Updated: 2026-08-09

SRO Programming Lab

Complete this lab to earn 4 league points.

Deadline: End of Wednesday session

Preparation

To install the simulator on the lab computers or your own computer, follow the instructions at:

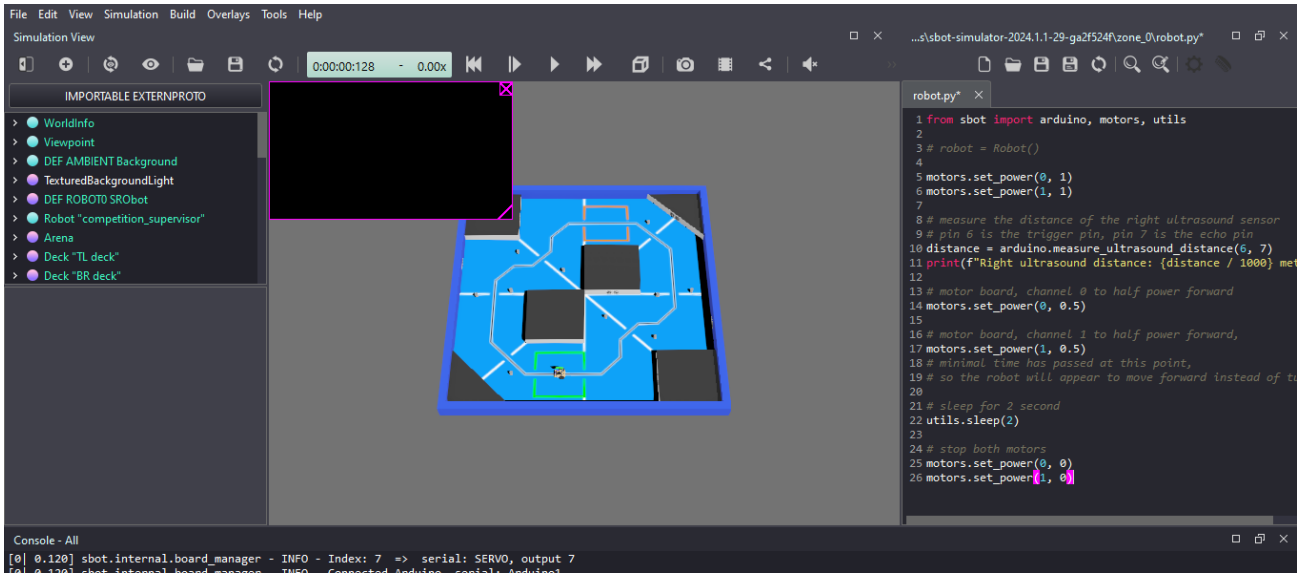
Setting up the Simulator | Docs

Just press the blue button and follow the install script.

1 – What are all these windows?

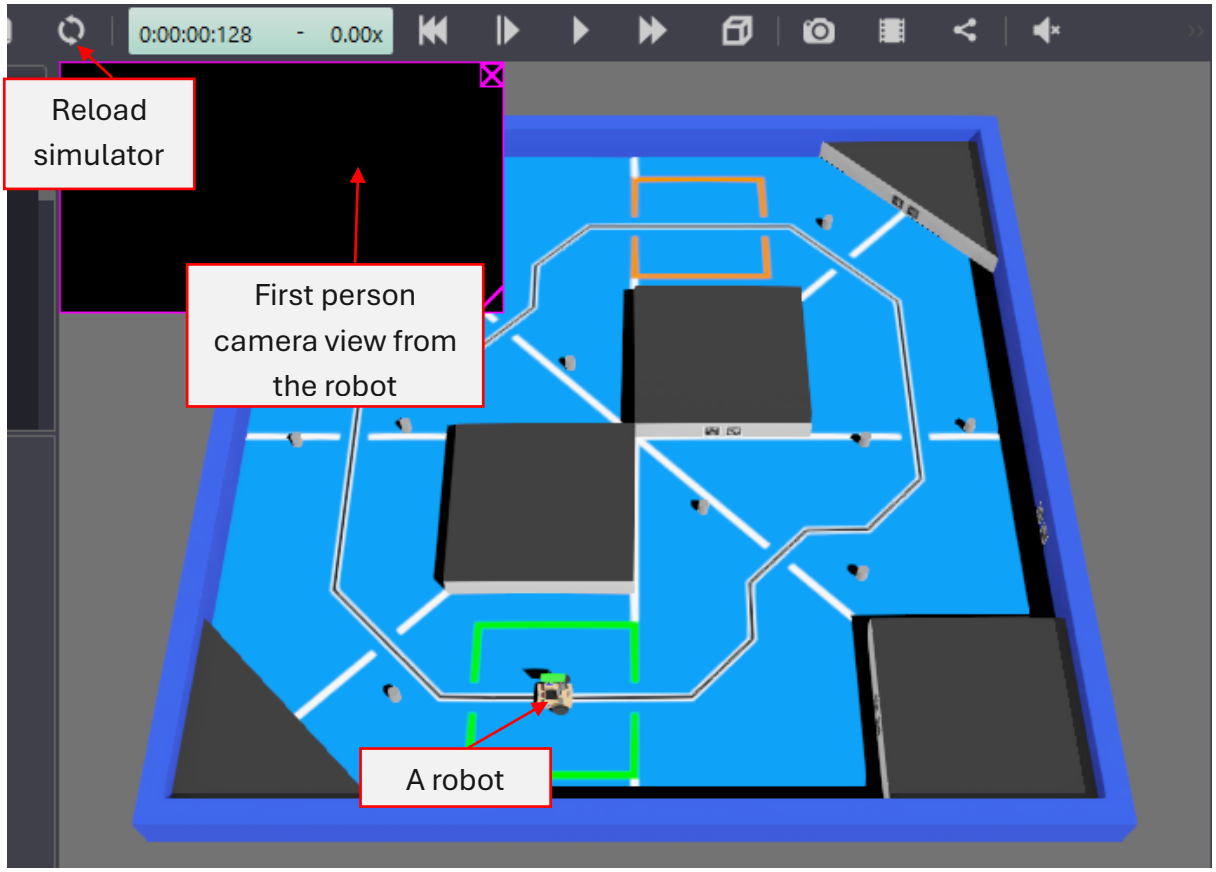
Open the simulator by running *run_simulator.py*.

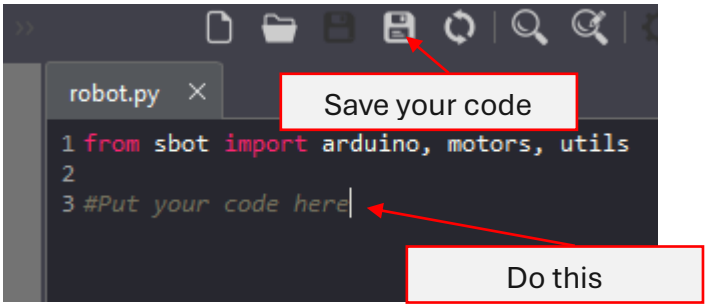
You should see something like this:

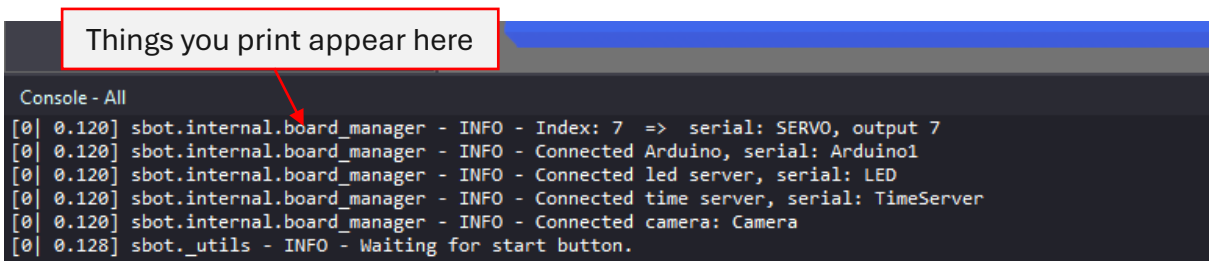


If you don't see something like this, ask your team mentor for help.

Here’s a quick tour.







Task 1 – Hello World!

To make sure you know how to use the simulator, tell the robot to print “Hello World!”.

You’ll need to save your code and reload the simulator.

Show your Team Mentor

2 – Making a Move

Now let's learn to make the robot move.

The simulated robot has **two motors** connected to **one motor board**.

The **left motor** is connected to **output 0** and the **right motor** is connected to **output 1**.

Motor power can go from **-1.00** (full reverse) to **+1.00** (full forward).

Open the docs page at https://docs.sro.bot/programming/motor-board/ for more information.

Try this code:

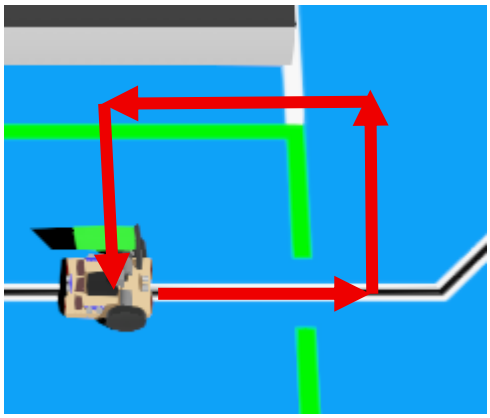
```
from sbot import motors, utils

#set the left motor to 50% power and right motor to 0% power
motors.set_power(0,0.5)
#Robot power goes off at end of code, so wait for 1 second
utils.sleep(1)
```

You should see the robot move!

In set_power(x,y), x is the **output channel.** In the simulator x = 0 is the left motor, x = 1 is the right motor. y is the motor power.

Task 2 – Moving Around

1. Make your robot move in a straight line forward for 2 seconds, then reverse for 2 seconds.
   Show your Team Mentor
2. Make the robot drive in a rectangle as shown:



   Hint: Moving one motor forward and the other motor backward will make the robot turn on the spot.
   Show your Team Mentor

Open Loop Control

Open Loop means the robot does not use any sensors, so it has no **feedback**.

Try the code below and see what happens.

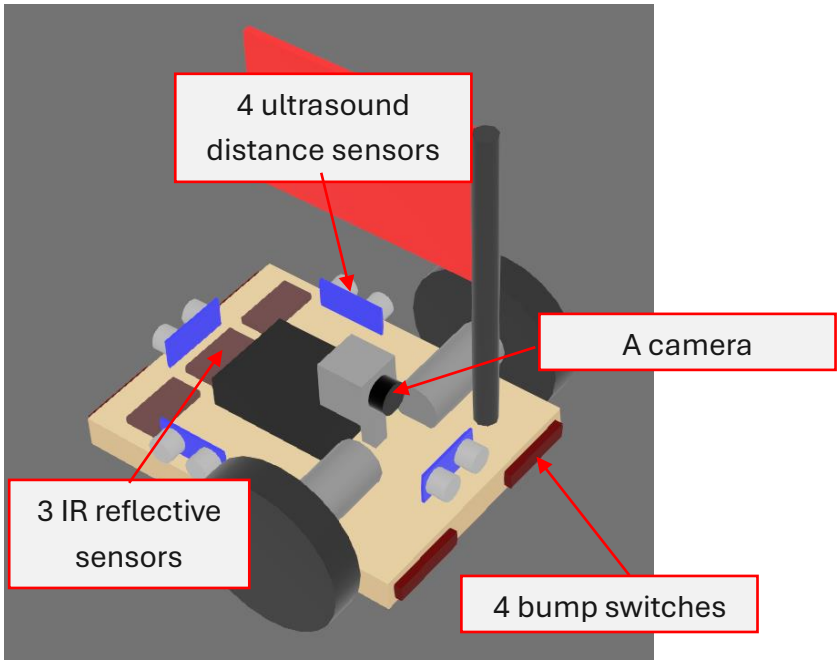
```
from sbot import motors, utils

#Go forward and backward at 50% power for 1 second 10 times, ending at the starting point
for i in range(10):
    motors.set_power(0,0.5)
    motors.set_power(1,0.5)
    utils.sleep(1)
    motors.set_power(0,-0.5)
    motors.set_power(1,-0.5)
    utils.sleep(1)
```

Open loop has some problems! Small errors build up over time, so the outcome deviates more and more.

Closed Loop Control

A closed loop system uses **feedback** from sensors to give the robot information about the world. This information is used to modify the behaviour of the robot.

The simulated robot has:



Full details of the sensors can be found at:

https://docs.sro.bot/simulator/robot

Useful Tip

Helper functions can make your code much more readable and quicker to write.

For example, you could create a helper to easily set both motors. This would be something like:

```
def set_motors(left,right):
    motors.set_power(0,left)
    motors.set_power(1,right)
```

Think about which functions could be useful for your robot as you go through this workshop.

3 – Using Sensors

3.1 Bump Switches

By placing a microswitch on the perimeter of the robot you can detect when it bumps into an object and do something to correct it. Microswitches are very cheap and easy to implement.

The sensors are connected using an Arduino board. To use it you will need to import the appropriate library.

```
from sbot import arduino
```

You can read the state of a switch using the command:

```
#pin 11 is the front right microswitch
#Consult the docs for the other switch pins
frontright = arduino.digital_read(11)
```

For other pin connections see: https://docs.sro.bot/simulator/robot#attached-sensors

This will return a value of true (switch pressed) or false (switch not pressed).

Task 3 – Bump

Write code to make your robot:

1. Drive forward until it hits a wall.
2. Print the message “I have hit a wall!”
3. Drive backwards for 1 second then stop.
4. Rotate left 90 degrees.

Show your team mentor the working code.

3.2 – Ultrasound Distance Sensors

Ultrasound distance sensors send out pulses of high frequency sound and measure the time for the reflection to return. This allows the arduino to calculate the distance to a surface.

Try the code below.

```
from sbot import motors, arduino

def set_motors(left,right):
    motors.set_power(0,left)
    motors.set_power(1,right)

set_motors(0.2,0.2)
while True:
    distance_left = arduino.measure_ultrasound_distance(4, 5) #This is the left sensor
    print(distance_left)
```

You should see the distance to the left wall printing over and over again.

Consult the docs page at https://docs.sro.bot/simulator/robot#attached-sensors to find the pins for each of the sensors. You will learn more detail about the sensors in the Electronics Lab Session.

Task 4 – Ultrasound Navigator

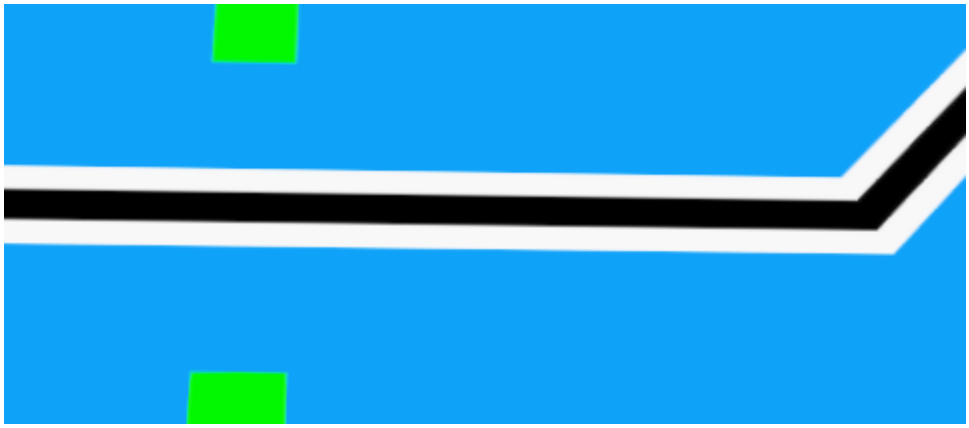
Write code to make your robot:

1. Drive forward until the **front distance sensor** is less than 500mm from the wall.
2. Turn left.
3. Drive forward until the **front distance sensor** is less than 300mm from the wall.
4. Turn right.

Show your team mentor the working code.

3.3 – Infrared Reflectance Sensors

These sensors send out IR radiation (using an LED) and detect the amount reflected by the surface. The arena floor has a white strip with a black line along the middle.



Each colour (blue, white and black) will reflect a different fraction of the incident radiation.

IR sensors are **analog** which means they can have any value between 0V and the positive power supply (5V). In our setup, more reflection gives lower voltage.

Try the code below:

```
from sbot import motors, arduino, AnalogPin

while True:
    left_IR = arduino.analog_read(AnalogPin.A0)
    centre_IR = arduino.analog_read(AnalogPin.A1)
    right_IR = arduino.analog_read(AnalogPin.A2)
    print(left_IR, centre_IR, right_IR)
```

The robot will initially have the left and right sensors on the white strips and the centre sensor on the black strip. The white strips reflect more IR, so they give a value close to 1.00 V. The black strip absorbs most of the IR, so it gives a value around 5.00 V.

As a rough guide **in the simulator**:

Black returns greater than 3.50 V.

White returns lower than 1.00 V.

The carpet will be somewhere in between.

This is a good time to introduce **state machines**.

Finite State Machines (Bang Bang Control)

To plan the code we can work out all the possible **states** for the robot. These are "things it should do". We can also work out the **transitions.** These are "when it should change state".

This table shows some of the possible states for the robot and the conditions for entering that state.

Transition			**State**
Left IR	**Centre IR**	**Right IR**	
White	Black	White	Go forward
Black	White	Carpet	Go left
Carpet	White	Black	Go right

This means we can **make a function for each state** and **change state depending on the sensor values**. This approach is called **bang bang control** as discussed in the control theory lecture.

The code below shows an **incomplete skeleton code** for this approach.

```
from sbot import motors, arduino, AnalogPin

def set_motors(left,right):
    motors.set_power(0,left)
    motors.set_power(1,right)

def set_state(state):
    if state == "forward":
        set_motors(0.2,0.2)
    elif state == "left":
        set_motors(0,0)#add your code here for left and right

while True:
    left_IR = arduino.analog_read(AnalogPin.A0)
    centre_IR = arduino.analog_read(AnalogPin.A1)
    right_IR = arduino.analog_read(AnalogPin.A2)
    print(left_IR, centre_IR, right_IR)
    if left_IR < 1.2 and centre_IR > 3.5 and right_IR < 1.2:
        current_state="forward"
    #Add your code here for left and right
    set_state(current_state)
```

Task 5 – State of Mind

Complete the skeleton code to make the robot drive around **at least one bend** using the IR sensors.

Show your team mentor the working code.

3.4 – Vision System

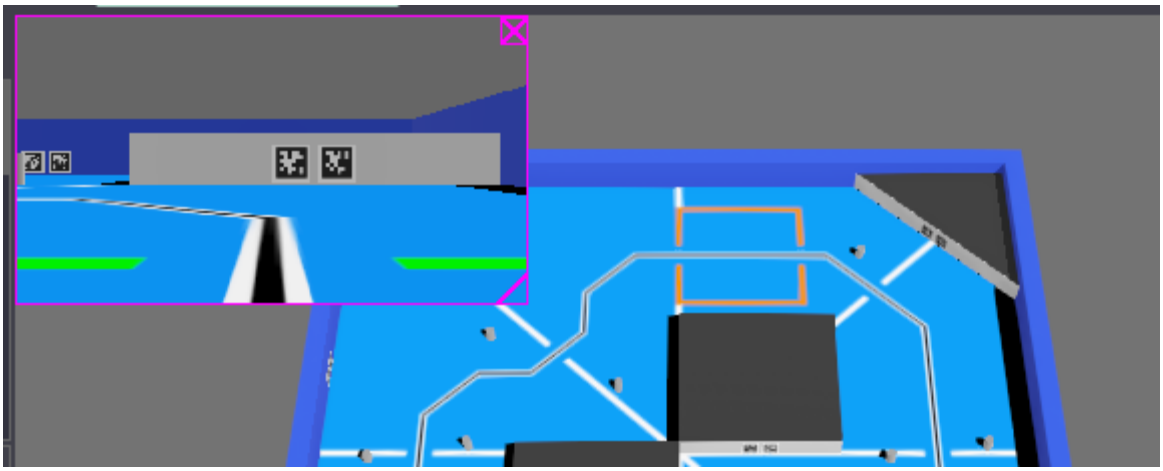
The robot has a camera. Taking images generates useful information but **is slow** and **the robot must stop moving** or motion blur will make it unreliable.

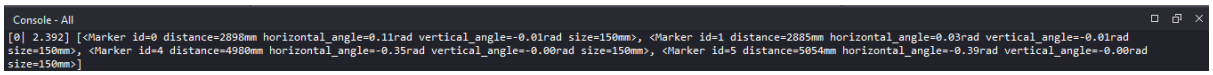
Try running the code below:

```
from sbot import vision, motors, utils, arduino


while True:
    markerlist = vision.detect_markers()
    print(markerlist)
    utils.sleep(1)
```

vision.detect_markers() returns a list of **Marker objects**.

You should see the first-person camera view update, and a lot of things print in the Console window.





Congratulations! You have a working vision system!

What's in a Marker?

Marker objects contain lots of useful information.

The **id** attribute holds the marker ID number. For details of the ID numbers, see the Rules document.

The **distance** attribute tells you the distance in mm between the centre of the camera and the centre of the marker.

The **horizontal_angle** and **vertical_angle** attributes give you the angle between the centre of the camera and the centre of the marker **in radians**.

Reminder: 180° = π radians

The code below will take an image and try to line up the robot with marker 0.

```
from sbot import vision, motors, arduino, utils

target_id = 0

#Helper to set both motors
def set_motors(left,right):
    motors.set_power(0,left)
    motors.set_power(1,right)

#Helper to find the target marker in the list of markers
def find_target(markerlist, target):
    for marker in markerlist:
        if marker.id == target:
            return marker
    return None

while True:
    markerlist = vision.detect_markers()
    targetmarker = find_target(markerlist, target_id)
    if targetmarker != None:
        angle_error = targetmarker.position.horizontal_angle
        if angle_error > 0.2: #target is on the right, 0.2 radians is about 11°
            set_motors(0.1,-0.1)
        elif angle_error < -0.2: #target is on the left
            set_motors(-0.1,0.1)
        else: #target is straight ahead
            set_motors(0.2,0.2)
        utils.sleep(0.2)
        set_motors(0,0) #stop to take a new photo
    else: #can't see the target
        set_motors(0,0)
```

Task 6 – Vision Quest

Write code to make the robot:

1. Find and drive toward marker 1 until it is 500mm away.
2. Find and drive toward marker 3 until it is 300mm away.
3. Find and drive toward marker 5 until it is 500mm away.

Show your team mentor the working code.

Congratulations! You have now completed the workshop!

Use the simulator to write and test your code throughout the rest of the week but remember the **real robot will not behave exactly like the simulated robot!**

Next Steps

Proportional Control

You can make your robot much more efficient by implementing **proportional** control.

A few examples:

1. When driving toward a marker you can **drive faster** when the **distance is bigger**.
2. To use ultrasound sensors to follow a wall you can change the left and right motors to turn toward or away from the wall as you drive forward.

To do this you need to:

1. Measure the **error** from the desired value
2. Set the motor speeds **proportional to the error.**

The general form of equation for this will be:

$$error = measured\ value - desired\ value$$

$$motor\ speed = minimum\ value + constant \times error$$

If you want the robot to track a wall you might use:

$$left\ motor\ speed = minimum\ value - constant \times error$$

$$right\ motor\ speed = minimum\ value + constant \times error$$

You will need to experiment and tune the constant to make it work best.

Bonus Task – Implement Proportional Control

Write code to implement proportional control for your forward movement, using either the ultrasound or the vision system.

Show your team mentor the working code.

PID Control

PID (Proportional, Integral, Derivative) is an even better system. Talk to Dan if you want to try it!