Robotics Lab 3: Basic Closed-Loop Control

Authors: Stuart Harley, Kiel Dowdle

Date: 9/21/2021

Abstract

In robotics, a closed-loop control system is one in which the robot has access to sensor information and feedback to inform its actions. These types of systems are generally more robust than open-loop systems, which have no form of feedback, but they also require more components and programming logic. In this lab we implement 3 different closed-loop systems to make the robot maintain a distance from an object ahead of the robot, follow a colored line on the ground, and follow a wall. We use a reactive system to make the robot follow the object ahead of it which ended up being very simple and efficient. We use a Finite State Machine to model the other two tasks since they are more complicated and so reactive control would not work as well here, though it may have still been possible. Following the line was done effectively using an FSM and a color sensor. Following a wall was able to succeed with the FSM and two touch sensors mounted on the front of the robot. But it would have benefitted from the use of alternative sensors in order to complete the task effectively and efficiently.

Methods

Task 1 (Ultrasonic Sensor Follow with Reactive Control)

Task 1 was designed to use reactive control. Reactive control is a control system where actions are a function of sensory input. This means that there is very limited computation being done. The robot is reacting simply to sensory data.

For task 1, we derived an equation to model the speed of our robot depending on how far the robot was from an object. The distance to the object was determined by an ultrasonic sensor attached to the front of the robot. Since the robot needed to stay at a specified target distance, we simply subtracted the target distance from the actual distance. If the robot was too far from the object, the equation would result in a positive value and the robot would move forward. Vice versa for backwards. If the robot was at the target distance the speed would be 0 and the robot would not move. This equation ended up moving the robot relatively slowly, so we sped it up being multiplying the result by a factor of two. Below is the final equation we developed.

Speed = (actual distance – target distance) * 2

Task 2 (Color Sensor Line Following - FSM)

For task 2, a line jagged was drawn on the ground using masking tape that the robot would follow as shown below in Image 1. The robot starts at one end of the line with the color sensor overtop of the tape line.

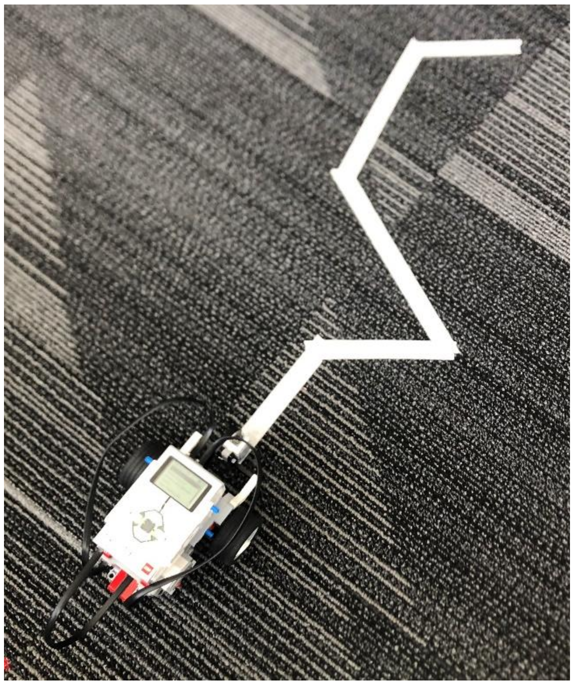


Image 1: Setup for task 2 (Color Sensor Line Following)

Unlike in task 1, in task 2 we used a finite state machine control system instead of using a reactive control system. For the purposes of this report, a finite state machine is a control system involving breaking the logic of the robot into reasonable sized chunks of modules. These states can be further broken down into smaller states. In terms of the code, this involves creating methods for each state which then call each other. Shown below in figure 1 is the finite state machine we developed to solve task 2.

As shown below, we had 2 high level states in our FSM. The first was the on-path state. When the robot sensed the tape below, it was “on-path,” and all the robot did was move forward. Once the robot went “off-path” it would rotate to find the path again. It would first rotate either left or right for three and a half seconds, approximately 140 degrees. The direction it would first rotate is the direction of the previous turn the robot made. If it did not find the path again in these three and a half seconds, it would rotate in the opposite direction until the path was found.

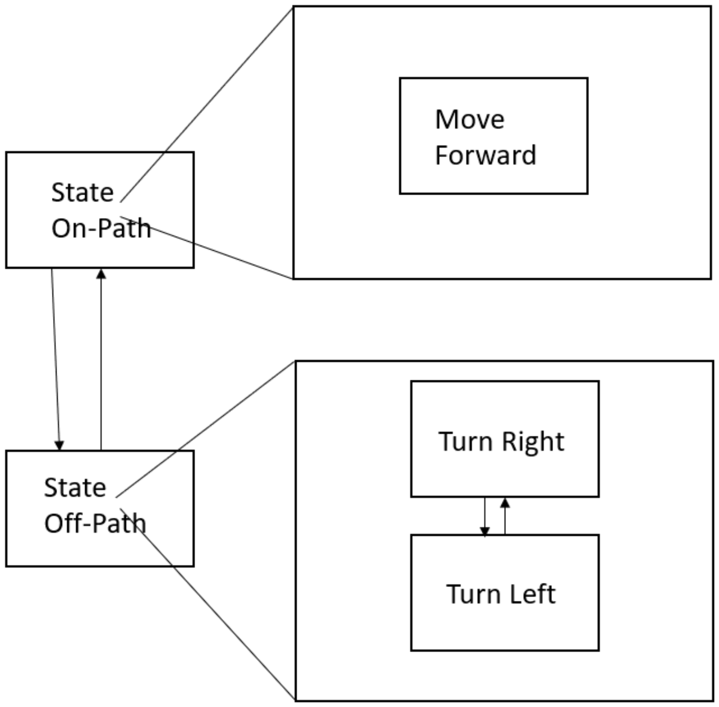


Figure 1: FSM for task 2

Task 3 (Wall Following - FSM)

For task 3, the robot is placed 8cm to the left of a wall in order to begin as shown below in Image 2. The code written for this task only works if the wall is to the right of the robot. The robot has 2 touch sensors mounted in the front that will signal if the robot runs into a wall directly ahead of it.

Image 2: Setup for task 3 (Wall Following)

Like in task 2, for task 3 we used a finite state machine control system. Shown below in figure 2 is the finite state machine we developed to solve task 3. As shown below, we had 3 high level states in our FSM. The first was the on wall state. In this state the robot would simply advance forward for three seconds or until it reached a wall, whichever occurred first. Three seconds was chosen as the time limit because the robot moved forward a medium sized amount without going too little so it advanced very slowly and not too far so that it could possibly get too far away from the wall.

If the robot hit a wall in this state it would advance to the hit wall ahead state. Here the robot would back away from the wall and then rotate 90 degrees left. Then it would go back to the move forward state.

If the robot did not hit a wall in the move forward state, then it would go to the check wall right state. Here, the robot would rotate 90 degrees right. It would then advance forward to check if it was still adjacent to the wall. If the wall was found it would proceed to the left turn state to revert to the state the robot was in at the beginning of the right turn state. If the wall was not found, then it would immediately return to the move forward state because the robot had rounded a corner.

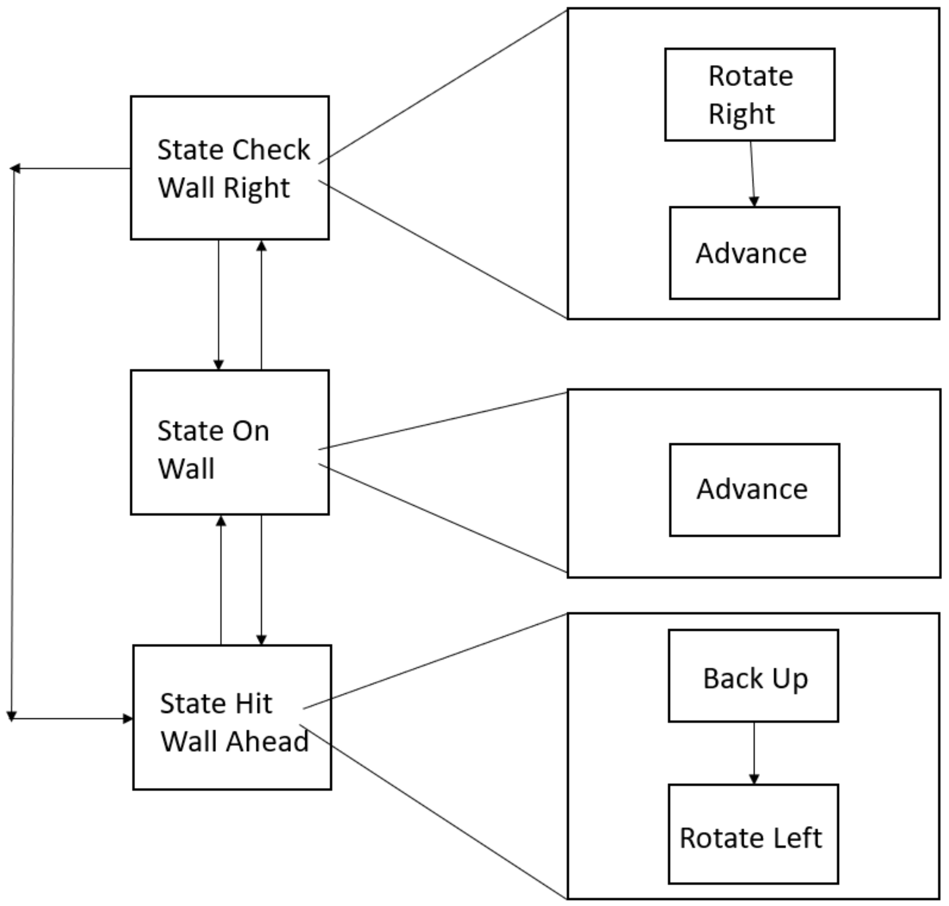
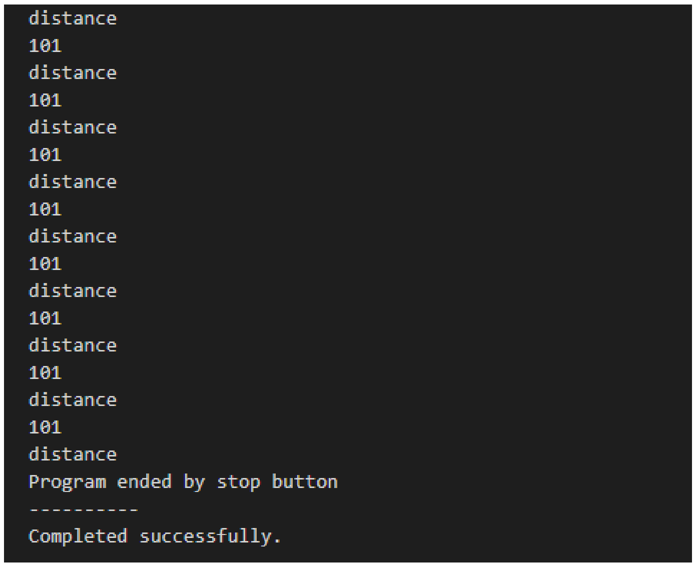


Figure 2: FSM for task 3

Results

Task 0 (Setup and Reading Sensors)

This task was a subtask of task 1 in which we implemented a simple function to print out the sensor values from the ultrasonic sensor. As shown below in Image 3, the output of the ultrasonic sensor was the distance to the object in front of the robot, reported in mm.



```
distance
101
distance
101
distance
101
distance
101
distance
101
distance
101
distance
101
distance
101
distance
Program ended by stop button
----------
Completed successfully.
```

Image 3: Output of the ultrasonic test function

Task 1 (Ultrasonic Sensor Follow with Reactive Control)

Shown below in Images 4 through 6 are screenshots from the video taken of the robot completing task 1. In Image 4, the robot is moving forward closer to the object. In Image 5, the robot is moving backwards away from the object, and in Image 6 the robot is at the target distance from the object, so it is sitting still.

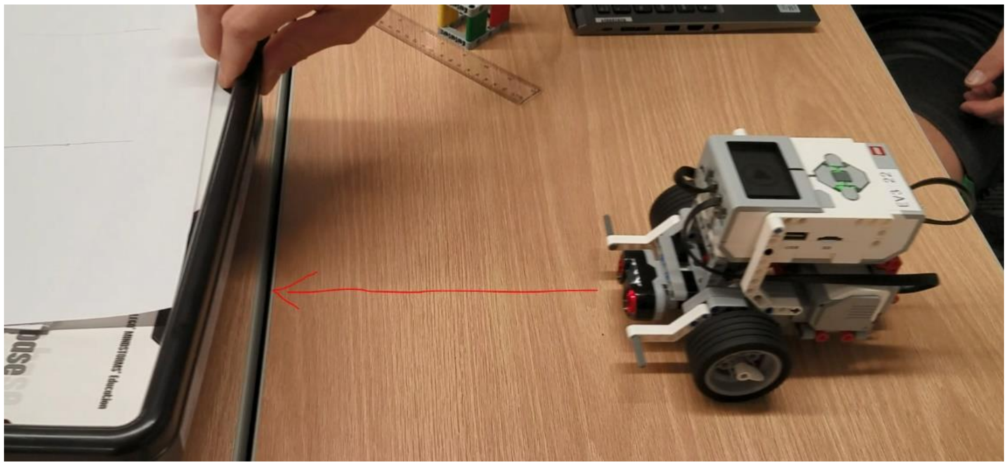


Image 4: When the robot's distance from the object is greater than the target distance it will move forward

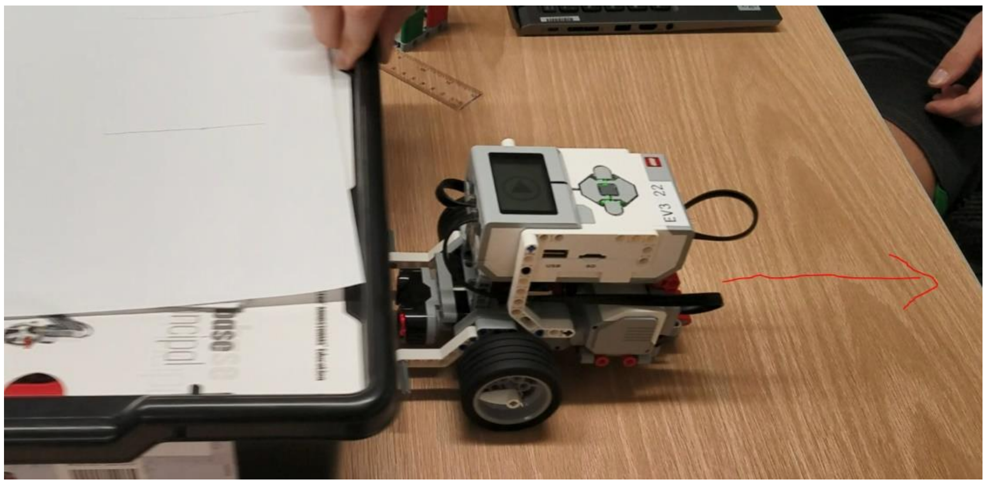


Image 5: When the robot’s distance from the object is less than the target distance it will move backwards

Image 6: When the robot's distance to the object is equal to the target distance the robot will do nothing

The robot was able to complete this task without any issues. Because of the way we designed our speed function, the robot would move forward more quickly than backwards because the maximum distance the robot could be from the object was much greater than the minimum distance the robot could be from the object.

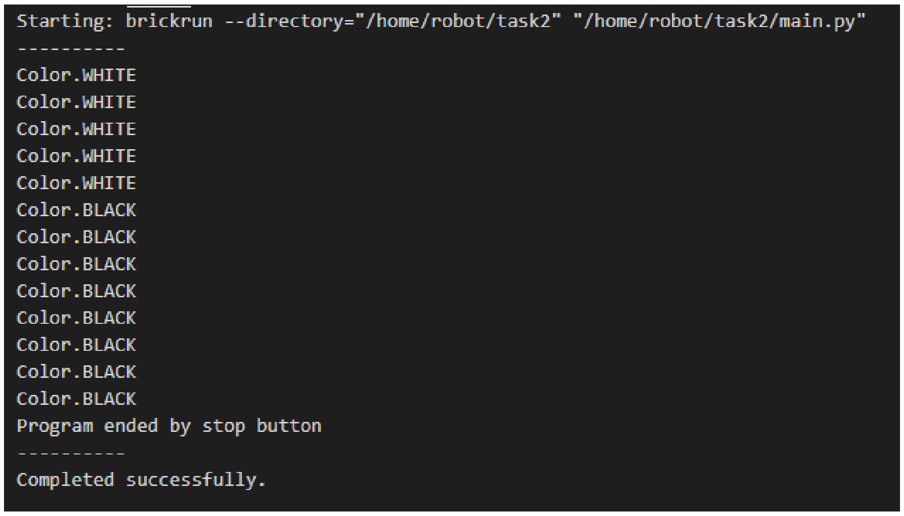
Two naïve observers were asked to watch the robot complete the task and describe what they thought the robot was doing. Their responses are shown in Table 1 below. The first observer said that the robot “Follows the object.” The second said that it “looks like it’s getting pushed and pulled by the object.”

Naïve Observers Descriptions of the robot completing the task	
Observer 1	“It follows the object.”
Observer 2	“It looks like it’s getting pushed and pulled by the object.”

Table 1: Naïve Observers Descriptions

Task 2 (Color Sensor Line Following – FSM)

Like in task 1, we started task 2 with a subtask of implementing a simple function to print out the sensor values from the color sensor. As shown below in Image 7, the output of the color sensor was the color of the ground directly under the color sensor. For our setup, the carpeted on the ground was reported as black by the sensor and the color of the masking tape we used to mark our line was reported as white.



```
Starting: brickrun --directory="/home/robot/task2" "/home/robot/task2/main.py"
----------
Color.WHITE
Color.WHITE
Color.WHITE
Color.WHITE
Color.WHITE
Color.BLACK
Color.BLACK
Color.BLACK
Color.BLACK
Color.BLACK
Color.BLACK
Color.BLACK
Color.BLACK
Program ended by stop button
----------
Completed successfully.
```

Image 7: Output of the function testing the color sensor running on the robot

Shown below in Images 8 and 9 are screenshots from the video taken of the robot completing task 2. The robot was able to follow the line we created successfully. In Image 8, the robot is following the tape line on the ground. In Image 9, the robot is initially rotating about 140 degrees left to try to find the line, and then rotating back to the right once it fails to find the line to the left.

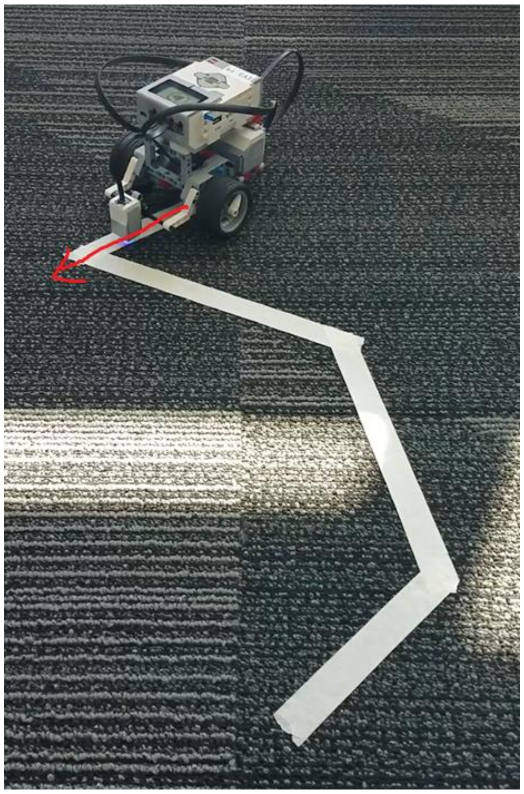


Image 8: Robot following the line (on-path)

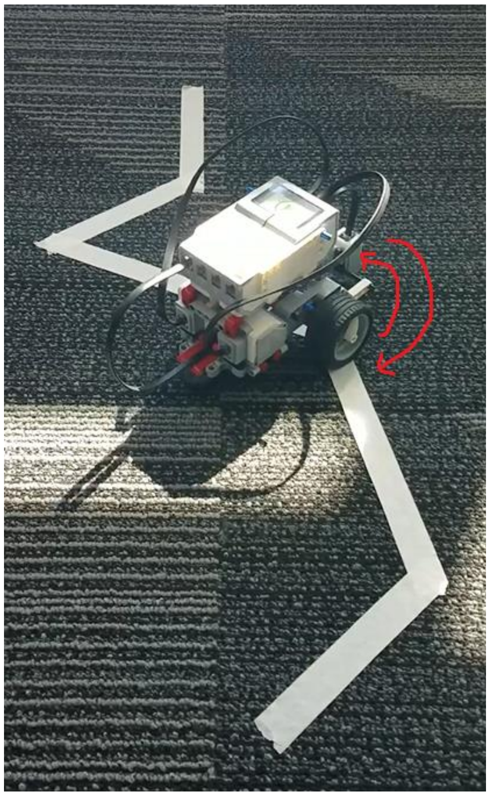


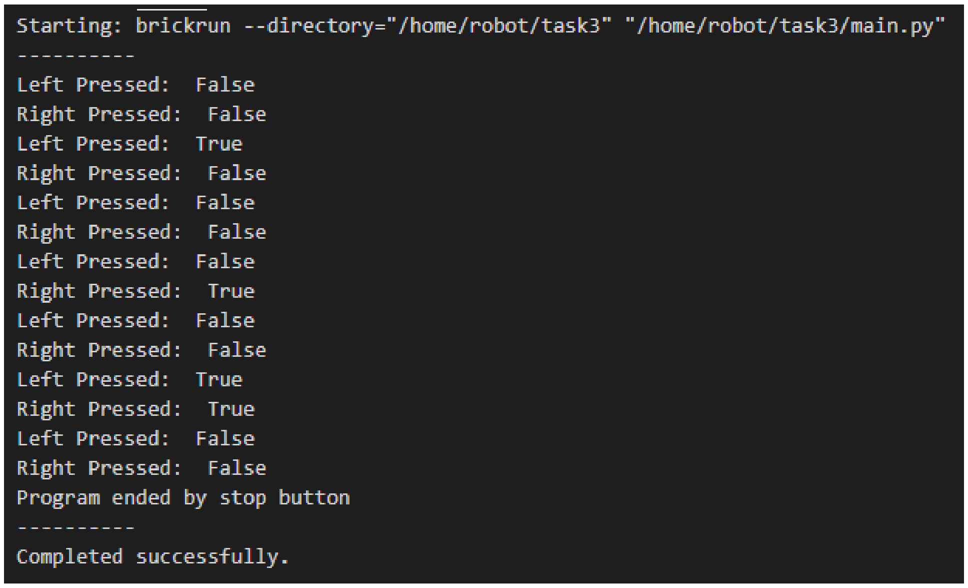
Image 9: Robot rotating to find the line (off-path)

For task 2, when the robot would complete a turn, it needed to turn several small times to become oriented completely straight to the new line. We were able to make the robot more efficient by recording direction of the last turn. Therefore, the robot wouldn't have to make multiple large turns to the right if it turned left multiple times in a row. Execution of a turn was only slow when the robot had to turn in the opposite direction than it turned last. This addition was able to cut the execution time of task 2 almost in half.

While the robot succeeded on the path, we created in task 2, the robot may not perfectly follow all paths you could possibly create. Because of the imperfections in the robot's movements and the width of the tape, when we rotate the initial direction to find a path, we only rotate approximately 140 degrees which takes about 3.5 seconds. If we rotated further, we sometimes ran into issues of the robot turning too far and finding the line behind it. Therefore, this means that there is a possibility that the path to follow had a turn that was greater than 140 degrees, the robot might not turn far enough in the correct direction to find the line. Then it would turn the opposite direction and end up finding the line behind it. There it would start to move in the wrong direction.

Task 3 (Wall Following - FSM)

Like in tasks 1 and 2, we started task 3 with a subtask of implementing a simple function to print out the sensor values from the touch sensor. As shown below in Image 10, the output of the touch sensor was a boolean value indicating if the sensor was pressed or not. The sensor only needed to be pushed approximately 50% of the way in before the sensor would read True.



```
Starting: brickrun --directory="/home/robot/task3" "/home/robot/task3/main.py"
----------
Left Pressed:  False
Right Pressed:  False
Left Pressed:  True
Right Pressed:  False
Left Pressed:  False
Right Pressed:  False
Left Pressed:  False
Right Pressed:  True
Left Pressed:  False
Right Pressed:  False
Left Pressed:  True
Right Pressed:  True
Left Pressed:  False
Right Pressed:  False
Program ended by stop button
----------
Completed successfully.
```

Image 10: Output of the function testing the touch sensor running on the robot

Shown below in Images 11 through 15 are screenshots from the timelapse video taken of the robot completing task 3. In Image 11, the robot is shown moving straight adjacent to the wall. In Image 12 the robot is shown after turning 90 degrees right and then moving forward and contacting the wall with the right sensor. The robot then backs up and turns back to the left. In Image 13 the robot is shown after it has moved passed the corner of the wall and turned right. In Image 14 the robot is shown having moved forward until it runs straight into a wall. In Image 15 the robot is shown after backing away from the wall directly in front of it and rotating left.

The robot was able to follow the wall in the video successfully, but not while staying perfectly parallel. Errors were found in the imperfect ability of the robot to rotate exactly 90 degrees. Collisions with the wall also cause the robot to move slightly. The longer the robot attempts to follow any wall, the greater the chance of the robot failing to follow the wall would occur.

The functions also use specific values for the robot to move on the carpet in the video. If the robot were to be placed on a hardwood surface, then the values would need to be adjusted. The robot also assumes that the wall only has 90-degree corners. If the wall the robot attempts to follow is curved or turns at an angle other than 90 degrees, the robot will fail to follow the wall. As mentioned in the methods section, the robot also assumes the wall is to its right when starting. If placed with the wall adjacent to the left side of the robot, the robot will be unable to follow the wall.

It should also be mentioned that having two touch sensors on the front of the robot is not the best setup to make the robot follow a way. In this setup, the robot must continuously turn right and check to make sure it is still following the wall which takes a lot of time. More time is spent checking for the wall than is moving adjacent to the wall. A better method for this task would include an ultrasonic sensor on the side of the robot which would keep the robot at a set distance from the wall. When the sensor sensed that the wall was no longer there, it would make the robot turn right to find it again. The two front touch sensors would still be useful to identify when the robot runs into a wall ahead of it.

Image 11: Robot moving forward adjacent to the wall

Image 12: Robot finding the wall still next to it and then backing up

Image 13: Robot turning around the corner of the wall

Image 14: Robot finding a wall directly in front of it and then backing up

Image 15: Robot turning left after finding the wall ahead

Discussion

1. The first naïve observer said that the robot "follows the object" which implies that this person thought the robot was reacting to the movement of the object, which is correct. The second naïve observer said that it "looks like it's getting pushed and pulled by the object", implying that the object is controlling the robot which is not correct. The robot is reacting to the object but the object is not controlling it.

2. To decide about how to breakdown the states in the FSM for task 2 and 3, we broke them into logical chunks. The high level states were the overarching actions that would be taken, like being on-path, checking if there was a wall to the right. While the smaller substates were singular actions, like rotating left, or moving forward.

3. Compared to following paths in Lab 1 using open loop control, the closed loop control has some pros and cons. The pros are that the robot is more accurate in the closed loop control because it has sensors so it stays on course and can adjust. However, the sensors need to be the correct sensors for the job. In task 3, simply having 2 touch sensors on the front of the robot was not an effective strategy for following a wall. The cons are that this approach was much more complex than open loop control. We had to deal with constantly reading data from sensors and moving based on that data instead of simply telling the robot to move forward 25cm.

4. Pros and cons of reactive control vs the FSM were also similar. Reactive control was very simple with our code being very short. The code for the FSMs was much longer and more complex and therefore tougher to get working correctly. However, the reactive control is more limited in what it can accomplish. The FSM divides logic into chunks that make sense and then you order them depending on how you need to complete the task.

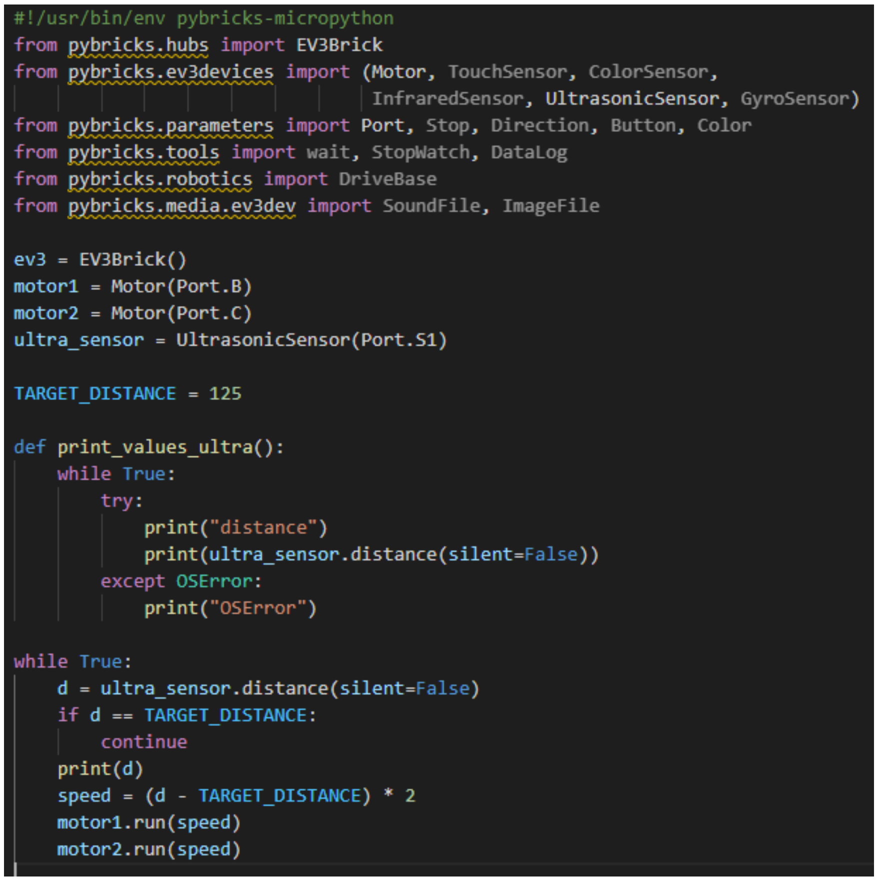
Supplementary Materials

Contained in this section are code screenshots for tasks 0 through 3. Videos of tasks 1 through 3 are also included in the submission.

Python Code

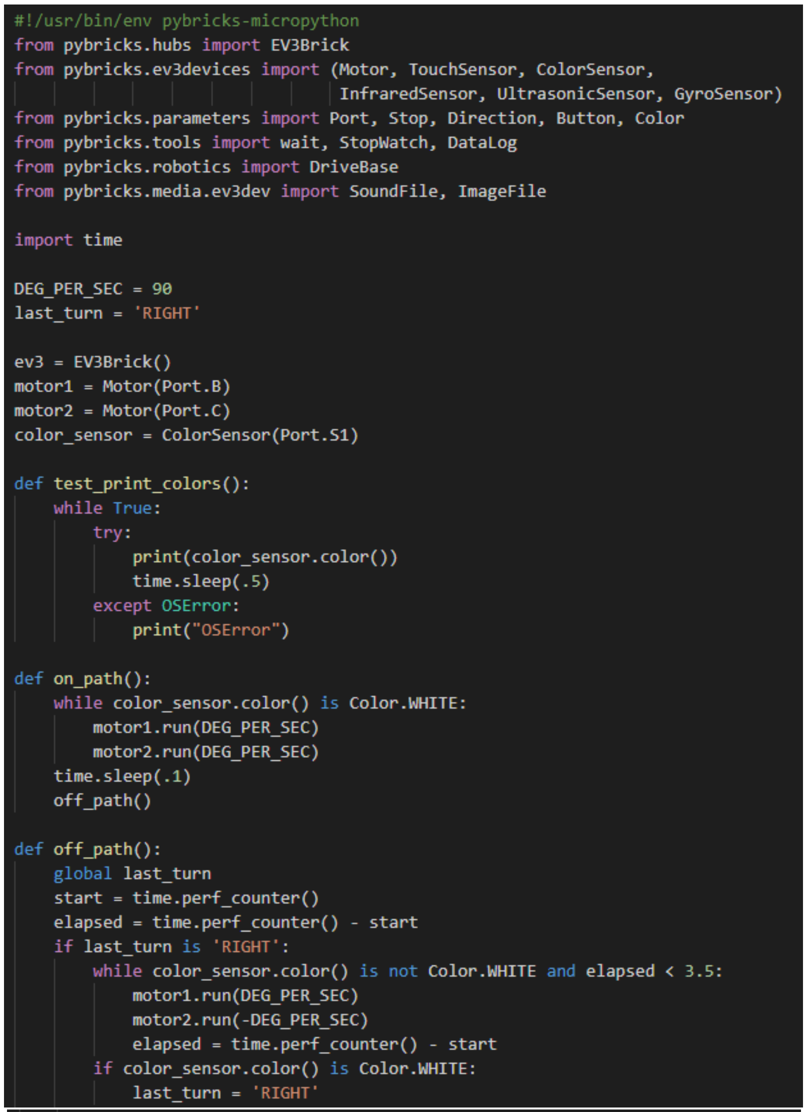
The python files included in the submission are task1_main.py which includes the code for tasks 0 and 1, task2_main.py, and task3_main.py. Screenshots of the python code are shown below.

Task 1 Code - Below is the code implementing our speed function to follow the object in front of the robot to a target distance.

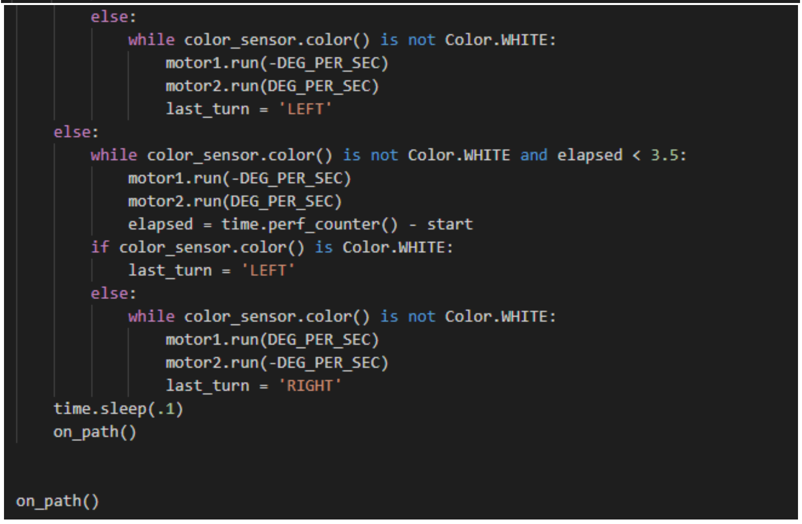


```python
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor, GyroSensor)
from pybricks.parameters import Port, Stop, Direction, Button, Color
from pybricks.tools import wait, StopWatch, DataLog
from pybricks.robotics import DriveBase
from pybricks.media.ev3dev import SoundFile, ImageFile

ev3 = EV3Brick()
motor1 = Motor(Port.B)
motor2 = Motor(Port.C)
ultra_sensor = UltrasonicSensor(Port.S1)

TARGET_DISTANCE = 125

def print_values_ultra():
    while True:
        try:
            print("distance")
            print(ultra_sensor.distance(silent=False))
        except OSError:
            print("OSError")

while True:
    d = ultra_sensor.distance(silent=False)
    if d == TARGET_DISTANCE:
        continue
    print(d)
    speed = (d - TARGET_DISTANCE) * 2
    motor1.run(speed)
    motor2.run(speed)
```

Task 2 Code - Below is the code implementing our test method for the color sensor as well as our solution to follow a line with the on-path and off-path methods.

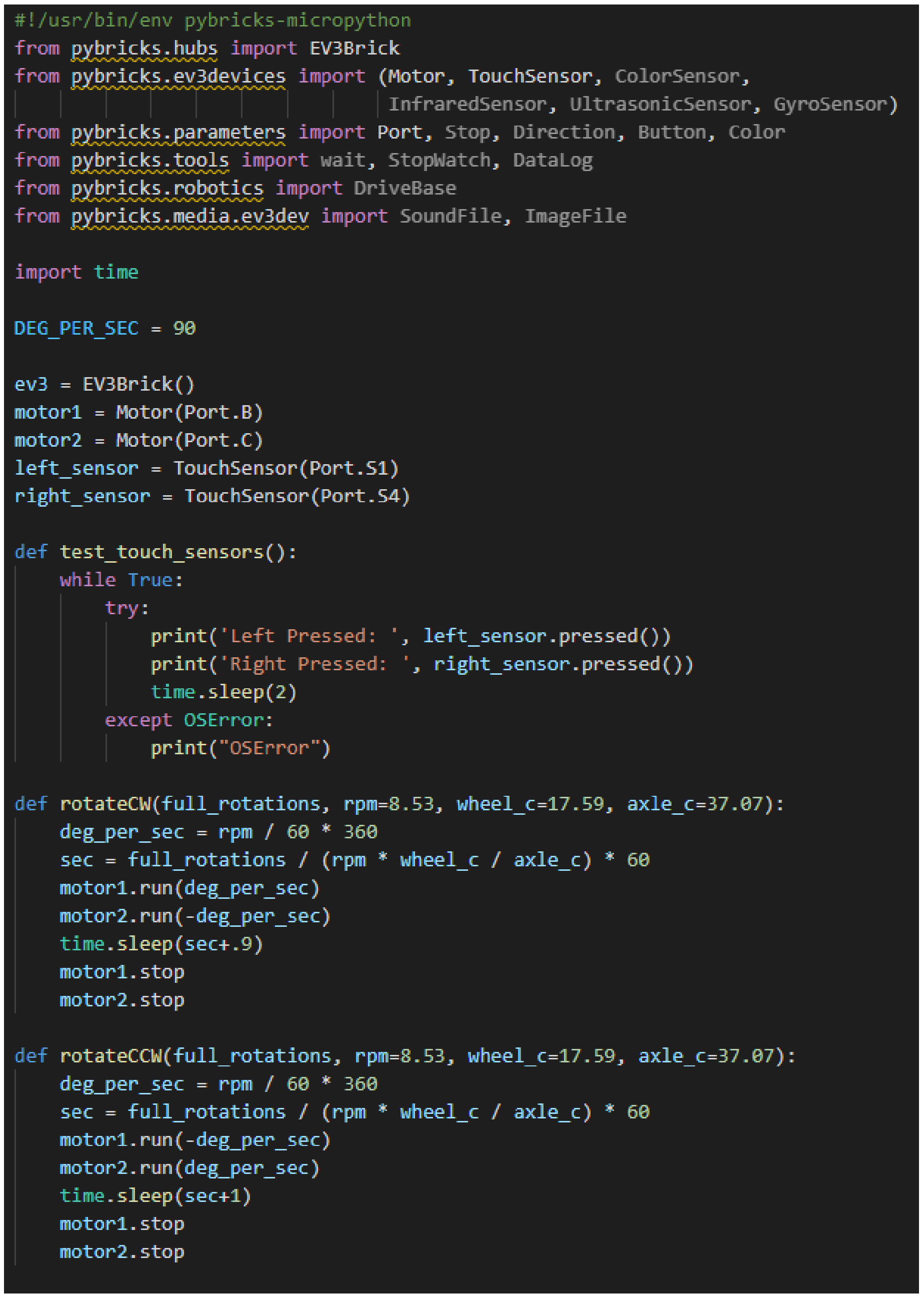


```python
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor, GyroSensor)
from pybricks.parameters import Port, Stop, Direction, Button, Color
from pybricks.tools import wait, StopWatch, DataLog
from pybricks.robotics import DriveBase
from pybricks.media.ev3dev import SoundFile, ImageFile

import time

DEG_PER_SEC = 90
last_turn = 'RIGHT'

ev3 = EV3Brick()
motor1 = Motor(Port.B)
motor2 = Motor(Port.C)
color_sensor = ColorSensor(Port.S1)

def test_print_colors():
    while True:
        try:
            print(color_sensor.color())
            time.sleep(.5)
        except OSError:
            print("OSError")

def on_path():
    while color_sensor.color() is Color.WHITE:
        motor1.run(DEG_PER_SEC)
        motor2.run(DEG_PER_SEC)
    time.sleep(.1)
    off_path()

def off_path():
    global last_turn
    start = time.perf_counter()
    elapsed = time.perf_counter() - start
    if last_turn is 'RIGHT':
        while color_sensor.color() is not Color.WHITE and elapsed < 3.5:
            motor1.run(DEG_PER_SEC)
            motor2.run(-DEG_PER_SEC)
            elapsed = time.perf_counter() - start
        if color_sensor.color() is Color.WHITE:
            last_turn = 'RIGHT'
```

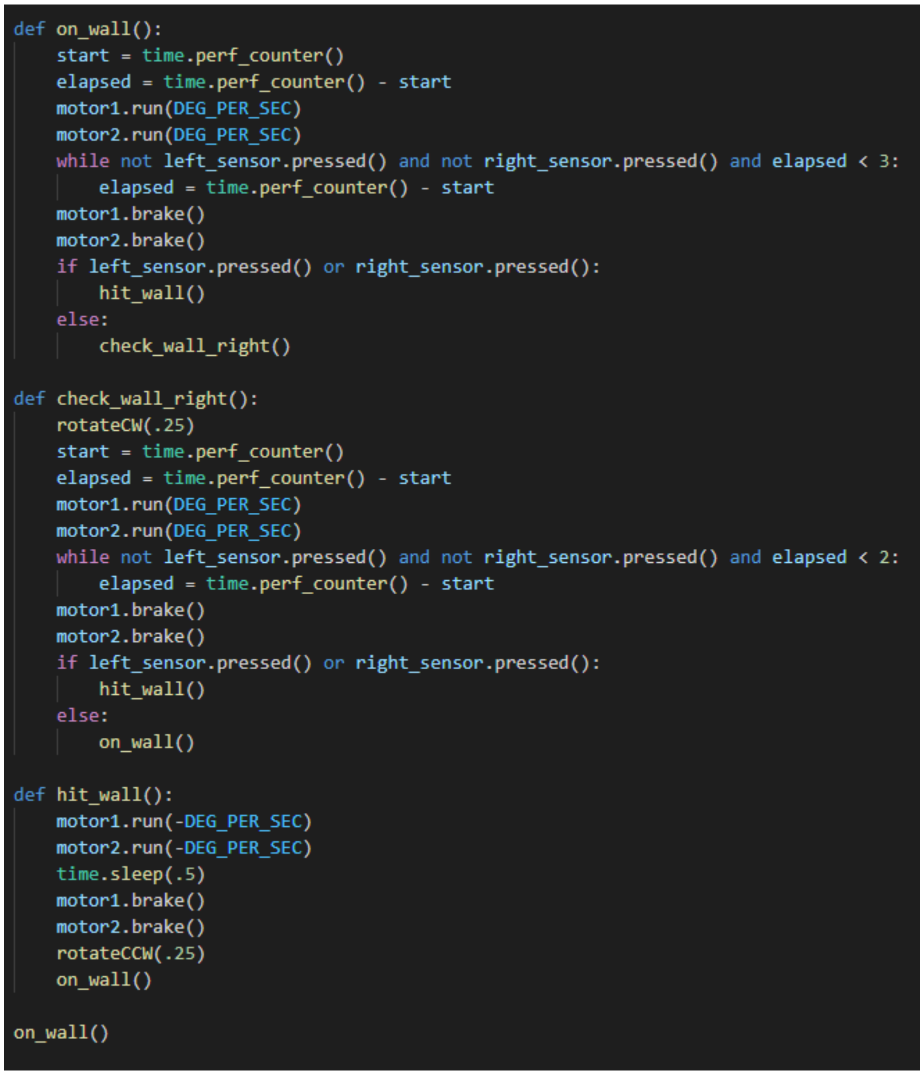


```python
        else:
            while color_sensor.color() is not Color.WHITE:
                motor1.run(-DEG_PER_SEC)
                motor2.run(DEG_PER_SEC)
                last_turn = 'LEFT'
    else:
        while color_sensor.color() is not Color.WHITE and elapsed < 3.5:
            motor1.run(-DEG_PER_SEC)
            motor2.run(DEG_PER_SEC)
            elapsed = time.perf_counter() - start
        if color_sensor.color() is Color.WHITE:
            last_turn = 'LEFT'
        else:
            while color_sensor.color() is not Color.WHITE:
                motor1.run(DEG_PER_SEC)
                motor2.run(-DEG_PER_SEC)
                last_turn = 'RIGHT'
    time.sleep(.1)
    on_path()


on_path()
```

Task 3 Code - Below is the code implementing our test method for the touch sensors as well as our solution to following a wall.



```python
#!/usr/bin/env pybricks-micropython
from pybricks.hubs import EV3Brick
from pybricks.ev3devices import (Motor, TouchSensor, ColorSensor,
                                 InfraredSensor, UltrasonicSensor, GyroSensor)
from pybricks.parameters import Port, Stop, Direction, Button, Color
from pybricks.tools import wait, StopWatch, DataLog
from pybricks.robotics import DriveBase
from pybricks.media.ev3dev import SoundFile, ImageFile

import time

DEG_PER_SEC = 90

ev3 = EV3Brick()
motor1 = Motor(Port.B)
motor2 = Motor(Port.C)
left_sensor = TouchSensor(Port.S1)
right_sensor = TouchSensor(Port.S4)

def test_touch_sensors():
    while True:
        try:
            print('Left Pressed: ', left_sensor.pressed())
            print('Right Pressed: ', right_sensor.pressed())
            time.sleep(2)
        except OSError:
            print("OSError")

def rotateCW(full_rotations, rpm=8.53, wheel_c=17.59, axle_c=37.07):
    deg_per_sec = rpm / 60 * 360
    sec = full_rotations / (rpm * wheel_c / axle_c) * 60
    motor1.run(deg_per_sec)
    motor2.run(-deg_per_sec)
    time.sleep(sec+.9)
    motor1.stop
    motor2.stop

def rotateCCW(full_rotations, rpm=8.53, wheel_c=17.59, axle_c=37.07):
    deg_per_sec = rpm / 60 * 360
    sec = full_rotations / (rpm * wheel_c / axle_c) * 60
    motor1.run(-deg_per_sec)
    motor2.run(deg_per_sec)
    time.sleep(sec+1)
    motor1.stop
    motor2.stop
```



```
def on_wall():
    start = time.perf_counter()
    elapsed = time.perf_counter() - start
    motor1.run(DEG_PER_SEC)
    motor2.run(DEG_PER_SEC)
    while not left_sensor.pressed() and not right_sensor.pressed() and elapsed < 3:
        elapsed = time.perf_counter() - start
    motor1.brake()
    motor2.brake()
    if left_sensor.pressed() or right_sensor.pressed():
        hit_wall()
    else:
        check_wall_right()

def check_wall_right():
    rotateCW(.25)
    start = time.perf_counter()
    elapsed = time.perf_counter() - start
    motor1.run(DEG_PER_SEC)
    motor2.run(DEG_PER_SEC)
    while not left_sensor.pressed() and not right_sensor.pressed() and elapsed < 2:
        elapsed = time.perf_counter() - start
    motor1.brake()
    motor2.brake()
    if left_sensor.pressed() or right_sensor.pressed():
        hit_wall()
    else:
        on_wall()

def hit_wall():
    motor1.run(-DEG_PER_SEC)
    motor2.run(-DEG_PER_SEC)
    time.sleep(.5)
    motor1.brake()
    motor2.brake()
    rotateCCW(.25)
    on_wall()

on_wall()
```

Videos

Task 1 video: This video shows the robot moving forwards and backwards in order to attempt to keep a set distance to the object that we were moving by hand in front of it.

Task 2 video: This video shows the robot following the jagged tape line on the ground.

Task 3 timelapse video: This video shows the robot following the wall to its right. This video was taken as a timelapse because the robot took a long time to perform the actions in the video due to it moving at a slow speed and constantly having to check if the wall was next to it.