

---

**XRP**

***Versión 2023.0***

**WPI**

**20 de febrero de 2024**



<b>1. Two robots in one</b>	<b>3</b>
<b>2. Software Tools</b>	<b>5</b>
<b>3. Building the XRP Robot</b>	<b>9</b>
<b>4. XRPCode Integrated Development Environment</b>	<b>29</b>
<b>5. Driving</b>	<b>35</b>
<b>6. Motors</b>	<b>39</b>
<b>7. Sensing the environment</b>	<b>43</b>
<b>8. Using the arm</b>	<b>47</b>
<b>9. Creating a dashboard</b>	<b>49</b>
<b>10. Miscellaneous topics</b>	<b>55</b>
<b>11. Indices and tables</b>	<b>57</b>





The XRP (Experiential Robotics Platform) [beta], started by WPI and DEKA Research & Development Corp., aims to level the STEM playing field globally and create a future generation of STEM innovators and technology leaders.

The robot kits you received are designed to operate autonomously and perform basic tasks. Its simple, tool-free assembly means robots can be built quickly, and replacement parts can be easily 3-D printed. As part of this platform, WPI will provide virtual support through online courses and will guide students and teachers through the new system, including the ability to scale up using the same hardware with free software updates.

The XRP platform is part of WPI's global STEM education initiative, which will bring inspiration and possibility to STEM education in ways that make it available to all.



---

## Two robots in one

---

The XRP can be used for two different applications:

- A **STEM learning platform using Python or Blockly** with custom tools designed to learn and experiment with robotics. Included is a curriculum to help learn about robotics and programming. This use of the XRP is described in this document.
- A robot to introduce new **FRC teams and team members to WPILib programming** with the same tools, languages, and libraries used in developing competition robots. To learn about using the XRP for the *FIRST* Robotics Competition, you should refer to the [standard WPILib documentation](#).



There are several software tools available to the programmer for the XRP. Some are available, especially for the XRP and other general-purpose tools that may also work with the XRP.

### 2.1 Programming Languages

The XRP team supports two programming languages for the XRP:

#### **Blockly**

A graphical programming system based on Scratch to make it easier to start coding your robot without the need to the syntax of Python. Internally, a Blockly program is translated to Python and saved on the robot. Users can even see the generated Python code to help them learn to use the language themselves.

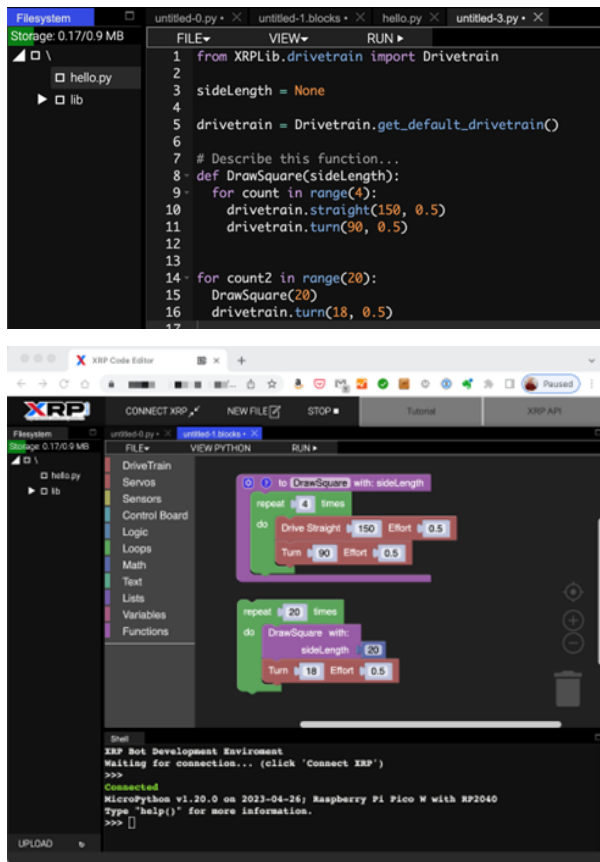
#### **Python**

An object-oriented text-based programming language used throughout industry and taught in many classrooms.

Other languages include C and C++. There may be other languages that can also work with the RP2040 microprocessor in the XRP.

### 2.2 XRPCode

The recommended programming tool for the XRP is [XRPCode](#). It is a web-based single tool designed specifically for the XRP to support programming in either Python or Blockly. It also can check and update firmware and library versions on the robot as new software releases become available. XRPCode is a web-based programming system that operates inside the Google Chrome or Microsoft Edge browsers, so users will always be running the most recent version of the tool.



## 2.3 XRPLib

**XRPLib** is a Python-based programming library that provides classes and functions to make it easy to use all the features of the XRP Robot. XRPLib is completely open source, so users can download the software to see how it works. We also encourage community involvement through pull requests to the library. However, we recommend contacting us before spending too much time to ensure that your ideas are compatible with our plans and development for XRPLib.

Here are some primary features of the XRP:

- The default drive function to control speed, direction, and power applied to the two motors. It can handle driving and turning, with and without sensors such as the IMU, for making accurate point turns.
- The sensors on the robot, such as the motor encoders, rangefinder, reflectance sensor, and IMU (Inertial Measurement Unit), which can get the robot heading and accelerations as it is driving.
- The WiFi connection so that programs can create a web server on the robot that can be used to display a dashboard on a connected phone, tablet, or computer. It is designed for displaying program status, driving controls for teleoperation, and buttons to run user functions when pressed for more control of user robot programs.
- Utility functions for sensing the user buttons, operating the LED, and robot program timing
- Several small sample programs to help illustrate how the various components are used to operate.



## 2.4 Other tools and languages

In addition to the supplied languages for the robot, users can program the robot using other standard tools such as C, C++ and WPILib using various IDEs like the Arduino IDE and Visual Studio Code. VS Code has several plugins specially designed to support Python programming and the Raspberry Pi Pico, which is the hardware that powers the XRP.

## 2.5 Where to find the tools

XRPCODE IDE: <https://xrpcode.wpi.edu/>

XRP API Documentation: [https://open-stem.github.io/XRP\\_MicroPython/index.html](https://open-stem.github.io/XRP_MicroPython/index.html)

XRP Curriculum: <https://introtoroboticsv2.readthedocs.io/en/latest/>

## 2.6 Getting help

We have set up a [Discourse server](#) where you can get help from our team as well as members of the community using XRP robots.



---

### Building the XRP Robot

---

Assembling the XRP robot is easy, but be sure to follow the steps here to be sure that the wiring is correct and all the pieces are added correctly to the chassis.

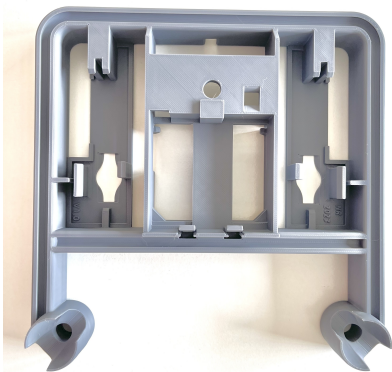
Below is a video provided by SparkFun Electronics showing how to assemble the robot followed by a step by step set of written instructions below.

<https://youtu.be/JQyKhzlMSms>

#### 3.1 The XRP kit (0:37)

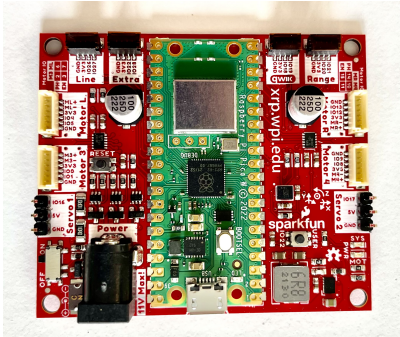
The XRP kit contains all the parts you need to assemble and use your robot. You only need to supply 4 AA Batteries (preferably rechargeable) and a micro USB cable to connect your computer to the robot. The contents of the kit are shown to help you identify the parts during assembly.

##### Robot chassis



The chassis is a single-piece design that holds all of the robot components. It is designed with a rail system that is designed to make adding additional components easy and without the need for tools. All the robot parts simply snap onto the chassis to make assembly as simple as possible. You can also 3D print your own parts to attach to the chassis.

### **Robot controller**



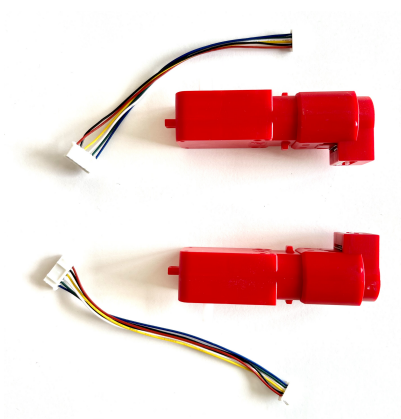
The robot controller has the RP2040 microprocessor that reads the sensors inputs, runs the Python or Blockly program and drives the actuators (motors). It also has additional components to sense accelerations and headings of the robot, and communicate over WiFi with your laptop or phone.

### **Electronics parts**



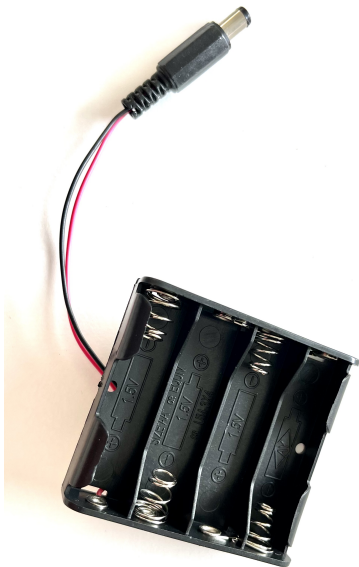
The components in the bag of electronics parts will each be shown individually below.

### **Motors and cables**



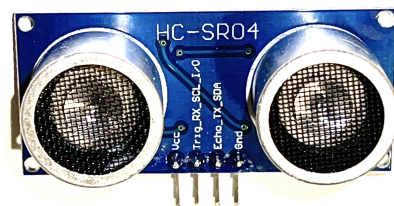
The motors are used to drive the robot and are attached to motor controller through the associated cables.

### Battery case



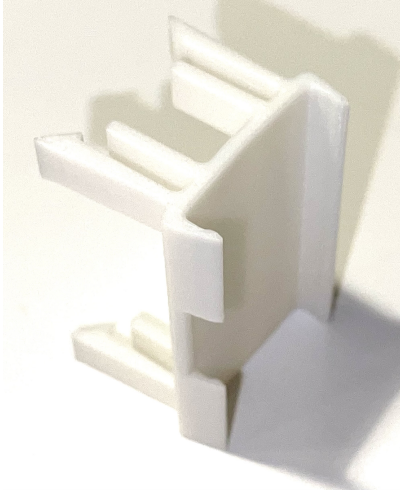
The battery case holds 4 AA batteries. You can use any standard alkaline cells but rechargeable cells are preferred so that you don't have to keep replacing them as they run out of energy.

### Ultrasonic rangefinder

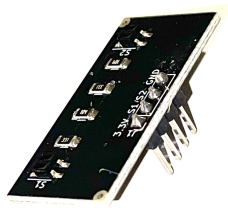


The ultrasonic wire has two power wires labeled Vcc (red wire) and Gnd (black wire). It also has two additional connections that operate the sensor and get range data. These are trig (blue wire) and echo (yellow wire). A common mistake when wiring this sensor is to get these two wired incorrectly.

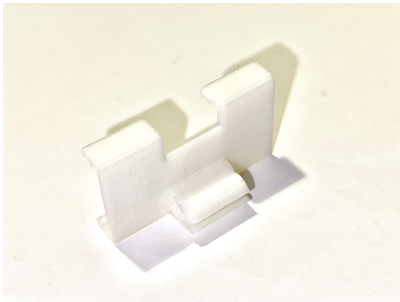
### Rangefinder bracket



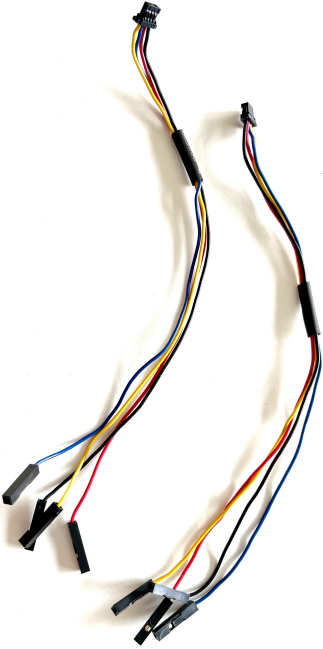
**Reflectance sensor**



**Reflectance sensor bracket**



**Sensor cables**



These cables connect the rangefinder and line following sensors to the robot controller. **When installing these on the sensor end, you must be careful to install the wires correctly, so be sure to carefully read the instructions when attaching them.** Miswiring is the motors is the most common cause of problems when assembling the XRP robot.

#### Tires (o-rings)



These o-rings are used to form tires to slip over the plastic wheels to give the robot more traction, especially on smooth surfaces.

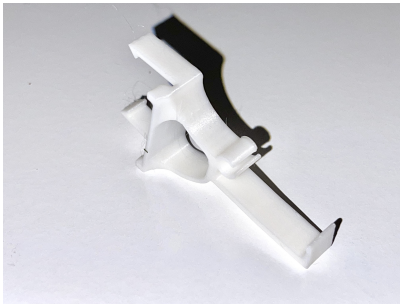
#### Servo motor



**Servo arm**



**Servo bracket**



The servo is a special type of motor such that when programmed with a position the shaft will automatically move to the specified angle. This is used to power the arm on your robot it can move to predetermined angles all by itself.

**Casters**



The casters simply provide a low friction contact point for the front of the robot to allow the two rear drive wheels to easily steer the robot forwards, backwards, or any angle.

## 3.2 Assembling the XRP Robot

Assembling the XRP robot can be done without the use of tools with the optional exception of screwing the servo arm to the servo. The total process should take about 15 minutes, especially once you understand how it goes together.

Each of the following sections has a time reference for the video at the top of this page so you can see how to assemble that part. We suggest that you view the entire video before starting the assembly so you can get a good overview of how it goes together.

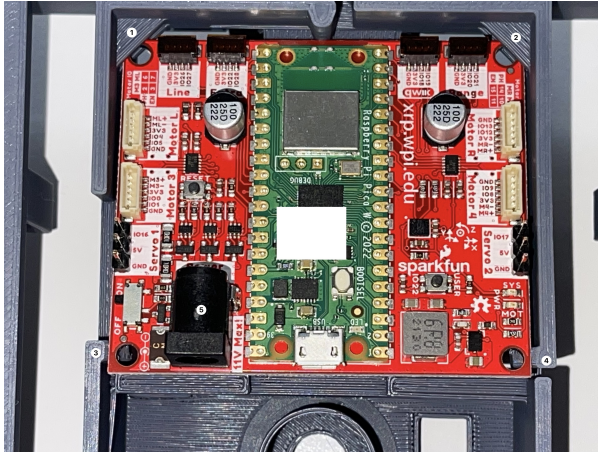
### 3.2.1 Inserting the robot controller into the chassis (1:18)

---

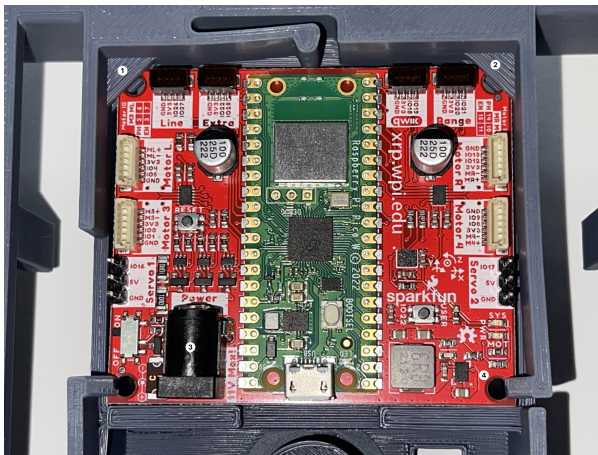
**Nota:** If you look at the connectors on the edge of the controller board labeled «Line», «extra», «qwiik», and «range» have very small pieces of tape covering the openings. Remove the tape from all four connectors before inserting the board in the chassis.

---

Insert the robot controller circuit board into the chassis as shown in the following picture. Observe the orientation of the board where the battery connector (5) istowards the back of the robot as shown. Also the top corners of the board are inserted part way into the corner pockets as shown at (1) and (2). The clips in the chassis (3) and (4) are designed to hold the chassis in place when it is pushed in.



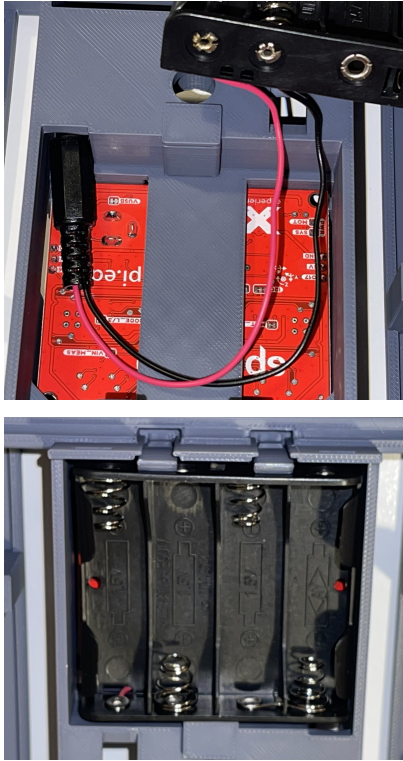
Then push down and forward on the back edges of the board so that the front corners are completely seated in the pocket as shown at (1) and (2) and the board snaps down as shown at (3) and (4) in the following photograph. It might be helpful to view this part of the assembly in the video from the top of this page.



### 3.2.2 Installing the battery pack (1:39)

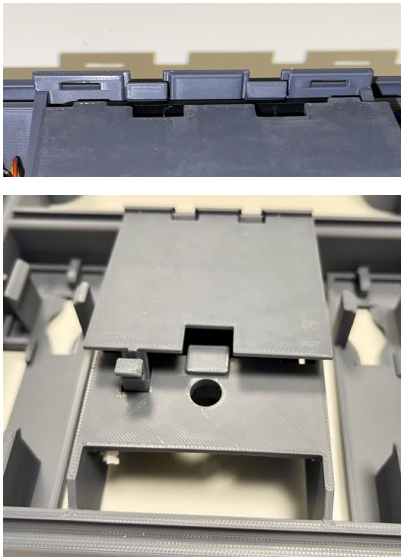
The battery pack is installed by:

1. Inserting the cable through the cutout in the battery pack area in the chassis.
2. Pushing the edge of the battery pack against the fingers in the chassis which hold it in place.
3. Push the battery pack in place into the robot chassis so that it is full seated.



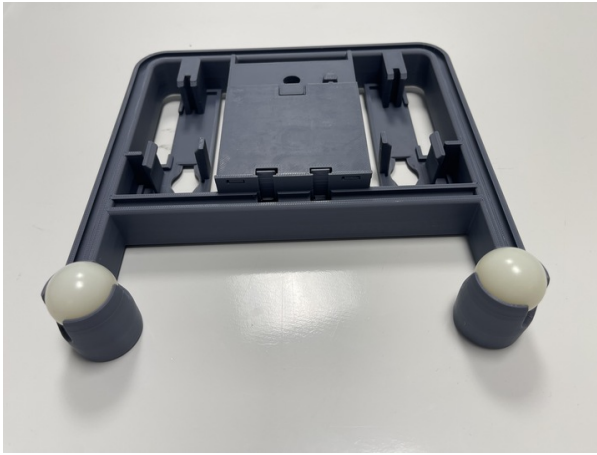
### 3.2.3 Adding the battery cover (2:29)

The battery cover is very easy to install, just line up the two tabs on the battery cover with the two slots in the chassis just outside of the battery case. Then the clip snaps into place as you push the battery cover into place.



### 3.2.4 Inserting the casters into the chassis (3:06)

Install the white front casters (balls) into the chassis by pushing them into place. Once they are installed, the casters should rotate freely.

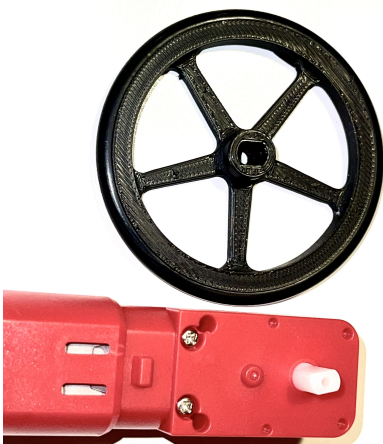


### 3.2.5 Adding the motors

The red hobby motors supplied with the kit include encoders (sensors to measure wheel rotation) to make it easy to program the robot to drive for specific distances and speeds. This will give your robots more control and accuracy as you are writing programs to operate it.

### 3.2.6 Putting the wheels onto the motors (3:22)

The wheels press fit onto the white motor shafts. Notice that the motor shafts have two flat sides that correspond to the flat edges in the center of the wheel. The wheel is pressed over the motor shaft so that the center part of the wheel that sticks out is closest to the motor body and that the wheel is pressed all the way onto the motor shaft.





### 3.2.7 Putting the tires onto the wheels (3:45)

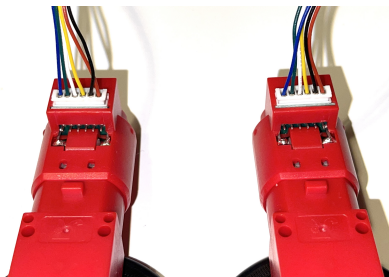
The tires are rubber o-rings that slip into the groove on the outside rim of the wheel. Simply stretch the o-ring to get it to move into place. These will provide friction when the robot is driving, especially on smooth surfaces.



### 3.2.8 Connecting the motor cables to the motors (3:52)

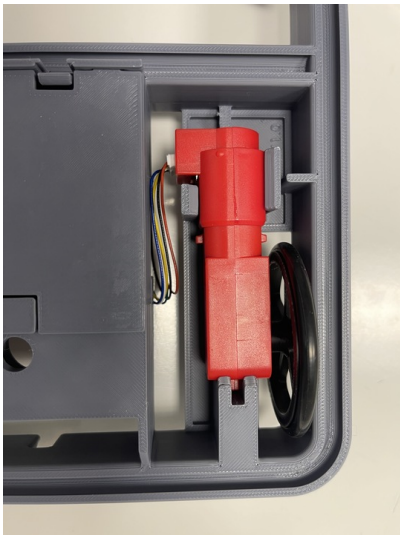
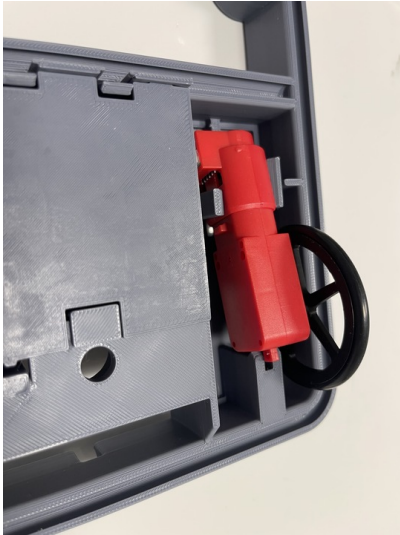
The motor cables connect the motor to the robot controller so that it can drive the drive the motors and receive data from the motor encoder sensors that provide position and speed information for your robot program. This encoders all the robot to drive at a desired speed and drive for a desired distance.

The wider connector on the cable is inserted into the motor. Notice that pins (wires) on the motor connector are closer to one side than the other. Similarly, the holes on the connector attached to the cable are closer to one side.

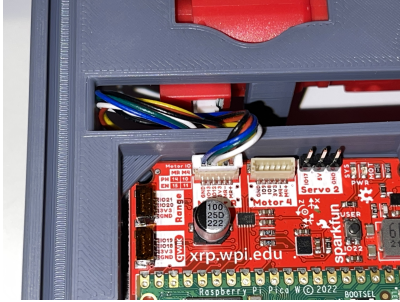


### 3.2.9 Installing the motors into the chassis (4:09)

The motors snap into the chassis from the bottom once the wheels and cables are installed. The motor is oriented so that the wheel goes through the slot on the chassis as shown in the picture. Ideally you should push the wires from the motor through the opening in the chassis to the top of the chassis so they can be attached to the robot controller. Then seat the end of the motor opposite the cable end, then snap the wheel side of the motor into place. Repeat for both motors.







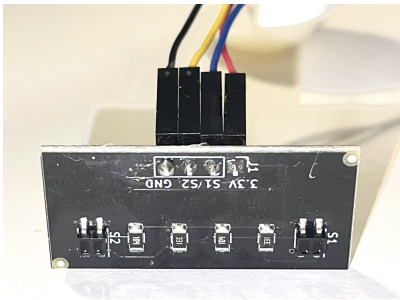
### 3.2.12 Adding the Sensors

The line following sensor can detect lines on the driving surface that have a different reflectivity. These are typically used in robot applications to follow lines or locating interesting places on a board or mat. It has two pairs of LEDs and photo sensors to emit infrared light and measure the reflected brightness.

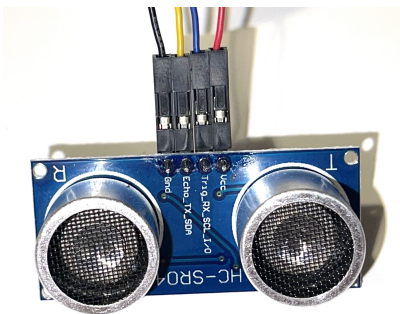
The ultrasonic rangefinder uses sound to measure the distance to objects in front of the sensor. An ultrasonic (inaudible high frequency) short sound is sent from one of the transducers which is reflected back by nearby objects and received by the second transducer. The time of the sound round-trip is measured to determine distance to nearby objects.

### 3.2.13 Wiring the sensors (5:11)

The sensor cable is connected to the line following (reflectance) sensor as shown in the picture below. Be sure to observe the order and color of the wires connecting to the sensor. The connectors simply push over the sensor pins. Be sure that they are fully seated as shown in the picture and video to ensure a good connection.

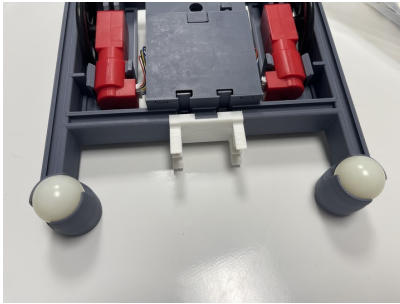


The rangefinder is wired by attaching the four wires from the sensor cable to the pins on the rangefinder as shown in the picture below. Be sure to connect the wires to the pins in the right order.

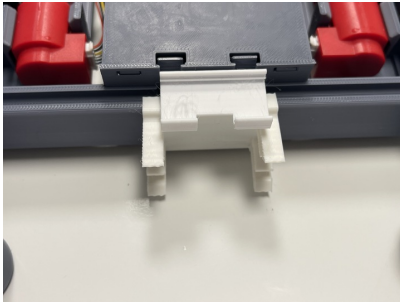


### 3.2.14 Attaching the brackets to the chassis (5:44)

The rangefinder bracket is attached to the front of the chassis just above the reflectance sensor as shown in the picture below.

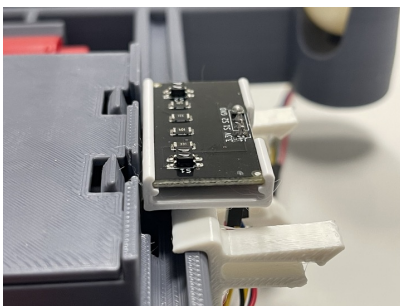
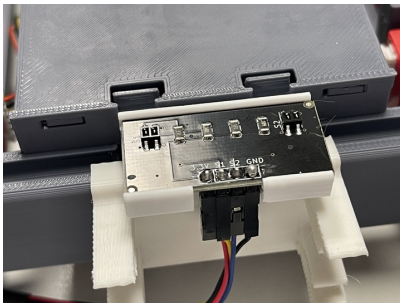


The reflectance sensor bracket is installed on the chassis as shown in the picture below. The ball end of the bracket is inserted into the slot in the front rail.



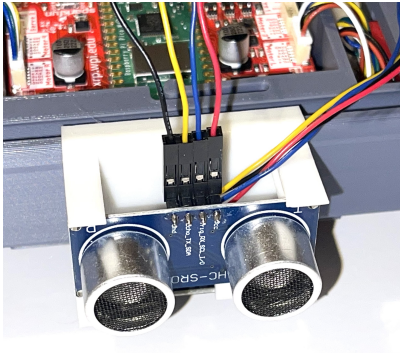
### 3.2.15 Inserting the line follower into the bracket (6:19)

The reflectance sensor is inserted into the bracket as shown in the picture below. Also look at the side view of the assembly to see how the sensor is correctly positioned in the bracket.



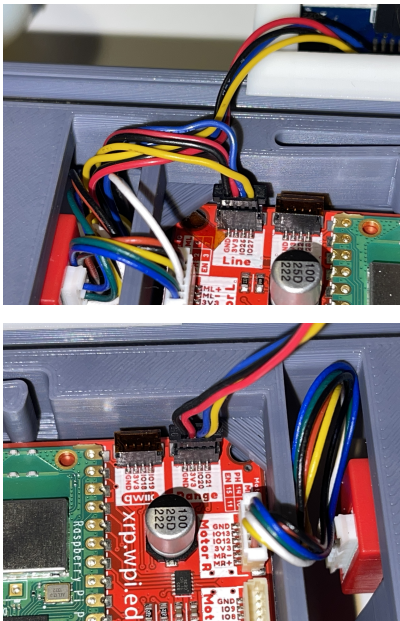
### 3.2.16 Attaching the rangefinder to the bracket (6:38)

Attach the rangefinder to the bracket as shown in the picture below.



### 3.2.17 Connecting the cables for the line follower and rangefinder (6:55)

The cables from the reflectance sensor (line follower) and the rangefinder are connected to the connectors on the controller board. Notice that there are labels on the board for each of these cables to help you get them into the right connectors. The line follower cable goes into the connector labeled Line and the rangefinder goes into the connector labeled Range. It is a good idea to put a small loop in the wire that can be tucked into the chassis before connecting it to help keep the wiring neat and less likely to get snagged.



### 3.2.18 Attaching the servo

The servo is used to rotate the arm to the desired position. It has the advantage over a normal motor in that it has sensors inside of it to allow it to move to a desired position that you can program.

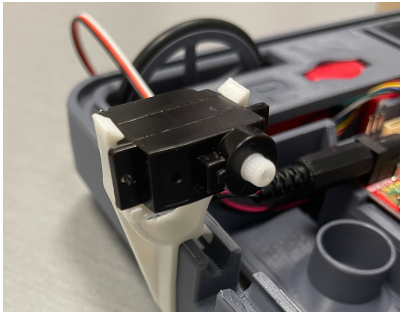
### 3.2.19 Attaching the servo bracket to the robot chassis (7:29)

The servo is attached to the robot by first inserting the ball end of the bracket into the upper slot on the back rail, then snapping the bottom part of the bracket over the bottom part of the rail.



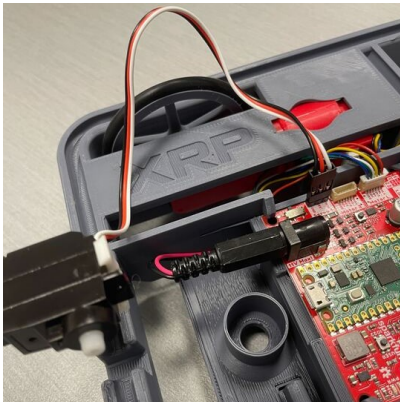
### 3.2.20 Mounting the servo to the servo bracket (7:54)

The servo snaps into the servo bracket as shown in the photo below.



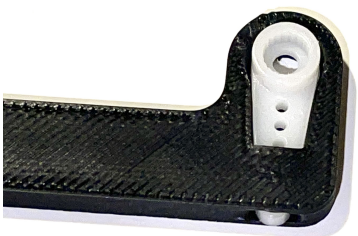
### 3.2.21 Connecting the servo cable to the robot controller (8:06)

The servo cable is connected to the slot labeled Servo 1 on the robot controller board as shown in the photo below. Be sure to connect it as shown with the black wire connecting to the Gnd terminal on the Robot Controller board.



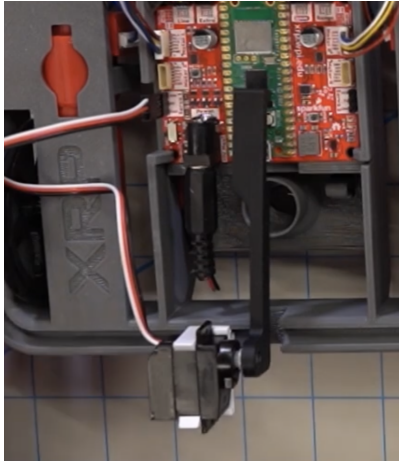
### 3.2.22 Inserting the servo horn into the robot arm (8:27)

The servo horn is the small white plastic arm that attaches to the servo by pressing onto the servo shaft. There are several servo horns that come with the servo accessories. The one that you should use has a hole for attaching to the servo shaft at one end, and a small arm at the other end. It gets installed into the slot at the end of the larger black servo arm as shown in the picture below and the video. **Be sure to install the servo arm so that it is oriented as shown in the photo, in particular make sure that the mounting flange is facing the correct direction.**



### 3.2.23 Mounting the arm to the servo (8:45)

The servo arm simply presses onto the white shaft on the servo. The servo shaft only has about 180 degrees of rotation so it's important to install the arm so that it can move through its full range of motion while mounted on the robot. Holding the servo so that it's flat with the wires coming out to the left, the arm should be mounted so that it has 180 degrees of motion from front to back. That is the arm will never travel below the level of the servo body. You can see how this is done by looking at the video at the indicated time stamp. This image shows the servo at the end of its travel inside the robot chassis. The other end of the travel will be slightly below horizontal behind the robot.



## 3.3 Initializing and testing your XRP (10:21)

Refer to SparkFun's video at the top of this page to set up your XRP and ensure that it's working correctly!

Once your XRP is connected, skip to (12:44) in the video and follow along with the built-in test to ensure that the sensors and motors are working properly.

## 3.4 Troubleshooting the robot build

Generally the build of the robot is very straightforward, but from feedback we have compiled this section that describes some of the common issues we have seen as people are building the XRP.

### 3.4.1 Rangefinder or the line following sensors don't work in the Installation Verification Test

It is very easy to accidentally attach the rangefinder and line following sensor cables to the wrong connectors on the controller board. Be sure to verify that the rangefinder is in the connector marked «Range» and the line following sensor is in the connector marked «Line».

If the connectors are reversed and you have to remove them, **be sure to only remove the connector by pulling on the plastic shell**. Do not pull on the wires as you might accidentally pull them out of the connector.



---

## XRPCode Integrated Development Environment

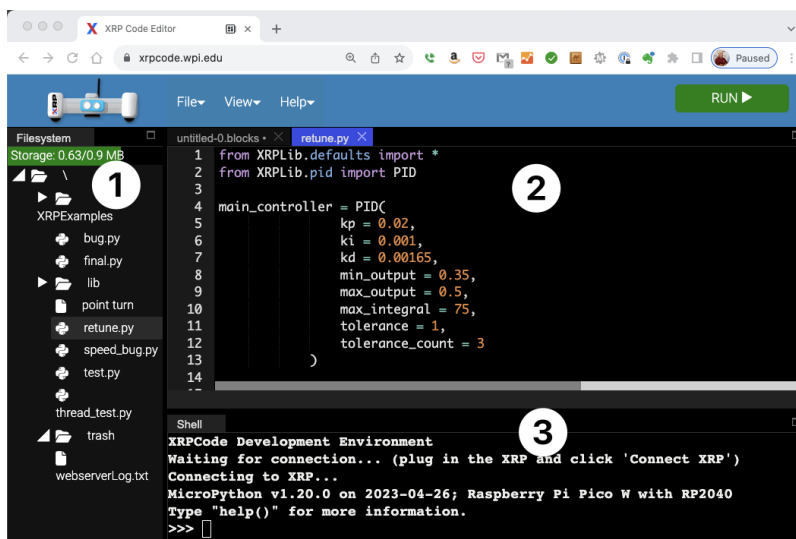
---

XRPCode is a web-based development tool that is run in either the Google Chrome or Microsoft Edge browsers. XRPCode can be used to create either Python or Blockly language programs that run natively on the XRP control board. Blockly programs are first translated to Python and then run. In fact, you can view or even translate the Blockly program into a Python program.

Run XRPCode by navigating to [XRPCode web site](https://xrplib.com/xrpcode).

If this is your first time accessing XRPCode or if there has been an update, a changelog will be displayed. Read through the changelog to learn what is new in the current release of XRPLib or the editor.

### 4.1 Exploring the XRPCode user interface



There are 3 major window areas for XRPCode.

On the left (1) is the Filesystem window. This will show the files on your XRP whenever an XRP is connected.

In the middle (2) is the editor. This is where you will be working on your programs.

At the bottom (3) is the shell window. This is where print statement output will be displayed. You can also use the shell as a command line to write Python code as an interactive way to quickly test ideas.

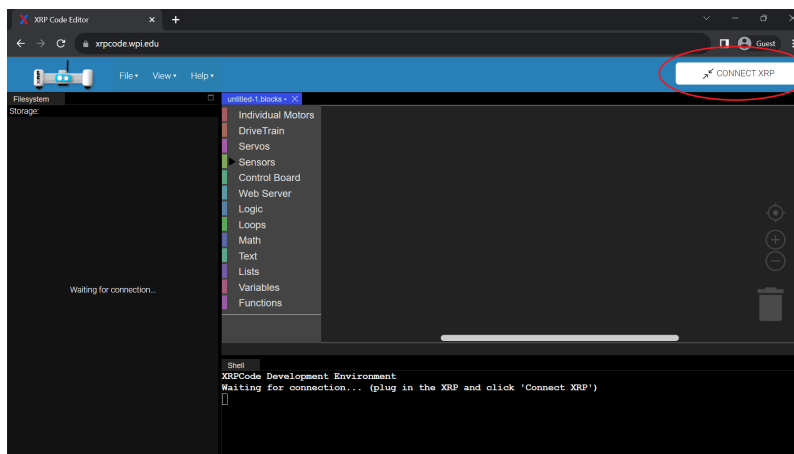
## 4.2 Connecting your XRP to XRPCode

The XRP robot has a micro USB connector on the controller board that is connected to your computers USB port with a cable.

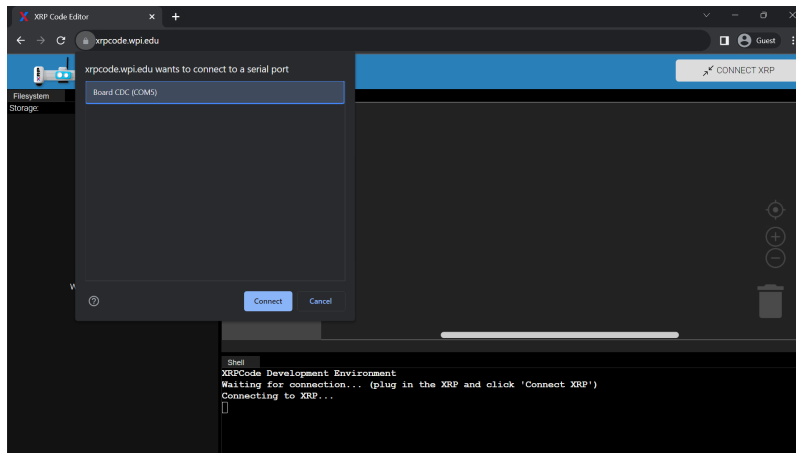
**Advertencia:** Many USB cables are for power only and do not transmit data. You will need a USB cable that can transfer data and power.

## 4.3 Connecting to the XRP

To establish the connection between the XRP robot and the computer, press the “CONNECT XRP” button. Often the connection will happen automatically when the XRP is plugged in and XRPCode is started.



This will bring up a dialog that lets you select the computer's serial port that XRPCode will connect to. In most cases, there will be just one serial device, but if there are more, select the one that is for your XRP robot. Click on the CONNECT button after selecting the desired serial port.

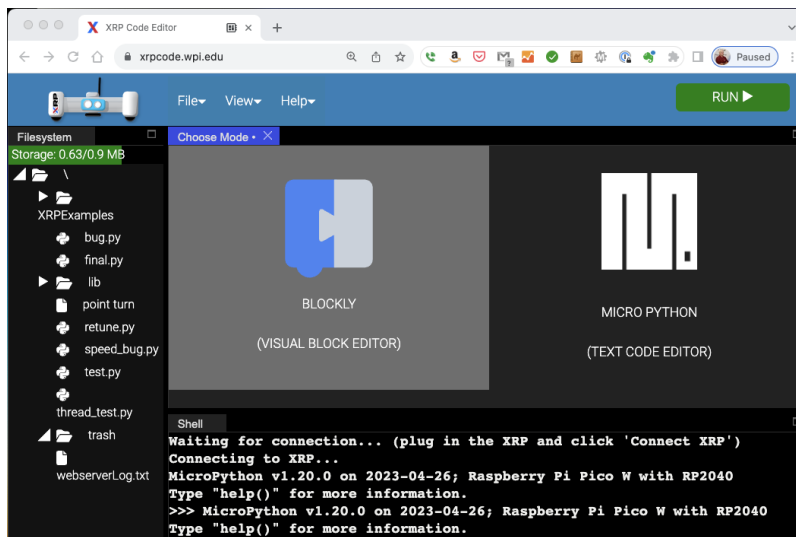


When the connection is made, the “CONNECT XRP” button will change to a green “RUN” button indicating that the connection has been made and a program can be run. The Filesystem window will show the files on the connected XRP.

If XRPCode cannot find your serial connection, or there are other connection issues please refer to the troubleshooting section at the bottom of this page.

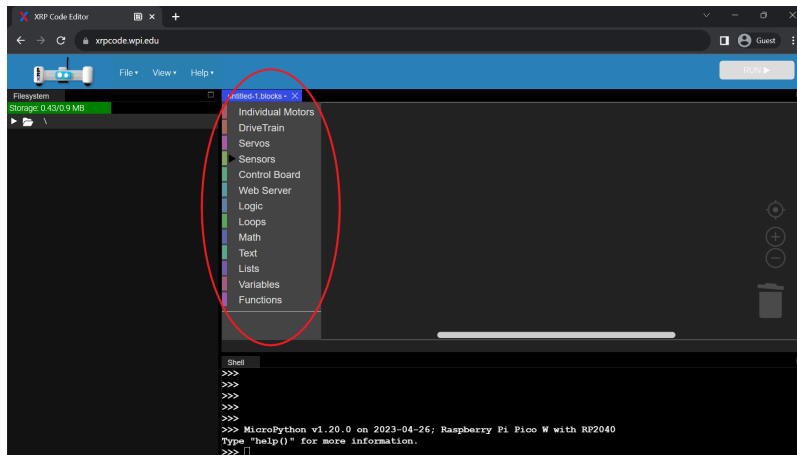
## 4.4 Using XRPCode

Now that the robot is connected, this is a good time to write a short program to learn about the editor.



In the editor area, there should be a window asking what type of editor to use, either BLOCKLY or MICROPYTHON. (If this is not showing, click on ‘New File’ under the ‘File’ Menu) and select BLOCKLY.

On the left of the editor is a palette of all the available blocks.

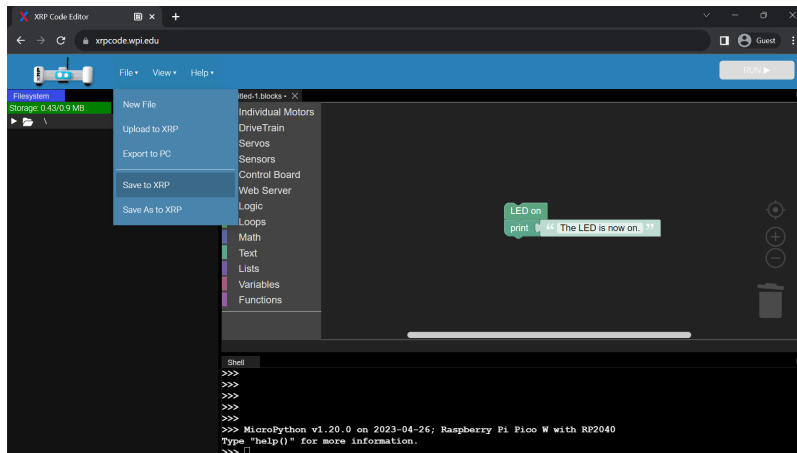


Notice that they are grouped by functionality. We recommend you click through each section to get a sense of where blocks are.

In this example, we'll create a program that will turn a controller board LED on and print a message in the XRPCode shell window at the bottom of the screen.

Click on the «Control Board» tab and then click on the **LED on** block. This will place this block onto your working canvas. You can move the block around and place it where you like. Now click on the «Text» tab. Then select the first block **print** "abc". This block is now also on your canvas. You can move this block around and place it right under the **LED on** block. You will notice that as you get close to the bottom of the **LED on** block it will show a yellow line indicating that the two fit together. When you let go of the **print** "abc" block it should click together with the **LED on** block. Feel free to change the "abc" to say what you want to print; ideally something useful to the program. For example, you might print something like "The LED is now on".

Save this new program to the XRP. Under the 'File' menu select 'Save to XRP'.



A dialog will be displayed for you to give this program a name. Change the *untitled* text to a name for your program such as 'first program' and click "OK". The program has now been saved to the XRP. You will see the name of your program in the Filesystem window on the left.

You can now press on the green "RUN" button, to run this program. Flip the power switch on your XRP to "on" and then click 'OK'. If your XRP is not turned on, a warning will pop up telling you to turn on the power switch.

**You will notice a few things:**

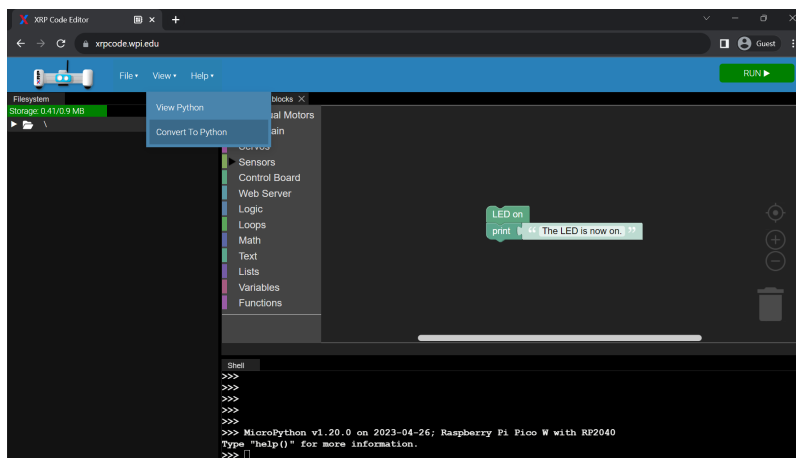
- The green LED next to where the USB cable connects to on the XRP will go on for a little while and then turn back off.
- The shell window at the bottom of XRPCode will print out your message from the print statement.

You may have also noticed that the green ‘RUN’ button changed to a red ‘STOP’ button while your program was running and then turn back after.

Clicking on the ‘STOP’ button will interrupt the program.

You might have noticed that the robot turns off when the program finishes or is interrupted. This is so that at the end of each run, the XRP is reset to a known state in preparation for the next time a program is started.

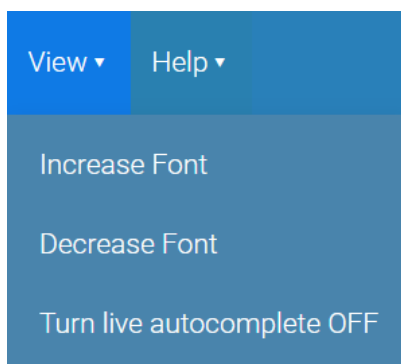
The ‘View’ menu contents change depending on the language you are using. For Blockly it will show View options for a Blockly program. Click on ‘View’ and then on ‘View Python’. This will bring up a view of the Python code that is generated from your Blockly file. Let’s actually convert this Blockly program program into a Python program. Click on ‘View’ and then on “Convert to Python”.



XRPCode will first give you a warning to make sure you want to convert the program as this cannot be undone. Click “OK”. It will do the operation and you should notice two things have happened:

1. There is a new ‘trash’ directory on your XRP.
2. Your program name now ends in .py instead of .blocks.

If you now go to the ‘View’ menu you will notice that the menu items have changed to be specific to Python.



Close the “View” menu and find the print statement in the program; it should be the last line. Change the message that is between the quotes. If you look at the file name tab at the top of the editor you will notice a white dot at the end of the name. That means that this file has been modified and has not yet been saved. Now if you click on the ‘RUN’ button it will save your program and run it again. The message in the shell window should be your new message.

You can close a file by clicking on the X next to the file name at the top of the editor. If you want to open the program again you can double click on the file name in the Filesystem window.

Congratulations, you have now learned how to create and run programs in XRPCode!

## **4.5 Troubleshooting XRPCode connection issues**

### **Cannot see the serial port when connecting**

- Be sure that the USB cable is a data cable and not just a power cable.
- Unplug the XRP from the computer and check the connection of the cable on the XRP side.
- Toggle the power switch on the XRP off. Confirm that the sys LED is on. This means it is properly receiving power from the USB cable. If the LED is not on, try a different cable.
- Make sure you are running either Google Chrome or Microsoft Edge browsers. At the time of writing, only those browsers support serial communication required for programming the XRP.

### **XRP was previously used for WPILib or some other purpose**

- In this case, XRPCode will try to load the current MicroPython firmware onto it; just follow the instructions.

### 5.1 Robot driving

The XRP is a mobile robot platform where driving from one place to another is central to the design of any program. The `differential_drive` class makes driving easy and has functions to:

- **Set the motor efforts**, which is the power or average voltage applied to the motors. The range of values can be set from -1 for full effort in reverse, to 0 for no voltage or stopped, to +1 for full effort in the forward direction.
- **Specify a speed to drive** in centimeters per second for each wheel. The robot will try to maintain the specified speed as best it can using the drive motor encoders.
- **Drive for a specified distance** using the drive motor encoders to sense how far the robot has traveled.
- **Make point turns** for a desired number of degrees, either clockwise or counterclockwise.

### 5.2 Effort vs. Speed

Throughout this document, we refer to the effort and speed of the drivetrain. Although they seem similar, they are distinguished as follows:

#### Effort

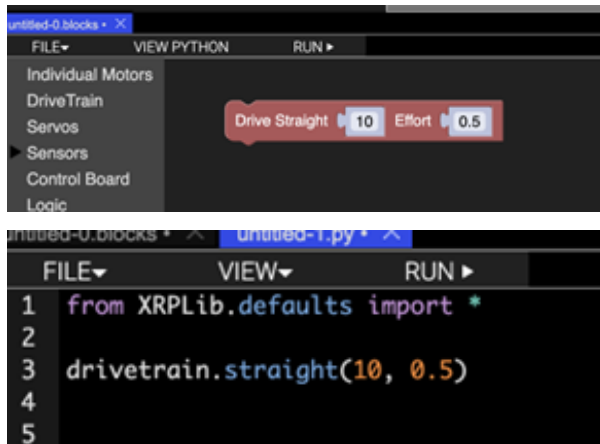
The effort reflects the amount of power (or average voltage) supplied to the motors. For a given effort, the speed will vary depending on things like the driving surface, the battery voltage, and the slope (either flat, uphill or downhill).

#### Speed

The speed is the actual number of centimeters per second that the robot will travel. When set in a program, the software will automatically adjust the motor effort within its capability to keep the robot moving at the desired speed.

## 5.3 Driving for a distance

The following program fragments show how to program the robot to drive forward for 10 centimeters with an effort of 0.5 or 50 percent power. This function uses the encoders to determine when the robot has traveled the requested 10cm. In addition, this function will ensure that the robot drives in a straight line by varying the speed of the left or right motors if one is slightly faster.



**Nota:** when requesting a distance to drive, the encoders are used to sense the number of degrees of wheel rotation to complete the operation. If the wheels slip while driving, the distance measurement will be incorrect.

## 5.4 Driving with an effort

This program will set the effort on the left and right drive motors to 50 percent, wait for 3 seconds, and stop the motors. The `set_effort()` function has parameters for the left and right drive motors to allow them to be set independently. No motor speed control is provided, so different driving surfaces, slopes, or battery voltage will affect the driving speed of the robot. The value of effort ranges from -1 for 100 percent backward to 0 for no effort or stopped to +1 for 100 percent effort forwards.

```
1 from XRPLib.differential_drive import DifferentialDrive
2 import time
3
4 differentialDrive = DifferentialDrive.get_default_differential_drive()
5
6
7 differentialDrive.set_effort(0.5, 0.5)
8 time.sleep(3)
9 differentialDrive.stop()
10
```



## 5.5 Driving at a speed

Set\_speed() attempts to maintain some linear speed in centimeters per second. The maximum speed is measured to be approximately 60cm/s tested on a flat surface.

This program will set the robot speed to 5 cm per second, in centimeters per second, of the left and right wheels separately. If both motors are set to the same speed, the robot will drive straight. If they are different, the robot will turn in a direction away from the faster wheel.

```
1 from XRPLib.defaults import *
2
3 drivetrain.set_speed(5, 5)
4 sleep(3)
5 drivetrain.stop()
6
```



## 5.6 Point turns

The robot can turn in place around a point directly centered between the two drive wheels. This is done by driving the left and right drive motors in opposite directions at the same speed. If the left wheel is spinning in the forward direction, the robot will rotate clockwise or to the right. If the right wheel is spinning in the forward direction, the robot will rotate counterclockwise or to the left.

The turn function specifies the number of degrees to turn, with a positive number indicating a counterclockwise turn, and a negative number indicating a clockwise turn. The second parameter specifies effort from -1 to 1.



```
1 from XRPLib.defaults import *
2
3 def test_turns():
4     drivetrain.turn(45, 0.5)
5     time.sleep(1)
6     drivetrain.turn(-45, 0.75)
7     time.sleep(1)
```

When you use the turn function, the IMU (Inertial Measurement Unit) gyro sensor on the robot will determine when the robot has completed the requested turn. This means the turn will continue until complete and is not affected by wheel slip.

---

**Nota:** If you were to pick up the robot while it is doing a turn, the wheels will continue turning until the gyro senses that the robot has turned the desired number of degrees.

---

## 5.7 Swing turns

This type of turn is where one wheel moves forward, and the other is stationary. The robot will pivot on the stationary wheel, making it the center of rotation. The circle's diameter traveled by the moving wheel will be twice the wheel track (the distance between the two wheels).

## 5.8 Smooth turns

Smooth turns are where the two wheels move in the same direction so that the robot drives in an arc, eventually completing a full circle. The circle's radius depends on the speed difference between the two wheels. The larger the difference, the smaller the circle diameter.

### 6.1 Motor classes

The XRP has two drive motors connected to the ports Motor L and Motor R on the robot controller board. The board also supports two additional motors labeled Motor 3 and Motor 4. These motors can be used to create additional mechanisms for the XRP.

There are four classes related to motors:

#### **Motor**

The motor class handles a single motor with a single method for setting the effort between -1 and 1.

#### **Encoder**

The encoder class is responsible for measuring the current position of a motor. This is useful to derive the speed of the motor, or the distance traveled by the motor.

#### **EncodedMotor**

Encoded motors contain a motor and an encoder, and has higher level logic for functionality that incorporate both objects. This class supports features like resetting and getting the motor position, setting the effort and speed of the motor, and configuring what controller is used for closed-loop speed control.

#### **MotorGroup**

It is often desirable to treat several motors as if they were one. For example, in a four wheel drive robot, the left side motors usually get the same settings when driving the robot. A `motor_group` is created with multiple motors, and functionality like setting effort can be applied to all the motors in the motor group.

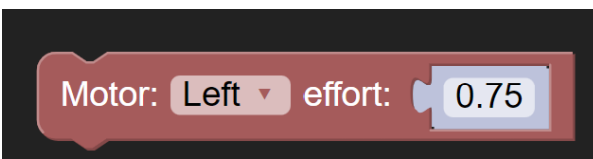
Since the XRP bot is built with an encoder on each motor, it usually is not necessary to directly deal with Motor or Encoder objects. Instead, use EncodedMotor or MotorGroup for higher level functionality.

## 6.2 Using EncodedMotor

Interacting with EncodedMotor objects is often the most convenient way to control motors on the XRP. The XRPLib.defaults module provides two ready-made EncodedMotor objects, left\_motor and right\_motor, which allow for fully independent control over the drive motors of the robot.

There are four motor controllers total on the XRP, numbered 1-4. 1 and 2 are the left and right motors, and 3 and 4 are labeled on the robot controller board. left\_motor and right\_motor objects are provided by default in the XRPLib.defaults module. Let's take a look at how we can use the left\_motor object to set the left motor to an effort of 0.75.

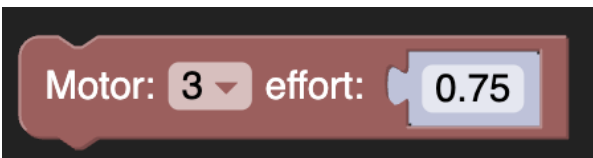
```
from XRPLib.defaults import *  
left_motor.set_effort(0.75)
```



Alternatively, you may also construct your own EncodedMotor objects, which is needed to control motors 3 and 4. The following code sets Motor 3 to an effort of 0.75.

```
from XRPLib.encoded_motor import EncodedMotor  
motor3 = EncodedMotor.get_default_encoded_motor(3)  
motor3.set_effort(0.75)
```

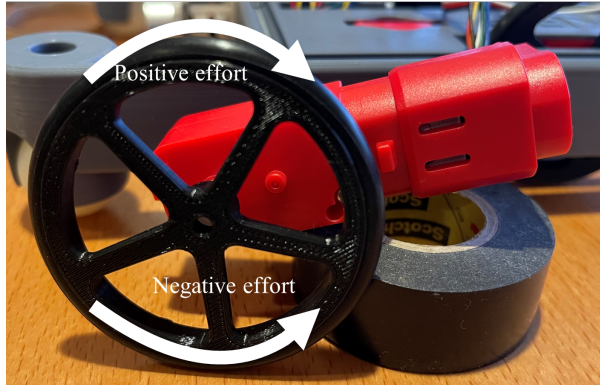
In Blockly, constructing motor objects is not necessary. Under the «Individual Motors» category, each block takes in a parameter to specify which motor to use. This is all that is needed to set Motor 3 to an effort of 0.75 in Blockly:



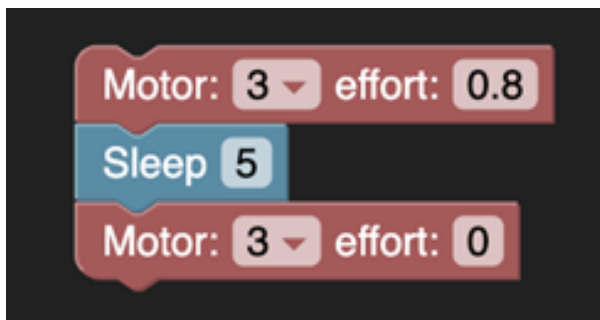
## 6.3 Methods for EncodedMotor objects

### set\_effort(effort\_value)

As seen earlier, this method spins the motor at some raw effort. The effort value ranges from -1 to +1, where negative efforts spin the motor in reverse, and positive efforts spin the motor forwards. An effort of 0 stops the motor.



The programs shown below set Motor 3 to 80 percent effort for 5 seconds, then afterwards, back to 0 percent effort to stop the motor. This example uses motor 3, but any motor can be used in its place.



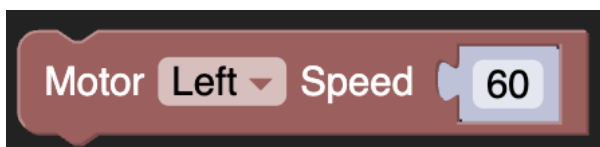
```
1 from XRPLib.encoded_motor import EncodedMotor
2 import time
3
4 motor3 = EncodedMotor.get_default_encoded_motor(3)
5
6
7 motor3.set_effort(0.8)
8 time.sleep(5)
9 motor3.set_effort(0)
```

### set\_speed(speed\_rpm)

Unlike the Drivetrain object which uses cm/second, the motor objects handle speed in rotations per minute. They read from the encoder to determine the current speed, and adjust based on a closed-loop controller, which by default is a PID controller. Similarly to set\_effort(), the sign of the speed determines the direction of the motor.

The example programs below set a speed of 60 rpm for the left motor:

```
from XRPLib.encoded_motor import EncodedMotor
motor1 = EncodedMotor.get_default_encoded_motor(1)
motor1.set_speed(60)
```

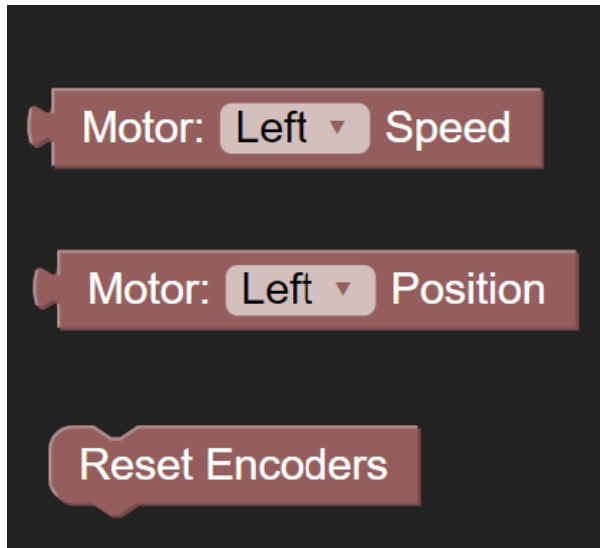


### set\_speed\_controller(controller)

The set\_speed() function relies on a controller to determine how to vary the effort of the motor to maintain the specified speed. By default, the controller is a PID controller, but it can be changed to any object that implements the Controller abstract class.

The example below sets the speed controller with custom PID tunings. For more information on controllers, refer to the page under Miscellaneous Topics. Currently, there is no support for custom controllers in Blockly.

```
from XRPLib.encoded_motor import EncodedMotor
from XRPLib.pid import PID
motor1 = EncodedMotor.get_default_encoded_motor(1)
motor1.set_speed_controller(PID(kp=2, ki=0, kd=0.1))
motor1.set_speed(60)
```



**get\_speed() -> float**

Returns the current speed of the motor in rotations per minute, as measured by the encoder.

**get\_position() -> float**

Returns the current position of the motor in rotations, as measured by the encoder.

**get\_count() -> integer**

Returns the current position of the motor in encoder counts, as measured by the encoder.

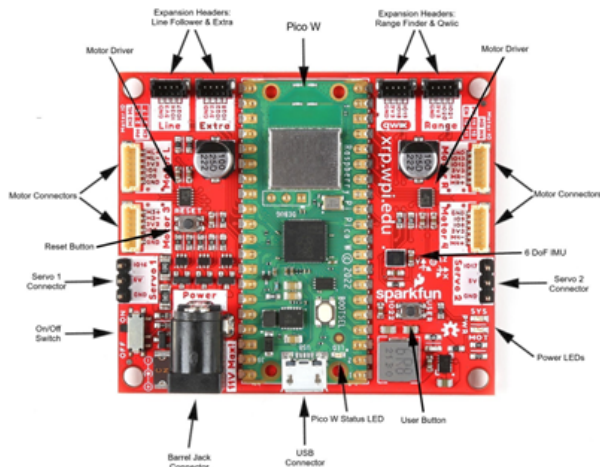
**reset\_encoder\_position()**

Resets the encoder counts to 0. `get_position()` and `get_count` return the difference in distance since the last reset.

---

## Sensing the environment

---



### 7.1 Measuring the distance to an object

The XRP includes an ultrasonic rangefinder that can measure the distance to objects in front of it. The sensor has two transducers; one acts as a speaker, and the other acts as a microphone. It does it by sending a burst of ultrasonic sound out of the speaker that hits an object in front of the robot. The sound reflects off the object back to the sensor and is captured by the microphone. The time for that round trip determines the distance to the object. Understanding how well the sound reflects off various objects of different sizes, profiles, and materials is important for using the sensor. A good exercise is to test the sensor by printing returned values at various distances from any object you want the robot to detect.



---

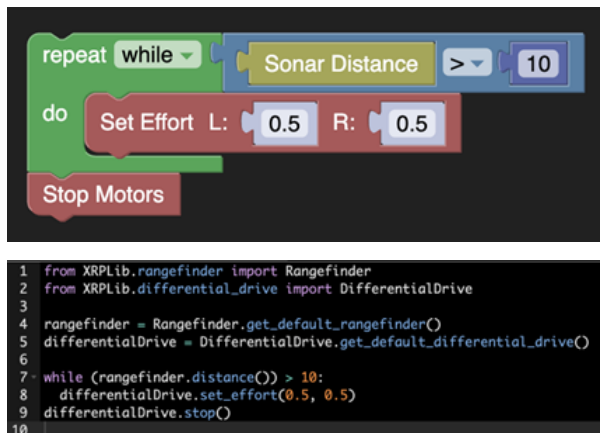
**Nota:** It is important to wire the sensor correctly, as described in the assembly instructions, to ensure it works properly. Interchanging the trigger and echo wires is a common error using that part.

---

XRPLib has a rangefinder class that takes care of the sending and receiving signals to the sensor. All the program has to do is request the distance, and the library returns it. There is a single method called `distance()` that returns the distance to the nearest object in centimeters. The range of operation is from 2cm to 4m.

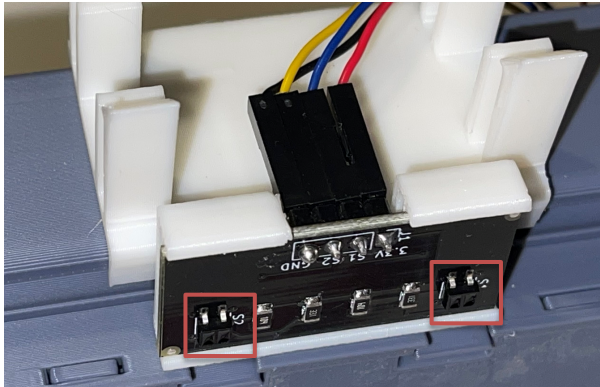
## 7.2 Example use of the rangefinder

The following program drives the XRP forwards until the code detects an object within 10cm of the ultrasonic rangefinder. Then it stops.

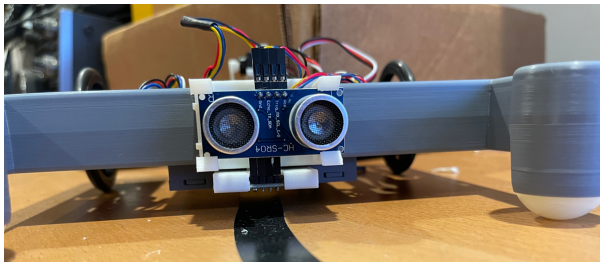


This program stops the motors when the object is detected. A better way of solving the same problem might be to use proportional or PID control to gradually bring the robot to a stop to avoid overshoot, where inertia might carry the robot beyond the 10cm set point before it comes to rest.

## 7.3 Following lines



A photo showing the two LEDs and photo sensor pairs



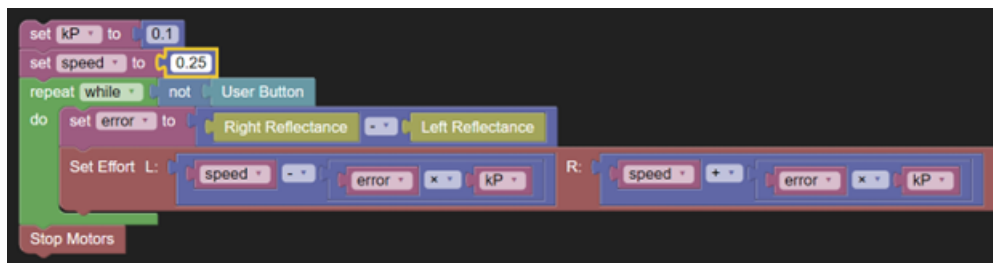
Robot following a line taped on a table

A reflectance sensor that can be used for line following is included with the XRP. It has two pairs of LEDs and light sensors. The LEDs emit infrared light that reflects off the driving surface. The light sensor measures the reflected light intensity, which depends on the surface below the sensor. Electrical tape is typically used to make a line that the robot can follow and has a different reflectivity than the surface, usually a whiteboard or tabletop. With a pair of sensors, the robot can read the reflectance value and tell where it is relative to the taped line.

The class `reflectance` has methods `get_right()` to retrieve the right reflectance value and `get_left()` to retrieve the left reflectance value. The reflectance ranges from 0 (white) to 1 (black).

## 7.4 Line following example program

The following program uses proportional control with the line sensors to follow a line across the driving surface for the robot. The `Kp` variable sets the gain for the controller.



```
from XRPLib.board import Board
from XRPLib.reflectance import Reflectance
from XRPLib.differential_drive import DifferentialDrive

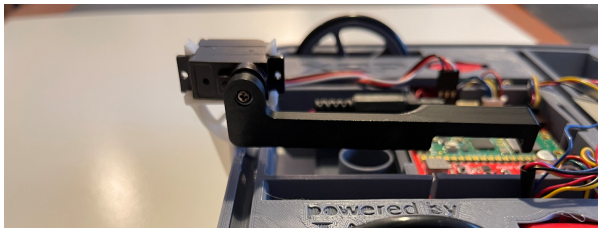
board = Board.get_default_board()
reflectance = Reflectance.get_default_reflectance()
differentialDrive = DifferentialDrive.get_default_differential_drive()

kP = 0.1
speed = 0.25
while not board.is_button_pressed():
    error = reflectance.get_right() - (reflectance.get_left()
    differentialDrive.set_effort(speed - error * kP, speed + error * kP)

differentialDrive.stop()
```

#### 8.1 Setting up the arm on the servo

The servo motor used to attach the arm on the XRP has about 200 degrees of rotation and can move to a desired position using internal sensors. When attaching the arm to the servo, it is important that one end of its range is with the arm relatively horizontal inside the robot chassis and the other end of the rotation outside the back of the robot for picking up objects. The full range of arm rotation is shown in the two images below.



Arm at one end of servo rotation inside the chassis for storage



Arm at the other end of rotation positioned behind chassis to pick up objects

To position the arm correctly, install it in any position and use it to rotate the servo to the full clockwise direction, as seen in the top photo. Then reinstall the arm so that it is in the shown position. Then the software will be able to move it to any position in between the two photos above.

## 8.2 Moving the arm under program control

Use the servo class to move the servo motor to the desired position. The method `set_angle()` sets the servo position to the desired angle. Below is an example program that moves the servo position from one end of its motion to the other.



When the servo is controlled from the program, it is held in the position it was last set to. To free it, that is to allow it to be moved by hand, the `free()` function may be called, and the program will stop sending the position signal to the servo.

---

## Creating a dashboard

---

A program can display a web page that can act as a dashboard or control panel for your robot. The web page can be used for:

1. Program debugging by wirelessly displaying values such as errors, messages about what the program is doing, sensor readings, and other status as the robot is driving. This allows the XRP to output important values without needing to be plugged in
2. Binding functions to web page buttons so that when pressed, the functions are executed by the web server. Using this technique, your device can act as a remote controller for your XRP. This is also a great technique to test functions as you write them by executing them from the web server.
3. To create more graphically pleasing buttons for controlling your robot, there are predefined buttons for forward, backward, left, right and stop operation. These will display on the web page like a game controller dashboard.

Before starting the web server, you should define all the displayed values, user buttons, and the driver panel.

First, one (but not both) of the following two methods must be called to initialize the network:

```
start_network(robot_id: int)
```

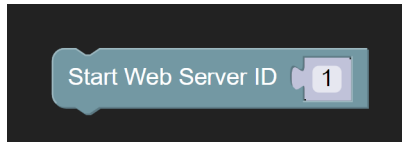
This opens an access point from the XRP board to be used as a captive host. The network password is "remote.xrp"

```
connect_to_network(ssid: str, password: str, timeout=10)
```

This will connect to a wifi network with the given ssid and password. If the connection fails, the board will disconnect from the network and return.

Then, XRPLib will create and start a web server using the robot built-in WiFi connectivity when the Webserver class method is called:

```
webserver.start_server(id)
```



Note that this will only succeed if one (not both) of the above two methods is called first. This function call will:

1. Bring up the WiFi interface on the robot using a robot number parameter to create a unique SSID. When a device (computer, phone, or tablet) connects to that SSID from the network settings, it will then be on the robot's local network and no longer on the internet.
2. Start listening on port 80 (HTTP) for a connection from a browser.
3. Dynamically create a web page based on user programming that can have driving controls, buttons corresponding to user functions, and display of program data for debugging or robot monitoring.

The user can then connect to the robot web server using a phone, tablet, or computer to see the web page the robot program creates.

---

**Nota:** If you connect to the robot to the computer you are using to programming it, you might lose connectivity with the internet, so that XRPCode stops working.

---

The user should enter the URL: ROBOT\_URL.

## 9.1 Using the web server

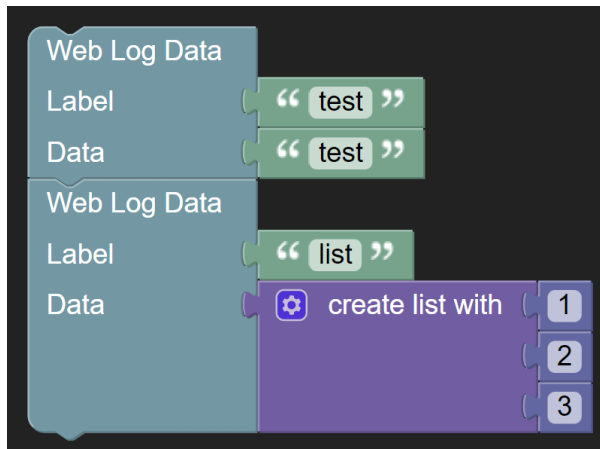
The `start_server()` function will never return. Once your program calls `start_server()`, the only way of executing code will be through the generated program. The program will then be event-driven, that is, only responding to pushbutton events from the web server.

## 9.2 Display running program data

A program can log any expression to the web server as a value by supplying a text label for the value and the value itself.

Some examples are:

```
webserver.log_data("test", "test")
webserver.log_data("List", [1,2,3])
webserver.log_data("Dict", {"a":1,"b":2,"c":3})
webserver.log_data("Tuple", (1,2,3))
```

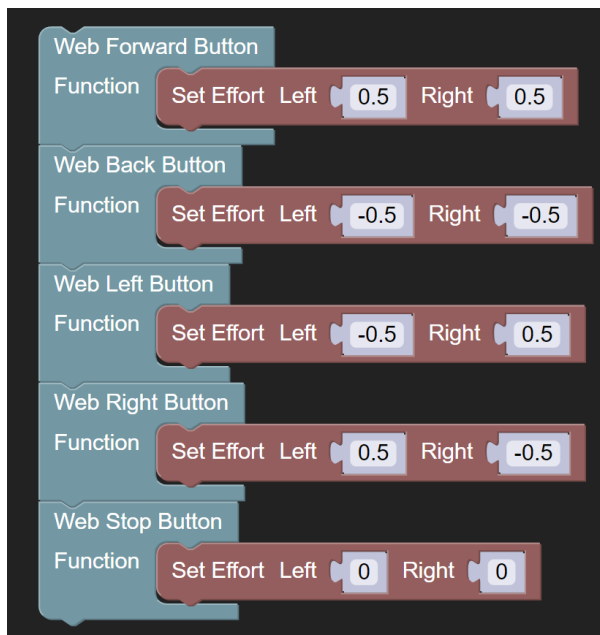


### 9.3 Teleop driving from the dashboard

To drive the robot, a program can create a driving interface by binding functions to left, right, forward, backward arrow buttons and a stop button. If bound, these buttons will appear in a diamond pattern in the browser and will call the bound functions when pressed.

Here is an example of how to do set up those bindings using lambdas (single line functions) that will operate a robot.

```
webserver.registerForwardButton(lambda: drivetrain.set_effort(0.5, 0.5))
webserver.registerLeftButton(lambda: drivetrain.set_effort(-0.5, 0.5))
webserver.registerRightButton(lambda: drivetrain.set_effort(0.5, -0.5))
webserver.registerBackwardButton(lambda: drivetrain.set_effort(-0.5, -0.5))
webserver.registerStopButton(lambda: drivetrain.set_effort(0, 0))
```

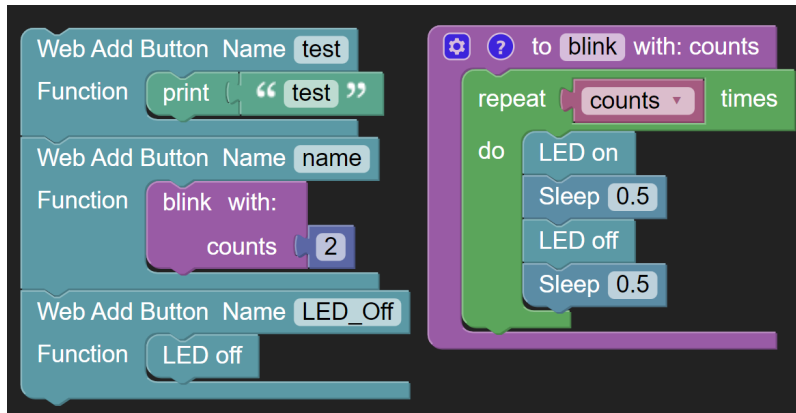


By using the `set_effort` function, the robot will continue to drive after a single button press until told to do something else.

## 9.4 Calling arbitrary functions from the dashboard

A program can create a button, that when pressed, will call a user function to do any operation that is required.

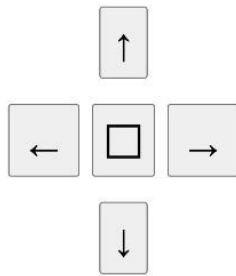
```
webserver.add_button("test", lambda: print("test"))
webserver.add_button("blink", lambda: print(led.blink(2)))
webserver.add_button("LED_Off", lambda: led.off())
```



## 9.5 Sample dashboard output

Below is an example of a dashboard that contains data logging, function buttons, and driving controls. This is a screen capture from a cell phone web browser where one can push buttons to run functions or use arrow keys as well as viewing values from the program.

## XRP control page



### Custom Function Bindings:

Test

Blink

Servo Up

Servo Down

LED Off

### Logged Data:

Button State: True

Left Motor: 1.603419

Right Motor: 3.801709

Time: 1609459871



### 10.1 Operating the LED

```
def test_led():  
    led.blink(5)  
    time.sleep(3)  
    led.off()
```

### 10.2 Waiting for a button press

### 10.3 Feedback-based control

### 10.4 Custom controllers

The `set_speed()` method of `EncodedMotor` uses a pretuned PID controller to maintain the speed through adjusting the the raw effort. However, through:

```
encodedMotor.set_speed_controller(controller)
```

a custom controller can be specified. The method expects a `Controller` object. The `Controller` abstract class is defined with the following three methods:

**update(error) -> float**

This method is called at every tick, and returns an output value given an input error value. This would be where PID logic would take place, for example.

**is\_done() -> bool**

This is a getter method that returns `True` if the controller has reached the target value, and `False` otherwise.

**clear\_history()**

The intent of this method is to reset any internal state of the controller, i.e. integral sum. This is called on the speed controller when `set_speed()` is called on the encoded motor.

By subclassing the Controller class and implementing these three methods, a custom controller can be used to control the speed of each motor.

## 10.5 Inbuilt PID controller

The PID class is a library-provided subclass of the Controller class, and provides a full PID implementation. The PID constructor takes in the following parameters, each with default values:

**kp = 1.0**

Proportional gain constant

**ki = 0.0**

Integral gain constant

**kd = 0.0**

Derivative gain constant

**minOutput = 0.0**

Constrain output to this minimum value

**maxOutput = 1.0**

Constrain output to this maximum value

**maxDerivative = None**

Constrain derivative term to this maximum magnitude

**tolerance = 0.1**

Error tolerance for `is_done()`

**toleranceCount = 1**

Number of consecutive ticks within tolerance to return True for `is_done()`

The default `set_speed()` controller uses a PID controller with the default parameters. However, it may be useful to pass in a PID object to `set_speed_controller()` with customized parameters to fine-tune the controller.

# CAPÍTULO 11

---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`