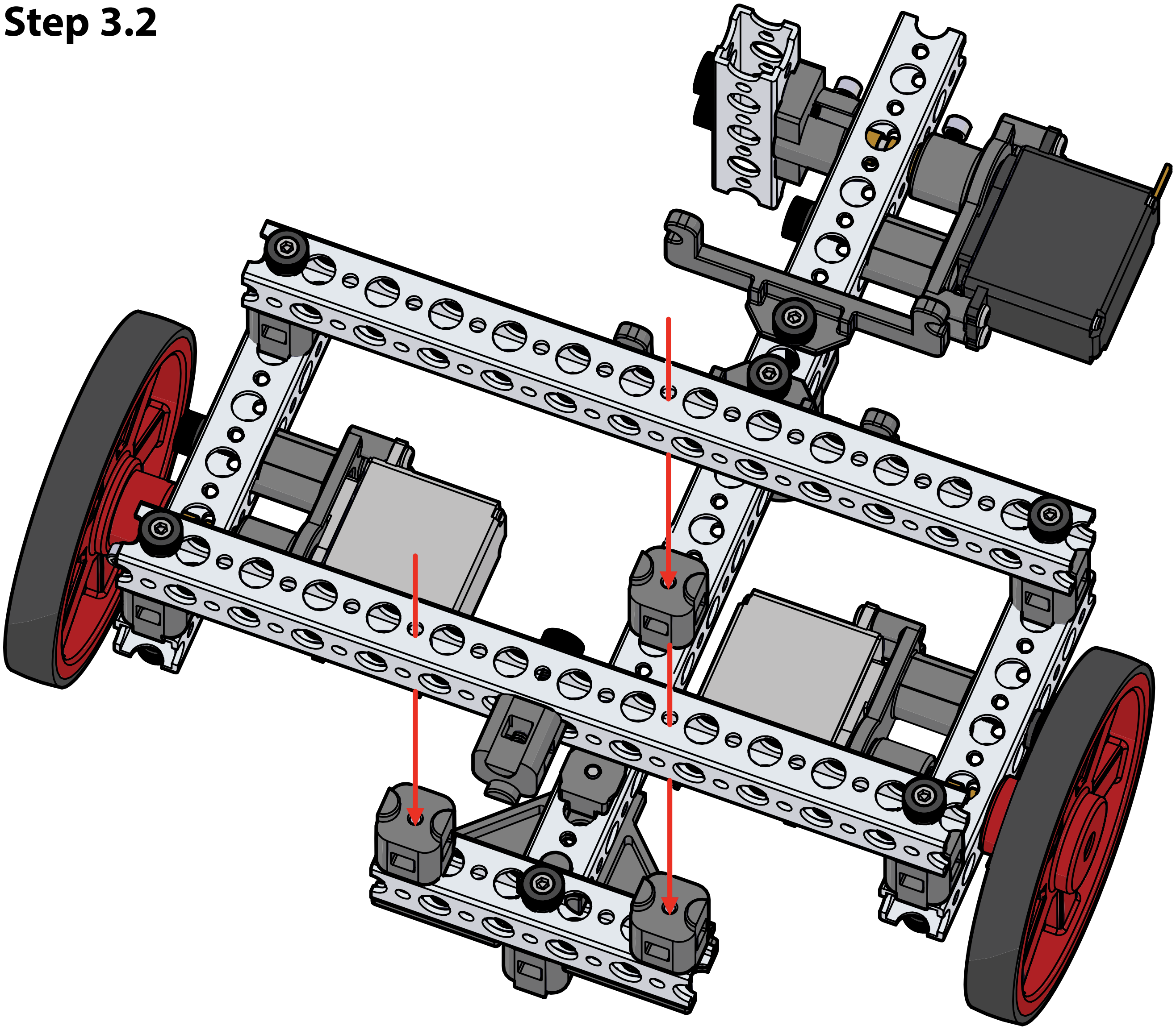
Step 3.0



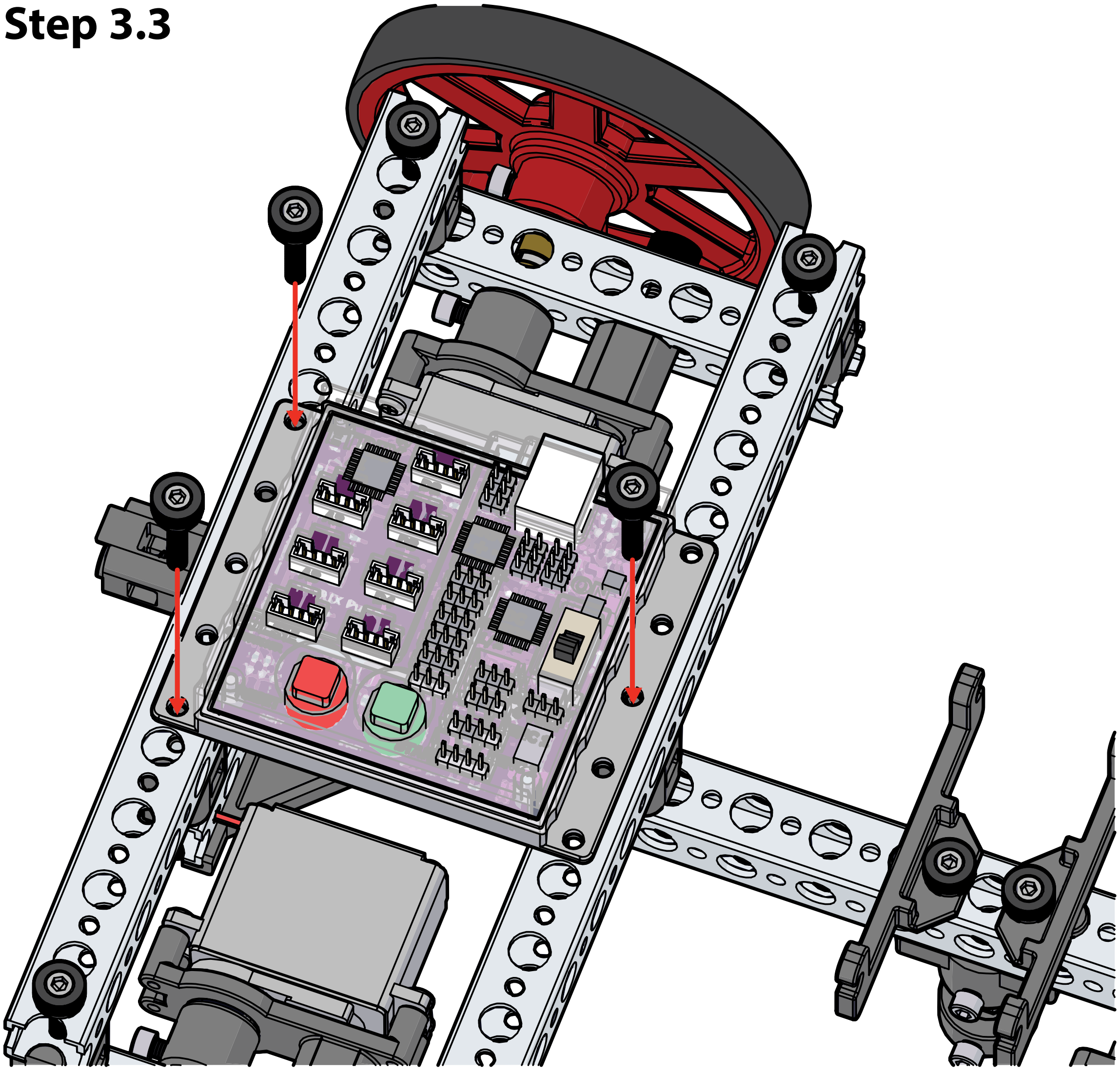
Step 3.1



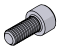
Step 3.2



Step 3.3

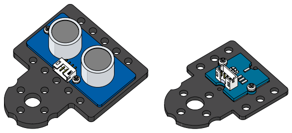


Step 4 Parts Needed



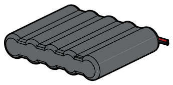
Connecting Hardware

Part No.	Part Name	Quantity
40516	Socket Head Cap Screw	2



Electronics & Control

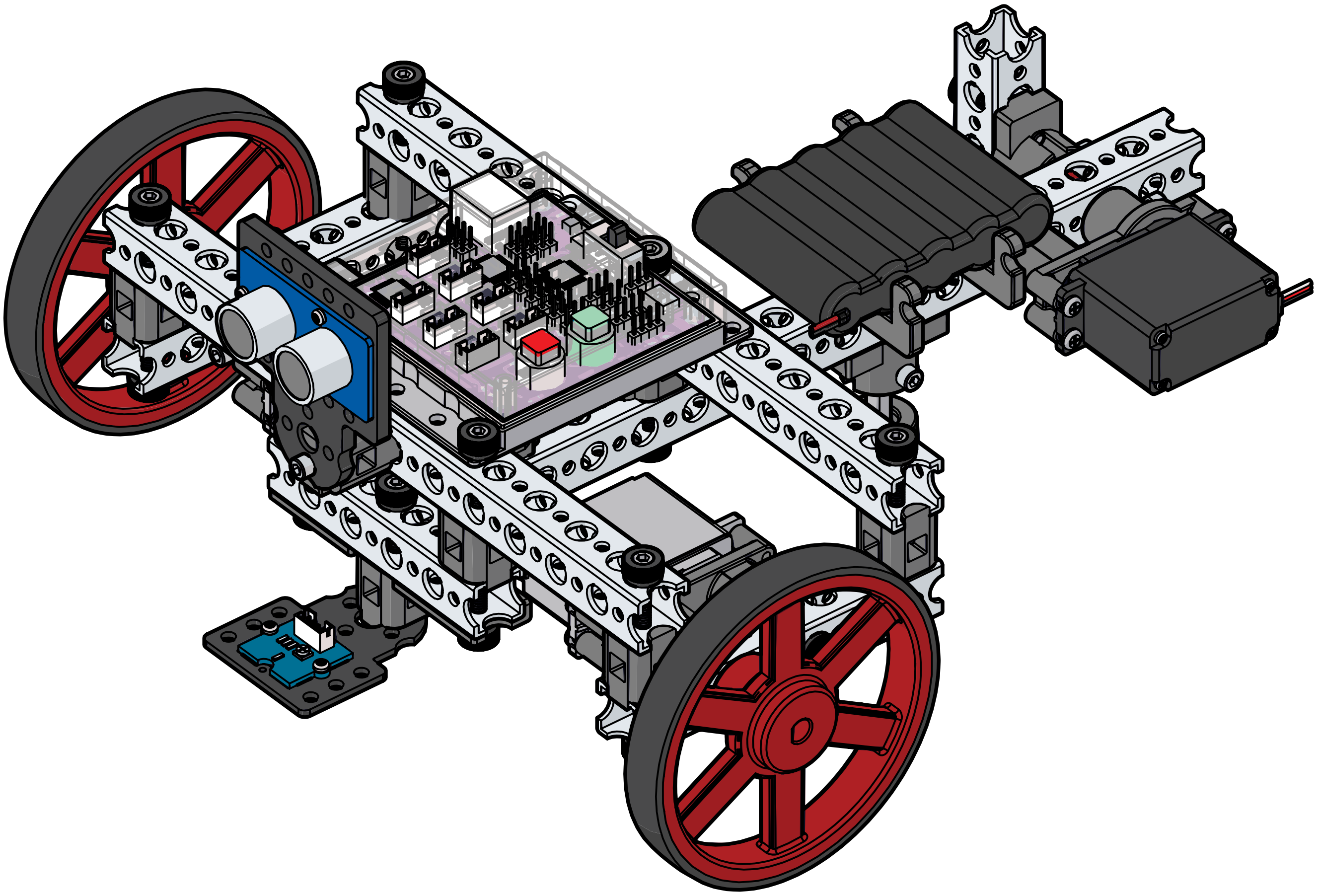
Part No.	Part Name	Quantity
43055	Ultrasonic Sensor Pack	1
43056	Line Finder Sensor Pack	1



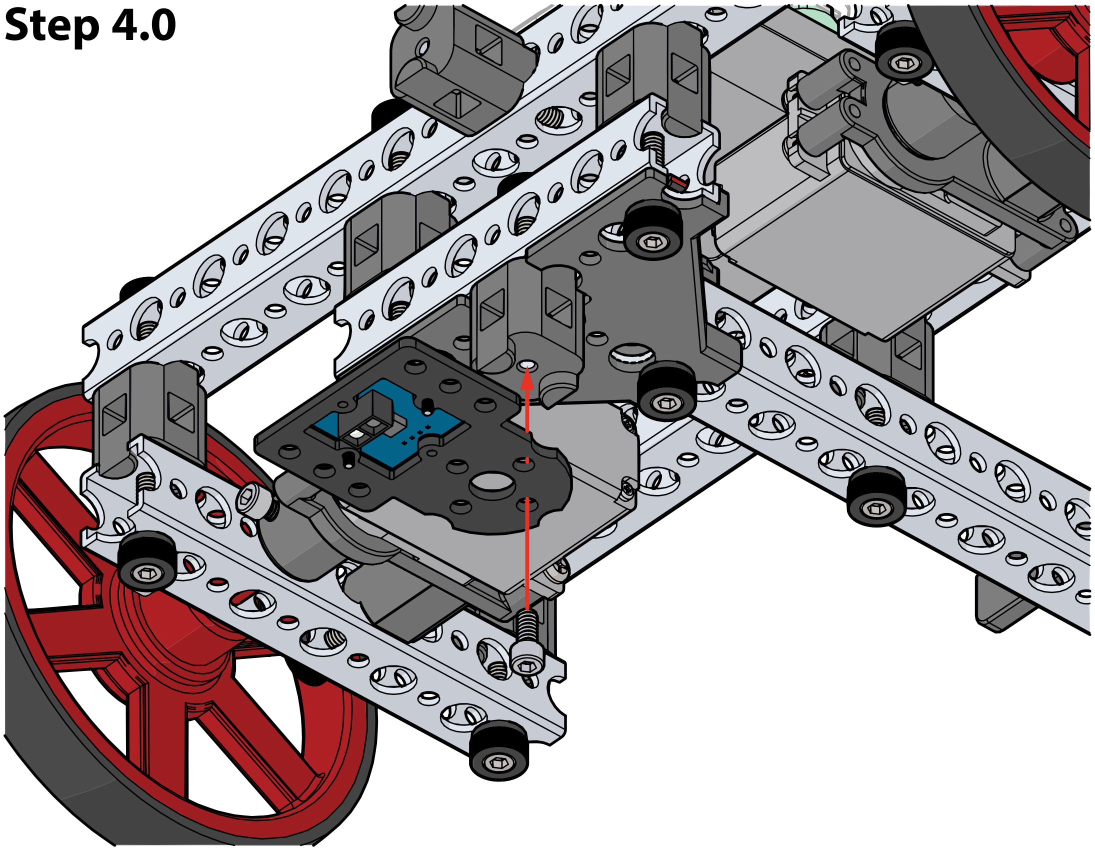
Batteries & Hardware

Part No.	Part Name	Quantity
40235	TETRIX PRIME 6 V NiMH Battery Pack	1

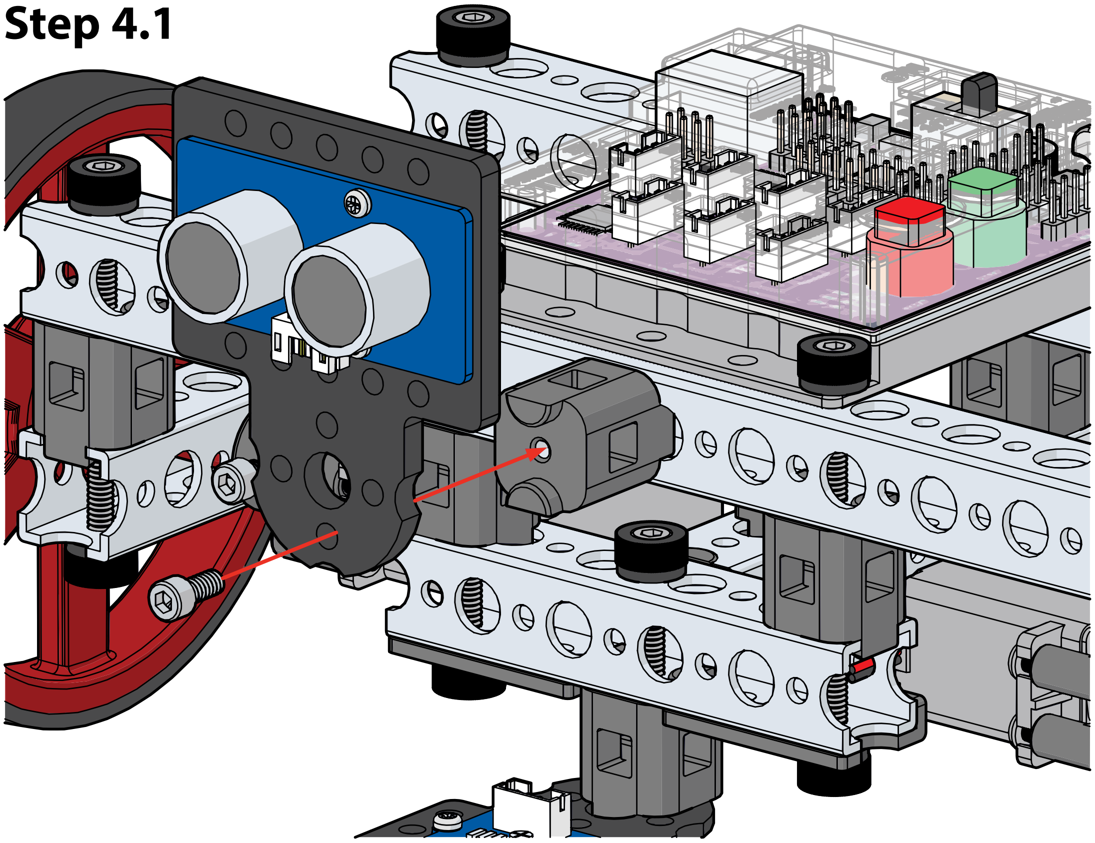
Partial assembly should look like this.



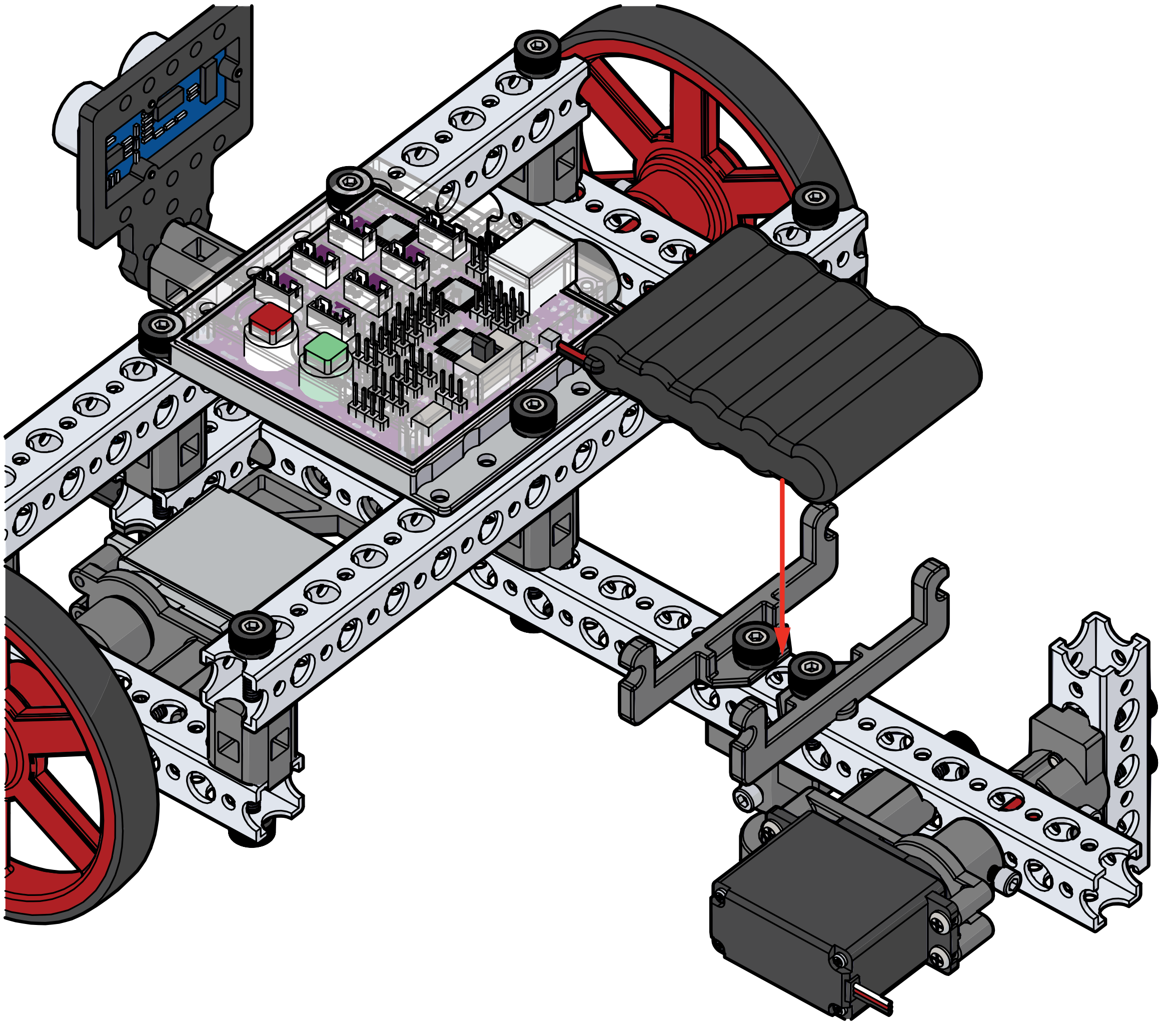
Step 4.0



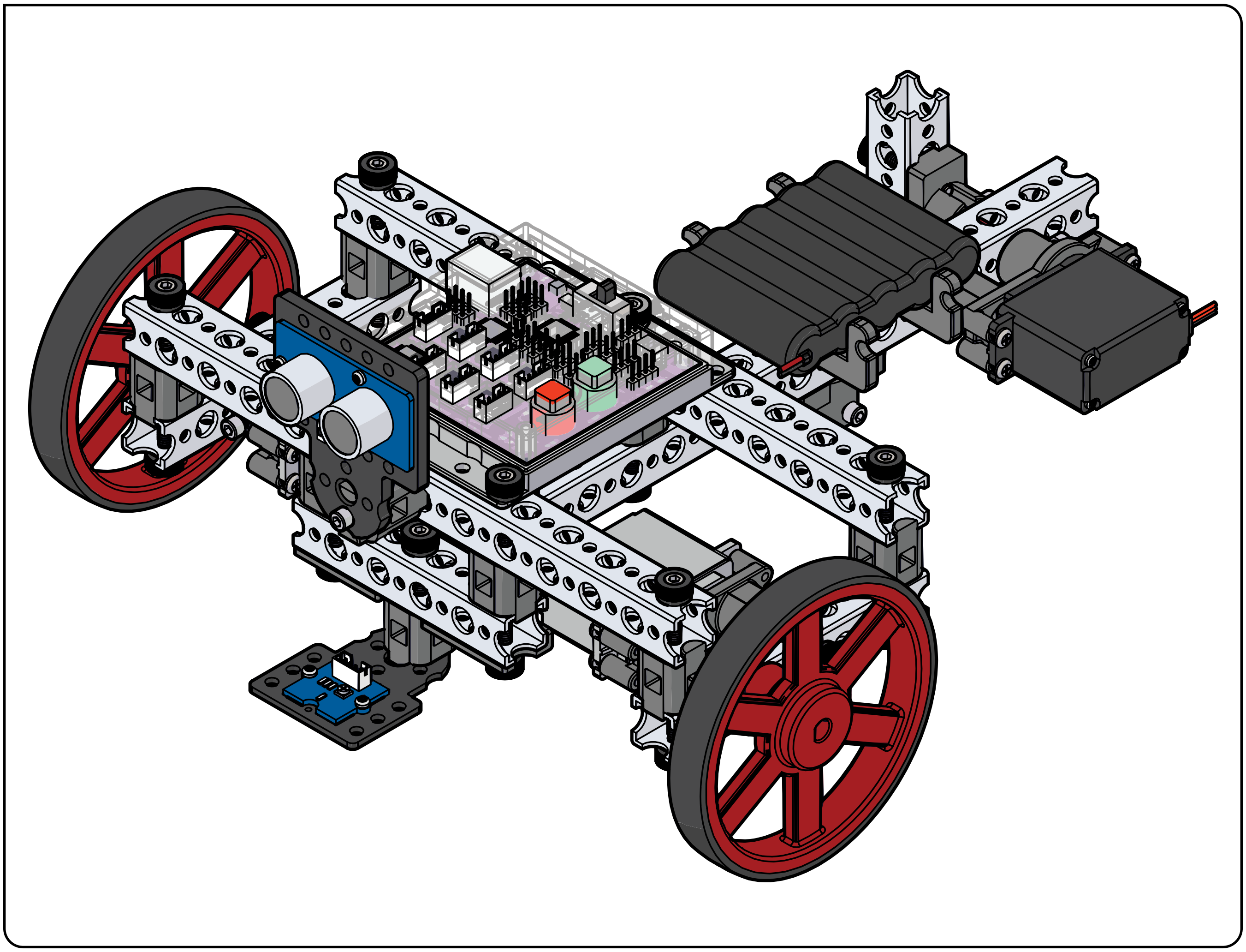
Step 4.1



Step 4.2



Finished assembly should look like this.



Activity 7: Drive Forward

Teacher Note: Other factors can affect building time, including matters such as set organization and whether the builder has a partner. The building time is only an estimate. Actual time might vary.

Introduction

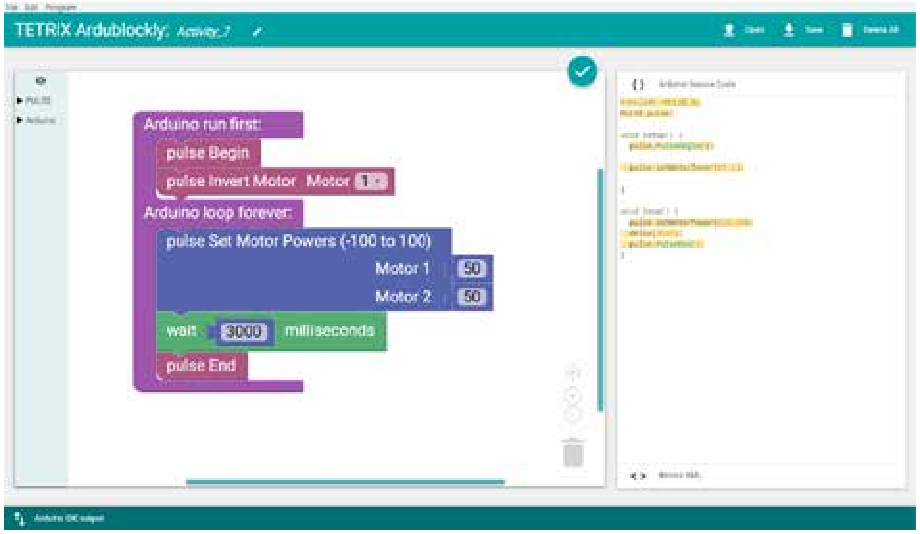
This is your first coding activity with the PULSE Codee Bot, so you'll keep it simple. In this activity, you will create a sketch to move the Codee Bot forward for three seconds and stop, and the program will end.

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

Parts Needed

Open the Program

Let's start by looking at the example sketch. Open the sketch by selecting **Examples** > **Activity_7**. A new sketch window will open titled Activity_7 (Figure 45).



```
Arduino run first:
    pulse Begin
    pulse Invert Motor  Motor 1
Arduino loop forever:
    pulse Set Motor Powers (-100 to 100)
                        Motor 1  50
                        Motor 2  50
    wait  3000  milliseconds
    pulse End
```

Figure 45

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your 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 Codee Bot on the floor.

Place the Codee Bot so the wheels are at the front of the bot. If your robot goes backward instead of forward, switch the DC motors' cords in the ports on the controller.

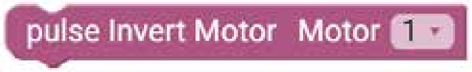
Press the green Start button to execute the sketch. Observe the direction and duration of the Codee Bot's motion. Based on the sketch comments, did the behavior match expectations? The Codee Bot should stop moving after three seconds.

Further Investigate

This sketch introduces three new blocks: pulse Invert Motor, pulse Set Motor Powers, and pulse End. The pulse Invert Motor block (Figure 46) enables you to invert the rotational direction of a motor.

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

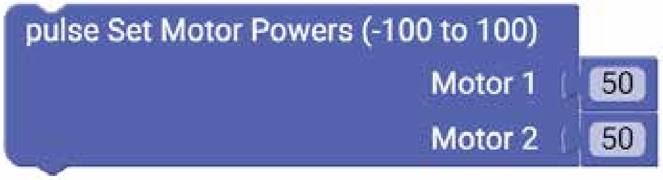
When two motors are mounted on opposite sides, this block enables you to give a single direction command to both motors and have them work together. This makes your job as a programmer easier. There are two parameters to the function. The first parameter designates the motor channel, and the second parameter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



```
pulse Invert Motor  Motor 1 ▾
```

Figure 46

The pulse Set Motor Powers block (Figure 47) 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 set at 50 percent.

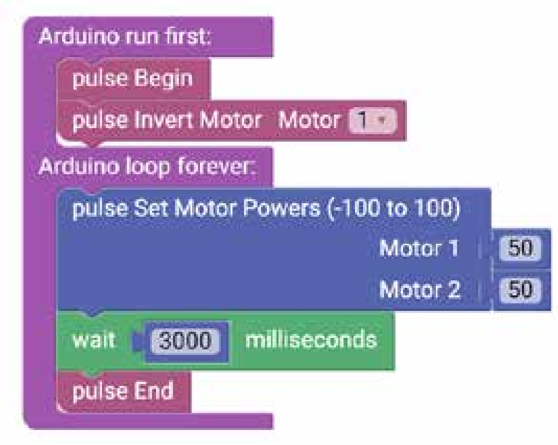


```
pulse Set Motor Powers (-100 to 100)
                            Motor 1  50
                            Motor 2  50
```

Figure 47

The pulse End block ends or terminates the sketch. If this block weren't present, your program would continue to execute the loop until you stopped the program by pressing the red button on the controller.

In this simple sketch, all these blocks work together so the robot can move forward for three seconds and then stop. Because you want the motors to always work together, the pulse Invert Motor block needs to be used only in the setup part of the loop block. The pulse Set Motor Powers block tells both motors to move at 50% power with a single block. To finish the sketch, you use pulse End instead of having it loop (Figure 48).



```
Arduino run first:
    pulse Begin
    pulse Invert Motor  Motor 1
Arduino loop forever:
    pulse Set Motor Powers (-100 to 100)
                               Motor 1  50
                               Motor 2  50
    wait  3000  milliseconds
    pulse End
```

Figure 48

Extension Activity

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

You have the fundamentals to explore speed because you can measure distance over time. In your sketch, when you use a specified time, you can physically measure how far the robot moves. When you change your 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 Codee Bot to get as close to the specified distance as possible without going over.

Real-World Link

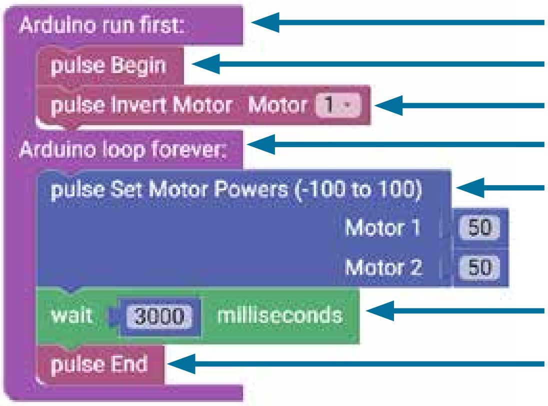
Trains are machines that can go only forward or backward with the motors they use. The motors are powered by thrust in either a forward or backward direction. A train could never directly turn around a corner. That's why all train tracks are curved instead of having 90-degree turns.

Careers: railroad worker, railroad conductor, locomotive engineer

STEM Connections

- Science
  - o Kinetic energy
  - o Speed
- Technology
  - o Direct drive
  - o DC motors
- Engineering
  - o Steering mechanism
  - o Programming
- Math
  - o Rotation
  - o Wheel circumference

Block-Text Correlation



```
Arduino run first:
  pulse Begin
  pulse Invert Motor  Motor 1
Arduino loop forever:
  pulse Set Motor Powers (-100 to 100)
                              Motor 1  50
                              Motor 2  50
  wait 3000 milliseconds
  pulse End
```

```
void setup() {
pulse.PulseBegin();
pulse.setMotorInvert(1,1);
void loop() {
pulse.setMotorPowers(50,50);
delay(3000);
pulse.PulseEnd();
```

Note: Use the pulse Invert Motor block to program the two motors to go in opposite directions.

Arduino Source Code

```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

pulse.setMotorInvert(1,1);
}

void loop() {
pulse.setMotorPowers(50,50);
delay(3000);
pulse.PulseEnd();
}
```

Activity 8: Drive in a Circle

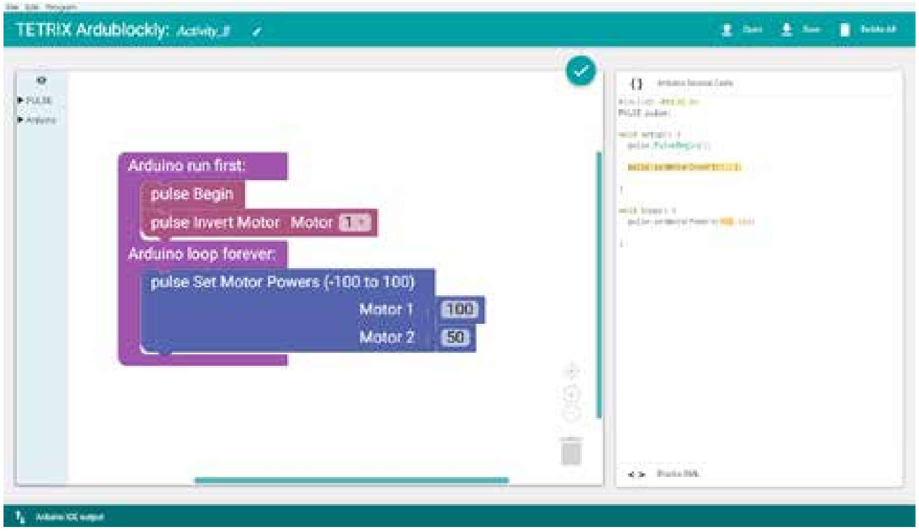
Introduction

For your eighth activity, you will apply your knowledge of motors to create a new behavior. While being able to move straight is important, you need to be able to expand on that and make turns. This activity will have your PULSE Codee Bot driving in circles by using different motor powers as the motors work in unison.

Parts Needed

Open the Program

Let’s start by looking at the example sketch. Open the sketch by selecting **Examples** > **Activity_8**. A new sketch window will open titled Activity_8 (Figure 49).



```
Arduino run first:
  pulse Begin
  pulse Invert Motor  Motor 1
Arduino loop forever:
  pulse Set Motor Powers (-100 to 100)
                             Motor 1  100
                             Motor 2  50
```

Figure 49

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your 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 Codee Bot on the floor.

Place the Codee Bot so the wheels are at the front of the bot. If your robot goes backward instead of forward, switch the DC motors' cords in the ports on the controller.

Press the green Start button to execute the sketch. Observe the direction and duration of the Codee Bot's motion. Press the red Stop/Reset button to end the sketch. Based on the sketch comments, did the behavior match expectations?

Tip: The Codee Bot might struggle to overcome the friction of carpet. If possible, test your bot on a smooth surface.

Tip: You will need about six feet, or two meters, of open space on the floor for the robot to complete a circle.

Further Investigate

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

All the blocks in this sketch work together to make the Codee Bot move in a circle. Because you want the motors to always work together, the pulse Invert Motor block is used only in the setup part of the loop. The pulse Set Motor Powers block tells both motors to move at different speeds in a single function. One motor is set to 100% while the other is set to 50%, resulting in a circular motion.

Extension Activity

With the example as a reference, try creating a new sketch to move the Codee Bot in a different-size circle. Remember what you have learned from your 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?

Real-World Link

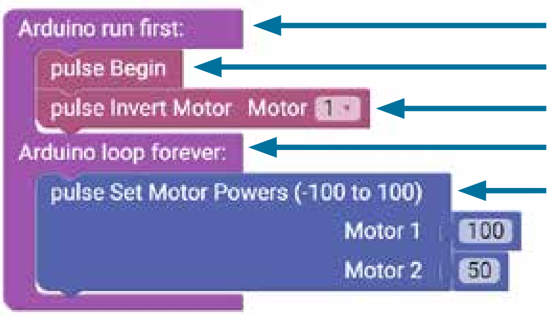
If you've ridden by a golf course, you have probably seen automatic sprinklers operating. These sprinklers go in a circle spraying water in a 360-degree radius. These sprinklers are programmed to turn off and on at different times. It's important that the motor turn the sprinkler head so it covers the entire area of its range.

Careers: groundskeeper, farm equipment mechanic, irrigation engineer

STEM Connections

- Science
  - o Power
  - o Speed
- Technology
  - o Motor control
  - o Motor precision
- Engineering
  - o Machine design
  - o Geometric patterns
- Math
  - o Radius
  - o Diameter

Block-Text Correlation



```
void setup() {
pulse.PulseBegin();
pulse.setMotorInvert(1,1);
void loop() {
pulse.setMotorPowers(100,50);
```

Note: Motor 1 is turning at twice the power of Motor 2.

Arduino Source Code

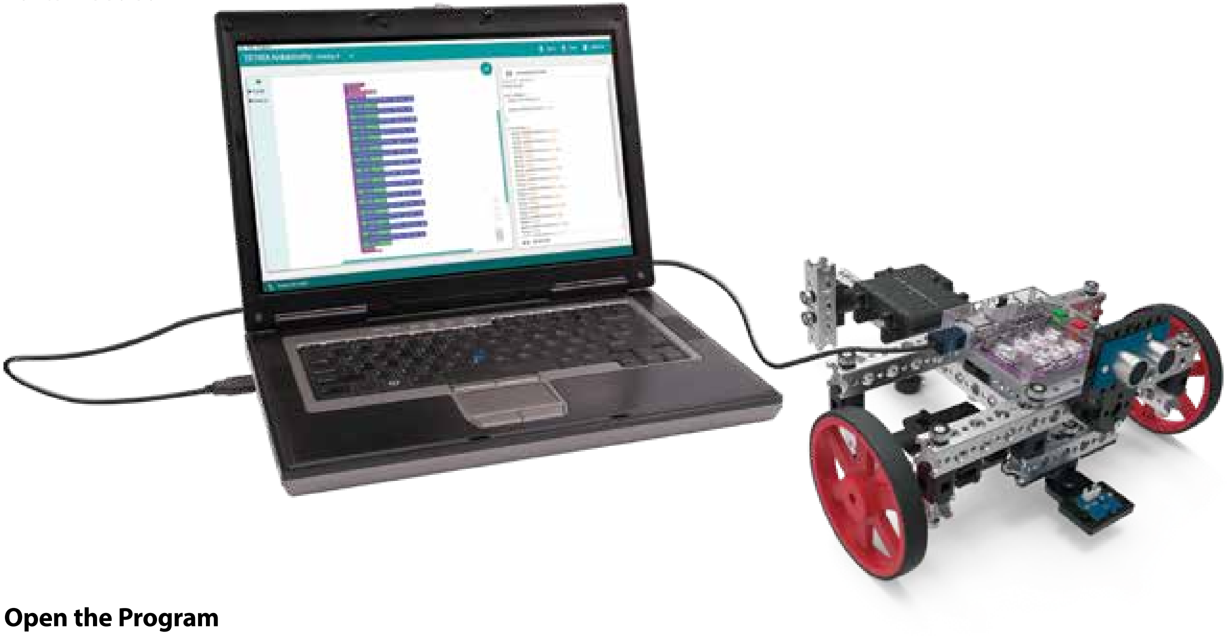
```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

pulse.setMotorInvert(1,1);
}

void loop() {
pulse.setMotorPowers(100,50);
}
```

Activity 9: Drive in a Square

Introduction

Now, you will continue to build your navigational skills with the PULSE Codee Bot by giving your robot the ability to make 90-degree turns. The ability to make 90-degree turns will be used to make the Codee Bot drive in a square.

Parts Needed



Open the Program

Let's start by looking at the example sketch. Open the sketch by selecting **Examples** > **Activity_9**. A new sketch window will open titled Activity_9 (Figure 50).

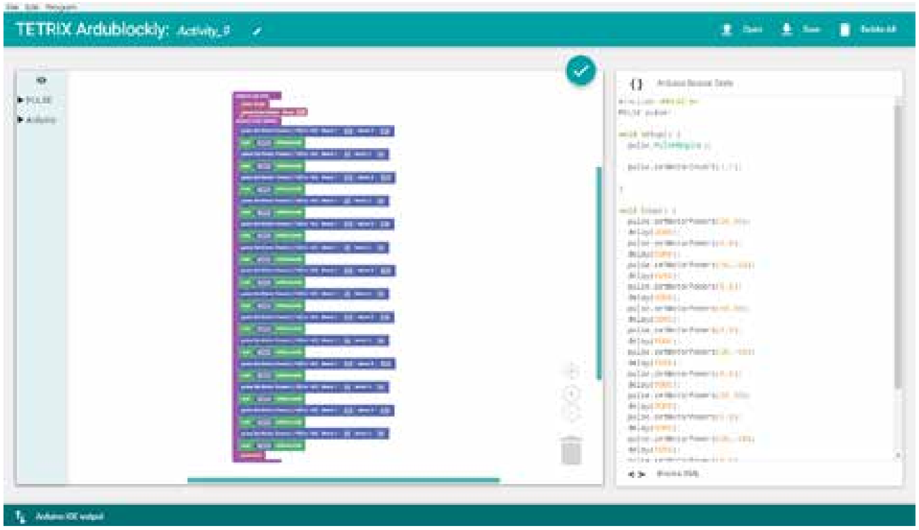


Figure 50

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your 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 Codee Bot on the floor.

Tip: Depending on the surface type, your bot might not execute a perfect square.

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

Further Investigate

When you have a lot of blocks on screen, it can hard to see your whole program. You can use the zoom in and out feature. Also, some blocks can be changed to inline inputs. The pulse Set Motor Powers block can be changed to an inline input. Below is an example of how this block can change appearance (Figure 51). This does not affect your program.

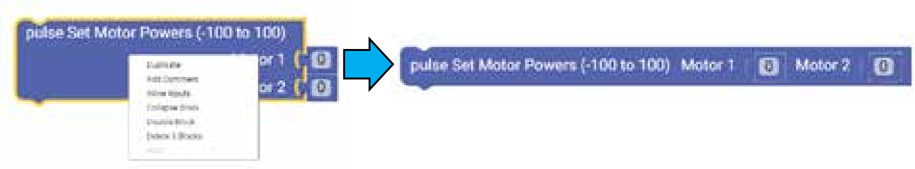


Figure 51

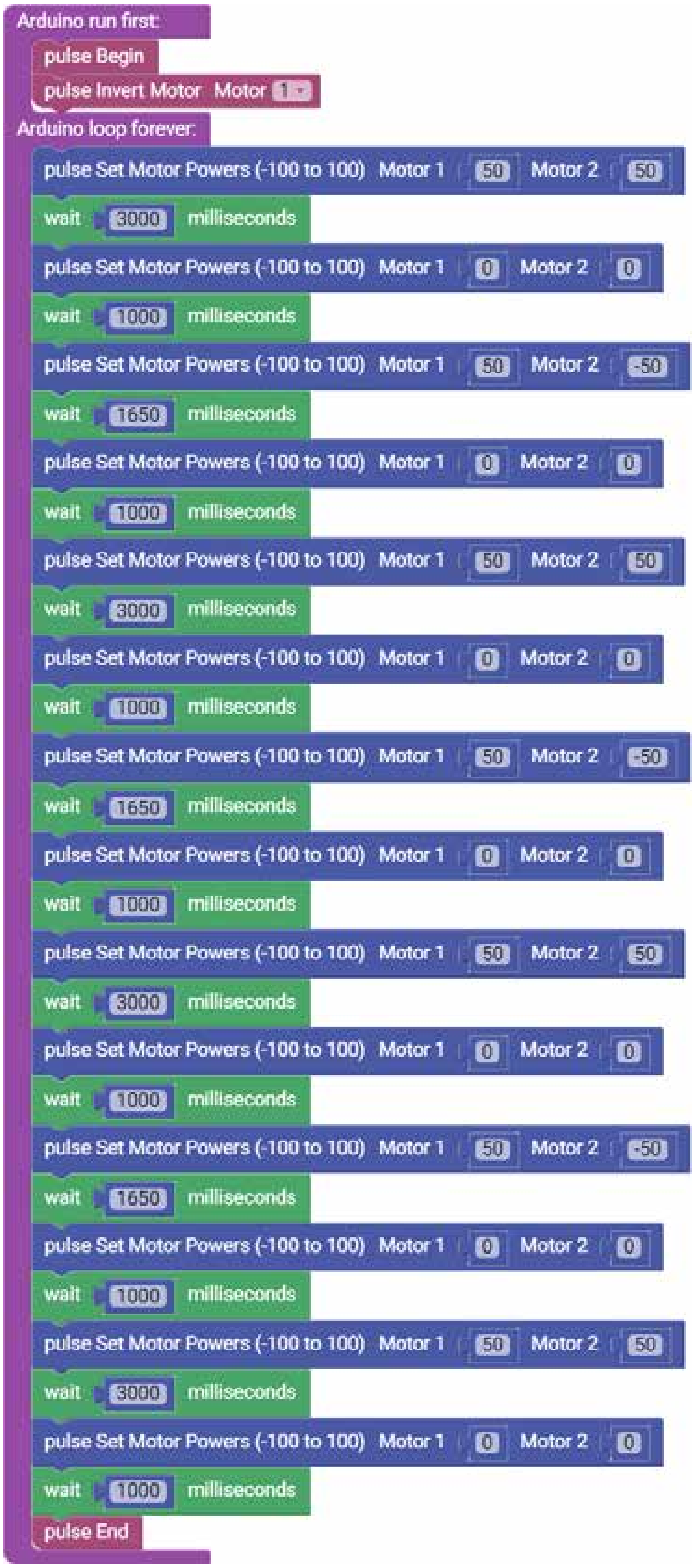
This sketch has a lot to it, but it is made of functions that you have used before. You have just combined multiple sequential behaviors to create one larger behavior – in this case, a square.

An important thing to understand about this program is that you 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, you are commanding the motors to turn on in opposite directions at 50% power and run for 1,650 milliseconds.

You estimate that if you spin the motors at a certain rpm for a certain time, you should come close to making a 90-degree right turn. However, this is not always accurate because the amount that your robot's battery is charged can vary, and any wheel slippage on the surface you are working on can cause a variation in results. You are using dead reckoning to make a right turn.

All the functions in this sketch work together to make the Codee Bot move in a square (Figure 52). Because you want the motors to always work together, the pulse Invert Motor block needs to be used only in the setup part of the loop. The pulse Set Motor Powers block tells both motors to move at different speeds in a single function.

Program Breakdown



```
Arduino run first:
    pulse Begin
    pulse Invert Motor  Motor 1
Arduino loop forever:
    pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  50     Drive forward
    wait  3000  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0       Brake
    wait  1000  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  -50    Turn 1
    wait  1650  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0       Brake
    wait  1000  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  50     Drive forward
    wait  3000  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0       Brake
    wait  1000  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  -50    Turn 2
    wait  1650  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0       Brake
    wait  1000  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  50     Drive forward
    wait  3000  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0       Brake
    wait  1000  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  -50    Turn 3
    wait  1650  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0       Brake
    wait  1000  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  50     Drive forward
    wait  3000  milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0       Brake
    wait  1000  milliseconds
    pulse End
```

Figure 52

Tip: Adjust the timing of the turns if the robot doesn't turn at right angles. For example, you can change the time from 1650 to 1300. The charge of the battery can affect the turns.

Extension Activity

With the example as a reference, try creating a new sketch to move the Codee Bot in a square. Remember what you have learned from your 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.

Real-World Link

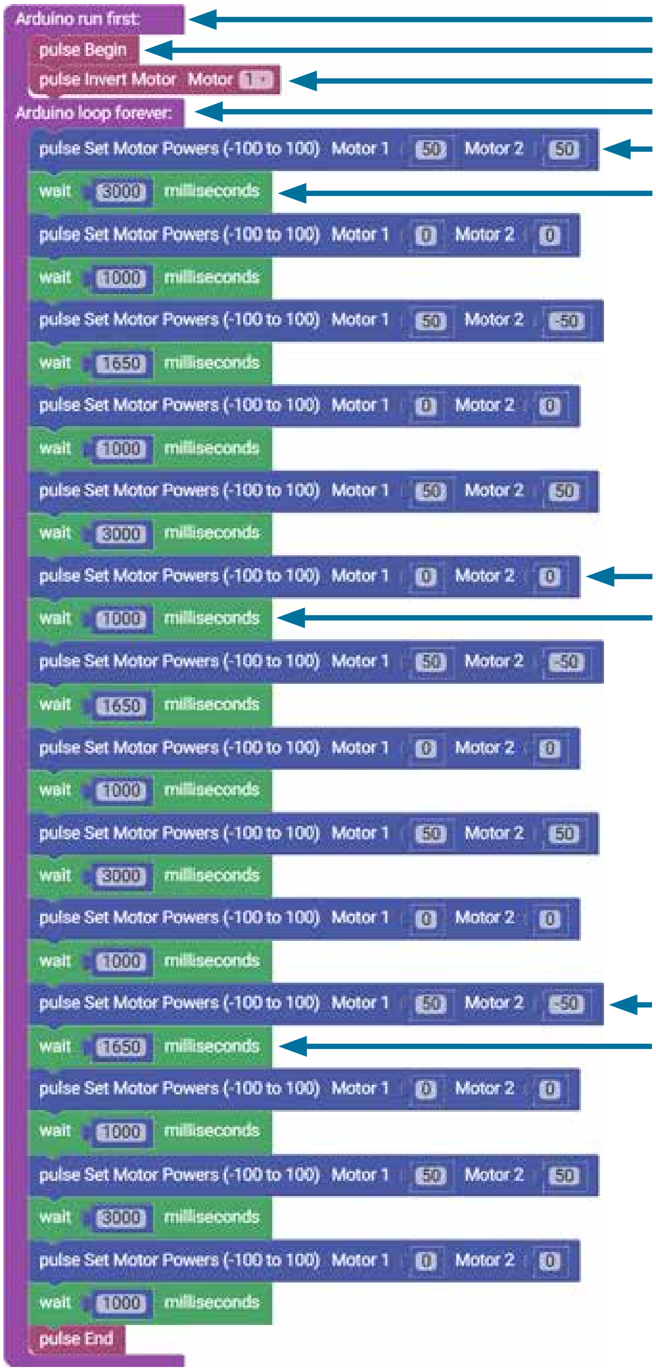
Some transportation vehicles take the same route time after time. If a vehicle is in a warehouse, it could transport items to three different locations before returning. Then it could pick up more items and move in a square to transport them.

Careers: mapping technician, air traffic controller, manufacturing engineer

STEM Connections

- Science
    - Distance
    - Time
- Technology
    - Microprocessor time
    - Turning radius
- Engineering
    - Motor power
    - Braking
- Math
    - Angles of a square
    - Measurement of angles

Block-Text Correlation



```
Arduino run first:
  pulse Begin
  pulse Invert Motor  Motor 1
Arduino loop forever:
  pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  50
  wait  3000  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
  wait  1000  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  -50
  wait  1650  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
  wait  1000  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  50
  wait  3000  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
  wait  1000  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  -50
  wait  1650  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
  wait  1000  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  50
  wait  3000  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
  wait  1000  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  -50
  wait  1650  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
  wait  1000  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  50
  wait  3000  milliseconds
  pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
  wait  1000  milliseconds
  pulse End
```

```
void setup() {
pulse.PulseBegin();
pulse.setMotorInvert(1,1);
void loop() {
pulse.setMotorPowers(50,50);
delay(3000);
```

Note: The motors drive forward for 3 seconds.

```
pulse.setMotorPowers(0,0);
delay(1000);
```

Note: The motors brake for 1 second.

```
pulse.setMotorPowers(50,-50);
delay(1650);
```

Note: The motors turn opposite directions for 1.65 seconds, causing the bot to turn.

Arduino Source Code

```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

pulse.setMotorInvert(1,1);
}

void loop() {
pulse.setMotorPowers(50,50);
delay(3000);
pulse.setMotorPowers(0,0);
delay(1000);
pulse.setMotorPowers(50,-50);
delay(1650);
pulse.setMotorPowers(0,0);
delay(1000);
pulse.setMotorPowers(50,50);
delay(3000);
pulse.setMotorPowers(0,0);
delay(1000);
pulse.setMotorPowers(50,-50);
delay(1650);
pulse.setMotorPowers(0,0);
delay(1000);
pulse.setMotorPowers(50,50);
delay(3000);
pulse.setMotorPowers(0,0);
delay(1000);
pulse.setMotorPowers(50,-50);
delay(1650);
pulse.setMotorPowers(0,0);
delay(1000);
pulse.setMotorPowers(50,50);
delay(3000);
pulse.setMotorPowers(0,0);
delay(1000);
pulse.PulseEnd();
}
```

Activity 10: Simplify the Square

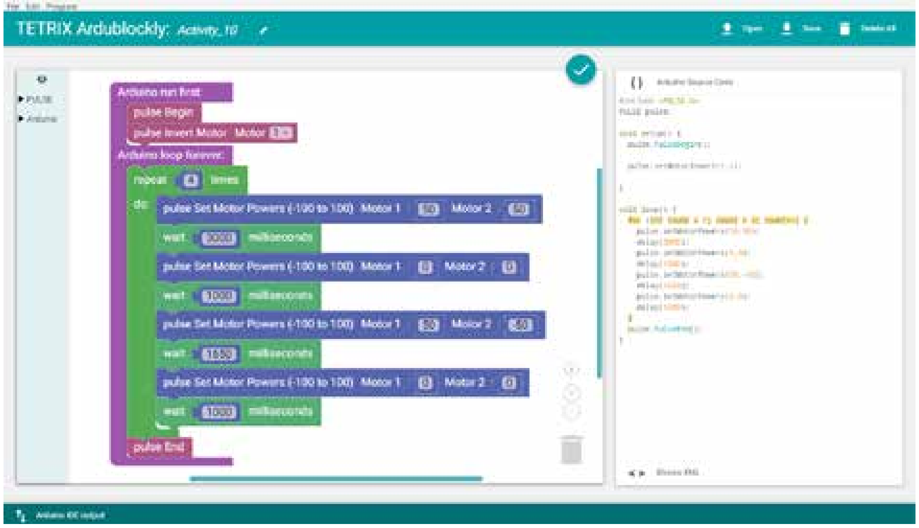
Introduction

Remember the square activity you did in Activity 9? For your next activity, you will use a more efficient way to code the same behavior.

Parts Needed

Open the Program

Let's start by looking at the example sketch. Open the sketch by selecting **Examples** > **Activity_10**. A new sketch window will open titled Activity_10 (Figure 53).



```
Arduino run first:
  pulse Begin
  pulse Invert Motor  Motor 1
Arduino loop forever:
  repeat 4 times
  do  pulse Set Motor Powers (-100 to 100)  Motor 1 50  Motor 2 50
      wait 3000 milliseconds
      pulse Set Motor Powers (-100 to 100)  Motor 1 0  Motor 2 0
      wait 1000 milliseconds
      pulse Set Motor Powers (-100 to 100)  Motor 1 50  Motor 2 -50
      wait 1650 milliseconds
      pulse Set Motor Powers (-100 to 100)  Motor 1 0  Motor 2 0
      wait 1000 milliseconds
  pulse End
```

```
#include <PULSE.h>
PULSE pulse;

void setup() {
  pulse.PulseBegin();

  pulse.setMotorInvert(1, 1);

}

void loop() {
  for (int count = 1; count <= 4; count++) {
    pulse.setMotorPowers(50, 50);
    delay(3000);
    pulse.setMotorPowers(0, 0);
    delay(1000);
    pulse.setMotorPowers(50, -50);
    delay(1650);
    pulse.setMotorPowers(0, 0);
    delay(1000);
  }
  pulse.PulseEnd();
}
```

Figure 53

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your 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 Codee Bot on the floor.

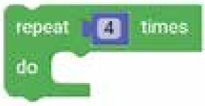
Press the green Start button to execute the sketch. Observe the direction and duration of the Codee Bot's motion. Based on the sketch comments, did the behavior match expectations?

Tip: Depending on the surface type, your bot might not execute a perfect square.

Further Investigate

This sketch implements one new program structure in the form of the repeat-do block. The repeat-do block allows for the statements within the block to repeat for a specific number of times. This makes a simpler, easier-to-read sketch.

The repeat-do block is used to repeat a block of code contained within it. An increment counter is used to count the number of times the code is executed before the loop is terminated. The repeat-do block is useful for limiting the number of times a loop will execute (Figure 54).

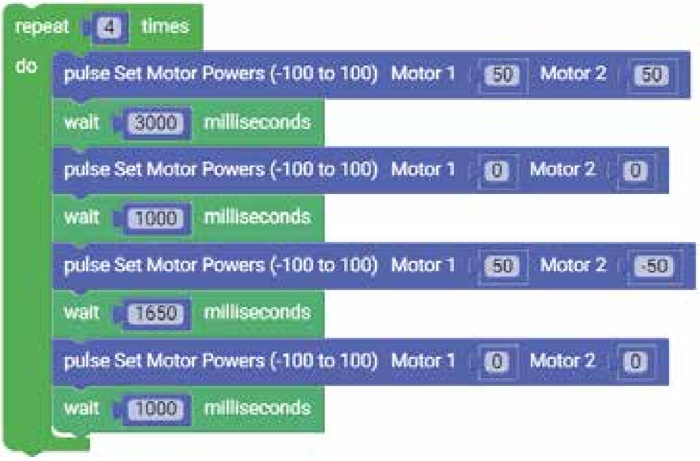


```
repeat 4 times
do
```

Figure 54

When you made the Codee Bot drive in a square in Activity 9, you listed each action line by line, resulting in excessive code that looked more complicated than it was.

You start the loop with the repeat-do block to define how many times you want the loop to repeat. The loop will repeat four times in the program. Within the loop, the robot is programmed to drive forward and turn (Figure 55). This is named the called function because it occurs outside the setup-loop block.



```
repeat 4 times
do  pulse Set Motor Powers (-100 to 100)  Motor 1 50  Motor 2 50
    wait 3000 milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1 0  Motor 2 0
    wait 1000 milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1 50  Motor 2 -50
    wait 1650 milliseconds
    pulse Set Motor Powers (-100 to 100)  Motor 1 0  Motor 2 0
    wait 1000 milliseconds
```

Figure 55

Extension Activity

With the example as a reference, try creating a new sketch to move the Codee Bot in a square using the repeat-do block and called functions. Or for an additional challenge, create complex paths beyond a square such as rectangles or hexagons.

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

Real-World Link

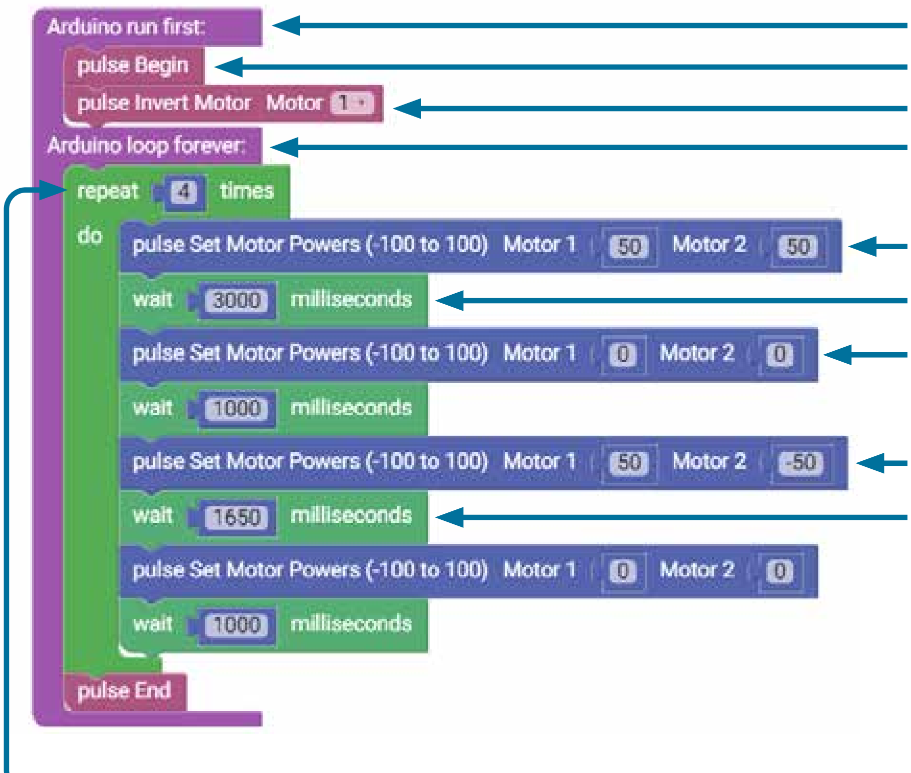
Robots are used to place items into packages in manufacturing. They have to repeat this process over and over again. The robots use sensors to sense when a package is underneath them to place an item.

Careers: PLC (programmable logic controller) programmer, CNC programmer, manufacturing technician

STEM Connections

- Science
  - Distance
  - Velocity
- Technology
  - Microprocessors
  - Turning radius
- Engineering
  - Path-following design
  - Programming
- Math
  - Angle measurements
  - Angles of a square

Block-Text Correlation


Arduino run first:
pulse Begin
pulse Invert Motor Motor 1
Arduino loop forever:
repeat 4 times
do
pulse Set Motor Powers (-100 to 100) Motor 1 50 Motor 2 50
wait 3000 milliseconds
pulse Set Motor Powers (-100 to 100) Motor 1 0 Motor 2 0
wait 1000 milliseconds
pulse Set Motor Powers (-100 to 100) Motor 1 50 Motor 2 -50
wait 1650 milliseconds
pulse Set Motor Powers (-100 to 100) Motor 1 0 Motor 2 0
wait 1000 milliseconds
pulse End

```
void setup() {
pulse.PulseBegin();
pulse.setMotorInvert(1,1);
void loop() {
pulse.setMotorPowers(50,50);
delay(3000);
pulse.setMotorPowers(0,0);
pulse.setMotorPowers(50,-50);
delay(1650);
```

```
for (int count = 0; count < 4; count++) {
}
```

Note: This is the initialization that happens first and only once.

Note: This is the condition that is tested in each loop.

Note: The increment is executed, and then the condition is tested again.

Arduino Source Code

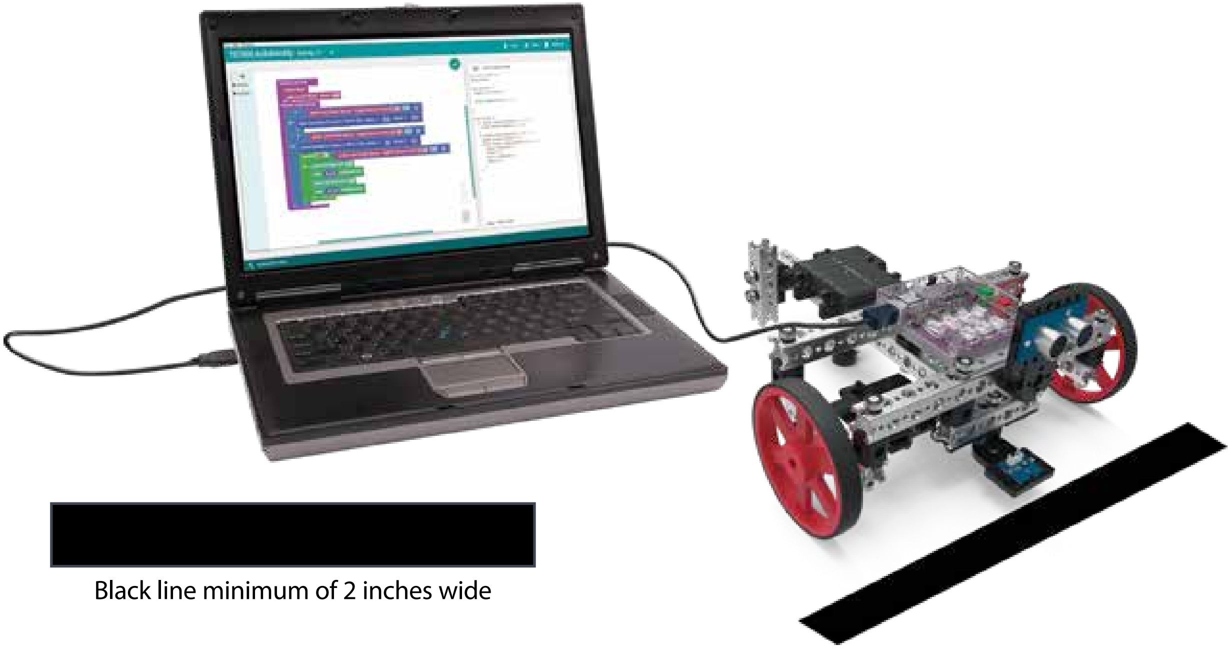
```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

pulse.setMotorInvert(1,1);
}
void loop() {
for (int count = 0; count < 4; count++) {
pulse.setMotorPowers(50,50);
delay(3000);
pulse.setMotorPowers(0,0);
delay(1000);
pulse.setMotorPowers(50,-50);
delay(1650);
pulse.setMotorPowers(0,0);
delay(1000);
}
pulse.PulseEnd();
}
```

Activity 11: Drive to a Line and Stop

Introduction

You will begin this activity by adding the Line Finder Sensor to a basic move forward behavior. This will enable the PULSE Codee Bot to stop based on the environment around it. In this activity, the Codee Bot will drive forward and stop at a line.

Parts Needed



Open the Program

Let's start by looking at the example sketch. Open the sketch by selecting **Examples** > **Activity_11**. A new sketch window will open titled Activity_11 (Figure 56).

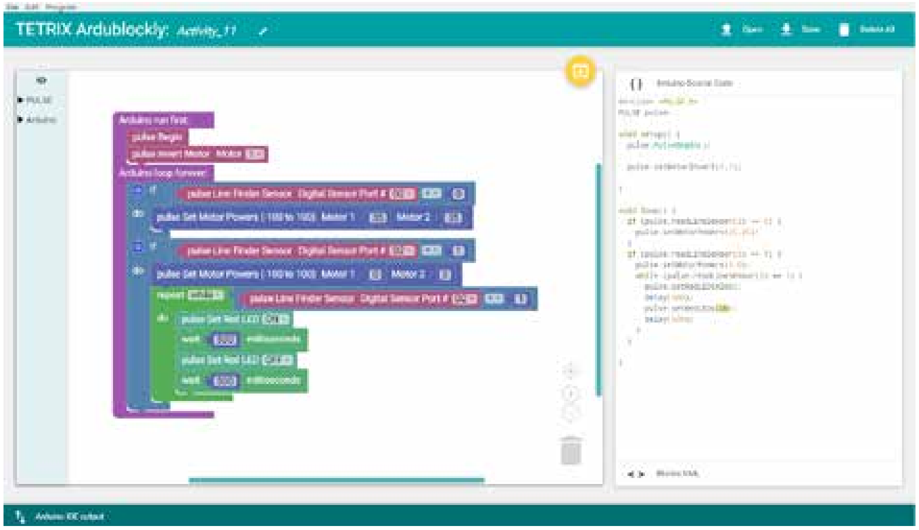


Figure 56

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your connections. The Line Finder Sensor should be in D2. 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 Codee Bot on the floor.

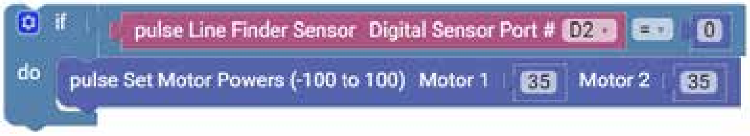
The Codee Bot should be on a white or reflective surface pointed toward a black line or nonreflective surface approximately 36 inches or one meter away. To run this code, you will need to place a black stripe on a white or reflective surface for the robot to follow. White glossy cardboard, foam board, poster board, or several sheets of paper tiled together can work for the surface.

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

Further Investigate

This sketch uses two if-do blocks, a program structure you used in Activity 4, and it introduces a while() loop block, which is a new program structure.

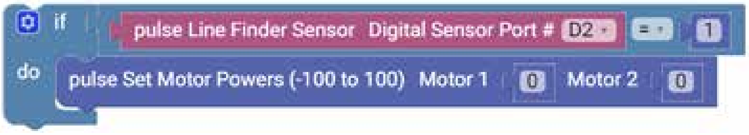
In this sketch, if-do blocks are looking at the condition of the Line Finder Sensor. The first if-do block tells the Codee Bot to move at 35% power when it's over a white or reflective surface. The Line Finder Sensor will read 0 over these surfaces (Figure 57).



```
if   pulse Line Finder Sensor  Digital Sensor Port # D2 = 0
do   pulse Set Motor Powers (-100 to 100)  Motor 1 35  Motor 2 35
```

Figure 57

The second if-do block contains two parts. The first part tells the Codee Bot to brake when it's over a black line or nonreflective surface. The Line Finder Sensor reads a value of 1. The motor power is set to 0 and brakes (Figure 58).



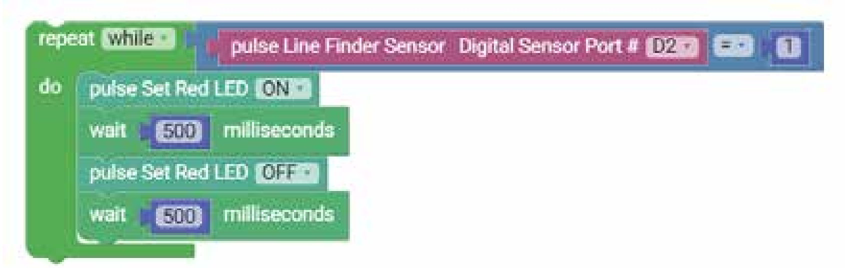
```
if   pulse Line Finder Sensor  Digital Sensor Port # D2 = 1
do   pulse Set Motor Powers (-100 to 100)  Motor 1 0  Motor 2 0
```

Figure 58

Tip: Depending on the surface type, your bot might not execute a perfect square.

Troubleshooting: 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-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.

The second part of the if-do block executes the while() loop block. A while() loop looks at a test condition in the block and will loop continually until the test condition set in the block becomes false. In this program, it will repeat what's in the while() loop as long as the Line Finder Sensor reads a value of 1 (Figure 59). The program will continue blinking the red LED until the sketch is reset.



```
repeat while ▾   pulse Line Finder Sensor  Digital Sensor Port # D2 ▾  = ▾  1
do  pulse Set Red LED ON ▾
    wait 500 milliseconds
    pulse Set Red LED OFF ▾
    wait 500 milliseconds
```

Figure 59

Extension Activity

With the example as a reference, try creating a new sketch to have the Codee Bot drive to a black line and stop. Remember what you have learned from your previous activities. Challenge yourself to create different behaviors for when the Codee Bot meets a line, such as backing up or turning a different direction.

Real-World Link

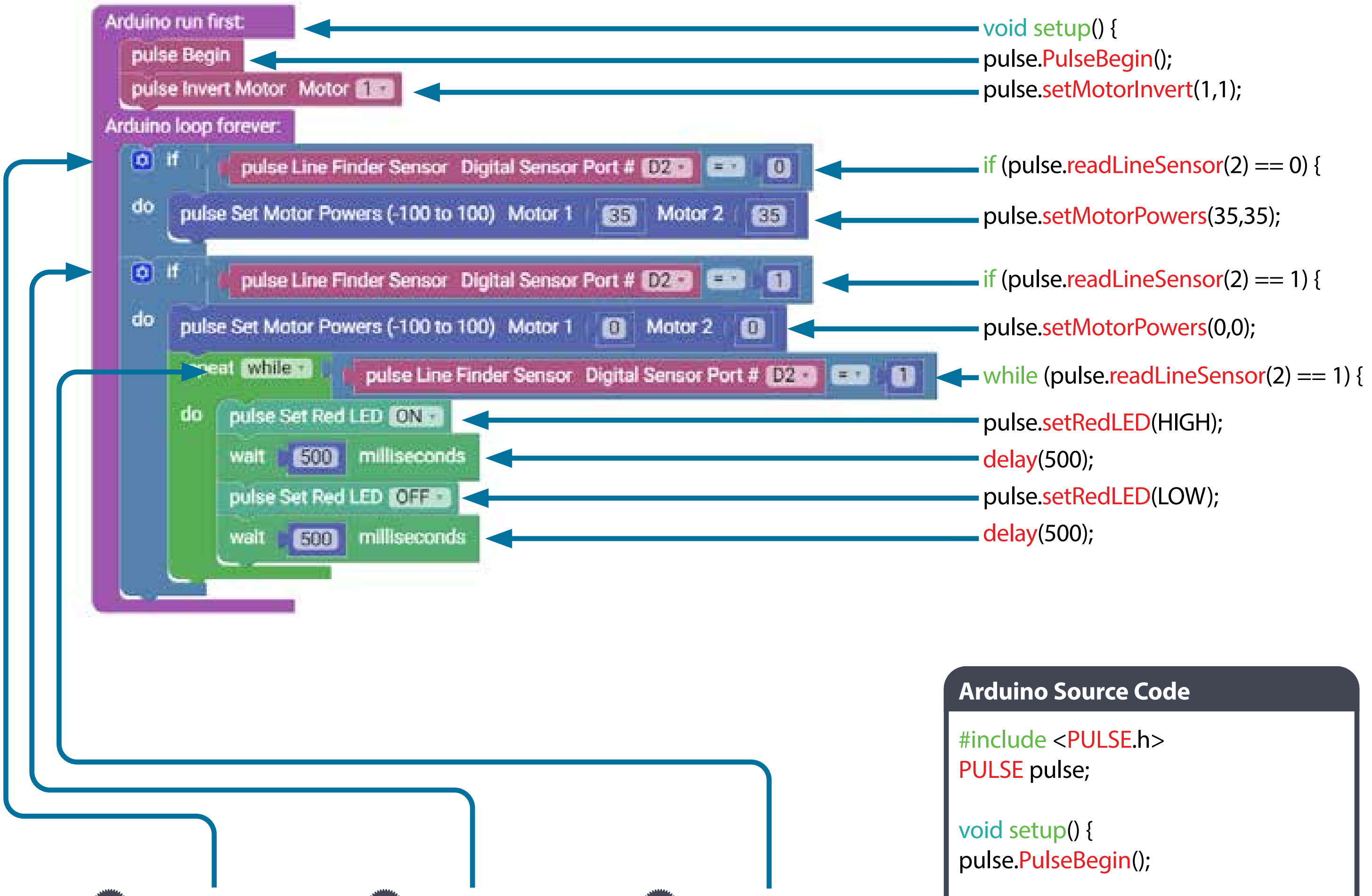
With the advent of self-driving cars, cars need to be able to sense lines in the road. If the car pulls up to an intersection, it needs to stop at the line. It can do this using a sensor that detects the line.

Careers: automotive software engineer, electrical engineer, digital sculptor

STEM Connections

- Science
  - Velocity
  - Acceleration
- Technology
  - Sensors
  - while() loops
- Engineering
  - Problem solving
  - Line following
- Math
  - Logic statements
  - Speed calculations

Block-Text Correlation



```
Arduino run first:
    pulse Begin
    pulse Invert Motor  Motor 1
Arduino loop forever:
    if  pulse Line Finder Sensor  Digital Sensor Port # D2  =  0
    do  pulse Set Motor Powers (-100 to 100)  Motor 1  35  Motor 2  35
    if  pulse Line Finder Sensor  Digital Sensor Port # D2  =  1
    do  pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
        repeat while  pulse Line Finder Sensor  Digital Sensor Port # D2  =  1
        do  pulse Set Red LED ON
            wait 500 milliseconds
            pulse Set Red LED OFF
            wait 500 milliseconds
```

```
void setup() {
pulse.PulseBegin();
pulse.setMotorInvert(1,1);
if (pulse.readLineSensor(2) == 0) {
pulse.setMotorPowers(35,35);
if (pulse.readLineSensor(2) == 1) {
pulse.setMotorPowers(0,0);
while (pulse.readLineSensor(2) == 1) {
pulse.setRedLED(HIGH);
delay(500);
pulse.setRedLED(LOW);
delay(500);
```

Note: This if() loop runs the motors if the reading is 0.

Note: This if() loop stops the motors if the reading is 1.

Note: This while() loop blinks the LED off and on while the reading is 1.

Arduino Source Code

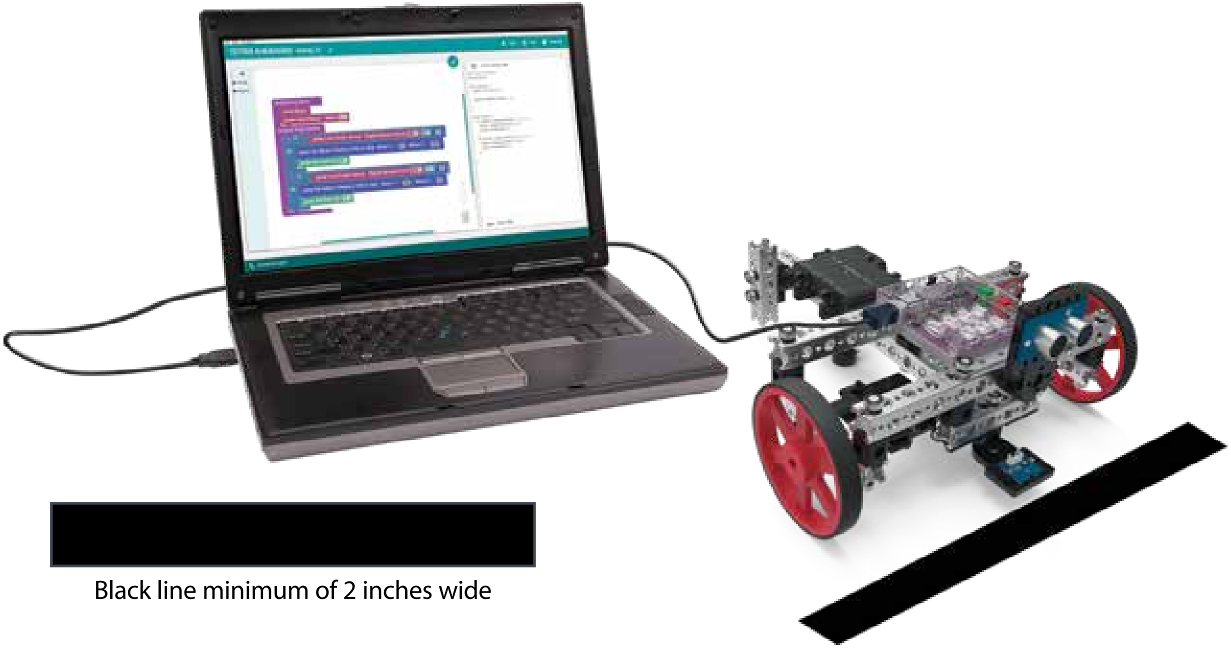
```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

pulse.setMotorInvert(1,1);
}
void loop() {
if (pulse.readLineSensor(2) == 0) {
pulse.setMotorPowers(35,35);
}
if (pulse.readLineSensor(2) == 1) {
pulse.setMotorPowers(0,0);
while (pulse.readLineSensor(2) == 1) {
pulse.setRedLED(HIGH);
delay(500);
pulse.setRedLED(LOW);
delay(500);
}
}
}
```

Activity 12: Follow a Line

Introduction

For this activity, you can take what you learned in the previous activity and apply it in a slightly different way to create a new behavior. This will enable the PULSE Codee Bot to follow a line.

Parts Needed



Open the Program

Let's start by looking at the example sketch. Open the sketch by selecting **Examples** > **Activity_12**. A new sketch window will open titled Activity_12 (Figure 60).

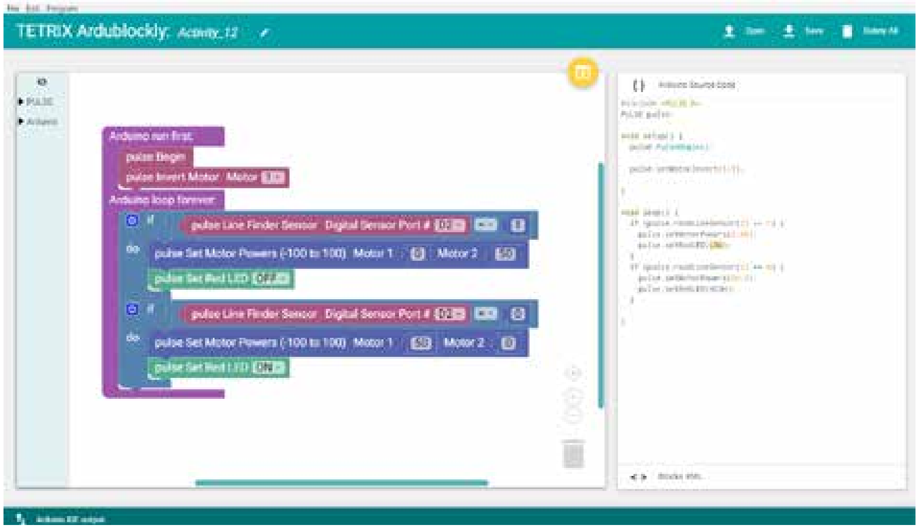

TETRIX Ardublockly: Activity_12
PULSE
Arduino
Arduino run first:
pulse Begin
pulse Invert Motor Motor 1
Arduino loop forever:
if pulse Line Finder Sensor Digital Sensor Port # D2 = 1
do pulse Set Motor Powers (-100 to 100) Motor 1 0 Motor 2 50
pulse Set Red LED OFF
if pulse Line Finder Sensor Digital Sensor Port # D2 = 0
do pulse Set Motor Powers (-100 to 100) Motor 1 50 Motor 2 0
pulse Set Red LED ON
{ } Arduino Source Code

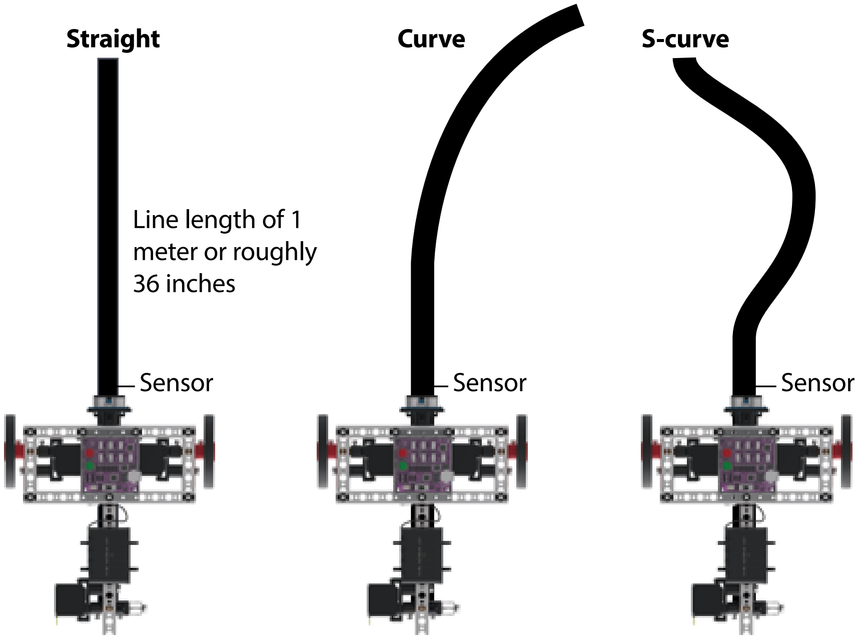
Figure 60

Execute the Code

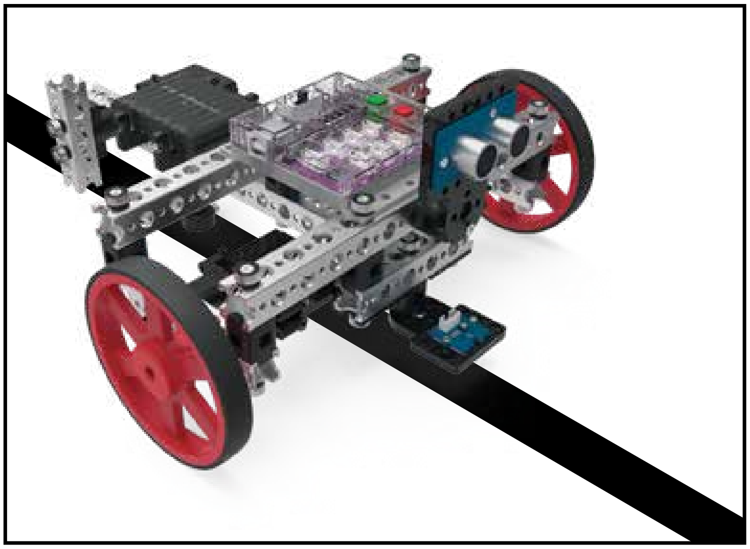
Before you can upload the sketch to the PULSE, remember to check your connections. The Line Finder Sensor should be in D2. Upload the sketch. The green LED will light up, indicating the code is ready to execute. When this has occurred, disconnect the USB cable and set the Codee Bot on the white or reflective surface.

To run this code, you will need to place a black stripe on a white or reflective surface for the robot to follow. White glossy cardboard, foam board, poster board, or several sheets of paper tiled together can work for the surface.

The black stripe can be made using electrical tape. You can make a curvy path or just a straight line. There will be a limitation on how sharp a curve your robot will be able to follow, so do not make the curves too tight.



The Codee Bot should be placed 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 Stop/Reset button to end the sketch. Based on the sketch comments, did the behavior match expectations?



Tip: You might need to nudge the bot to get it to move because the wheels might slip on smooth surfaces.

Troubleshooting: 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-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.

Further Investigate

This sketch uses two if-do blocks. Two actions will be executed within each if-do block.

The first action of the if-do block deals with the motors. The if-do block tells the Codee Bot to turn by sending power to one motor and braking the other based on the condition of the Line Finder Sensor. Each if-do block is the opposite of the other. This creates a series of turns that the Codee Bot uses to follow the line.

The second action of the if-do block deals with the red LED on the PULSE. The if-do block 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-do block work together to make the Codee Bot follow the line while producing visual cues in the form of the blinking red LED.

Extension Activity

With the example as a reference, try creating a new sketch to have the Codee Bot follow a black line. Remember what you have learned from your previous activities. Challenge yourself to make the Codee Bot go faster and still follow the line accurately or change the code to make the Codee Bot follow the line in the opposite direction.

Real-World Link

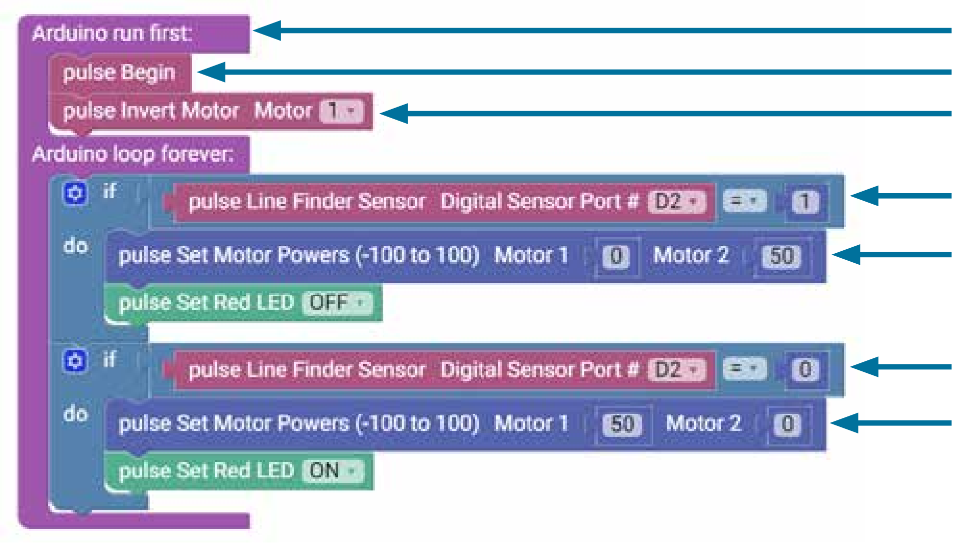
The next generation of children's rides are utilizing line-following sensors to operate the rides. Little electric cars can drive toddlers along a path. A tiny train can follow a figure eight path on the ground.

Careers: theme park manager, technical director, electrical ride control technician

STEM Connections

- Science
  - o Light reflection
  - o Angle of incidence
- Technology
  - o Thresholds
  - o Analog and digital
- Engineering
  - o Edging machines
  - o Self-driving vehicles
- Math
  - o Logic statements
  - o Numeric values

Block-Text Correlation



```
Arduino run first:
  pulse Begin
  pulse Invert Motor  Motor 1
Arduino loop forever:
  if  pulse Line Finder Sensor  Digital Sensor Port # D2  =  1
  do  pulse Set Motor Powers (-100 to 100)  Motor 1 0  Motor 2 50
      pulse Set Red LED OFF
  if  pulse Line Finder Sensor  Digital Sensor Port # D2  =  0
  do  pulse Set Motor Powers (-100 to 100)  Motor 1 50  Motor 2 0
      pulse Set Red LED ON
```

```
void setup() {
pulse.PulseBegin();
pulse.setMotorInvert(1,1);
if (pulse.readLineSensor(2) == 1) {
pulse.setMotorPowers(0,50);
if (pulse.readLineSensor(2) == 0) {
pulse.setMotorPowers(50,0);
```

Note: One wheel will turn and the other wheel will brake, depending on the readings from the Line Finder Sensor.

Arduino Source Code

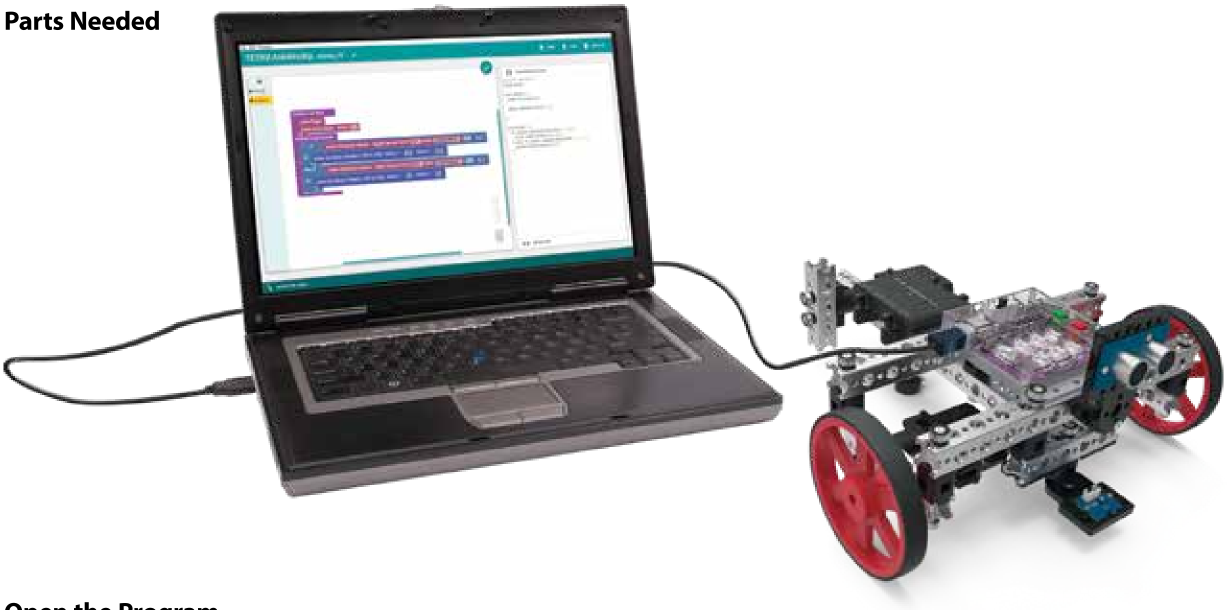
```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

pulse.setMotorInvert(1,1);
}

void loop() {
if (pulse.readLineSensor(2) == 1) {
pulse.setMotorPowers(0,50);
pulse.setRedLED(LOW);
}
if (pulse.readLineSensor(2) == 0) {
pulse.setMotorPowers(50,0);
pulse.setRedLED(HIGH);
}

}
```

Activity 13: Drive Toward a Wall and Stop

Introduction

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

Parts Needed



Open the Program

Let's start by looking at the example sketch. Open the sketch by selecting **Examples** > **Activity_13**. A new sketch window will open titled Activity_13 (Figure 61).

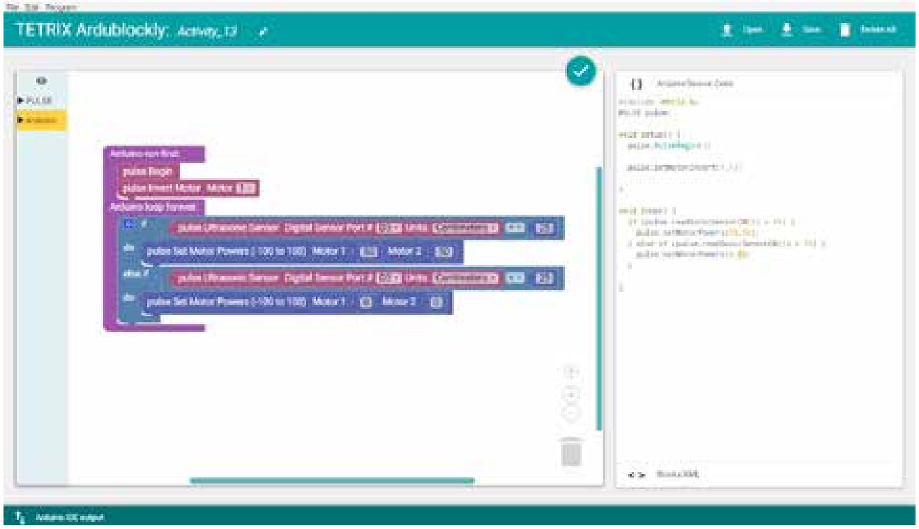

TETRIX Ardublockly: Activity_13
Open
Save
PULSE
Arduino
Arduino run first:
pulse Begin
pulse Invert Motor Motor
Arduino loop forever
if
pulse Ultrasonic Sensor Digital Sensor Port # Units Centimeters
do
pulse Set Motor Powers (-100 to 100) Motor 1 Motor 2
else if
pulse Ultrasonic Sensor Digital Sensor Port # Units Centimeters
do
pulse Set Motor Powers (-100 to 100) Motor 1 Motor 2
Arduino Source Code
Arduino IDE output

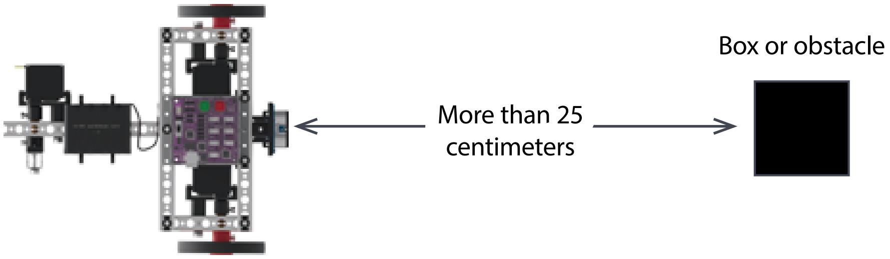
Figure 61

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your connections. The Ultrasonic Sensor should be in D3. 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 Codee Bot on the floor.

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

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



Further Investigate

This sketch uses an if-else block. An if-else block gives greater control over the flow of code than the basic if block. 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 block says if this condition is true, do this. The else part of the block says if this condition is false, do this instead.

Troubleshooting: Be sure to check that the Ultrasonic Sensor is plugged into the correct sensor port and is plugged in correctly. Keep in mind that objects without adequate surface size or that have an irregular surface might not be detected or might affect the distance the sensor reads.

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

On the if block, you can choose from different programming options. You can use the basic if-do block, and then the program will do the action within the loop. In this activity, you are using the if-do; else if-do block. Another option is the if-do, else-do block. You have to click the blue gear icon to change the block type (Figures 62 and 63). Then you drag over the statement you plan to use underneath the if block on the right side (Figure 64).

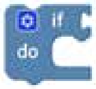


Figure 62

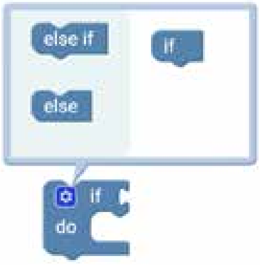


Figure 63

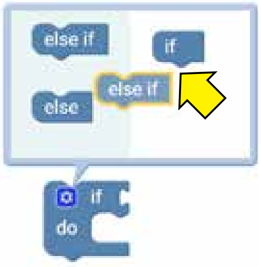


Figure 64

Extension Activity

With the example as a reference, try creating a new sketch to have the Codee Bot drive toward a wall or object and stop. Remember what you have learned from your 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.

Real-World Link

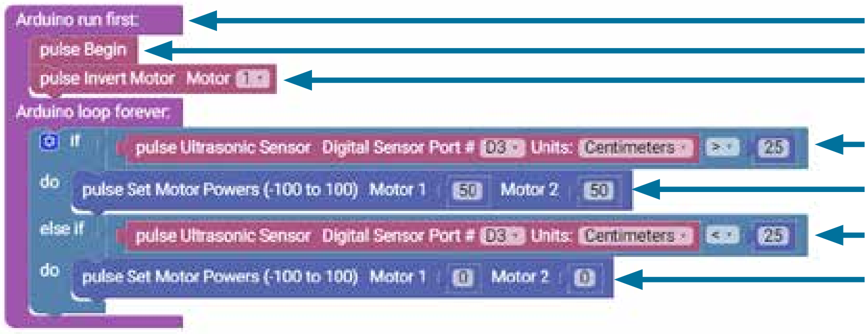
Small vacuum robots that clean the floors of homes use sensors to detect if there are obstacles in their path. If the little robot encounters a couch, it is programmed to back up and turn to avoid the obstacle. It also has sensors to detect stairs so that it stops and doesn't fall down.

Careers: vacuum engineer, vacuum coater, mechanical drafter

STEM Connections

- Science
    - o Speed of sound
    - o Sound waves
- Technology
    - o Sensor calibration
    - o Ultrasonic
- Engineering
    - o Obstacle detection
    - o Braking
- Math
    - o Distance from objects
    - o Metric and standard units

Block-Text Correlation



```
Arduino run first:
    pulse Begin
    pulse Invert Motor  Motor 1
Arduino loop forever:
    if      pulse Ultrasonic Sensor  Digital Sensor Port # D3  Units: Centimeters  >  25
    do      pulse Set Motor Powers (-100 to 100)  Motor 1  50  Motor 2  50
    else if pulse Ultrasonic Sensor  Digital Sensor Port # D3  Units: Centimeters  <  25
    do      pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
```

```
void setup() {
pulse.PulseBegin();
pulse.setMotorInvert(1,1);
if (pulse.readSonicSensorCM(3) > 25) {
pulse.setMotorPowers(50,50);
} else if (pulse.readSonicSensorCM(3) < 25)  {
pulse.setMotorPowers(0,0);
```

Note: The bot will drive forward until it encounters an obstacle 25 cm away, and then it will brake.

Arduino Source Code

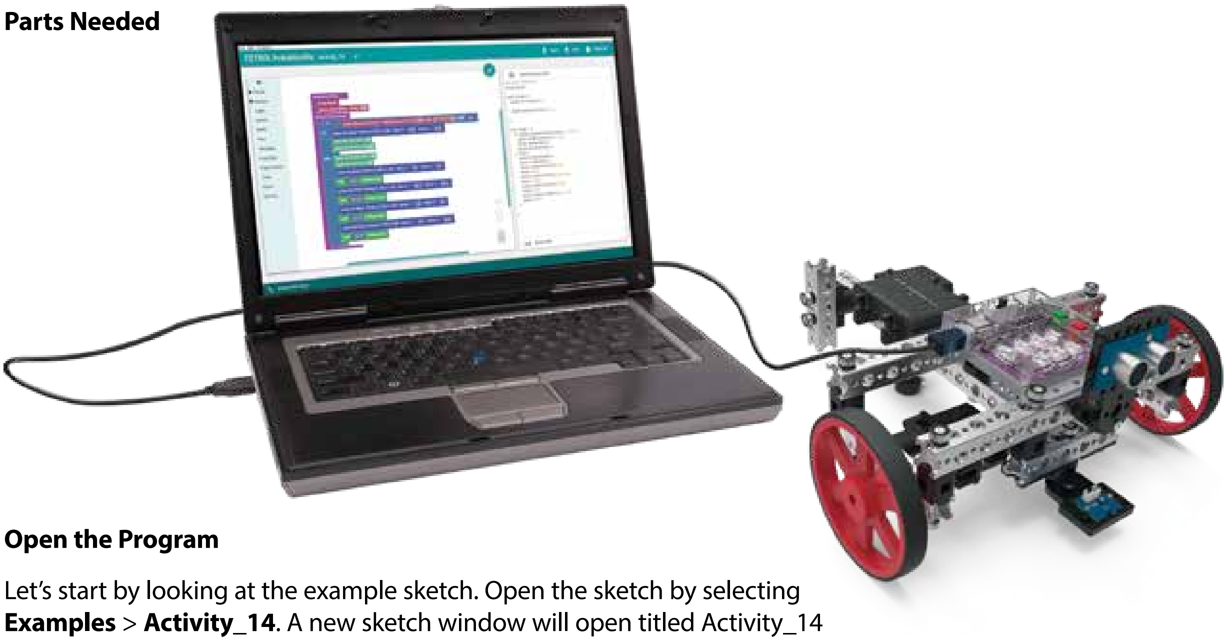
```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

pulse.setMotorInvert(1,1);
}

void loop() {
if (pulse.readSonicSensorCM(3) > 25) {
pulse.setMotorPowers(50,50);
} else if (pulse.readSonicSensorCM(3) < 25) {
pulse.setMotorPowers(0,0);
}

}
```

Activity 14: Avoiding Obstacles

Introduction

For this activity, you are going to expand on the if-else block by adding actions. This will enable the PULSE Codee Bot to avoid obstacles in its path.

Parts Needed



Open the Program

Let's start by looking at the example sketch. Open the sketch by selecting **Examples** > **Activity_14**. A new sketch window will open titled Activity_14 (Figure 65).

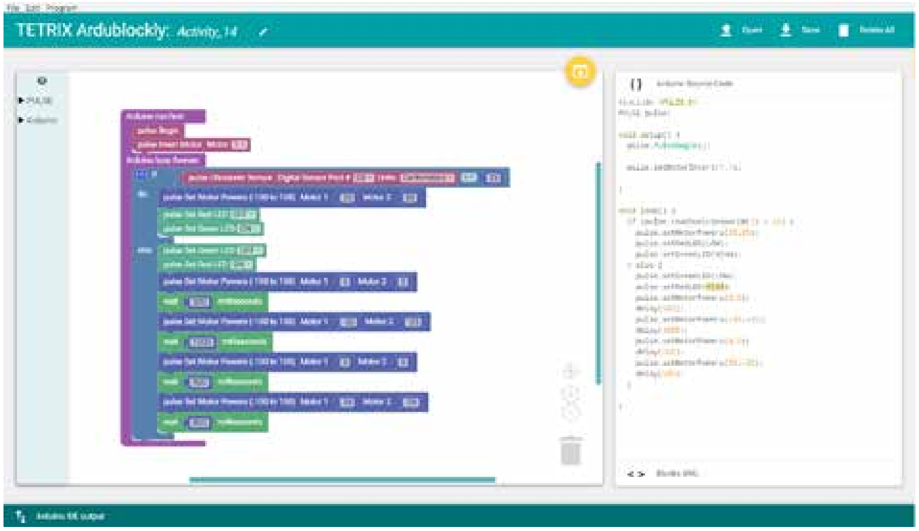


Figure 65

Execute the Code

Before you can upload the sketch to the PULSE, remember to check your connections. The Ultrasonic Sensor should be in D3. 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 Codee Bot on the floor.

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

To begin, press the green Start button. Your robot will travel forward until it senses that it is within 25 cm 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 Stop/Reset button to end the sketch. Based on the sketch comments, did the behavior match expectations?

Further Investigate

This sketch uses the if-else block and uses multiple actions within the loop.

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

Extension Activity

With the example as a reference, try creating a new sketch to have the Codee Bot avoid obstacles. Remember what you learned from your previous activities.

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

Tip: When the robot approaches an obstacle at an angle, it might have a challenge sensing the object.

Real-World Link

Some new vehicles have sensors in the front that sense if an object is in front of them. Imagine a deer running out in front of the car you're riding in. The car would automatically brake because it sensed the deer in front of it. The car avoided an accident that would have occurred because of the collision with a deer.

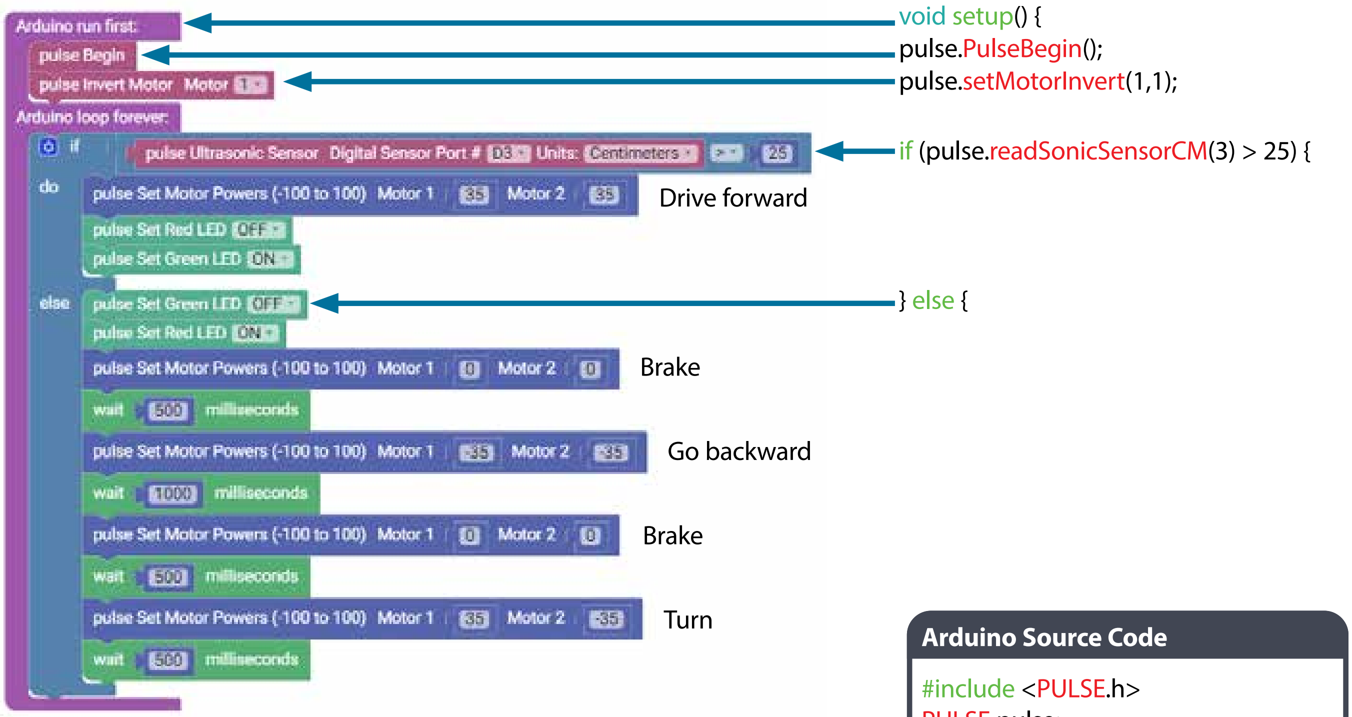
Careers: test engineer, materials development engineer, measurement lab technician

Troubleshooting: Be sure to check that the Ultrasonic Sensor is plugged into the correct sensor port and is plugged in correctly. Keep in mind that objects without adequate surface size or that have an irregular surface might not be detected or might affect the distance the sensor reads.

STEM Connections

- Science
  - o Sound waves
  - o Echolocation
- Technology
  - o Logic statements
  - o Motor applications
- Engineering
  - o Obstacle avoidance
  - o Steering
- Math
  - o Graphing
  - o Angles

Block-Text Correlation



```
Arduino run first:
    pulse Begin
    pulse Invert Motor  Motor 1
Arduino loop forever:
    if  pulse Ultrasonic Sensor  Digital Sensor Port # D3  Units: Centimeters  >  25
    do    pulse Set Motor Powers (-100 to 100)  Motor 1  35  Motor 2  35
          pulse Set Red LED OFF
          pulse Set Green LED ON
    else  pulse Set Green LED OFF
          pulse Set Red LED ON
          pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
          wait  500  milliseconds
          pulse Set Motor Powers (-100 to 100)  Motor 1  -35  Motor 2  -35
          wait  1000  milliseconds
          pulse Set Motor Powers (-100 to 100)  Motor 1  0  Motor 2  0
          wait  500  milliseconds
          pulse Set Motor Powers (-100 to 100)  Motor 1  35  Motor 2  -35
          wait  500  milliseconds
```

void setup() {
pulse.PulseBegin();
pulse.setMotorInvert(1,1);
if (pulse.readSonicSensorCM(3) > 25) {
Drive forward
} else {
Brake
Go backward
Brake
Turn

Arduino Source Code

```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

pulse.setMotorInvert(1,1);
}

void loop() {
if (pulse.readSonicSensorCM(3) > 25) {
pulse.setMotorPowers(35,35);
pulse.setRedLED(LOW);
pulse.setGreenLED(HIGH);
} else {
pulse.setGreenLED(LOW);
pulse.setRedLED(HIGH);
pulse.setMotorPowers(0,0);
delay(500);
pulse.setMotorPowers(-35,-35);
delay(1000);
pulse.setMotorPowers(0,0);
delay(500);
pulse.setMotorPowers(35,-35);
delay(500);
}

}
```

Activity 15: Combining the Sensors

Introduction

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

Parts Needed

Open the Program

Let's start by looking at the example sketch. Open the sketch by selecting **Examples** > **Activity_15**. A new sketch window will open titled Activity_15 (Figure 66).

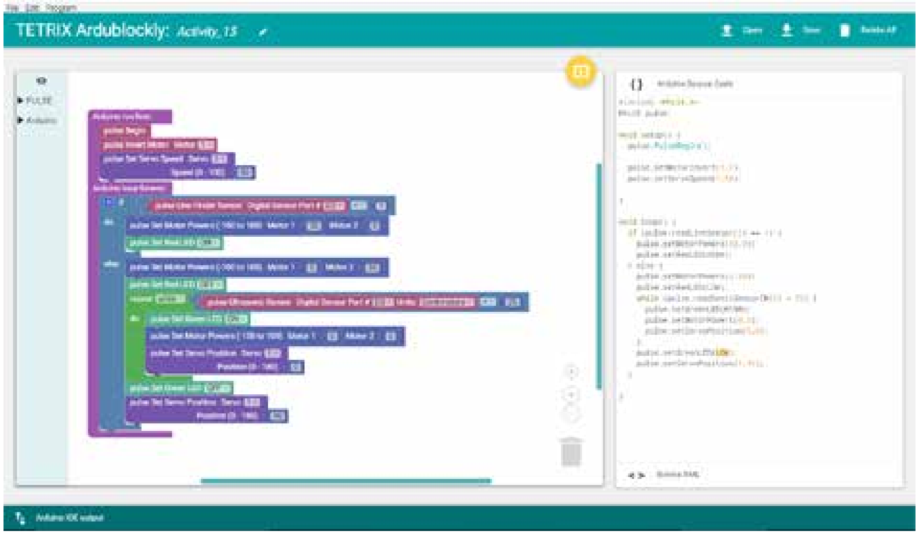


Figure 66

Execute the Code

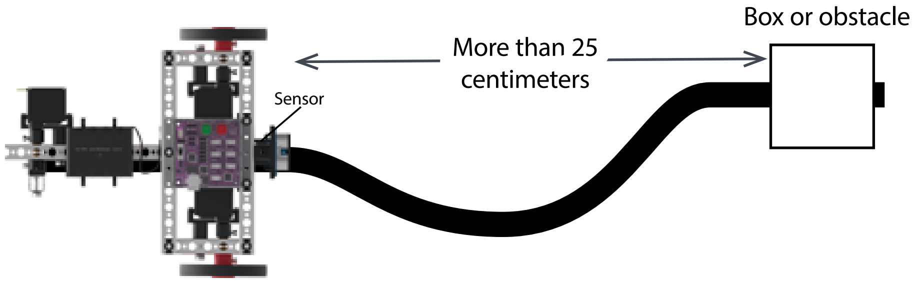
Before you can upload the sketch to the PULSE, remember to check your connections. The Line Finder Sensor will be in D2, and the Ultrasonic Sensor will be in D3.

You 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 cm, the robot will stop to avoid crashing into it. The robot will raise its beam 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 Codee Bot on the floor.

The Codee Bot 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 your robot at a distance well outside the 25-centimeter detection range. Cardboard boxes work well for this purpose. You do not want anything too heavy that could cause damage to your robot just in case your 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.

Tip: If you're struggling with your sensors, look back at previous activities for troubleshooting tips.

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

Further Investigate

This sketch uses an if-else block and a while() loop block, which you have covered before. This sketch is not about adding new elements. It is about combining the elements from the previous activities.

The if-else block in this sketch does the same line-following behavior as in Activity 12 with two if blocks but in a more controlled way. The while() loop block monitors the Ultrasonic Sensor and creates a behavior when an object is detected within 25 cm. This new behavior stops the Codee Bot and uses the servo to raise the Codee Bot's arm until the object is cleared from the Codee Bot's path. When the object is cleared, the Codee Bot will lower its arm and continue the original behavior of line following.

Extension Activity

With the example as a reference, try creating a new sketch to have the Codee Bot follow a black line and stop for obstacles. Remember what you learned from your previous activities.

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

Real-World Link

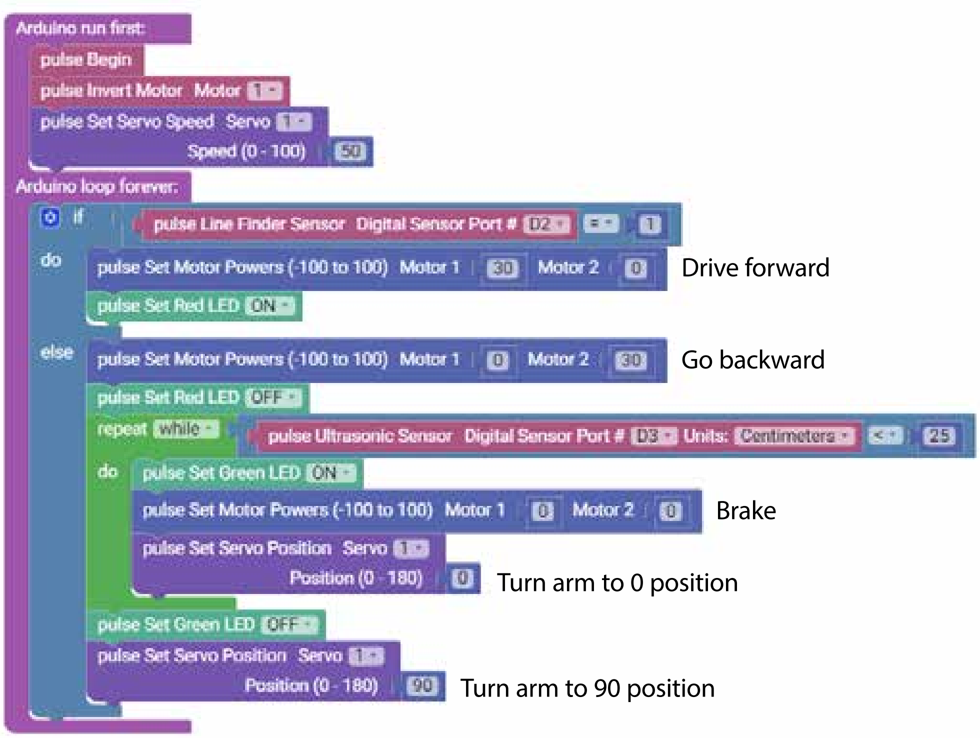
Many different sensors are used in the medical field. Some examples include those that determine your temperature, heart rate, breathing rate, and heartbeat pattern. Some robots are even used to perform surgeries or take pictures of the inside of your body.

Careers: surgical technologist, medical technician, biomedical engineer

STEM Connections

- Science
    - Senses
    - Functions
- Technology
    - Sensor readings
    - Course design
- Engineering
    - Problem solving
    - Sensory data integration
- Math
    - Data distribution
    - Duration

Block-Text Correlation



```
Arduino run first:
    pulse Begin
    pulse Invert Motor  Motor 1
    pulse Set Servo Speed  Servo 1
                           Speed (0 - 100) 50
Arduino loop forever:
    if    pulse Line Finder Sensor  Digital Sensor Port # D2  =  1
    do    pulse Set Motor Powers (-100 to 100)  Motor 1 30  Motor 2 0
          pulse Set Red LED ON
    else  pulse Set Motor Powers (-100 to 100)  Motor 1 0  Motor 2 30
          pulse Set Red LED OFF
          repeat while  pulse Ultrasonic Sensor  Digital Sensor Port # D3  Units: Centimeters  <  25
          do  pulse Set Green LED ON
              pulse Set Motor Powers (-100 to 100)  Motor 1 0  Motor 2 0
              pulse Set Servo Position  Servo 1
                                        Position (0 - 180) 0
          pulse Set Green LED OFF
          pulse Set Servo Position  Servo 1
                                    Position (0 - 180) 90
```

Drive forward

Go backward

Brake

Turn arm to 0 position

Turn arm to 90 position

Arduino Source Code

```
#include <PULSE.h>
PULSE pulse;

void setup() {
pulse.PulseBegin();

pulse.setMotorInvert(1,1);
pulse.setServoSpeed(1,50);
}

void loop() {
if (pulse.readLineSensor(2) == 1) {
pulse.setMotorPowers(30,0);
pulse.setRedLED(HIGH);
} else {
pulse.setMotorPowers(0,30);
pulse.setRedLED(LOW);
while (pulse.readSonicSensorCM(3) < 25) {
pulse.setGreenLED(HIGH);
pulse.setMotorPowers(0,0);
pulse.setServoPosition(1,0);
}
pulse.setGreenLED(LOW);
pulse.setServoPosition(1,90);
}

}
```

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

You have built a robot, you have programmed PULSE, 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 PRIME robot using the PULSE controller, TETRIX Ardublockly, 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, here are some favorite resources.

Please be sure to check out the appendix of this guide, which includes a vast array of additional resources.

Want more challenges?

Complete the activities contained within this guide through the text-based Arduino Software (IDE). Re-create the block-based programs in actual text code. Make modifications and attempt the extension activities using this code. This can vastly extend the amount of time you can spend on this guide.

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

TETRIX PULSE is built on Arduino architecture and is thus usable with programming languages that support Arduino. This enables PULSE 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 the TETRIX building system:

The parts and pieces used in this guide are only a portion of the elements available in the TETRIX PRIME system. Visit **Pitsco.com/TETRIX** 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.

Scope and Sequence

The scope and sequence contains estimated timelines for each activity based on the students completing all sections. Students work at different paces, so the time that each activity could take will vary. The time estimate also includes the completion of all the extension activities. This scope and sequence does not include time to complete the introduction section of the programming guide.

Lesson	Time Frame
Introduction	60 minutes
Activity 1: Hello World!	20 minutes
Activity 2: Moving Your DC Motors	20 minutes
Activity 3: Moving Your Servo Motors	20 minutes
Activity 4: Using the Line Finder Sensor	30 minutes
Activity 5: Introduction to the Ultrasonic Sensor	30 minutes
Activity 6: Build the PULSE Codee Bot	60 minutes
Activity 7: Drive Forward	60 minutes
Activity 8: Drive in a Circle	60 minutes
Activity 9: Drive in a Square	60 minutes
Activity 10: Simplify the Square	60 minutes
Activity 11: Drive to a Line and Stop	60 minutes
Activity 12: Follow a Line	60 minutes
Activity 13: Drive Toward a Wall and Stop	60 minutes
Activity 14: Avoiding Obstacles	60 minutes
Activity 15: Combining the Sensors	60 minutes

Total Time: 780 minutes (13 hours)

Standards Addressed

Common Core State Standards – English Language Arts/Literacy

Grade 6

- SL.6.1
    - SL.6.1.A
    - SL.6.1.B
    - SL.6.1.C
    - SL.6.1.D
- SL.6.4
- RST.6-8.3
- RST.6-8.4

Grade 7

- SL.7.1
    - SL.7.1.A
    - SL.7.1.B
    - SL.7.1.C
    - SL.7.1.D
- SL.7.4
- RST.6-8.5
- RST.6-8.6

Grade 8

- SL.8.1
    - SL.8.1.A
    - SL.8.1.B
    - SL.8.1.C
    - SL.8.1.D
- SL.8.4
- RST.6-8.10

Common Core State Standards – Mathematics

Math Practice

- MP1
- MP2

Math Content

- 6.NS.C.5

ITEEA Standards for Technological Literacy

Grades 6-8

- 1.F
- 1.G
- 1.H
- 2.M
- 2.N
- 2.O
- 2.V
- 3.D
- 3.E
- 3.F
- 4.D
- 7.D
- 7.E
- 8.E
- 9.H
- 10.H
- 12.H
- 12.I
- 12.J
- 12.K
- 14.G
- 15.G
- 16.G
- 16.H
- 17.H
- 18.F
- 18.G
- 18.I
- 19.F

Glossary

abort: to stop a program or sketch from continuing to execute (on PULSE, press the red Stop/Reset button)

Arduino Software (IDE): open-source software used to program Arduino-based hardware, such as PULSE

autonomous: a robot whose actions occur by use of a microprocessor without human intervention

beam: a square aluminum part with a set pattern of holes used as a structural building component

behavior: the physical action(s) of a robot

called function: an instruction set that is performed from within the main program or sketch

code: programming instructions

comparison statement: code that compares the value of two variables

condition: code that uses both logic and comparison of values

dead reckoning: process of calculating position by estimating the direction and distance previously traveled

declaration: a statement providing a name for a given function, variable, or constant

delay: code that provides a time interval between statements or actions

differential drive: a drive system in which the speeds of the motors located on opposite sides of the robot control the direction of travel

end effector: device on a robot that interacts with its environment

execute: to carry out an instruction or a program

expected behavior: the action of a robot based on how you think it will execute a set of instructions

for loop: a set of instructions that repeats a set number of times

function: a procedure within a computer program

HIGH: typically the "on" value of a switch or sensor

if/else: instruction statement that chooses between two paths based on value comparisons

if/then: instruction statement that provides an action based on a value comparison

increment: a specific amount of increase, typically used in code loops

initialize: to set a beginning value of a variable or the starting point of a code configuration

integer: a positive or negative whole number, including zero

LED: light-emitting diode

Line Finder Sensor: hardware that can detect contrast in light and dark surfaces to direct the path of a robot

logic: the science of formal principles of reasoning or correct inference; logic is governed by mathematical principles

LOW: typically the "off" value of a switch or sensor

main loop: also known as a void loop within C-based programming, this loop contains the instructional code for controlling the robot

millisecond: one-one-thousandth of a second

neutral position: the center position of a servo motor

obstacle: a physical object that blocks the path of a robot

parameter: a factor or condition to meet within the final solution of a problem

pseudocode: descriptive steps (based on simple language) to be taken within a computer program

reflected light: the amount of light that travels from a source to an object and returns to the source

sensor: an electronic device used to detect light, sound, motion, or energy and relay the information to a processor

servo: short for *servo motor* – a rotary actuator capable of providing precise control of angular position and speed; it consists of a motor coupled with a sensor for position feedback

sketch: a set of instructions (a computer program) within the Arduino Software (IDE) platform

standard servo: a servo that can rotate to a known angular position (typically 0°-180°) through internal sensor feedback

statement: one line of code; an instruction to be executed

syntax: the precise way computer code is written to be understood by the computer or microprocessor

TETRIX Ardublockly: a plug-in interface that enables code to be written using graphic blocks rather than syntax-based coding and displays the coding to assist you in troubleshooting and learning to code

Ultrasonic Sensor: hardware that uses sound waves to determine the distance of an object from the sensor

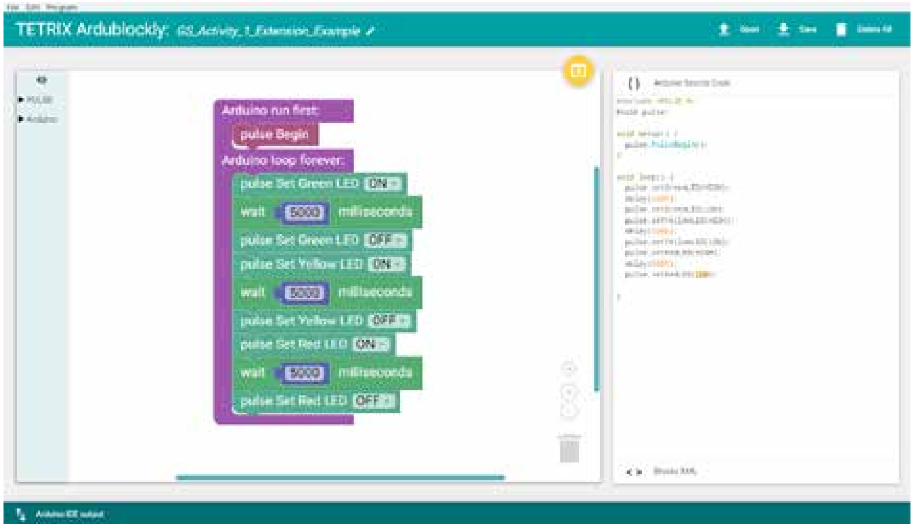
upload: to move a sketch or file to a computer or microprocessor

variable: a named data item that might have one or more values during the execution of a program

void setup: instruction that provides the initialization of a program

while loop: a set of instructions that continues to execute over and over based on a logic statement

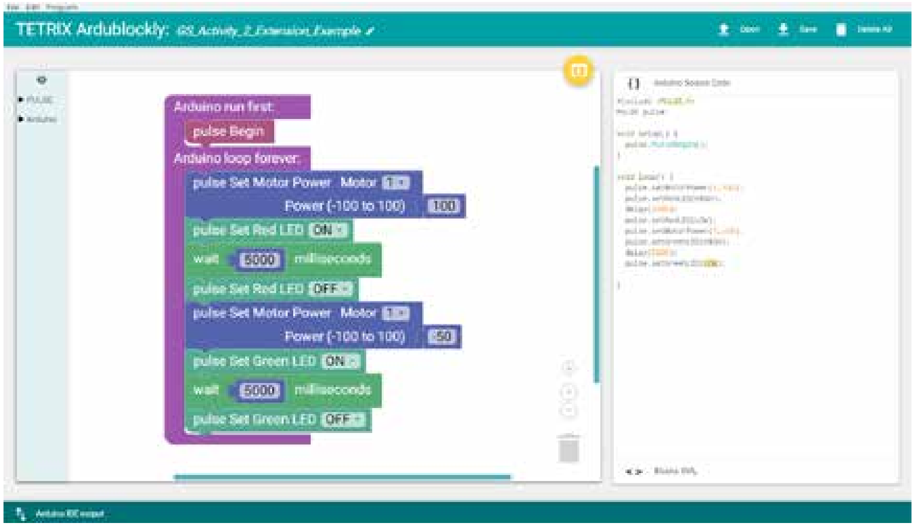
Getting Started Extension Activities



```
Arduino run first:
  pulse Begin
Arduino loop forever:
  pulse Set Green LED ON
  wait 5000 milliseconds
  pulse Set Green LED OFF
  pulse Set Yellow LED ON
  wait 5000 milliseconds
  pulse Set Yellow LED OFF
  pulse Set Red LED ON
  wait 5000 milliseconds
  pulse Set Red LED OFF
```

```
#include <PULSE.h>
PULSE pulse;

void setup() {
  pulse.PulseBegin();
}

void loop() {
  pulse.setGreenLED(HIGH);
  delay(5000);
  pulse.setGreenLED(LOW);
  pulse.setYellowLED(HIGH);
  delay(5000);
  pulse.setYellowLED(LOW);
  pulse.setRedLED(HIGH);
  delay(5000);
  pulse.setRedLED(LOW);

}
```

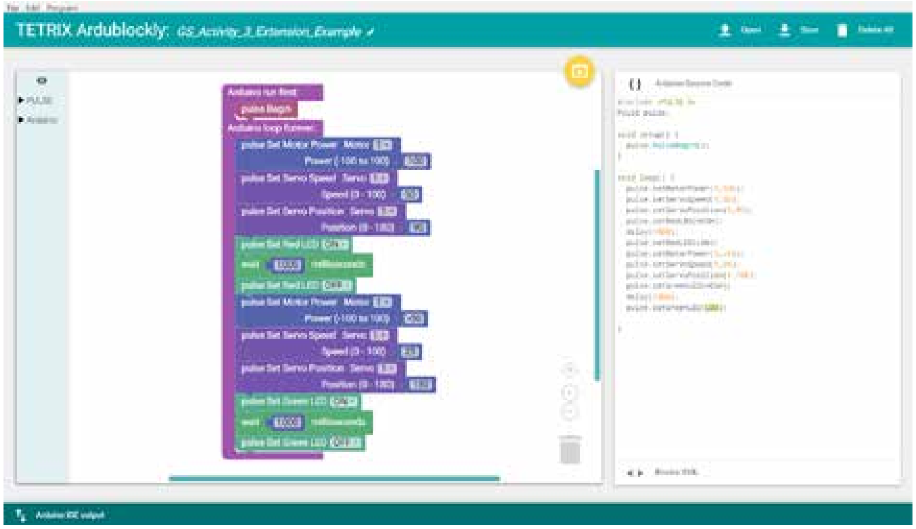
GS Activity 1 extension



```
Arduino run first:
  pulse Begin
Arduino loop forever:
  pulse Set Motor Power  Motor 1
                 Power (-100 to 100) 100
  pulse Set Red LED ON
  wait 5000 milliseconds
  pulse Set Red LED OFF
  pulse Set Motor Power  Motor 1
                 Power (-100 to 100) -50
  pulse Set Green LED ON
  wait 5000 milliseconds
  pulse Set Green LED OFF
```

```
#include <PULSE.h>
PULSE pulse;

void setup() {
  pulse.PulseBegin();
}

void loop() {
  pulse.setMotorPower(1, 100);
  pulse.setRedLED(HIGH);
  delay(5000);
  pulse.setRedLED(LOW);
  pulse.setMotorPower(1, -50);
  pulse.setGreenLED(HIGH);
  delay(5000);
  pulse.setGreenLED(LOW);

}
```

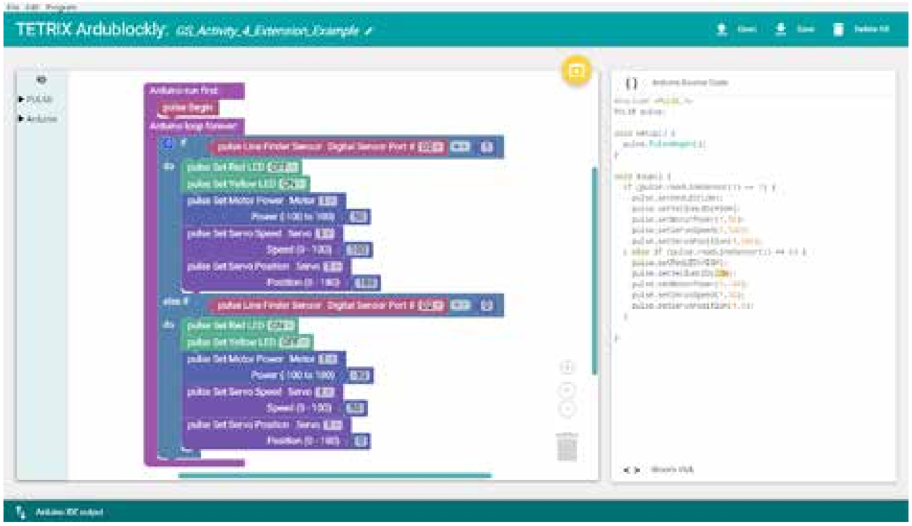
GS Activity 2 extension



```
Arduino run first:
  pulse Begin
Arduino loop forever:
  pulse Set Motor Power  Motor 1
                 Power (-100 to 100) 100
  pulse Set Servo Speed  Servo 1
                 Speed (0 - 100) 50
  pulse Set Servo Position  Servo 1
                 Position (0 - 180) 90
  pulse Set Red LED ON
  wait 1000 milliseconds
  pulse Set Red LED OFF
  pulse Set Motor Power  Motor 1
                 Power (-100 to 100) -50
  pulse Set Servo Speed  Servo 1
                 Speed (0 - 100) 25
  pulse Set Servo Position  Servo 1
                 Position (0 - 180) 180
  pulse Set Green LED ON
  wait 1000 milliseconds
  pulse Set Green LED OFF
```

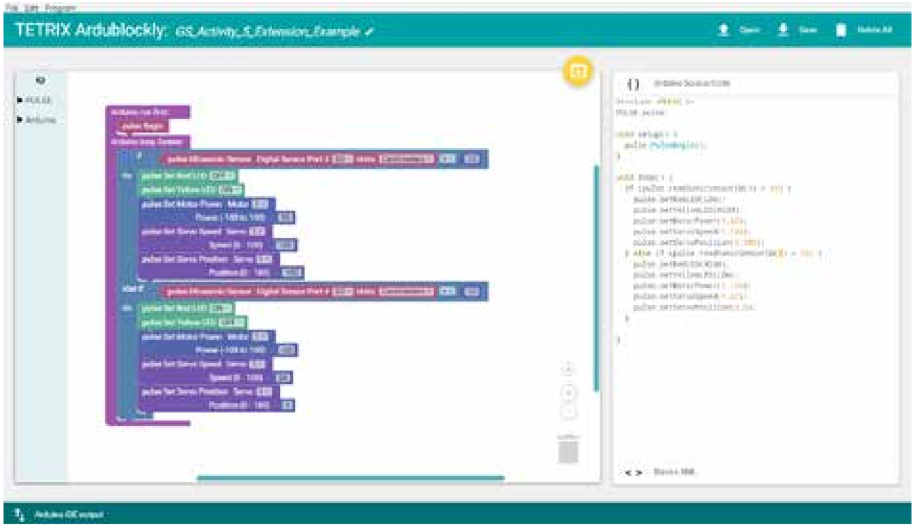
```
#include <PULSE.h>
PULSE pulse;

void setup() {
  pulse.PulseBegin();
}

void loop() {
  pulse.setMotorPower(1, 100);
  pulse.setServoSpeed(1, 50);
  pulse.setServoPosition(1, 90);
  pulse.setRedLED(HIGH);
  delay(1000);
  pulse.setRedLED(LOW);
  pulse.setMotorPower(1, -50);
  pulse.setServoSpeed(1, 25);
  pulse.setServoPosition(1, 180);
  pulse.setGreenLED(HIGH);
  delay(1000);
  pulse.setGreenLED(LOW);

}
```

GS Activity 3 extension



```
#include <PULSE.h>
PULSE pulse;

void setup() {
  pulse.PulseBegin();
}

void loop() {
  if (pulse.readLineSensor(1) == 1) {
    pulse.setRedLED(LOW);
    pulse.setYellowLED(HIGH);
    pulse.setMotorPower(1, 50);
    pulse.setServoSpeed(1, 100);
    pulse.setServoPosition(1, 180);
  } else if (pulse.readLineSensor(1) == 0) {
    pulse.setRedLED(HIGH);
    pulse.setYellowLED(LOW);
    pulse.setMotorPower(1, -50);
    pulse.setServoSpeed(1, 50);
    pulse.setServoPosition(1, 0);
  }

}
```

GS Activity 4 extension



```
#include <PULSE.h>
PULSE pulse;

void setup() {
  pulse.PulseBegin();
}

void loop() {
  if (pulse.readSonicSensorCM(8) < [illegible]) {
    pulse.setRedLED(LOW);
    pulse.setYellowLED(HIGH);
    pulse.setMotorPower(1, 50);
    pulse.setServoSpeed(1, 100);
    pulse.setServoPosition(1, 180);
  } else if (pulse.readSonicSensorCM(8) > [illegible]) {
    pulse.setRedLED(HIGH);
    pulse.setYellowLED(LOW);
    pulse.setMotorPower(1, -50);
    pulse.setServoSpeed(1, 50);
    pulse.setServoPosition(1, 0);
  }

}
```

GS Activity 5 extension

Careers Complete Listing

- air traffic controller
- automotive software engineer
- biomedical engineer
- car designer
- CNC programmer
- digital sculptor
- electrical engineer
- electrical ride control technician
- electromechanical technician
- electronic engineer
- engineer
- fabricator
- farm equipment mechanic
- groundskeeper
- industrial engineer
- irrigation engineer
- locomotive engineer
- machine operator
- machinery maintenance worker
- machinist
- manufacturing engineer
- manufacturing technician
- mapping technician
- materials development engineer
- materials engineer
- measurement lab technician
- mechanical drafter
- mechanical engineer
- medical technician
- PLC programmer
- railroad conductor
- railroad worker
- roboticist
- small-engine mechanic
- software developer
- sonar technician
- sound engineering technician
- surgical technologist
- technical director
- test engineer
- theme park manager
- traffic light engineer
- traffic technician
- vacuum coater
- vacuum engineer

TETRIX PULSE Robotics Controller Technical Specifications

Microcontroller:	ATmega328P with Arduino Optiboot bootloader installed
Memory:	32 KB flash programmable memory (ATmega328P)
Power:	6 volts DC using TETRIX PRIME 6 V NiMH Battery Pack
DC motor ports:	2 three-position header pins; H-bridge PWM controlled; 3 A continuous current each channel, 5 A peak
Compatible DC motor:	TETRIX PRIME 6-Volt DC Motor (44298)
DC motor control modes:	Constant power (-100% to 100%) Support for PID constant speed (-100 to 100 degrees per second) Support for PID constant speed to encoder target position and hold Support for PID constant speed to encoder degree position and hold
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
Compatible servo motor:	Standard-Scale Servo Motor (40538)
Total servo power limit:	6 volts DC, 6 A max
Servo control modes:	Set servo speed (0% to 100%) Set servo position (0-180 degrees)
Battery voltage monitoring:	0-7.5 volts range
3 digital sensor ports (D2-D4):	Each can be configured as digital input, digital output, or serial communication.
3 analog sensor ports (A1-A3):	Each can be configured as analog input or digital input or output ports.
1 I2C port (I2C):	100 kHz speed. This connection shares the same I2C bus as the internal DC motor and servo motor control chips. I2C addresses 0x01-0x06 reserved by the PULSE controller.
Battery connection port:	3-position pin header. Use only TETRIX PRIME 6 V NiMH Battery Pack.
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 yellow 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 LED:	Indicates DC motor rotation and direction for each DC motor channel

TETRIX PULSE Controller Sensor Port Pinout Diagrams

PULSE Sensor Ports

The PULSE controller uses Arduino UNO-compatible pin assignments. The sensors that are supported in the PULSE Arduino 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 code with additional sensors not yet supported in the PULSE library. With the exception of the I2C port, all can be configured as inputs or outputs using the Arduino pinMode() function. To learn more, visit the Language Reference section at **https://www.arduino.cc/en/Reference/HomePage**.

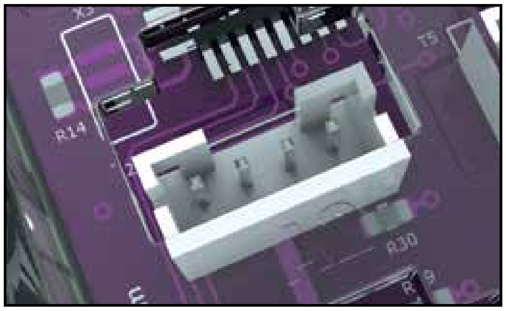


Figure 67: PULSE Sensor Port (Pins are left to right: 1, 2, 3, 4.)

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) | digital I/O (18)
Pin 4	SCL (I2C serial clock)	ADC5 input channel (A5) | digital I/O (19)

Note: The PULSE I2C port can be used only in I2C mode. It cannot 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: Analog sensor ports A1-A3 can also be configured as digital input/output.

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 sensor ports D2-D4 can also be configured as software-implemented serial ports using the Arduino Software (IDE) Serial Library.

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	Digital input/output	Digital I/O (10)
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	Digital input/output	Digital I/O (11)
Pin 4	Digital input/output	Digital I/O (4)

TETRIX PULSE Arduino Library Functions Chart

Description	Function	Coding Example
PULSE Begin Is called in the Arduino code setup() loop. Initializes the PULSE controller.	**PulseBegin**(); Data Type: None	**PulseBegin**(); *Reset and initialize PULSE controller.*
PULSE End When called, immediately terminates a program and resets the PULSE controller.	**PulseEnd**(); Data Type: None	**PulseEnd**(); *Terminate a PULSE program and reset controller.*

Set Red LED
Sets the PULSE 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 Yellow LED
Sets the PULSE yellow indicator LED to on or off.
	setYellowLED(state);
	
	Data Type:
	state = integer
	
	Data Range:
	state = 1 or 0
	or
	state = HIGH or LOW
		setYellowLED(HIGH);
		or
		setYellowLED(1);
		Turn red LED on.
		setYellowLED(LOW);
		or
		setYellowLED(0);
		Turn red LED off.

Set Green LED
Sets the PULSE 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 PULSE DC motor ports. Power level range is 0 to 100. Direction is set by the sign (+/-) of the power level. Power level 0 = stop motor.
	setMotorPower(motor#, power);
	
	Data Type:
	motor# = integer
	power = integer
	
	Data Range:
	motor# = 1 or 2
	power = -100 to 100
		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.

Set DC Motor Powers
Simultaneously sets the power level and direction of **both** TETRIX DC Motors connected to the PULSE motor ports. Both PULSE 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 motor.
	setMotorPowers(power1, power2);
	
	Data Type:
	power1 = integer
	power2 = integer
	
	Data Range:
	power1 = -100 to 100
	power2 = -100 to 100
		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.

Description	Function	Coding Example

Set DC Motor Speed
Uses velocity PID control to set the constant speed of a TETRIX DC Motor with a TETRIX motor encoder installed. 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.

Set DC Motor Speeds
Uses velocity PID control to simultaneously set the constant speeds of **both** TETRIX DC Motor channels with TETRIX motor encoders installed. Both PULSE 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 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 installed. Both PULSE 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 installed. 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 2 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 installed. Both PULSE 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 count 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 PULSE power-up or reset.*

Read Motor Busy Status
Reads the busy flag 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 (0 or 1)
		readMotorBusy(1);
		Return the busy status of Motor 1.
		readMotorBusy(2);
		Return the busy status of Motor 2.

Description	Function	Coding Example

Read Encoder Count
Reads the encoder count value. The PULSE controller uses encoder pulse data to implement PID control of a TETRIX DC Motor connected to the motor ports. The PULSE 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 PULSE controller uses encoder pulse data to implement PID control of a TETRIX DC Motor connected to the motor ports. The PULSE 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 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 PULSE 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-D4, or analog ports A1-A3 configured as digital input.*
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-D4, or analog ports A1-A3 configured as digital input.*
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-D4, or analog ports A1-A3.*
Read Battery Pack Voltage Reads the voltage of the TETRIX battery pack powering the PULSE 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 PULSE controller.* *Example: A value of 918 equals 9.18 volts.*
Read Start Button State Reads the state of the green PULSE 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 PULSE 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 PULSE 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 PULSE 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 PULSE 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 PULSE 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

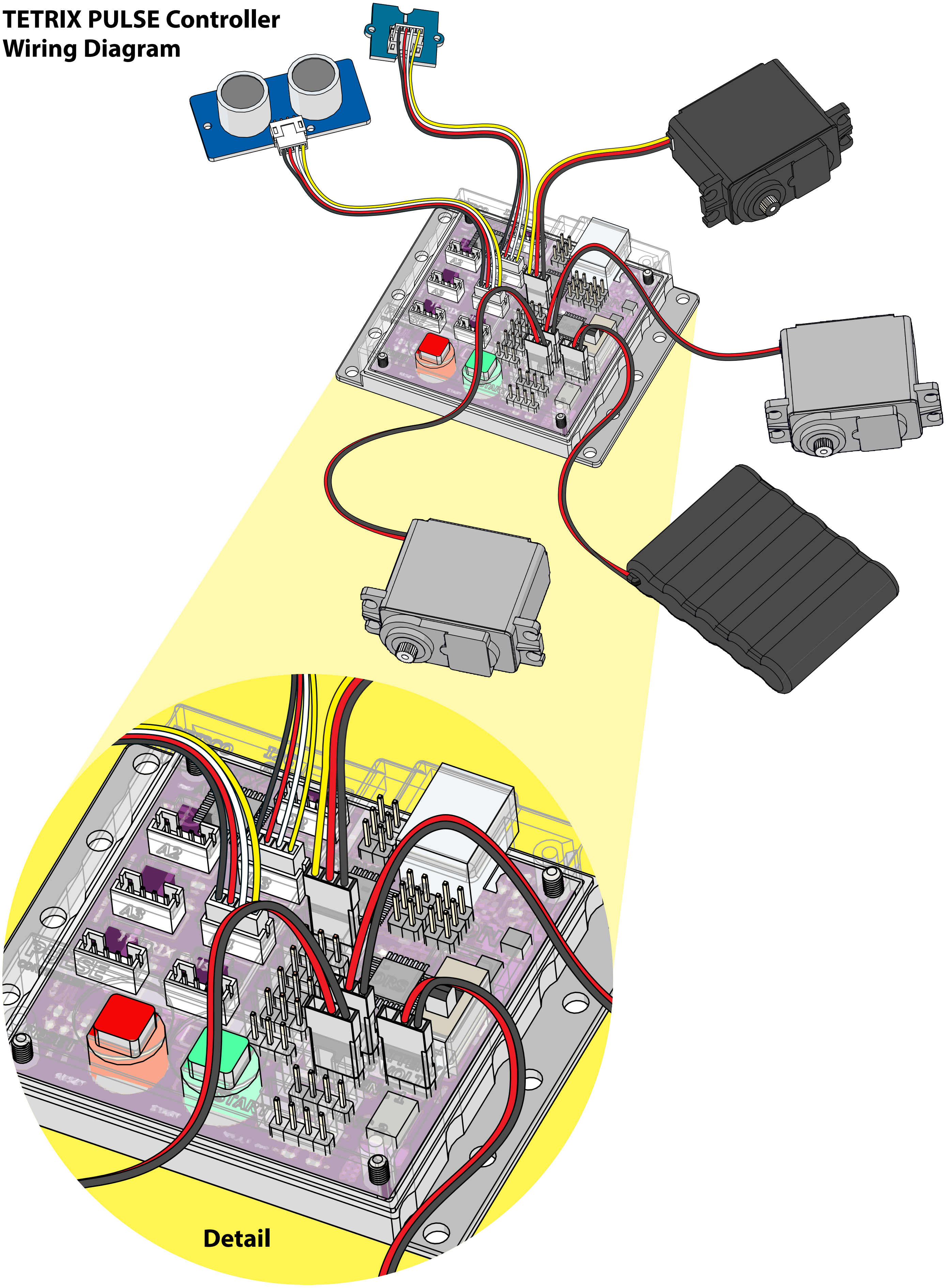
Read a Servo Position
Reads the most recent commanded position of a servo motor connected to PULSE servo ports 1-6. The value returned will be 0-180.
	readServoPosition(servo#);
	
	Data Type:
	servo# = integer
	
	Data Range:
	servo# = 1 to 6
	
	Data Type Returned:
	value = integer (0 to 180)
		readServoPosition(1);
		Read the most recent commanded position of Servo 1.
		readServoPosition(2);
		Read the most recent commanded position of Servo 2.

TETRIX PULSE Arduino Library Functions Cheat Sheet

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

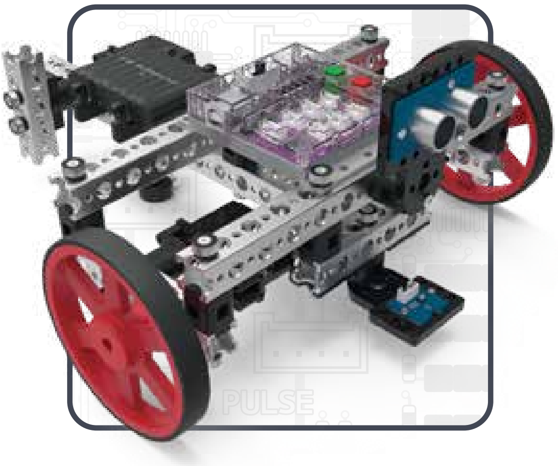
```
PulseBegin();
PulseEnd();
setRedLED(HIGH/LOW);
setYellowLED(HIGH/LOW);
setGreenLED(HIGH/LOW);
setMotorPower(motor#, power);
setMotorPowers(power1, power2);
setMotorSpeed(motor#, speed);
setMotorSpeeds(speed1, speed2);
setMotorTarget(motor#, speed, target);
setMotorTargets(speed1, target1, speed2, target2);
setMotorDegree(motor#, speed, degrees);
setMotorDegrees(speed1, degrees1, speed2, degrees2);
setMotorInvert(motor#, invert);
readMotorBusy(motor#);
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#);
```

TETRIX PULSE Controller Wiring Diagram



PITSCO TETRIX® PRIME

TETRIX® PULSE™ Robotics Controller Programming Guide



Call Toll-Free
800•835•0686

Visit Us Online at
Pitsco.com

PITSCO EDUCATION