SR2024 Microgames 🚀

Welcome Space Cadets!

Welcome to the SR2024 bootcamp, here we have a series of challenges designed to assist you getting comfortable with using your provided robotics kits to build an amazing asteroid-collecting robot!

[Optional] Mission 0: Join our Discord

Before we send you back to your respective home planets, we need *stable comms*! Our first challenge is (arguably) the hardest of all, trying to navigate the dreaded *discord user interface*.

Our Discord server is the way to chat with us and other competitors. Our friendly volunteer support crew will be available there to help you solve any problems you have building your robot. (Our unfriendly volunteer support crew was not given Discord access).

1. Click or scan the join link here:

   JOIN US ON DISCORD

   https://discord.gg/BpSWKfUPS2

2. Click on the '`#welcome-<username>`' channel (it should have a red badge on it), you need to read and accept our rules by clicking on the button at the bottom before you can type anything in that chat.

To gain full access to our super-secure Discord server, you will need the **passphrase that was shared with your team supervisor**; you haven't lost it, have you?

3. **Enter your team's password** into the '`#welcome-<username>`' channel. If the message is correct you'll immediately get access. Your supervisor should have the password for your team.

Mission series 1: Kit

This series of missions gives you the basics on how to use your Student Robotics kit. Don't have your kit yet? Feel free to skip ahead to the Simulator section to get acquainted with our API and Simulator.

Mission 1 - Battery Charging 🔋

Battery safety is very important!

The two Lithium Polymer (LiPo) batteries in your kit together store the same energy as a **hand grenade** 💣, it just releases it much more slowly. So it's very important to treat them with care. Having said that, if treated properly these batteries are very safe.

Obviously, you should never:

- Pierce
- Compress
- Drop

...your batteries, what we ask of you is to also make sure you don't put them in situations where they can be pierced, compressed, or dropped (e.g. letting your battery run loose inside your robot, with exposed screws and a grabbing mechanism).

If you do treat these badly, batteries can enter 'thermal runaway', where the heat of a short-circuit can break down the insulation between cells and cause further shorts. When a battery enters thermal runaway, it's very hard to stop. So, please do treat them with care.

If you don't know how to do something, the best place to start is our documentation, at studentrobotics.org/docs. In the docs, we have detailed instructions on how to use the battery chargers in your kit.

We have two different chargers, make sure you use the instructions for the one you have in your kit.

Mission Summary:

Connect a battery and start charging it in the battery bag. Read the instructions in our docs, **the obvious way to charge is not the correct way**. Please get someone to check things are plugged in correctly before you start charging!

Mission 2 - Assemble your kit 🏗️

Now you know how to charge your battery, you can start plugging everything together.

You should start by assembling your kit and getting used to what's in the box. You need to get used to checking for information in the docs: studentrobotics.org/docs. It's very important to follow the instructions carefully.

Before you can assemble your kit, you will need to make your power cables. You'll need:

- The small flathead screwdriver in your kit
- Some black and red wire that should be in your kit
- Some wire cutters/strippers

Strip about 0.5cm off the end of short lengths of red and black cables, then screw them into the green screw connector (which we call Camcons), the ones that are the right size to fit into your power board L0/L1/H0/H1..etc sockets. Make sure you match red to '+' and black to '-'.

Once you've made these, you can start plugging it all in, at our [Assembly page in the docs](), there's a video tutorial on how to put it all together. If you have any more questions, don't hesitate to ask in person or in Discord.

> Note: Your brain board must be plugged into the 'L2' socket on the power board for it to receive power.
>
> Note: Your kit also won't turn on unless you have a jumper cable plugged into the 'on|off' terminal! If there isn't already one in your kit, you can make a jumper cable by wiring both terminals of a 5mm camcon (medium-sized green connector) together, and plugging it into the slot next to the on|off button. The thickness of the cable does not matter.
>
> **Warning**: The white plug on your battery is only used for charging. **Do not try to plug it into anything in your kit. It will damage the battery and the kit.**

Take your time, once you're happy that you've assembled your kit, it powers on, and you know which board is which, you can move on to the next mission.

Mission Summary:

Assemble your kit!

Mission 3 - Hello Kit 👋🧰

Now you've got a shiny kit assembled and working, it's time to run some code on it!

The purpose of this mission is to learn how to program your robot!
In order to put code on a robot, all you need is:

- A computer with a USB-A port (see picture)
- The USB stick included in your kit.

You can follow the instructions on how to program the robot in the 'Getting Code on the robot' section of the docs.

Now, in order to put code on your robot, you first need to write it! Our robots are programmed in python, we recommend you use a code editor to write your code, so check out our Code Editors page in the docs to see what you could use.

As a start, the programming tradition is to write a "Hello world", write this line into a robot.py file in the root of your USB stick, then plug it into the robot!

```
print("Hello world")
```

You should see this in a log file on your USB stick if it ran correctly! Once you're printing "Hello, World!", you can try and find the secret password, add the lines below and see what it prints. This code is deliberately complex and unreadable, so you only get the correct answer if you run this on your robot.

```
import sys
from sr.robot3 import Robot
print(f"SP{hex(int(sys.version[3:4])+1218)[2:]}")
```

The code is also available at `https://pastebin.com/iZxBwAG0` *for you to copy/paste on a device.*

You can check in your team channel on Discord to verify the secret code.

Mission Summary:

Find the secret code!

Mission 4 - Change LED colours! 🚨

Once you've got your code ready, let's see if you can write some code that makes your robot do something! We'll start off easy, blinking an LED (Light emitting diode)!

Your brain board is a Raspberry Pi, with the 'KCH' hat, which has controllable LEDs, you can control these very easily in your code. We've found LEDs are really useful to quickly figure out what's happening in your code, so please don't forget to use them when you're testing your real robot code!

Firstly, you need the two lines of code to import what you need, and secondly to set up your robot:

```
from sr.robot3 import *
robot = Robot()
```

Note: Your code editor may complain about 'sr.robot3' not existing. You can ignore this error, it's just that you don't have (and don't need) the robot code installed on your PC. If it bugs you, you can fix this in some editors! check the Code Editors page in our docs.

After the above lines, you can turn each of the Red, Green, and Blue channels of each LED on or off, like so:

```
# Make the 'A' LED Red
robot.kch.leds[LED_A].colour = Colour.RED
```

Please check the LED page in our docs for all the things you can do!

Once you've plugged the USB stick in with your code, you will need to wait for the flashing light and then press the black 'Start' button to run the code!

Once you've got the code running, make all of the LEDs switch between **Red**, **Green**, then **Blue** every second.

Hints:

- To wait for 1 second you can use the robot.sleep() method.
- You can use a 'while True' loop to make your code run forever

Mission Summary:

Make LEDs A, B, and C switch between Red, Green and Blue every 1.0 seconds.

Mission 5 - Broken Code 🩹

Engineers often come across many problems, anything from missing brackets in code or a cat jumping onto their keyboard — hopefully you'll only be dealing with the former.

To help prepare you for this, we would like to look at the code to the right. Point out any errors, and then fix them. Copy the code (also found on https://pastebin.com/T5B4uyqQ) into your code editor, then find the fixes, and then run your code on your kit!

This can be a very tricky task for people who are new to Python, if you need help, first look in the docs at [Python Troubleshooting page](). If you have any other issues, ask us in Discord!

Hint: there are 5 bugs in total.

```python
from sr.robot3 import *

robot = Robot()

def spin(duration, speed):
    # make robot spin
    robot.motor_board.motors[0].power = speed
    robot.motor_board.motors[1].power = -speed
    robot.sleep(duraiton)
    robot.motor_board.motors[0].power = 0
    robot.motor_board.motors[0].power = 0

def look_for_any_marker():
    markers = robot.camera.see()
    if len(markers) > 0:
        print("Found a marker!")
        return markers[0]

# Main code, tries to face a marker
marker = None
duration = 0.01
    speed = 0.1
while True
    marker = look_for_any_marker()
    if marker is not None:
        if marker.position.horizontal_angle > "0":
            spin(duration, speed)
        else:
            spin(duration, -speed)
```

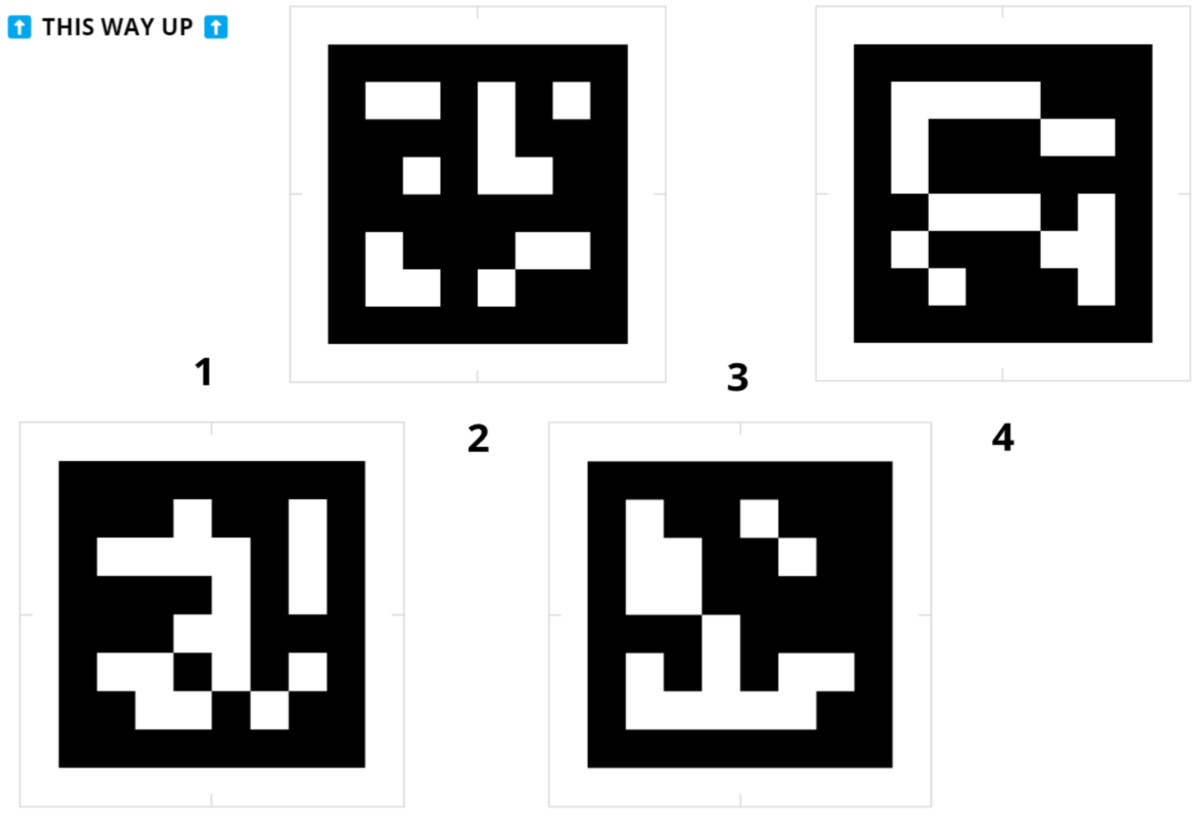
Mission Summary:

Find the 5 bugs in the above code

Mission 6 - Look for things 👀

Your virtual robot comes equipped with a camera on the top! With this, your robot can spot markers from far away. Our robot API does the complex vision work, and outputs the relative position of the markers in addition to both the orientation of the marker face relative to the robot. You can read our vision docs to find out more about vision.

To test your vision learning skills, we've prepared a secret to decode;



View Full Size & Print

Print out, or show the above image on your screen to your robots webcam. Use the robots vision system to extract the IDs of the markers, from left to right. Then use the cipher below to decode the secret word!

ID	Letter
150	J
151	X
152	S
153	Z
154	B

ID	Letter
163	L
164	P
165	U
166	Y
167	V

155	W
156	M
157	F
158	C
159	H
160	K
161	Q
162	T

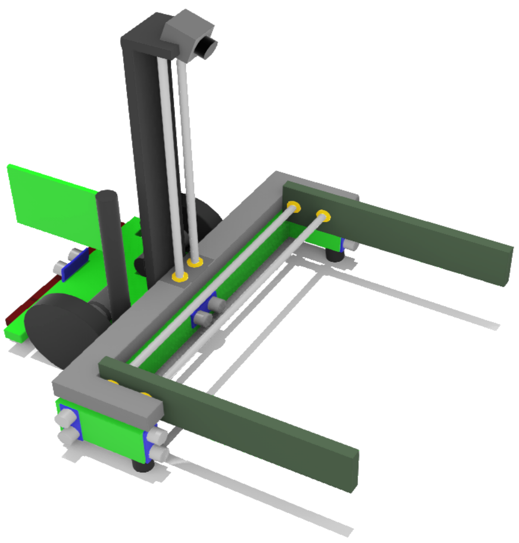
168	I
169	A
170	O
171	G
172	R
173	V
174	E
175	N

Our documentation on vision is the place to start. Try to write a program that prints the entire answer in one go! That is, it prints the markers from left to right.

Mission Summary:

Use the provided image, get the marker IDs from left-to-right, and convert them into a passcode.

Mission series 2: Simulator



We’ve developed a state-of-the-art simulator with a robot that’s a veritable asteroid-relocating-machine! These are to assist you with programming your own robot. This robot is equipped with many sensors, including ultrasound distance sensors and a camera on top. Plus, it has a big grabber which can lift objects up.

If you complete this series of missions, you’ll end up with a robot that can reliably score **12 points** in the game.

Mission 1 - Install the Simulator 🔽

First, you need to get the simulator working! Our Simulator supports Mac, Windows, and Linux.

Follow the instructions on the Simulator page in our docs. Our documentation should be your go-to place to find information about our robots, so you should learn how to read and navigate them well!

You will need to download both Webots and our simulator files, and **make sure you have at least Python 3.8 installed** on your machine.

Once you've downloaded both files, install Webots, then extract the 'competition simulator' zip. This is going to be your main workspace for the simulator. In the 'worlds' folder, double click on the `arena.wbt` file to run the simulation. It should launch Webots (if not, ask for help!), then start running a simulation of our sample robot. If you get an error, ask for help, usually it's an issue with your python installation.

Mission 2 - Hello Simulator 👋

You've installed the Simulator, congratulations! Next step is to run some code.

After successfully running the simulation, a *new* file named '`robot.py`' should have appeared just outside the 'competition-simulator' folder. This is our sample robot code (if a file ends in .py, it means it is a python file).

Now, in order to run code on your robot, you first need to write it! Our robots are programmed in python, we recommend you use a code editor to write your code, so check out our Code Editors page in the docs to see what you could use.

Tip: If you can't find your robot.py, it gets created when you first run the simulator, so make sure you've ran the simulator once already!

Now, copy and paste the following code into the top of your `robot.py` (after you create your robot), which should be just outside the 'competition-simulator' folder:

```
print("Secret Code:", "rvo3eAt4eWM"[::-2])
```

This code is also available at pastebin.com/NDdS0BAB if you have this printed out.

(Don't worry if you don't understand it, we made it to be deliberately hard to understand, so it's tricky to figure out what it does before you run it, kudos to those who do understand it however.)

Then, after saving, you can run the code by double-clicking on the `arena.wbt` file (in the 'worlds' folder), or by clicking on the restart button: ↻.

The message should appear in the 'Console' at the bottom.

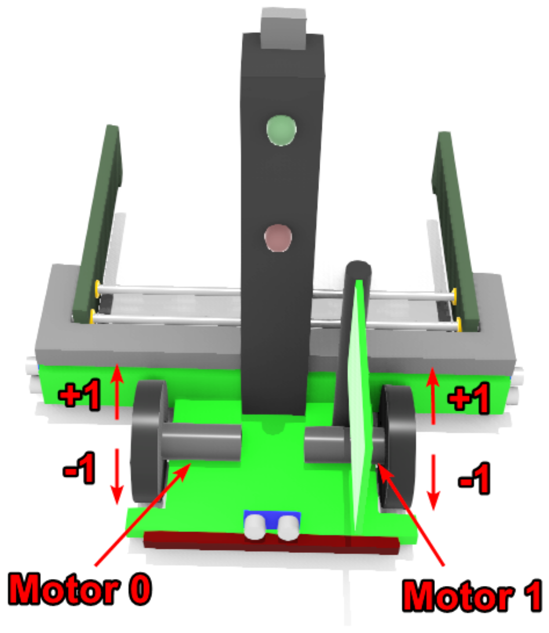
If you don't have a console at the bottom of the window, try resizing the main view of the simulation -- it may just be hidden. If that doesn't work, you can create the console by going to Tools > New Console.

If you get an error message, it might be that you have a version of python lower than our minimum, Python 3.8. You can install a later version at www.python.org/downloads/. If you have any other issues, don't hesitate to contact us!

If you find the secret code, well done! You can check the answer with us (either in Discord in your team channel or via your supervisor), and move on to the next mission.

Mission 3 - Movement 🚀

For your first step in your asteroid-collecting career, you're going to need to actually get to the asteroids! The robot has 2 big motors attached to wheels, one on each side:



To drive the motor, you simply tell it how much power to put in each motor. You can set the motor power to anything between -1 and 1. Your first port of call (as always) is our documentation, it would be wise to take a look at the programming category, In particular, the motor board page.

Our simulation robot has 1 motor board, with 2 motors plugged in, in your physical kit we give you two motor boards, meaning you can power up to 4 motors for your actual robot (this doesn't count servos!) There's a special page on our documentation that describes the setup of the virtual robot, it has details like which motor does what. It's worth looking at!

First, make your robot drive forwards a little bit by setting both motors to **0.4 power for 0.5 seconds**, (Warning, You'll find the simulator robot is pretty unstable if the motors are at 1.0 power!)

Motors only ever change speed if you tell them to, so if you set the speed to 0.5, then sleep for 1 second, then set the speed to 0, the robot will have moved forward at 0.5 speed for 1 second.

Important: If you've programmed in python before, you may know about `time.sleep(duration)` as a way to wait for a duration. For our simulator and our kits, you should use `robot.sleep(duration)` instead, as it synchronises with the simulator, so it'll be more consistent if your simulator is laggy.

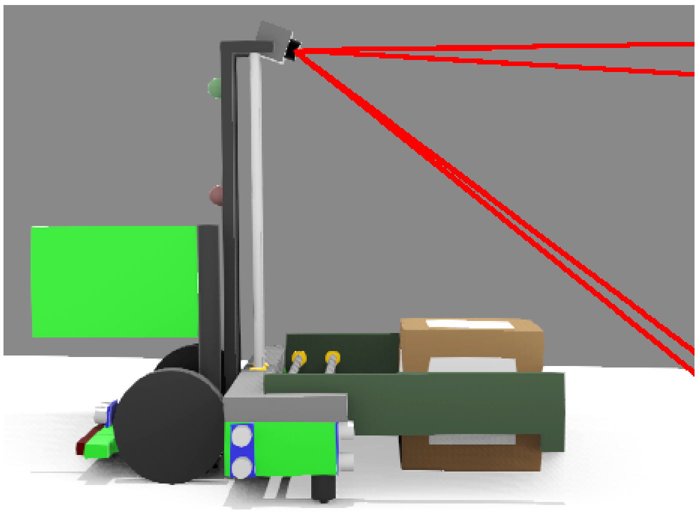
When you've managed that, make your robot turn on the spot, about **20 degrees** (doesn't need to be accurate, just make sure it can still see the asteroid in front of it for the next task!)

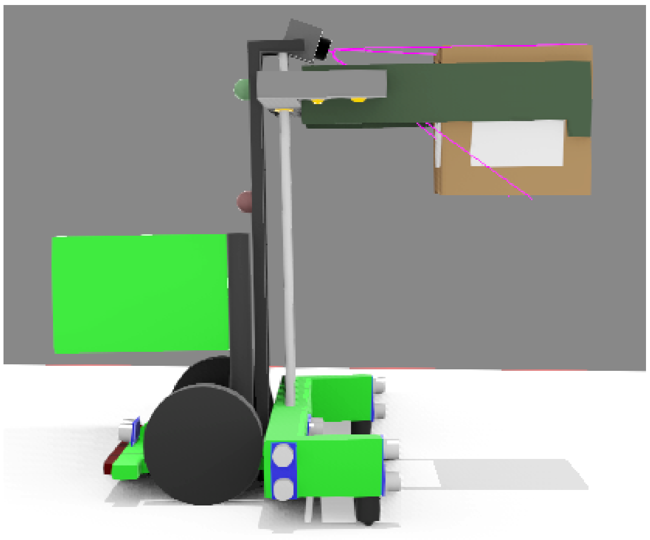
Mission Summary:

Make the robot drive forwards for 0.5 seconds, then turn 45 degrees.

Mission 4 - Vision 👀

Your robot has a camera attached to the top post. There are square 'fiducial' markers on the arena walls, asteroids, and spaceships. Your camera can not only see these, but it can measure their distance, position, and orientation, relative to the robot!





Beware, the simulator robot camera is positioned in a way that it can see the entire arena, but it can't quite see asteroids directly in front of it! You'll need to code your way around this. Don't worry about it for now!

Also good to know: If you lift your grabber up high enough, the asteroid will block the camera's view!

In this mission, you need to make the robot face a marker; Once the marker is within 2 degrees (0.035 radians) of forward, drive forwards for 1 second at 0.4 power.

Check the programming category of our docs to see how to program your robot! In particular, the vision page.

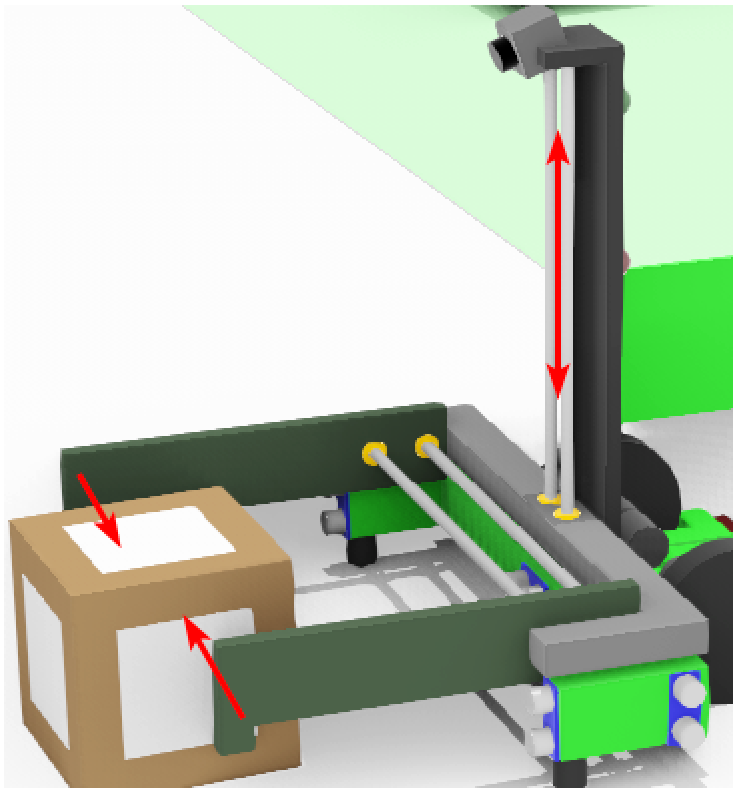
Once you call `markers = robot.camera.see()`, your `markers` variable will contain a list of marker objects. These objects have a position that gives you the relative position. For this task you can use the first marker in the list, `markers[0]`.

If you put this in a `while` loop, you can keep calling `robot.camera.see()`, then turning the robot a very very small amount, (like 0.05 power for 0.1 seconds). The direction you turn the robot can be determined by the horizontal angle of the marker's position.

Mission Summary:

Make the robot line up to a marker, then drive forward for 1 second at 0.4 power.

Mission 5 - Servo Movement 🦾



In addition to dumb motors, your robot can power **servos**. Servos are smart motors, they know how far they've rotated. You can tell servos to go to a specific position. Though continuous servos do exist, most servos have a limited range of movement. All in-all, this makes them a decent choice when you want to make a grabber!

Your robotics kits can power up to *12* servos through the servo board. You can see how to get your code to move your servos in the Servo board programming page.

The simulator robot has 3 linear actuator servos, one on each grabber, and one on the lifting mechanism.

If everything went well with your previous mission, you'll find yourself with an asteroid in reach!

For this mission, add to the end of code for the previous mission, to make your robot close its grabber, and lift the asteroid upwards. You'll want to take a look at the simulator programming section to see which servo does what.

Mission Summary:

Make your robot close its grabber, then lift up an asteroid

Mission 6 - Light up the Sky 💡

The simulator robot has two LEDs on it, LEDs are useful to immediately see what has happened to your robot. You can hook LEDs up to your physical robot using the Arduino.

We use the arduino as a general-purpose input-output board. This means it doesn't really know what's plugged into it, so you need to tell it what's plugged in; before you can use your LEDs, you need to set the 'PIN Mode' for those pins. For the LEDS, this mode is `OUTPUT`.

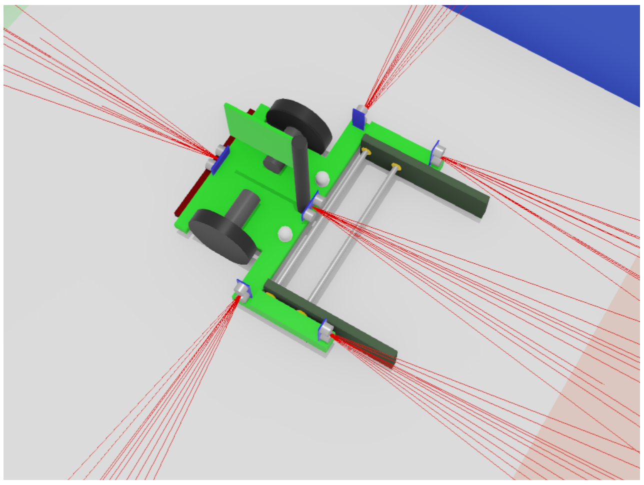
The simulator programming docs explain which LEDs to use in the LEDs section, and the Arduino programming page explains pin modes and how to use the `.digital_write()` function.

Note:
There is currently a bug in the simulator where you cannot change LEDs in the first 0.5s of a simulation. We recommend you add a `robot.sleep(0.5)` straight after `robot = Robot()` to avoid hitting this issue.

Mission task:

Modify your program from the previous mission so that your robot lights its green LED when aligning to a marker, then the red LED when your code is finished.

Mission 7 - Distance Sensors 🤏



Distance sensors, (usually ultrasound sensors in the physical world), report the closest distance away from them in a cone perpendicular to the sensor. Our sensors can read up to 2m away, but get more noisy the further away things are. Normally they tell you the distance of the *closest* thing in the cone.

They can be used for many situations, like to make sure you're not about to crash into a wall. There are 6 in total (see the picture above). Our virtual robots have the distance sensors connected up to the I/O pins of the 'arduino', which is a board in our robotics kit that has general purpose pins. In our physical kit, you can connect whatever you want to these pins. However, in the simulator we've connected 6 distance sensors in analog pins A0-A5. You can check which is which in the simulator programming section.

These sensors give you a range of values, from 0 to 5 for distance in metres, thus you'll need to use the `analog_read()` function to read their distances. As always, check our docs!

You can hook up an ultrasound sensor on the physical kit too, it's a bit more involved than the other sensors, ask us and we'll be happy to explain how you'll need to do it.

From the previous task, you should be proficient at grabbing and lifting up asteroids, but maybe you've noticed your robot doesn't always grab the asteroid every time. Plus, you can't see it with the camera when it's that close! Thankfully, there's a handy distance sensor at the front centre of the robot. The perfect place for verifyingif there actually is an asteroid in your grips!

Mission task:

Change the code from the last mission, (in addition to your code from the previous missions), to light up the red LED if there’s no token in front of the robot when the servos need to move.

Mission 8 - Bump sensors 👊

Bump sensors are great for detecting if something has bumped into them. Normally, bump sensors are switches which connect two wires when they bump into something. They're really useful, and really easy to hook up on your real robot, you just need a switch and two wires!

It's an excellent way of spotting when you're stuck up against a wall, or if you managed to grab something correctly.

The bump sensor in the simulator is set up very similar to how you would do it in real life, it's connected to the IO pins of the 'arduino', which is a board in our robotics kit that has general purpose pins. In our physical kit you can connect whatever you want to these pins. In the simulator, it has a bump sensor attached to digital pin 2.

As always, we have information on how to read data from the arduino in our documentation: studentrobotics.org/docs/programming/arduino/sr_firmware

You do need to set the pin modes for your robot, see the simulator programming page for what modes to put them into. The bump switches should be in INPUT mode. '`digital_read(<pin number>)`' them.

Now, make your robot drive backwards, then stop when it hits the wall. Once it can, you can move to the next mission!

Mission 9 - Find & pick up an asteroid 🔄👀➡️🧊

This mission puts together all that you've learned so far! Your goal is to make a robot which can score 12 points - That is, a robot that gets 1 asteroid from the field into your scoring zone.

1. 🔄 use your knowledge of motors to make your robot turn around to roughly face the asteroid.
2. 👀Use vision to spot the asteroids, use our vision documentation to see how to identify markers. Do make note of what the rules say about Marker IDs, or you'll end up spotting a wall instead of an asteroid.
3. Find the closest asteroid to the camera.
   Hint: you can do this using R.camera.see(), which returns a list of Marker objects, then use m.distance (the distance property on the marker object), which reports the distance of the marker from the camera.
4. ➡️Use the motors to drive towards the asteroid.
5. 🧊Use the 'Front' distance sensor to find when the asteroid is close enough, and use the grabber servos to grab the asteroid.

If you've got your robot reliably doing this, you've finished this mission! Be sure to boast about it in Discord if you can, we'd love to see it!