The Engineering Design Notebook

Project Four

Customer Overview

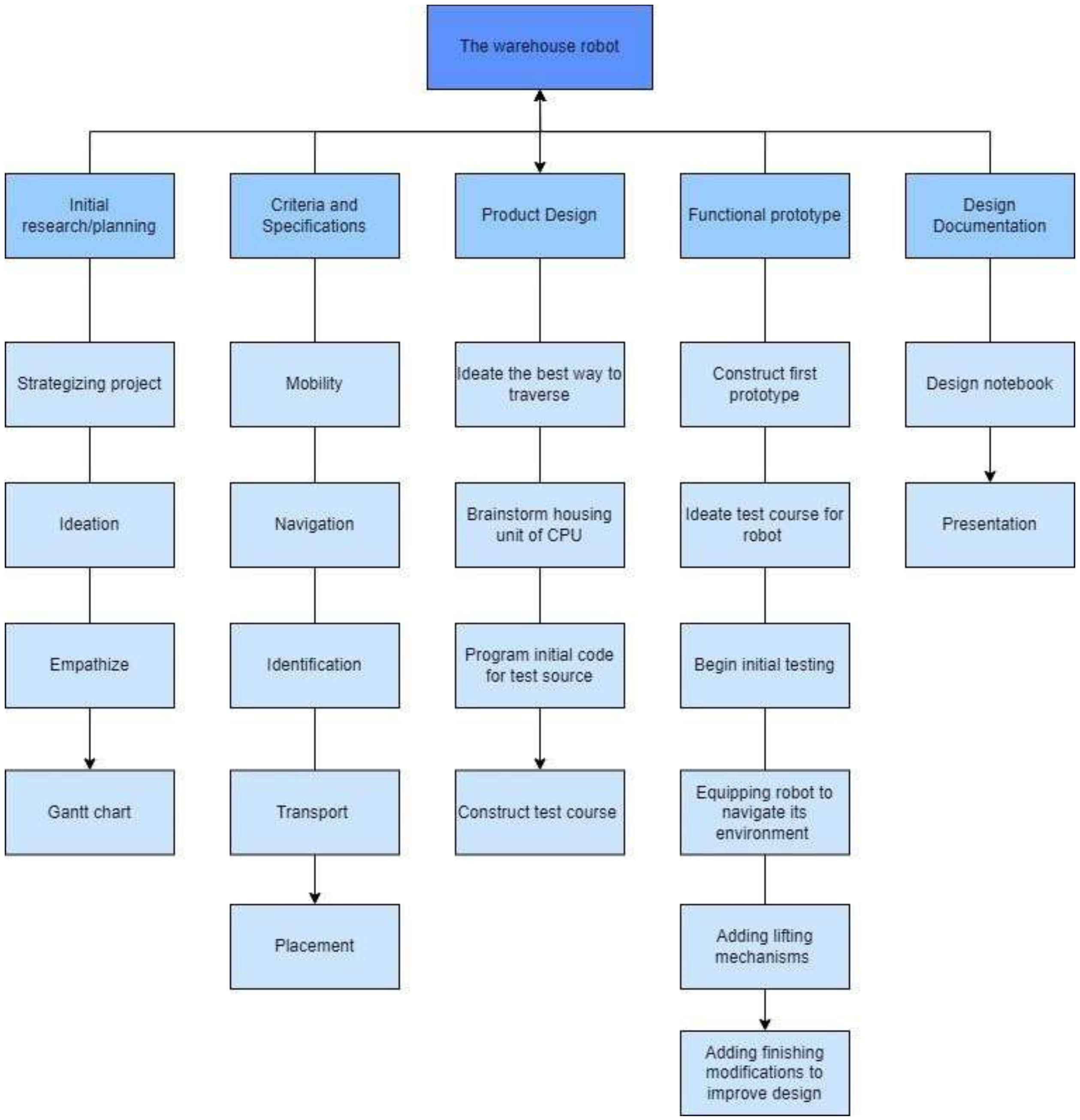
Buy More, a home entertainment, computer, and gaming store, has issued a Request for Proposal (RFP) for an autonomous robot, the Autonomous Product Retriever (APR), to be used in their warehouse to retrieve and deliver boxes to an order fulfillment area. The APR system is intended to address the labor shortage at the warehouse caused by the company's decision to close its physical stores and focus on online sales. The APR will eventually interface with Buy More's current inventory system, Where's the Ware (WTW), to retrieve items for order fulfillment. Our team has been tasked to design this APR for Buy More.

Table of Contents

Project Management Information	5
Empathize	7
Major Sub-Components	7
Define & Ideate	8
Morphological Chart	8
Decision Matrix (For final prototype)	9
Prototype building	10
Codes	12
Initial Testing and Ideation	14
Revised Codes	17
First Project Demonstration	19
Locomotion Test Plan (Ideation)	20
Remodeling	24
Prototype 2.0	25
Prototype 2.0 - Improvements Were Made	26
Prototype 3.0- Ideation	26
Introducing Axiom I	27
Behind the scenes (Final Code)	28
Axiom I - On the Demo	35
Future Enhancement	36

Project Management Information

1. Work Break-Down Structure


The warehouse robot
Initial research/planning
Strategizing project
Ideation
Empathize
Gantt chart
Criteria and Specifications
Mobility
Navigation
Identification
Transport
Placement
Product Design
Ideate the best way to traverse
Brainstorm housing unit of CPU
Program initial code for test source
Construct test course
Functional prototype
Construct first prototype
Ideate test course for robot
Begin initial testing
Equipping robot to navigate its environment
Adding lifting mechanisms
Adding finishing modifications to improve design
Design Documentation
Design notebook
Presentation

Empathize

The proposed Autonomous Product Retriever (APR) system requires a sturdy and modular chassis to accommodate changes in design plans and requirements. The chassis will house the gyroscopic sensor, the NXT brick, and potentially the ultrasonic sensor. The robot arm must be strong and powerful enough to lift weights as high as 250g minimum and could potentially host the color sensors for scanning the barcodes and the ultrasonic sensor. The sensors must be positioned accurately to collect accurate data (reading the barcode), and the code is crucial for stringing together all the components to create the desired APR, including instructions for navigating corridors and picking up and dropping items based on barcode information. With these components in place, the APR should be able to fulfill the demands made by Buy More to automate their warehouse.

Major Sub-Components

Mobility – The robot must be able to move accurately without colliding with the environment or straying far from the intended path.

Navigation - The robot must be able to navigate the warehouse and create a map of its environment.

Identification – The robot must be able to easily distinguish the different cargo types and place them according to the barcode.

Transport – The robot must have the ability to properly secure any cargo and avoid dropping it.

Placement – The robot must be able to place transported cargo accurately in its intended location consistently.

Define & Ideate

Specifications based on Goals & Criteria

Dimensions - The robot should be compact and able to navigate through narrow aisles in the warehouse.

Mobility – The robot should traverse smoothly on surfaces commonly found in a warehouse, such as concrete, and tiles. It should also be able to climb over small obstacles and navigate tight turns.

Identification – By using a variety of sensors the robot must be able to distinguish and locate cargo types with 90% accuracy.

Transport – The robot should have arms and grippers to hold and move the boxes around without dropping them.

Sensors - The robot should be equipped with sensors to allow it to navigate the warehouse and locate different boxes.

Computing - The robot should be able to perform real-time navigation and mapping.

Morphological Chart

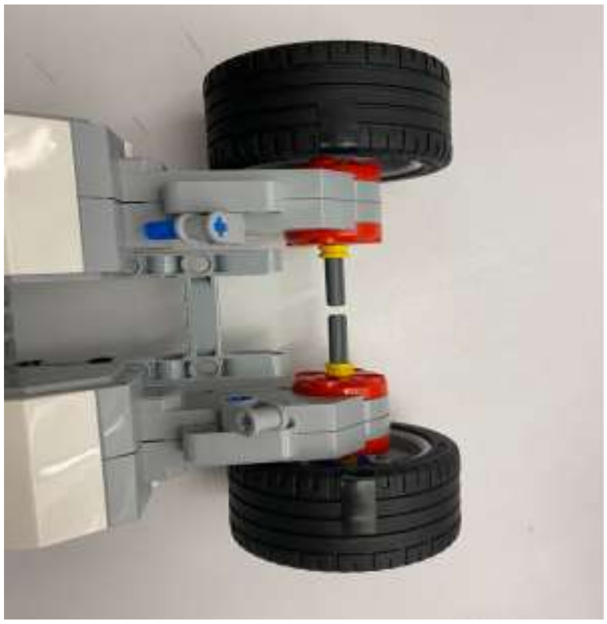
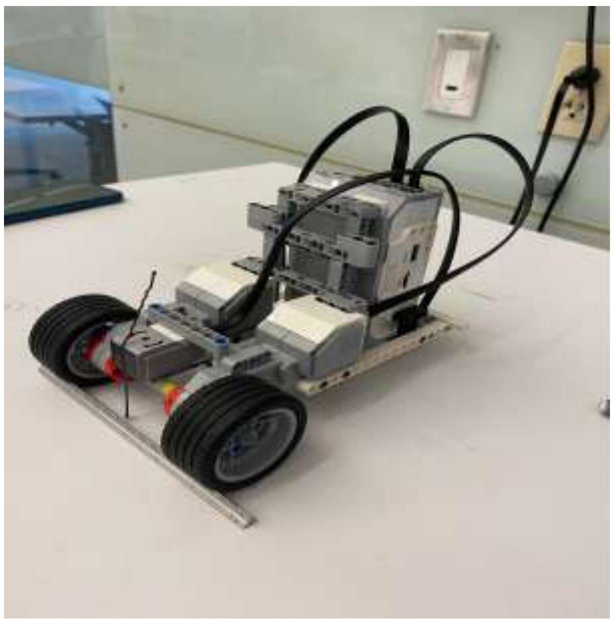
Function	**Option 1**	**Option 2**	**Option 3**
Navigation (hardware)	Two wheels and a ball	Two wheels and a small wheel	Four wheel tank design
Navigation (software)	Coordinate map	Linear programming	Depth First Searching
Barcode scanning	Horizontal on the side	Horizontal front	Vertical front
Lifting mechanism	Magnetic lift	Hydraulic lift	Forklift
Object Avoidance	Placed at the bottom center	Placed on the bottom side	Placed at top center

Decision Matrix (For final prototype)

All the criteria mentioned below are rated out of 5.

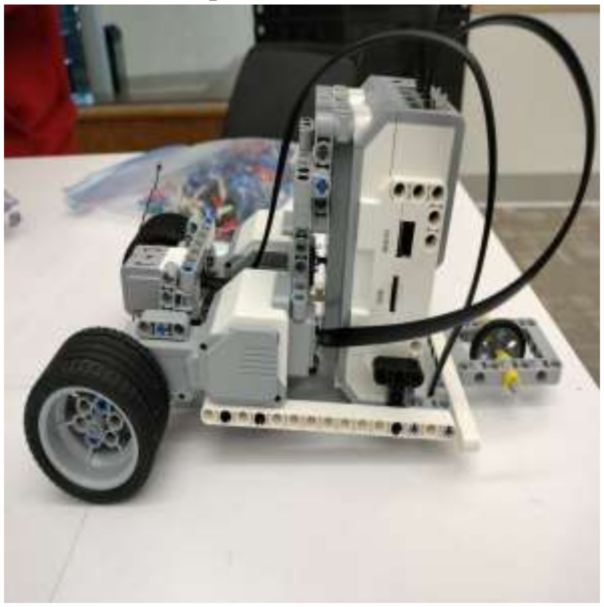
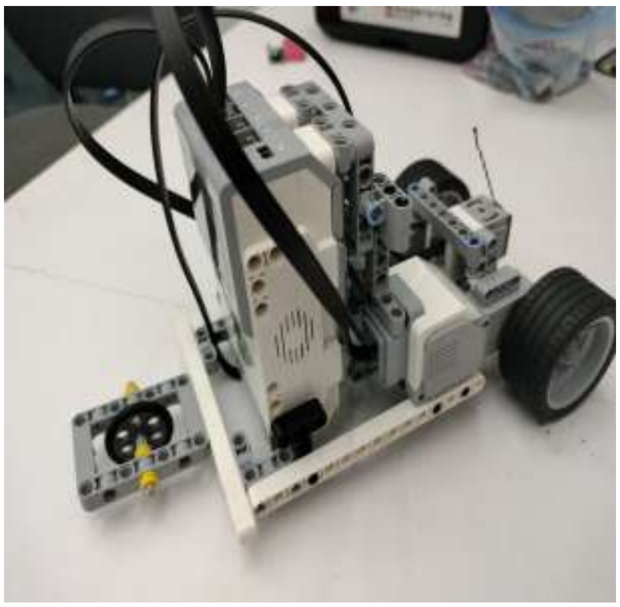
Function	**Option 1**	**Option 2**	**Final prototype**
Navigation (hardware)	Two wheels and a small wheel(2)	Two wheels and a ball(3)	Four wheel tank design(5)
Navigation (software)	Depth First searching(2)	Linear programming(2)	Coordinate map(5)
Barcode scanning	Horizontal on the side(2)	Horizontal front (3)	Vertical front (4)
Lifting mechanism	Magnetic lift (3)	Hydraulic lift (2)	Forklift (4)
Object Avoidance	Placed at the bottom center(3)	Placed at the top (2)	Placed on the bottom side(4)
Total	12	12	22

Prototype building

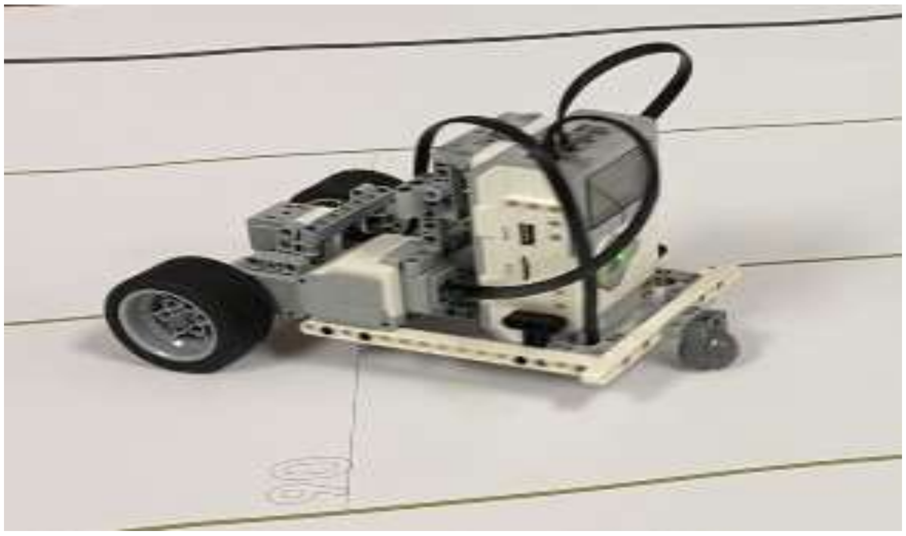



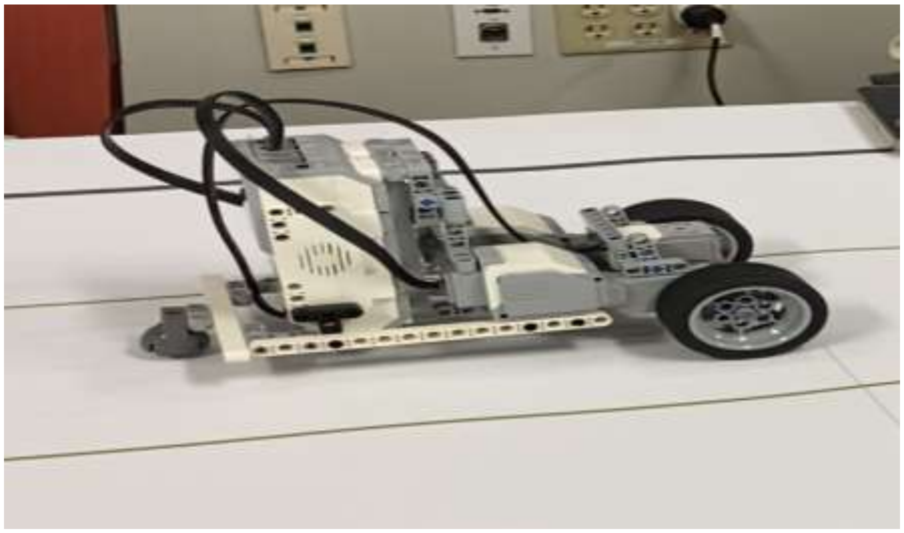
For the first prototype, two tires with motors were placed on the anterior part of the robot while the EV3 brick was placed on the posterior part. The tires on the front, twisted and turned to navigate the direction of the robot. However, the bulk of the weight of the robot was on the posterior part of the robot, and in order to make the robot turn, a mechanism had to be deployed and that is where we had to be more creative.

First, we came up with a wheel mechanism:

The problem with the wheel mechanism was it hindered turning by skidding perpendicular instead of facilitating the turn. We needed a freely rotating mechanism for the robot to turn successfully with minimum error. Then, we attached a marble on the back.

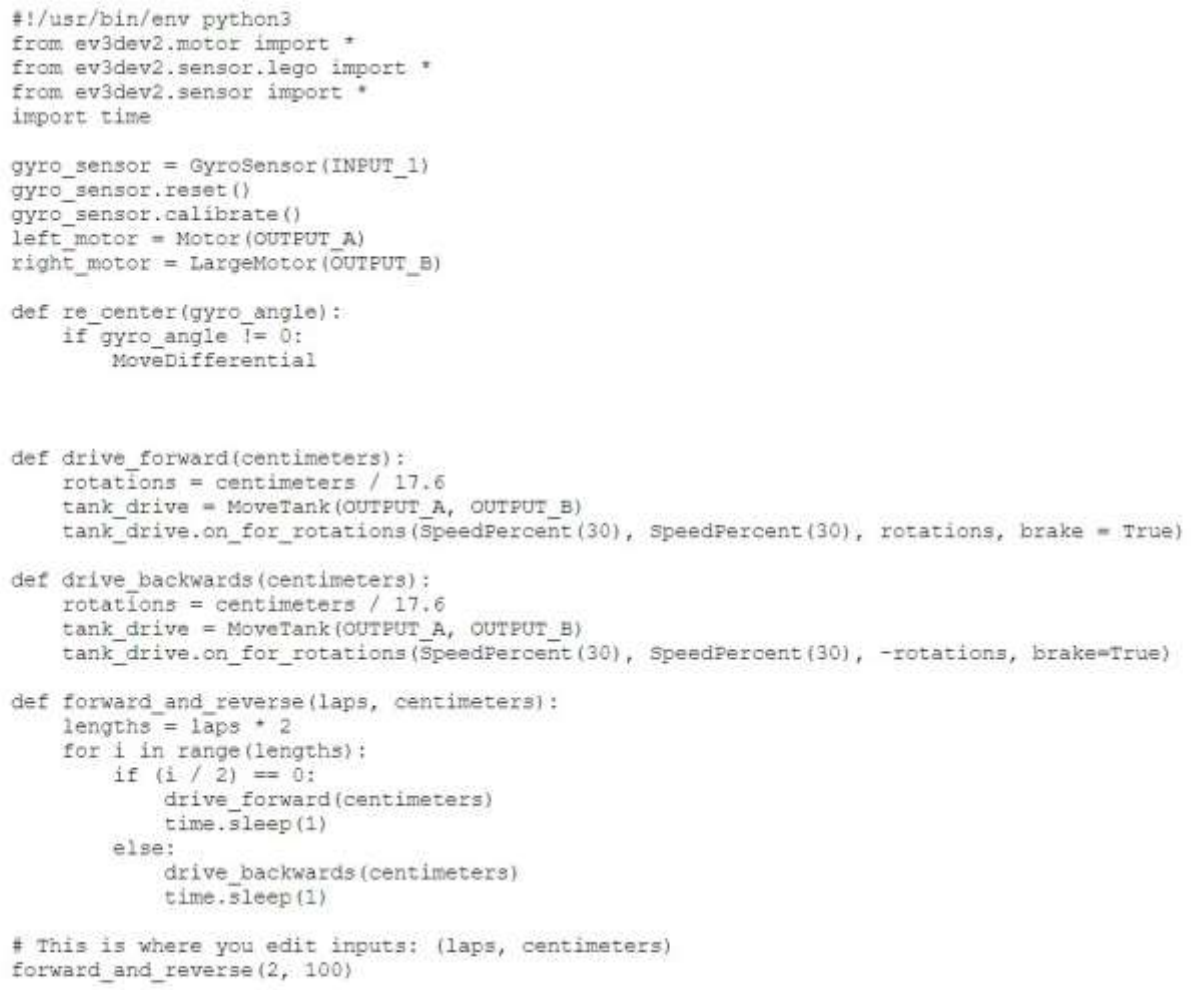




We saw significant improvement in the movement, especially in turning the robot after adding the marble mechanism, so this model was taken for the first project demonstration.

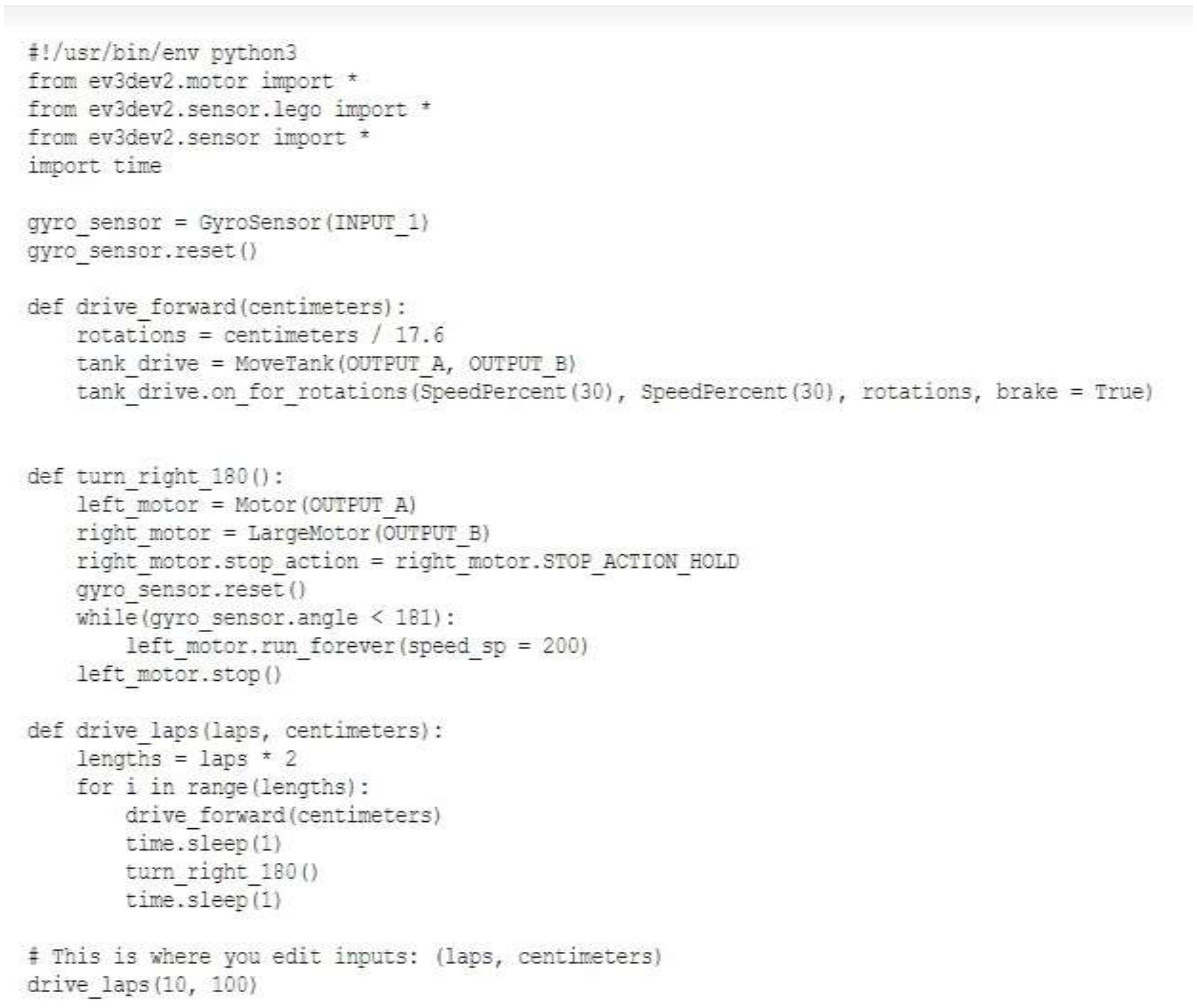
Codes

Codes for subtask 1a(To move forward and come back)



```
#!/usr/bin/env python3
from ev3dev2.motor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor import *
import time

gyro_sensor = GyroSensor(INPUT_1)
gyro_sensor.reset()
gyro_sensor.calibrate()
left_motor = Motor(OUTPUT_A)
right_motor = LargeMotor(OUTPUT_B)

def re_center(gyro_angle):
    if gyro_angle != 0:
        MoveDifferential



def drive_forward(centimeters):
    rotations = centimeters / 17.6
    tank_drive = MoveTank(OUTPUT_A, OUTPUT_B)
    tank_drive.on_for_rotations(SpeedPercent(30), SpeedPercent(30), rotations, brake = True)

def drive_backwards(centimeters):
    rotations = centimeters / 17.6
    tank_drive = MoveTank(OUTPUT_A, OUTPUT_B)
    tank_drive.on_for_rotations(SpeedPercent(30), SpeedPercent(30), -rotations, brake=True)

def forward_and_reverse(laps, centimeters):
    lengths = laps * 2
    for i in range(lengths):
        if (i / 2) == 0:
            drive_forward(centimeters)
            time.sleep(1)
        else:
            drive_backwards(centimeters)
            time.sleep(1)

# This is where you edit inputs: (laps, centimeters)
forward_and_reverse(2, 100)
```

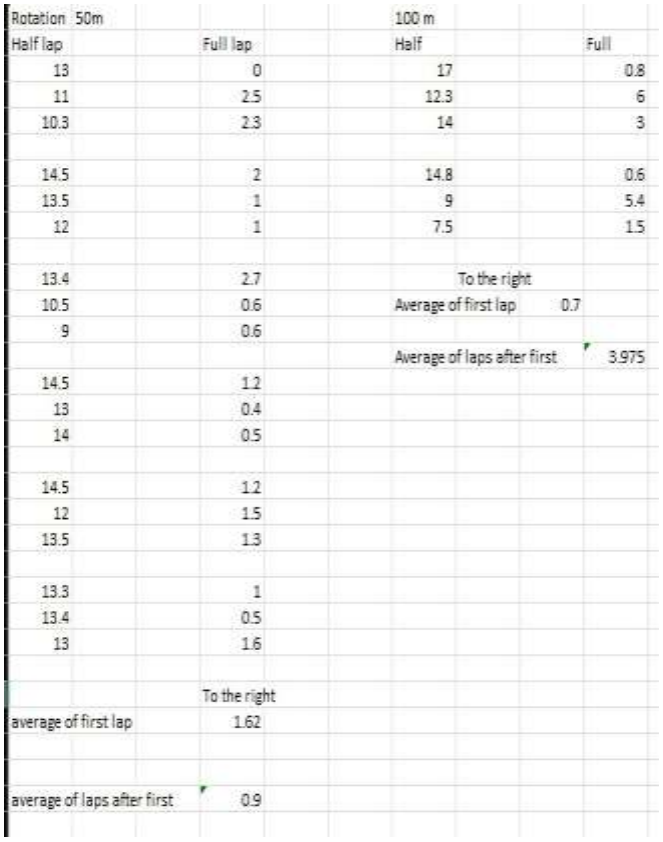
Codes for subtask 1b(To move forward and rotate the robot)



```
#!/usr/bin/env python3
from ev3dev2.motor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor import *
import time

gyro_sensor = GyroSensor(INPUT_1)
gyro_sensor.reset()

def drive_forward(centimeters):
    rotations = centimeters / 17.6
    tank_drive = MoveTank(OUTPUT_A, OUTPUT_B)
    tank_drive.on_for_rotations(SpeedPercent(30), SpeedPercent(30), rotations, brake = True)


def turn_right_180():
    left_motor = Motor(OUTPUT_A)
    right_motor = LargeMotor(OUTPUT_B)
    right_motor.stop_action = right_motor.STOP_ACTION_HOLD
    gyro_sensor.reset()
    while(gyro_sensor.angle < 181):
        left_motor.run_forever(speed_sp = 200)
    left_motor.stop()

def drive_laps(laps, centimeters):
    lengths = laps * 2
    for i in range(lengths):
        drive_forward(centimeters)
        time.sleep(1)
        turn_right_180()
        time.sleep(1)

# This is where you edit inputs: (laps, centimeters)
drive_laps(10, 100)
```

Initial Testing and Ideation

With the first test that we ran, the goal was to determine the error of the x and y coordinates based on the length traveled and the number of laps. To determine the values, we first attached a paper clip to the front middle of the robot as our measuring point. Next, for each trial run, we set up the wheels at the same point to ensure they were aligned at the beginning to reduce the error within the trials. We then used a series of lengths measuring the error generated on the x and y coordinates with half laps and while laps. To do this, we measured the distance the robot traveled from the location it started at.

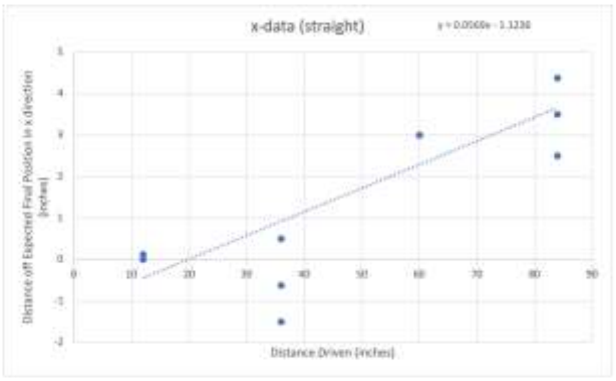


Rotation 50m		100 m		
Half lap	Full lap	Half		Full
13	0	17		0.8
11	2.5	12.3		6
10.3	2.3	14		3
14.5	2	14.8		0.6
13.5	1	9		5.4
12	1	7.5		1.5
13.4	2.7	To the right		
10.5	0.6	Average of first lap	0.7	
9	0.6			
		Average of laps after first		3.975
14.5	1.2			
13	0.4			
14	0.5			
14.5	1.2			
12	1.5			
13.5	1.3			
13.3	1			
13.4	0.5			
13	1.6			
	To the right			
average of first lap	1.62			
average of laps after first	0.9			

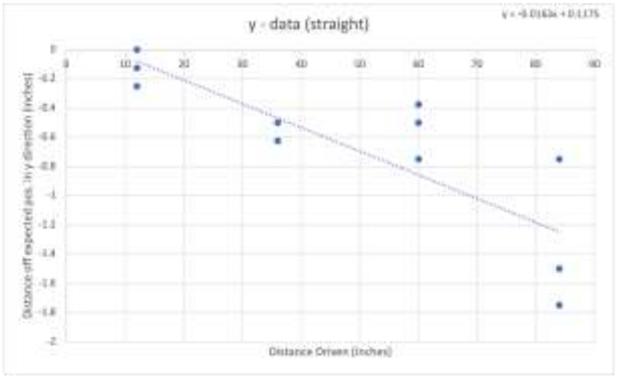
Locomotion test plan findings:

After conducting our locomotion functionality tests, we analyzed our data to create mathematical models that we could use to predict the error given the distance the robot has to travel. We used Excel to graph our x and y error data for the straight tests and turning tests separately. Using Excel, we then graphed a line of best fit using a linear equation to create our models. We ended up with **four** total models:

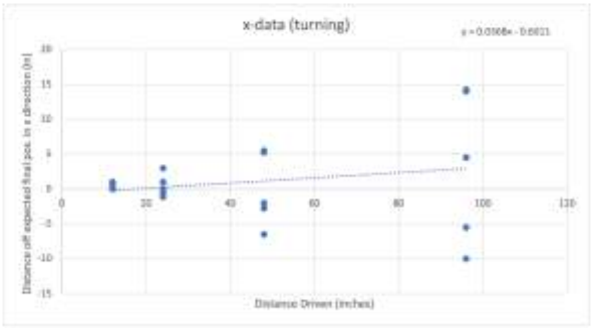
- Predicting x - error (based on straight data)



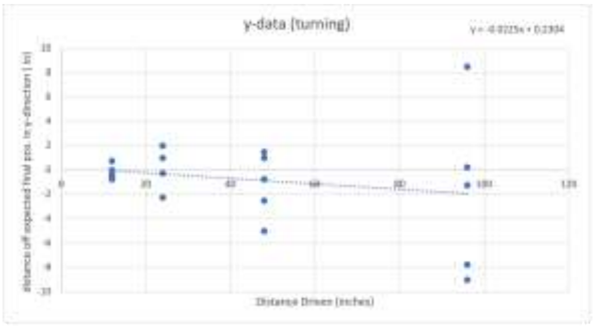
- Predicting y - error (based on straight data)



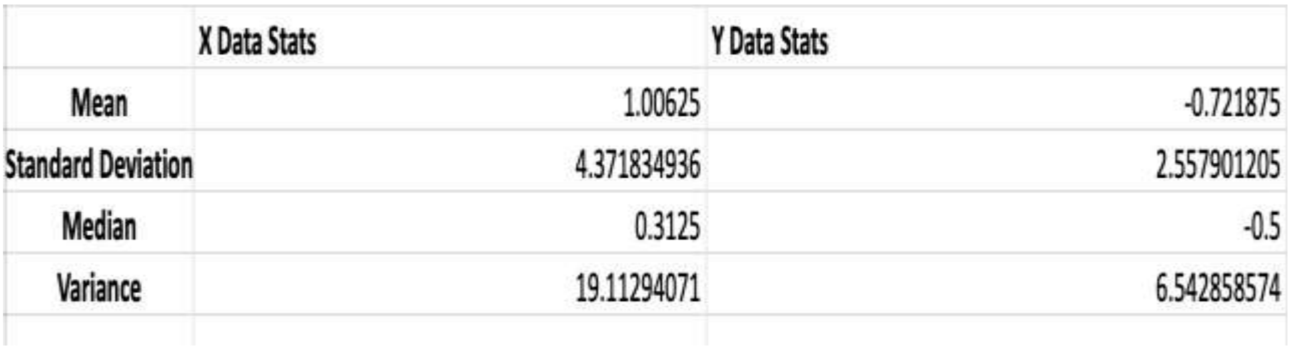
- Predicting x - error (based on turning data)



- Predicting y - error (based on turning data)



We also took the standard deviation, mean, and variance for the different tests:



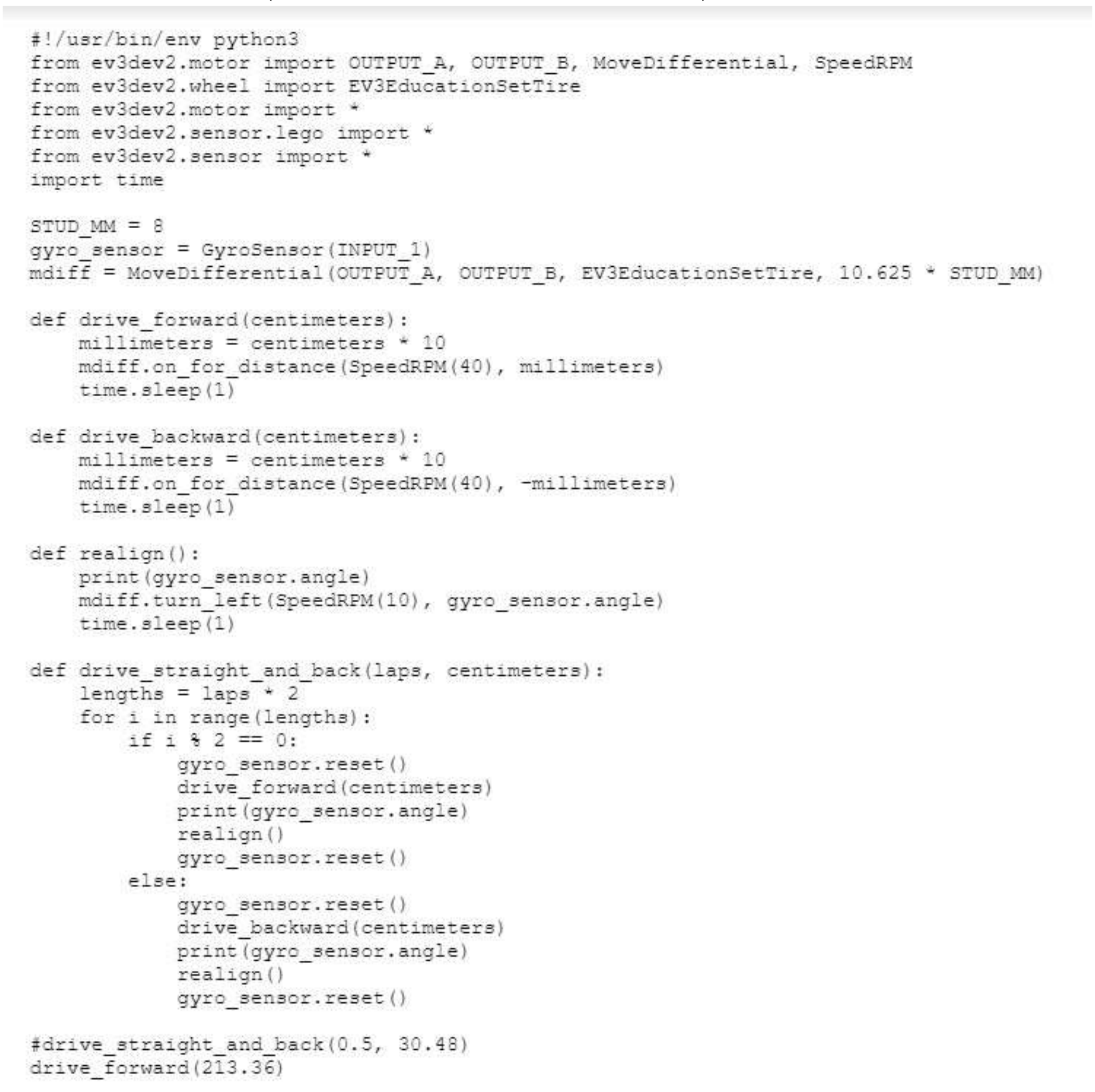
	X Data Stats	Y Data Stats
Mean	1.00625	-0.721875
Standard Deviation	4.371834936	2.557901205
Median	0.3125	-0.5
Variance	19.11294071	6.542858574

Using these statistics, we were able to conclude that our robot had a serious problem with consistency. Our data collected was neither precise nor accurate.

Revised Codes

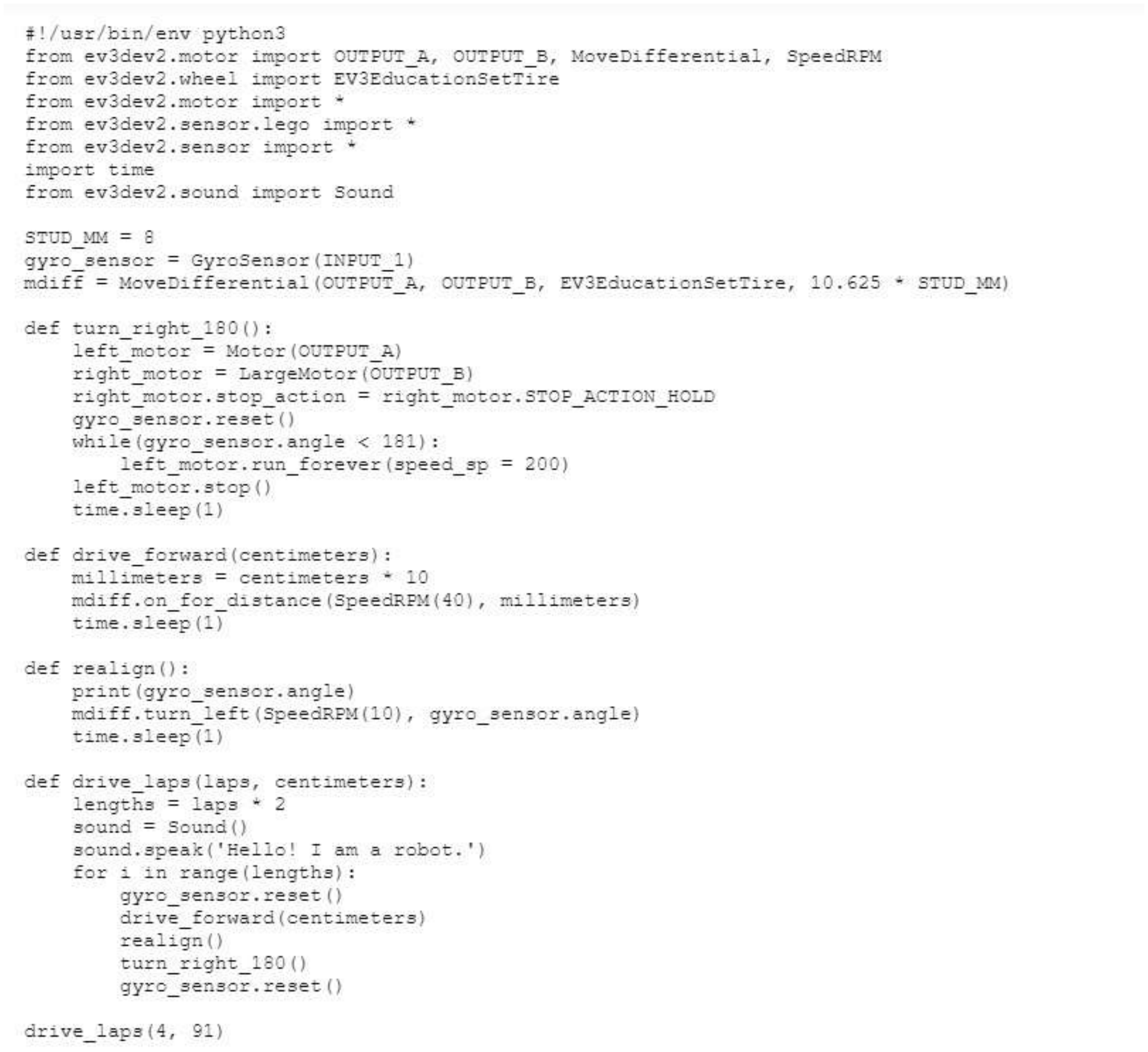
Minor changes were employed in order to lower the error while landing the robot in the revised code.

Codes for subtask 1a(To move forward and come back)



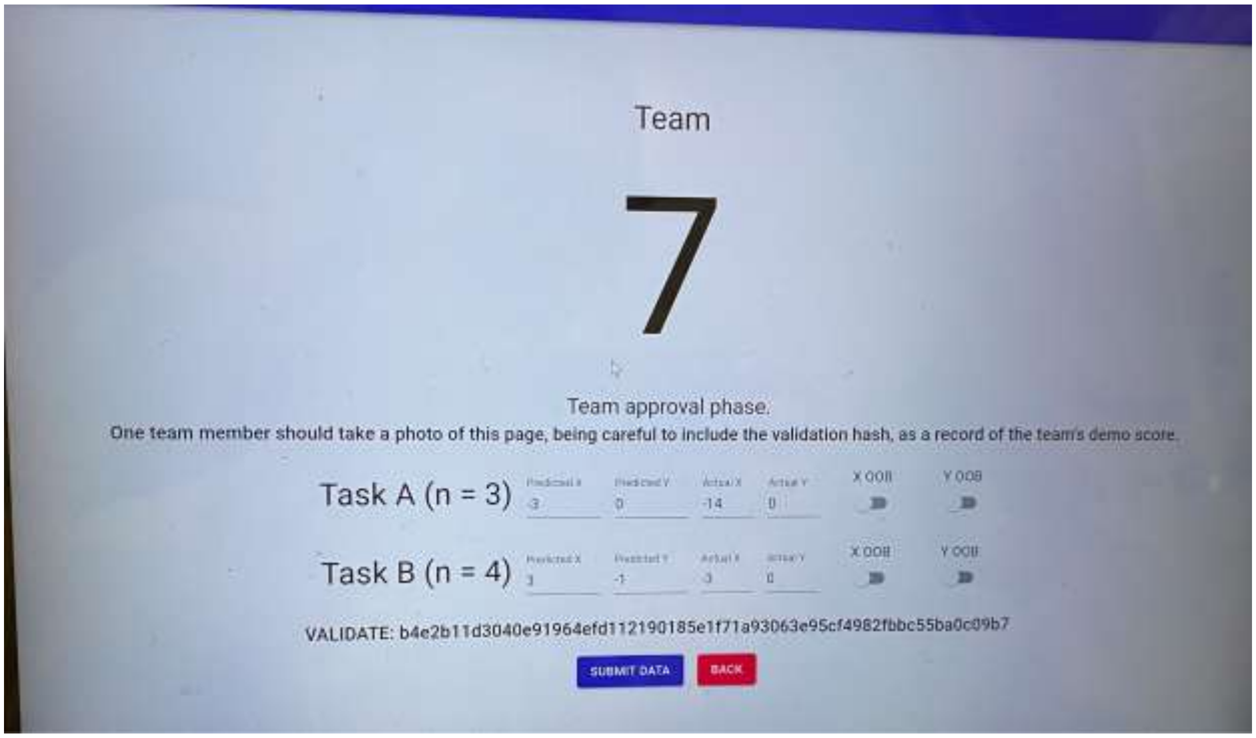
```
#!/usr/bin/env python3
from ev3dev2.motor import OUTPUT_A, OUTPUT_B, MoveDifferential, SpeedRPM
from ev3dev2.wheel import EV3EducationSetTire
from ev3dev2.motor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor import *
import time

STUD_MM = 8
gyro_sensor = GyroSensor(INPUT_1)
mdiff = MoveDifferential(OUTPUT_A, OUTPUT_B, EV3EducationSetTire, 10.625 * STUD_MM)

def drive_forward(centimeters):
    millimeters = centimeters * 10
    mdiff.on_for_distance(SpeedRPM(40), millimeters)
    time.sleep(1)

def drive_backward(centimeters):
    millimeters = centimeters * 10
    mdiff.on_for_distance(SpeedRPM(40), -millimeters)
    time.sleep(1)

def realign():
    print(gyro_sensor.angle)
    mdiff.turn_left(SpeedRPM(10), gyro_sensor.angle)
    time.sleep(1)

def drive_straight_and_back(laps, centimeters):
    lengths = laps * 2
    for i in range(lengths):
        if i % 2 == 0:
            gyro_sensor.reset()
            drive_forward(centimeters)
            print(gyro_sensor.angle)
            realign()
            gyro_sensor.reset()
        else:
            gyro_sensor.reset()
            drive_backward(centimeters)
            print(gyro_sensor.angle)
            realign()
            gyro_sensor.reset()

#drive_straight_and_back(0.5, 30.48)
drive_forward(213.36)
```

Codes for subtask 1b(To move forward and rotate the robot)



```python
#!/usr/bin/env python3
from ev3dev2.motor import OUTPUT_A, OUTPUT_B, MoveDifferential, SpeedRPM
from ev3dev2.wheel import EV3EducationSetTire
from ev3dev2.motor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor import *
import time
from ev3dev2.sound import Sound

STUD_MM = 8
gyro_sensor = GyroSensor(INPUT_1)
mdiff = MoveDifferential(OUTPUT_A, OUTPUT_B, EV3EducationSetTire, 10.625 * STUD_MM)

def turn_right_180():
    left_motor = Motor(OUTPUT_A)
    right_motor = LargeMotor(OUTPUT_B)
    right_motor.stop_action = right_motor.STOP_ACTION_HOLD
    gyro_sensor.reset()
    while(gyro_sensor.angle < 181):
        left_motor.run_forever(speed_sp = 200)
    left_motor.stop()
    time.sleep(1)

def drive_forward(centimeters):
    millimeters = centimeters * 10
    mdiff.on_for_distance(SpeedRPM(40), millimeters)
    time.sleep(1)

def realign():
    print(gyro_sensor.angle)
    mdiff.turn_left(SpeedRPM(10), gyro_sensor.angle)
    time.sleep(1)

def drive_laps(laps, centimeters):
    lengths = laps * 2
    sound = Sound()
    sound.speak('Hello! I am a robot.')
    for i in range(lengths):
        gyro_sensor.reset()
        drive_forward(centimeters)
        realign()
        turn_right_180()
        gyro_sensor.reset()

drive_laps(4, 91)
```

First Project Demonstration



Team

7

Team approval phase.

One team member should take a photo of this page, being careful to include the validation hash, as a record of the team's demo score.

	Predicted X	Predicted Y	Actual X	Actual Y	X OOB	Y OOB
Task A (n = 3)	-3	0	-14	0		
Task B (n = 4)	3	-1	-3	0		

VALIDATE: b4e2b11d3040e91964efd112190185e1f71a93063e95cf4982fbbc55ba0c09b7

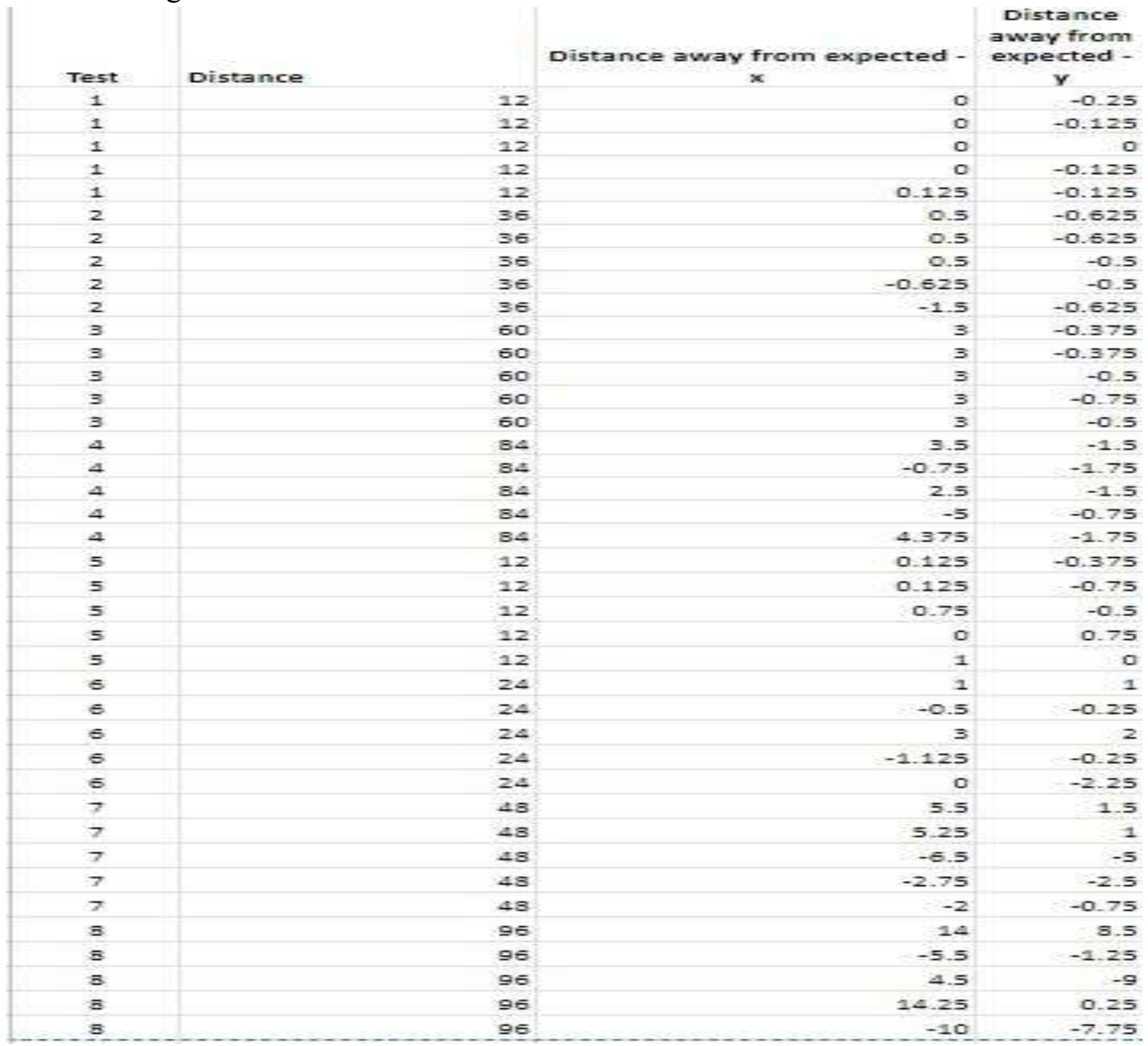
From the data that we collected and the statistics we calculated, at the first demonstration, we predicted that the robot would be at (-3,0). To find this we did a series of tests measuring the x and y coordinates. We used the values to determine what the average error of x would be per centimeter. Distances were calculated for half laps and whole laps. During the demonstration, we saw that the marble was not dragging on the paper very well which resulted in the error point being (-14,0).

For the turning test, our estimated error point was much closer to the actual point that resulted. We believe this is due to the gyro sensor calculating the angle, but there was still an issue with the marble rolling rather than sliding across the paper. This poor performance finalized our decision to change the design to a four-wheel tank design.

Locomotion Test Plan (Ideation)

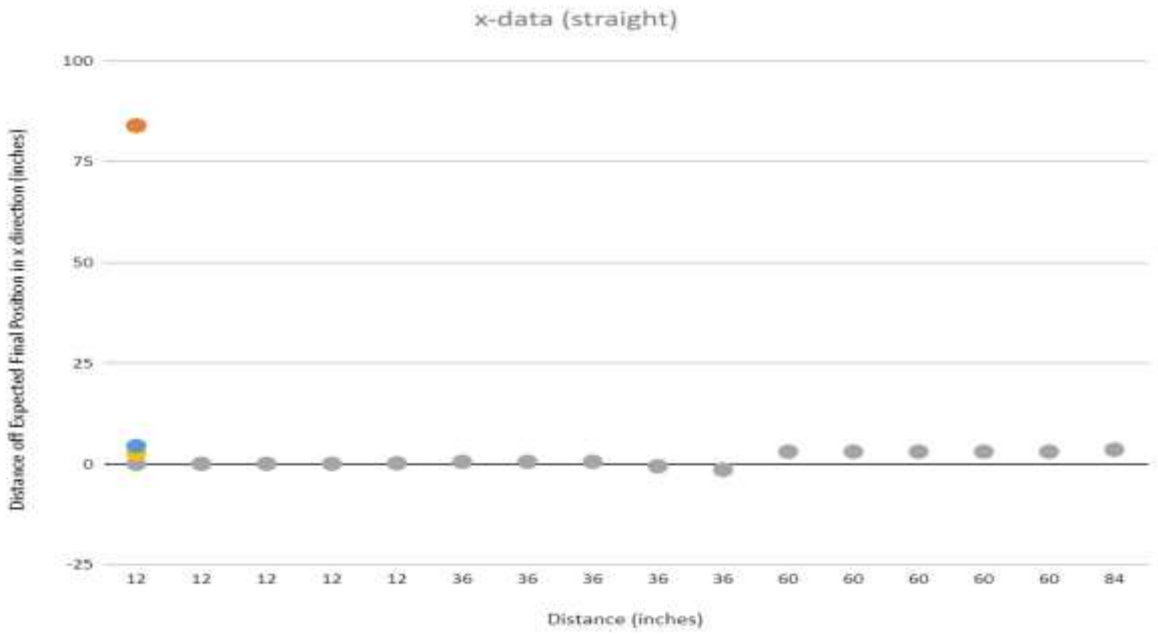
The ability of the robot to move in a straight line and to turn has been analyzed in the test. The data includes the distance traveled and the robot's distance away from the expected location in the x and y direction. First, we collected our data using our test plan, where we only modified one thing. We changed the lengths it drives on the turning tests to make them equal so that we could test our robot over longer distances (that is what it had the most trouble with). Then, we tested our robot for each of these conditions and measured its error in the x and y direction. We plotted this data on two different graphs, one using the straight data and one using the turning data. We did this for two reasons: 1) so we can compare the two equations and see which is more consistent 2) because our turning data did not yield "predictable" results.

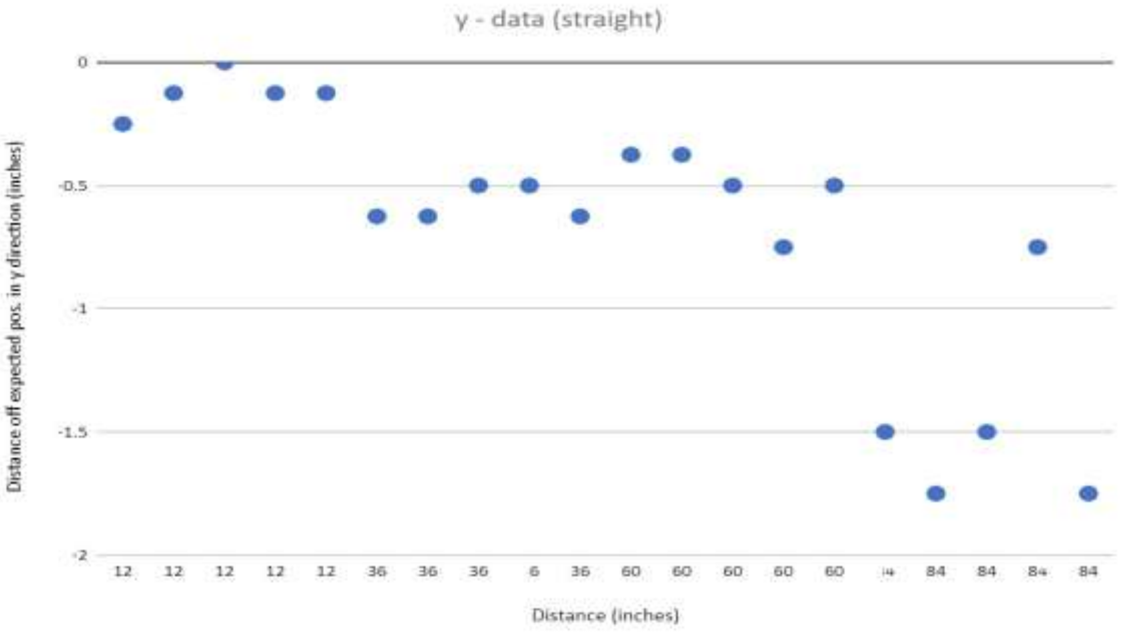
The following is the data for the tests:

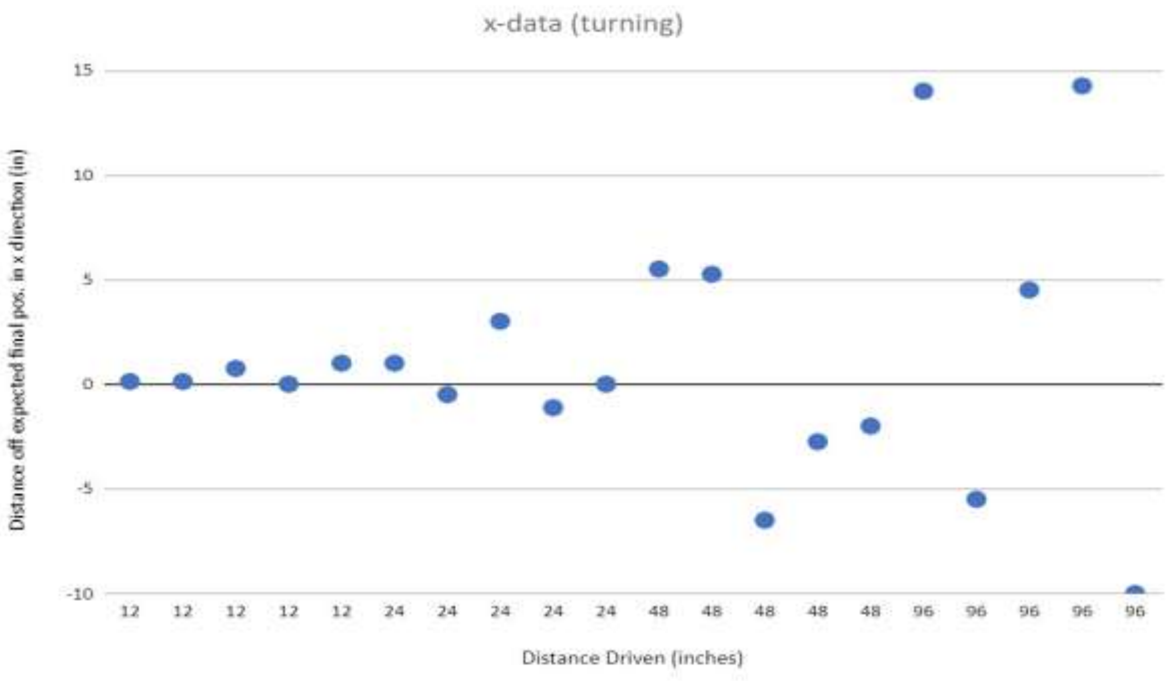


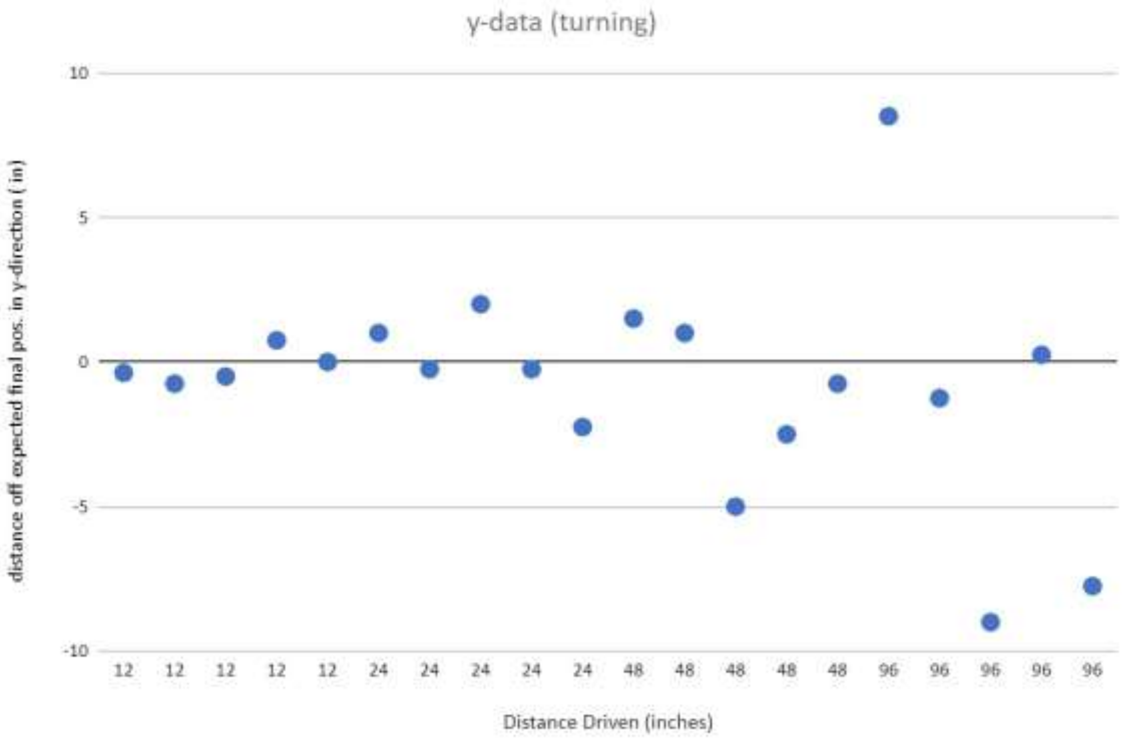
Test	Distance	Distance away from expected - x	Distance away from expected - y
1	12	0	-0.25
1	12	0	-0.125
1	12	0	0
1	12	0	-0.125
1	12	0.125	-0.125
2	36	0.5	-0.625
2	36	0.5	-0.625
2	36	0.5	-0.5
2	36	-0.625	-0.5
2	36	-1.5	-0.625
3	60	3	-0.375
3	60	3	-0.375
3	60	3	-0.5
3	60	3	-0.75
3	60	3	-0.5
4	84	3.5	-1.5
4	84	-0.75	-1.75
4	84	2.5	-1.5
4	84	-5	-0.75
4	84	4.375	-1.75
5	12	0.125	-0.375
5	12	0.125	-0.75
5	12	0.75	-0.5
5	12	0	0.75
5	12	1	0
6	24	1	1
6	24	-0.5	-0.25
6	24	3	2
6	24	-1.125	-0.25
6	24	0	-2.25
7	48	5.5	1.5
7	48	5.25	1
7	48	-6.5	-5
7	48	-2.75	-2.5
7	48	-2	-0.75
8	96	14	8.5
8	96	-5.5	-1.25
8	96	4.5	-9
8	96	14.25	0.25
8	96	-10	-7.75

The following are the graphs for the data above:


x-data (straight)
100
75
50
25
0
-25
Distance off Expected Final Position in x direction (inches)
12
12
12
12
12
36
36
36
36
36
60
60
60
60
60
84
Distance (inches)


y - data (straight)
0
-0.5
-1
-1.5
-2
Distance off expected pos. in y direction (inches)
12
12
12
12
12
36
36
36
6
36
60
60
60
60
60
i4
84
84
84
84
Distance (inches)





Conclusion for the data analysis for the locomotion test:

Based on the data, it seems that the robot had some difficulty in maintaining its expected position during both the straight line and turning tests. The distance away from the expected position varied quite a bit, with some measurements very close to the expected position and others quite far away. The turning data also did not yield “predictable” results. This may indicate that the robot had more difficulty turning than with moving in a straight line.

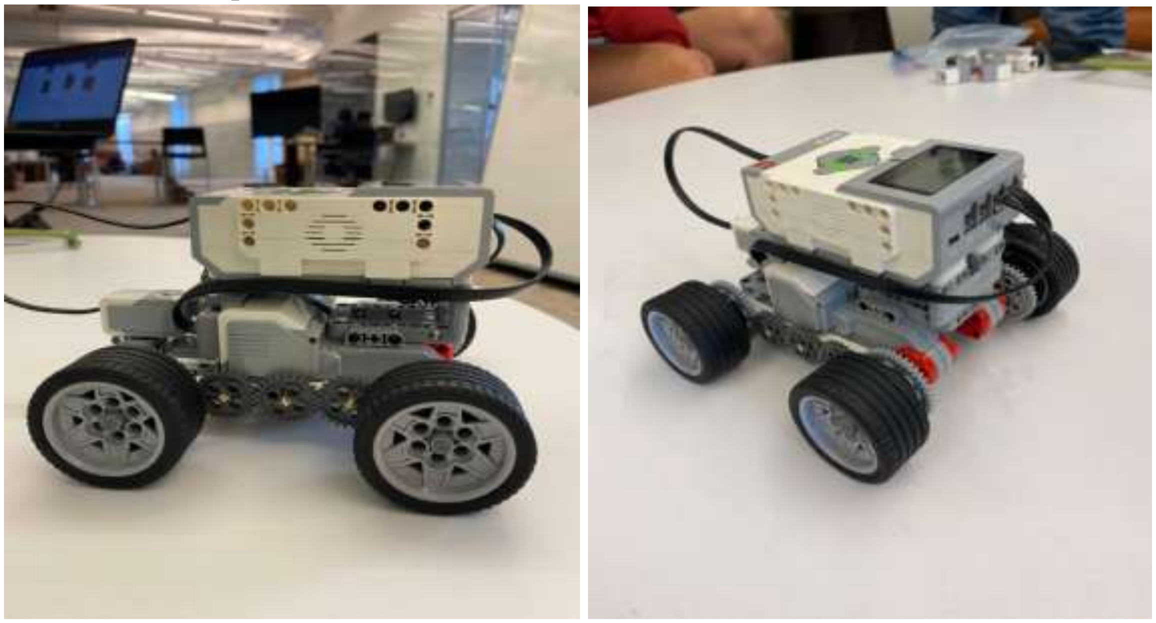
Remodeling

After meditating on the shortcomings of the movement of our robot and to fulfill other criteria, remodeling was proposed.

The following are the problems we addressed with the first prototype:

1. Difficulty in predicting where it would stop after starting it
2. Irregularity in the coordinates obtained
3. Difficulty in maintaining its expected position during both the straight line and turning tests
4. Difficulty in turning the robot

So, to address the problem, the robot was remodeled:



1. The marble was removed and four wheels were added to maintain the position
2. The entire structure becomes more stable
3. The robot lands precisely in the expected position
4. It will be able to handle the additional weight of the crane and other components.

Prototype 2.0

Other than moving in a straight line and turning, there are other goals that the robot needs to fulfill such as

Navigation - The robot must be able to navigate the warehouse and create a map of its environment.

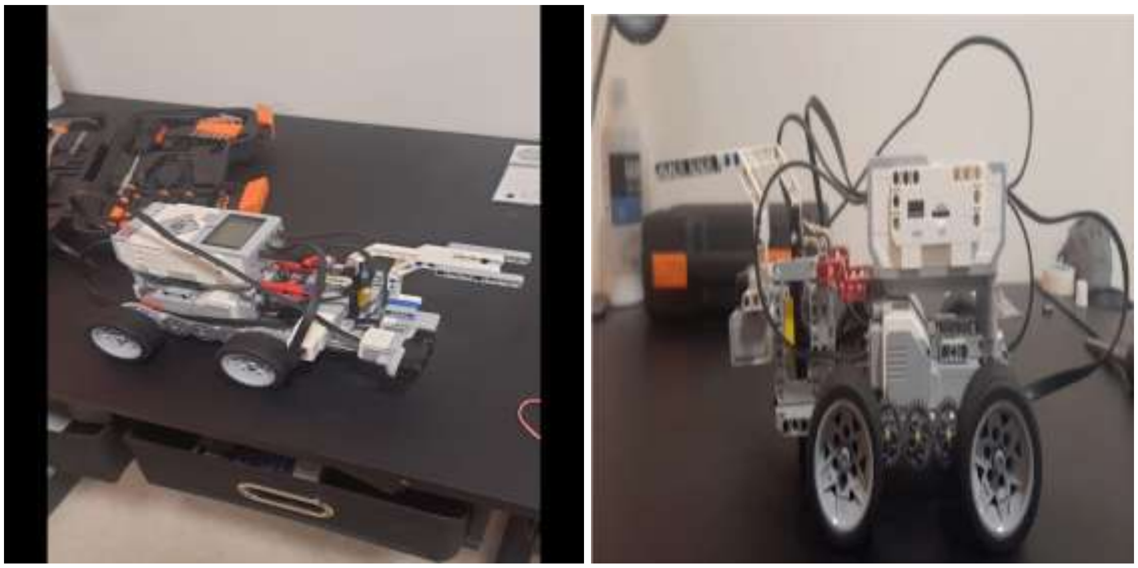
Identification – The robot must be able to easily distinguish the different cargo types and place them accordingly.

Transport – The robot must have the ability to properly secure any cargo and avoid dropping it.

Placement – The robot must be able to place transported cargo accurately in its intended location consistently.

The following modifications were made to the robot to fulfill the criteria given above:

1. **Addition of the crane system.**
2. **Addition of the color sensor.**
3. **Addition of the gyroscope.**
4. **Addition of an ultrasonic sensor.**



Prototype 2.0 - Improvements Were Made

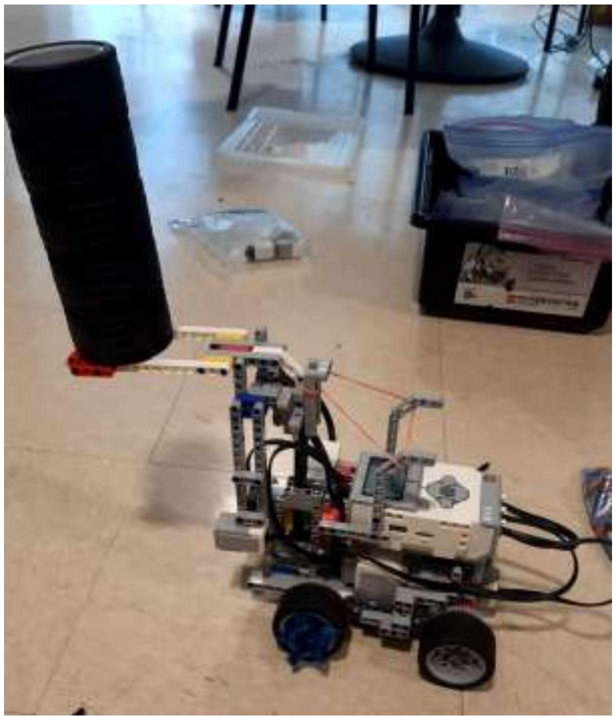
Although switching to four wheels from two wheels and marble helped us solve the movement accuracy problem, it was far from what Buy More intended us to design. The crane system, color sensors, ultrasonic sensor, and gyroscope were added to the prototype for the tasks aforementioned but the performance was not on par. The lifting mechanism failed to lift the boxes and the color sensor was positioned where it could not read the barcode accurately so the team decided to remodel the robot and recode it.

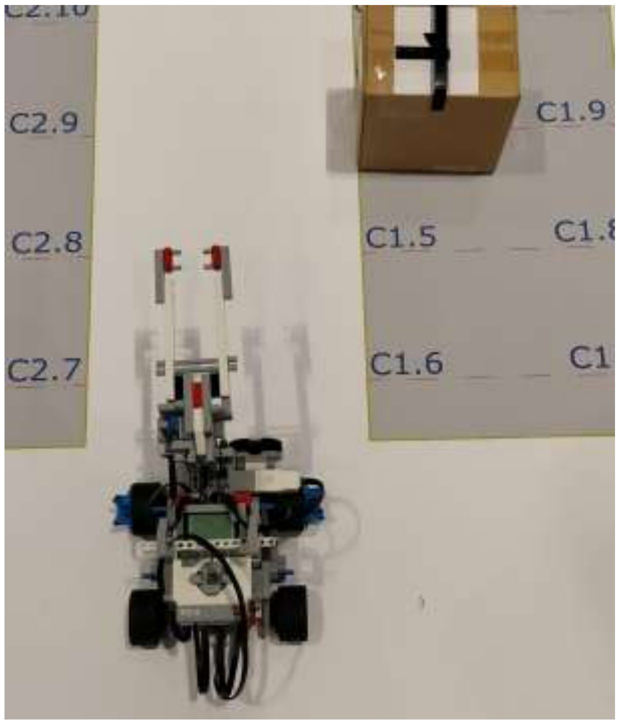
Prototype 3.0- Ideation

Prototype 3.0 is an improvement from its predecessor 2.0. While the skeleton and chassis remained the same, the crane system was designed to be extendable i.e., the fork with the help of a mechanical belt could move up and down to lift the box off the ground and support the box's weight. The color sensor too was attached to the moving part for it to manually read the entire bar code. The crane system was reinforced in such a way that it could lift up heavy weight. In summary, the following were the improvements made:

1. **Retractable Crane system**: This allowed the robot to lift and move objects that it previously could not. The fork could move up and down to lift the box off the ground.
2. **Color sensor:** This allowed the robot to read barcodes on boxes using reflected light, which can be used to identify the contents of the box and determine where it needs to be moved within the warehouse. This can help streamline the picking and sorting process.
3. **Gyroscope**: This helped the robot determine its position and orientation more accurately. This was useful for navigation, especially in environments where there may be obstacles or other hazards that the robot needs to avoid.
4. **Ultrasonic sensor:** This will allow the robot to detect objects that may be out of its line of sight. This is useful for identifying and avoiding obstacles, as well as moving a desired distance away from a box.

Introducing Axiom I





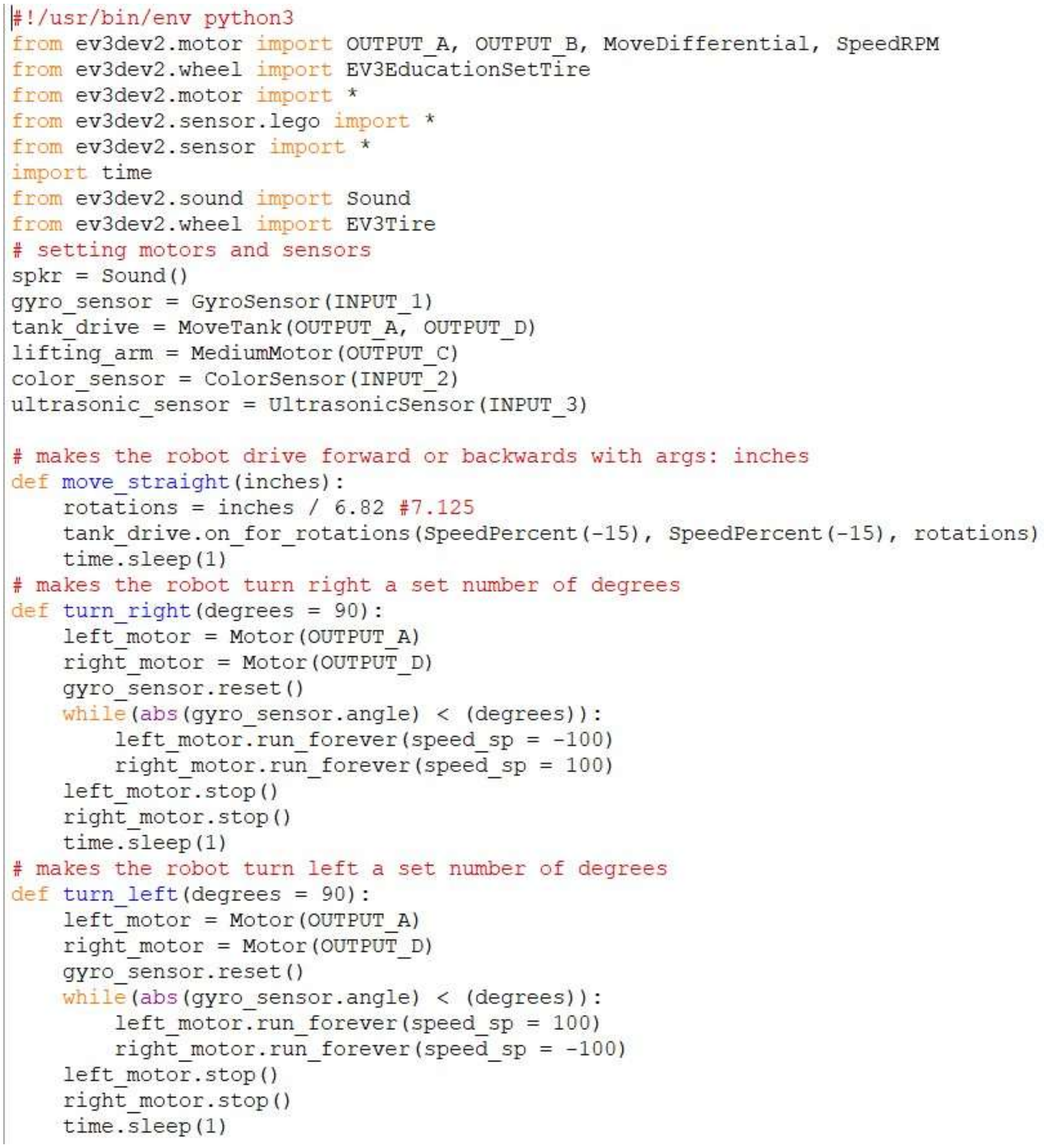
Developing a sturdy and modular chassis was important to accommodate changes in design plans and requirements. The robot arm was strong enough to lift weights as high as 250g minimum, and accurately positioned sensors collected data precisely to navigate the robot around the warehouse perfectly.

Mobility and navigation were critical factors. Axiom I was able to move accurately without straying far from the intended path, navigate the warehouse and create a map of its environment, and traverse smoothly on surfaces commonly found in a warehouse.

Identification and transport were also key components of the system, as the robot was able to scan barcodes accurately and identify the right cargo and properly secure any cargo to avoid dropping it. The robot had rubber grippers at the end of the fork to hold and move the boxes around without dropping them, and be able to place transported cargo accurately in its intended location consistently.

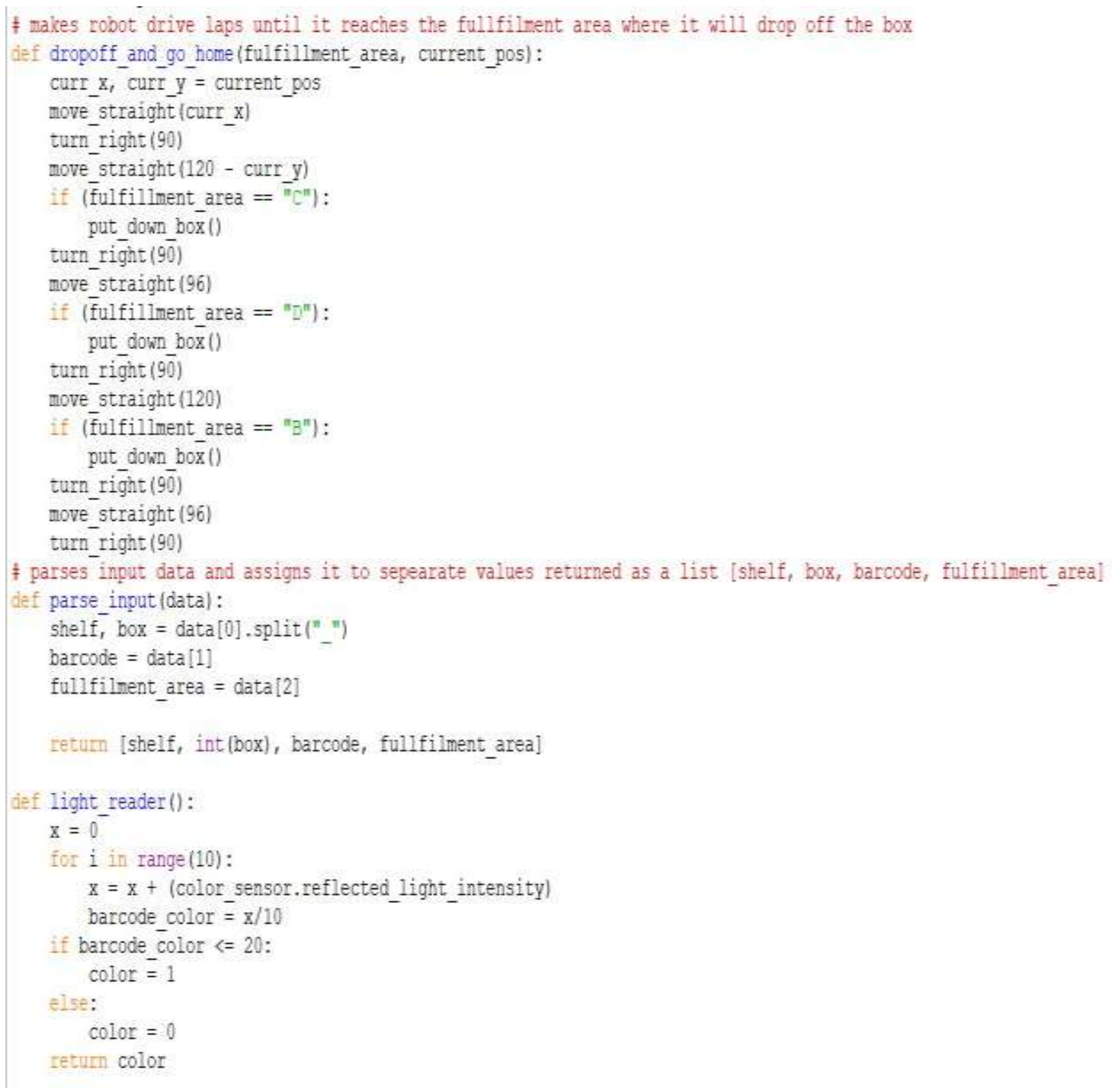
Lastly, equipping it with sensors to allow it to navigate the warehouse and locate different boxes, as well as computing capabilities to perform real-time navigation and mapping.

Behind the scenes (Final Code)

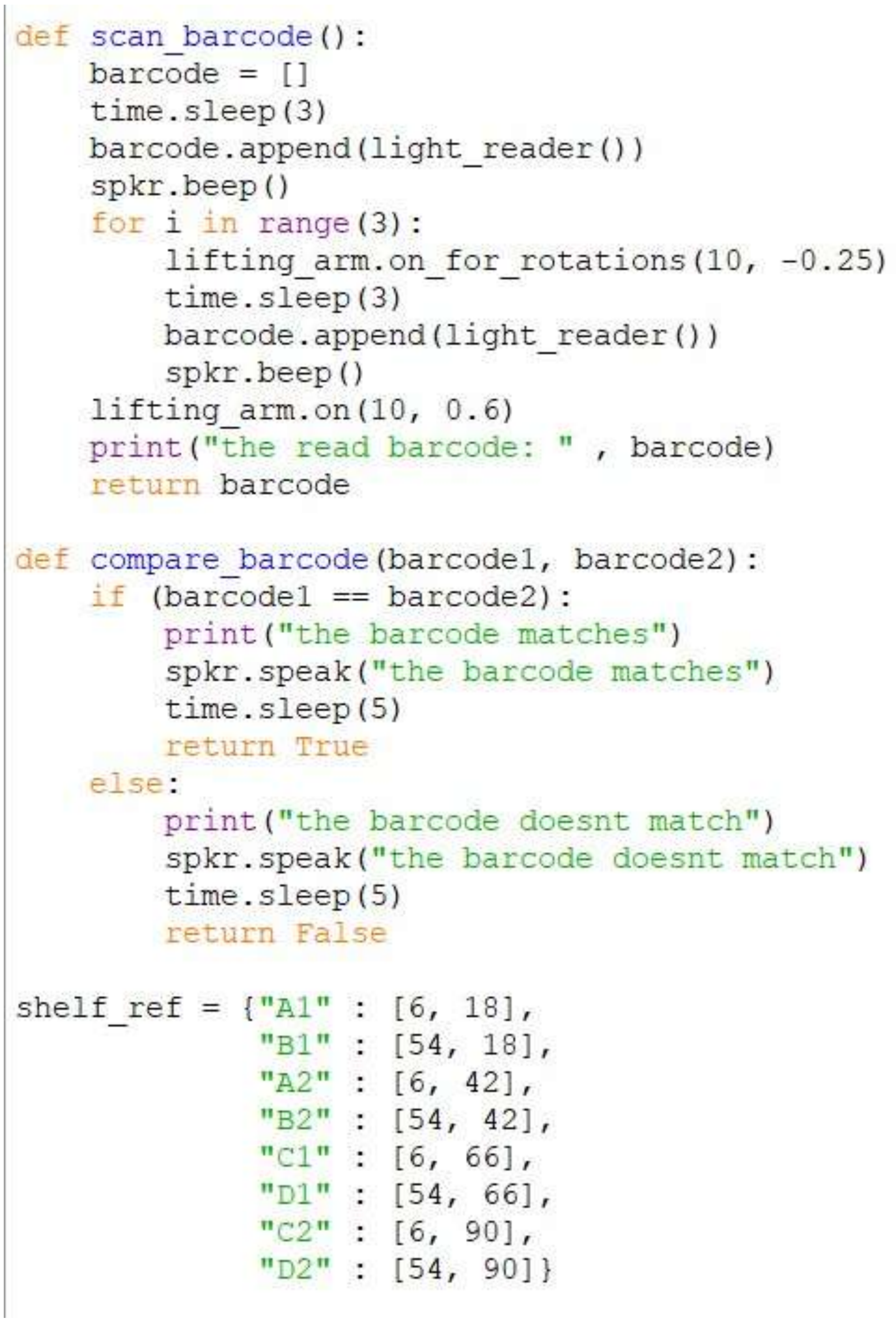


```python
#!/usr/bin/env python3
from ev3dev2.motor import OUTPUT_A, OUTPUT_B, MoveDifferential, SpeedRPM
from ev3dev2.wheel import EV3EducationSetTire
from ev3dev2.motor import *
from ev3dev2.sensor.lego import *
from ev3dev2.sensor import *
import time
from ev3dev2.sound import Sound
from ev3dev2.wheel import EV3Tire
# setting motors and sensors
spkr = Sound()
gyro_sensor = GyroSensor(INPUT_1)
tank_drive = MoveTank(OUTPUT_A, OUTPUT_D)
lifting_arm = MediumMotor(OUTPUT_C)
color_sensor = ColorSensor(INPUT_2)
ultrasonic_sensor = UltrasonicSensor(INPUT_3)

# makes the robot drive forward or backwards with args: inches
def move_straight(inches):
    rotations = inches / 6.82 #7.125
    tank_drive.on_for_rotations(SpeedPercent(-15), SpeedPercent(-15), rotations)
    time.sleep(1)
# makes the robot turn right a set number of degrees
def turn_right(degrees = 90):
    left_motor = Motor(OUTPUT_A)
    right_motor = Motor(OUTPUT_D)
    gyro_sensor.reset()
    while(abs(gyro_sensor.angle) < (degrees)):
        left_motor.run_forever(speed_sp = -100)
        right_motor.run_forever(speed_sp = 100)
    left_motor.stop()
    right_motor.stop()
    time.sleep(1)
# makes the robot turn left a set number of degrees
def turn_left(degrees = 90):
    left_motor = Motor(OUTPUT_A)
    right_motor = Motor(OUTPUT_D)
    gyro_sensor.reset()
    while(abs(gyro_sensor.angle) < (degrees)):
        left_motor.run_forever(speed_sp = 100)
        right_motor.run_forever(speed_sp = -100)
    left_motor.stop()
    right_motor.stop()
    time.sleep(1)
```

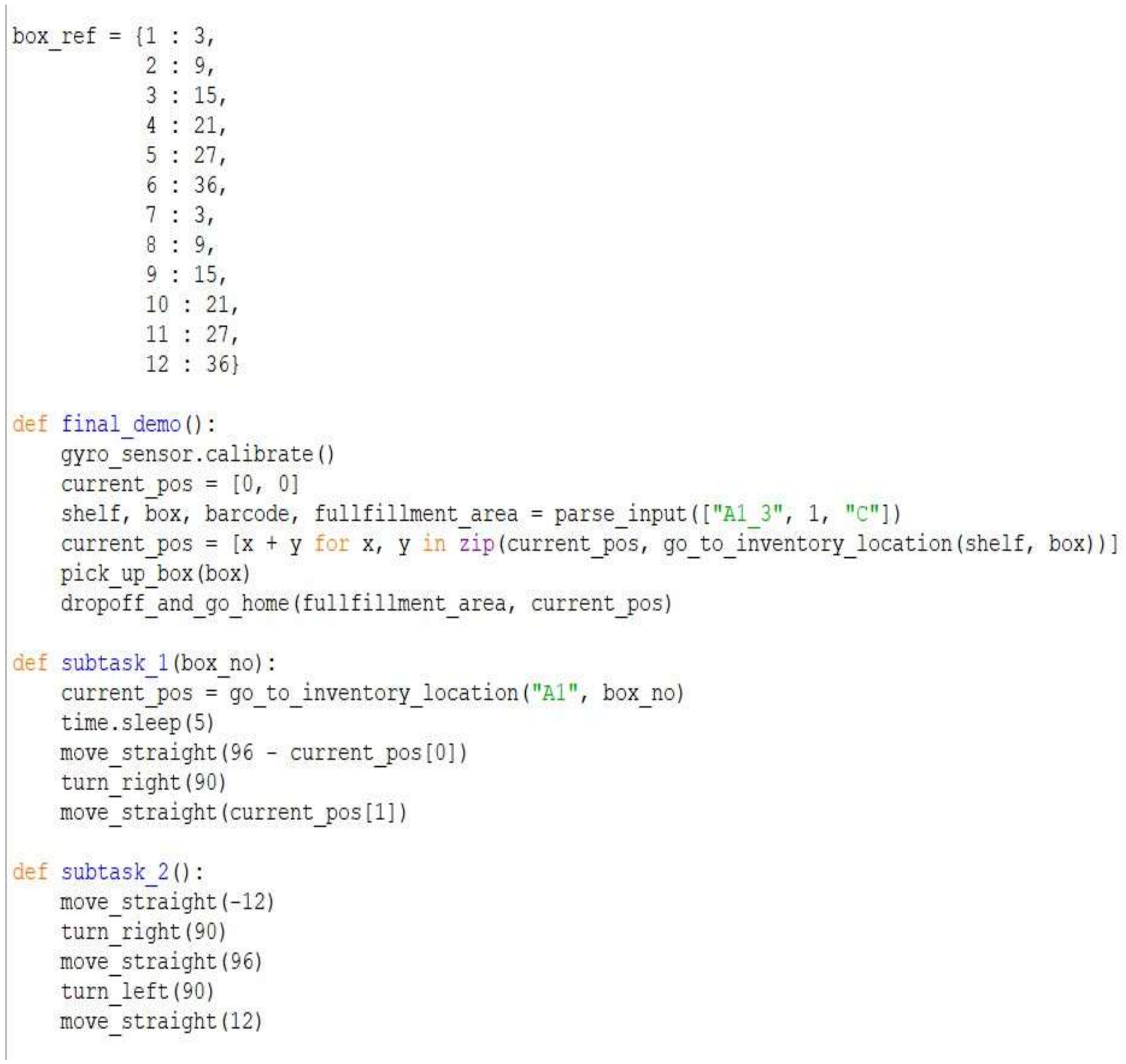
```
# makes the robot go to shelf and box
def go_to_inventory_location(shelf, box):
    if int(box) <= 6:
        destination = shelf_ref[shelf]
        destination[1] = destination[1] - 6
    else:
        destination = shelf_ref[shelf]
        destination[1] = destination[1] + 18
    destination[0] = destination[0] + box_ref[int(box)]
    move_straight(destination[1])
    turn_right(90)
    move_straight(destination[0])
    return destination
# makes robot drive into box, pick it up, reverse back, and turn to exit
def pick_up_box(box):
    if box <= 6:
        turn_left(90)
        move_straight(3)
        lifting_arm.on_for_rotations(10, -1)
        move_straight(3)
        lifting_arm.on_for_rotations(10,1)
        time.sleep(3)
        move_straight(-6)
        turn_left(90)
    else:
        turn_right(90)
        move_straight(3)
        lifting_arm.on_for_rotations(10, -1)
        move_straight(3)
        lifting_arm.on_for_rotations(10,1)
        time.sleep(3)
        move_straight(-6)
        turn_left(90)
# makes robot put down box
def put_down_box():
    lifting_arm.on_for_rotations(10, -1)
    time.sleep(3)
    move_straight(-3)
    lifting_arm.on_for_rotations(10, 1)
    time.sleep(3)
```



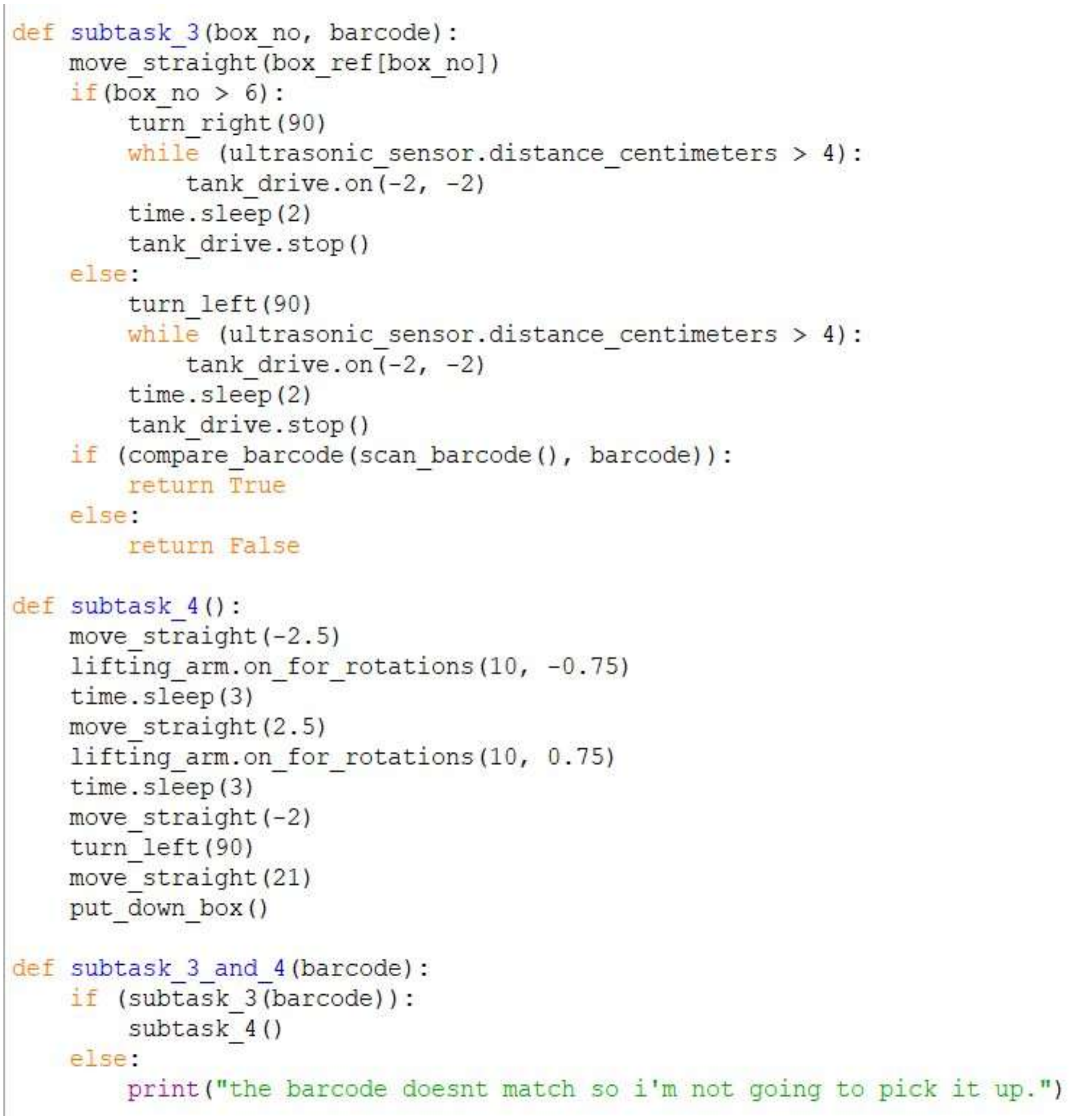
```python
# makes robot drive laps until it reaches the fullfilment area where it will drop off the box
def dropoff_and_go_home(fulfillment_area, current_pos):
    curr_x, curr_y = current_pos
    move_straight(curr_x)
    turn_right(90)
    move_straight(120 - curr_y)
    if (fulfillment_area == "C"):
        put_down_box()
    turn_right(90)
    move_straight(96)
    if (fulfillment_area == "D"):
        put_down_box()
    turn_right(90)
    move_straight(120)
    if (fulfillment_area == "B"):
        put_down_box()
    turn_right(90)
    move_straight(96)
    turn_right(90)
# parses input data and assigns it to sepearate values returned as a list [shelf, box, barcode, fulfillment_area]
def parse_input(data):
    shelf, box = data[0].split("_")
    barcode = data[1]
    fullfilment_area = data[2]

    return [shelf, int(box), barcode, fullfilment_area]

def light_reader():
    x = 0
    for i in range(10):
        x = x + (color_sensor.reflected_light_intensity)
        barcode_color = x/10
    if barcode_color <= 20:
        color = 1
    else:
        color = 0
    return color
```



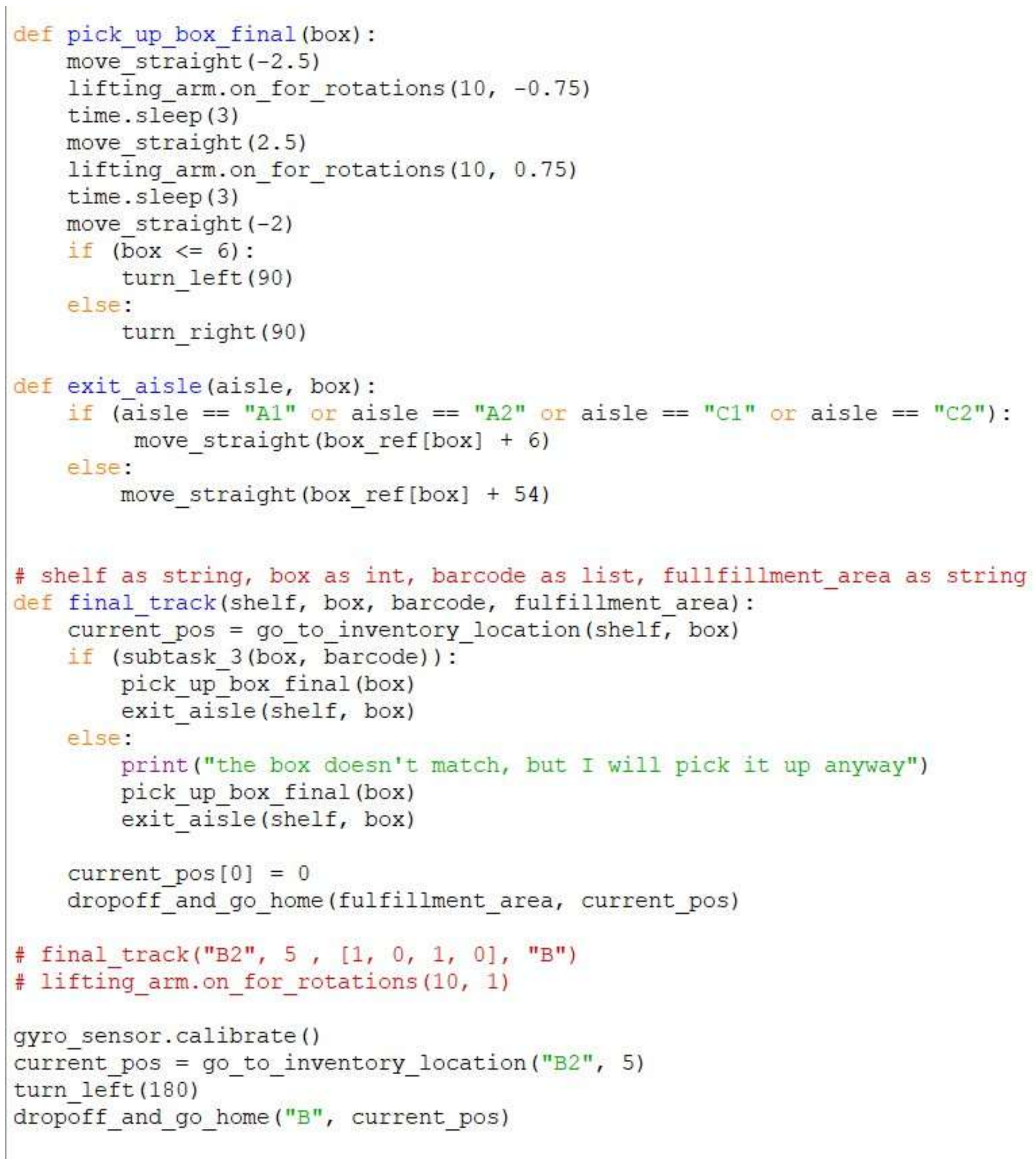
```python
def scan_barcode():
    barcode = []
    time.sleep(3)
    barcode.append(light_reader())
    spkr.beep()
    for i in range(3):
        lifting_arm.on_for_rotations(10, -0.25)
        time.sleep(3)
        barcode.append(light_reader())
        spkr.beep()
    lifting_arm.on(10, 0.6)
    print("the read barcode: " , barcode)
    return barcode

def compare_barcode(barcode1, barcode2):
    if (barcode1 == barcode2):
        print("the barcode matches")
        spkr.speak("the barcode matches")
        time.sleep(5)
        return True
    else:
        print("the barcode doesnt match")
        spkr.speak("the barcode doesnt match")
        time.sleep(5)
        return False

shelf_ref = {"A1" : [6, 18],
             "B1" : [54, 18],
             "A2" : [6, 42],
             "B2" : [54, 42],
             "C1" : [6, 66],
             "D1" : [54, 66],
             "C2" : [6, 90],
             "D2" : [54, 90]}
```



```python
box_ref = {1 : 3,
           2 : 9,
           3 : 15,
           4 : 21,
           5 : 27,
           6 : 36,
           7 : 3,
           8 : 9,
           9 : 15,
           10 : 21,
           11 : 27,
           12 : 36}

def final_demo():
    gyro_sensor.calibrate()
    current_pos = [0, 0]
    shelf, box, barcode, fullfillment_area = parse_input(["A1_3", 1, "C"])
    current_pos = [x + y for x, y in zip(current_pos, go_to_inventory_location(shelf, box))]
    pick_up_box(box)
    dropoff_and_go_home(fullfillment_area, current_pos)

def subtask_1(box_no):
    current_pos = go_to_inventory_location("A1", box_no)
    time.sleep(5)
    move_straight(96 - current_pos[0])
    turn_right(90)
    move_straight(current_pos[1])

def subtask_2():
    move_straight(-12)
    turn_right(90)
    move_straight(96)
    turn_left(90)
    move_straight(12)
```

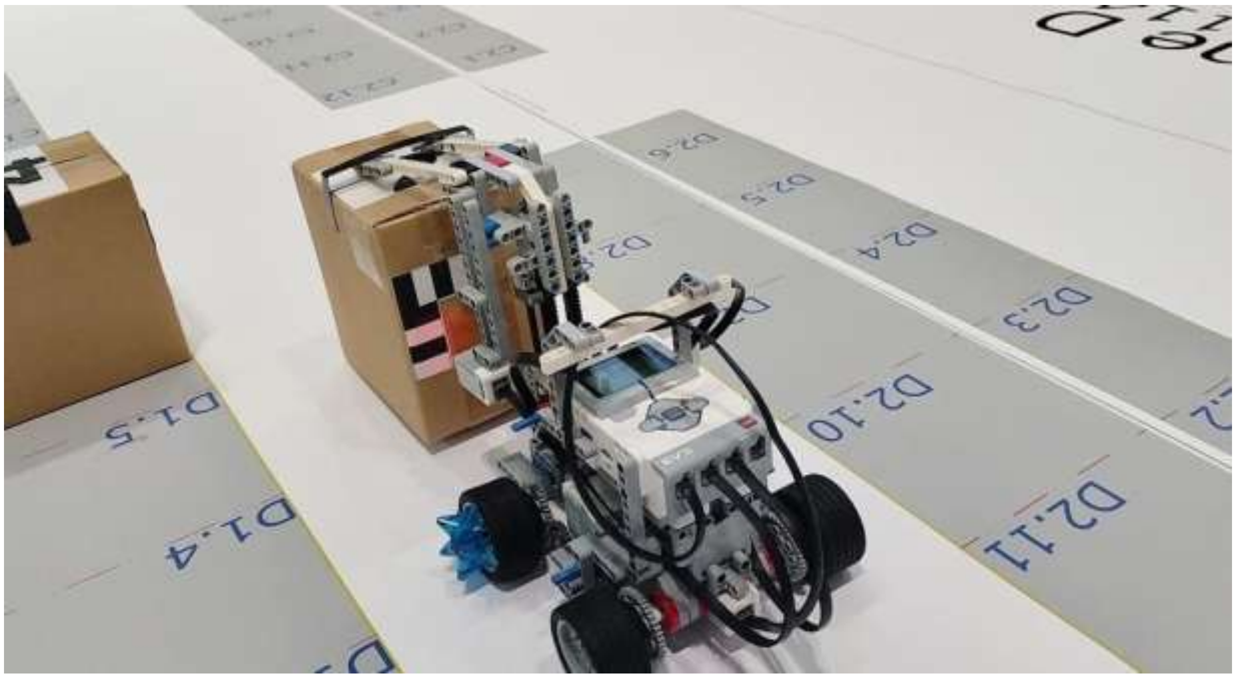


```python
def subtask_3(box_no, barcode):
    move_straight(box_ref[box_no])
    if(box_no > 6):
        turn_right(90)
        while (ultrasonic_sensor.distance_centimeters > 4):
            tank_drive.on(-2, -2)
        time.sleep(2)
        tank_drive.stop()
    else:
        turn_left(90)
        while (ultrasonic_sensor.distance_centimeters > 4):
            tank_drive.on(-2, -2)
        time.sleep(2)
        tank_drive.stop()
    if (compare_barcode(scan_barcode(), barcode)):
        return True
    else:
        return False

def subtask_4():
    move_straight(-2.5)
    lifting_arm.on_for_rotations(10, -0.75)
    time.sleep(3)
    move_straight(2.5)
    lifting_arm.on_for_rotations(10, 0.75)
    time.sleep(3)
    move_straight(-2)
    turn_left(90)
    move_straight(21)
    put_down_box()

def subtask_3_and_4(barcode):
    if (subtask_3(barcode)):
        subtask_4()
    else:
        print("the barcode doesnt match so i'm not going to pick it up.")
```

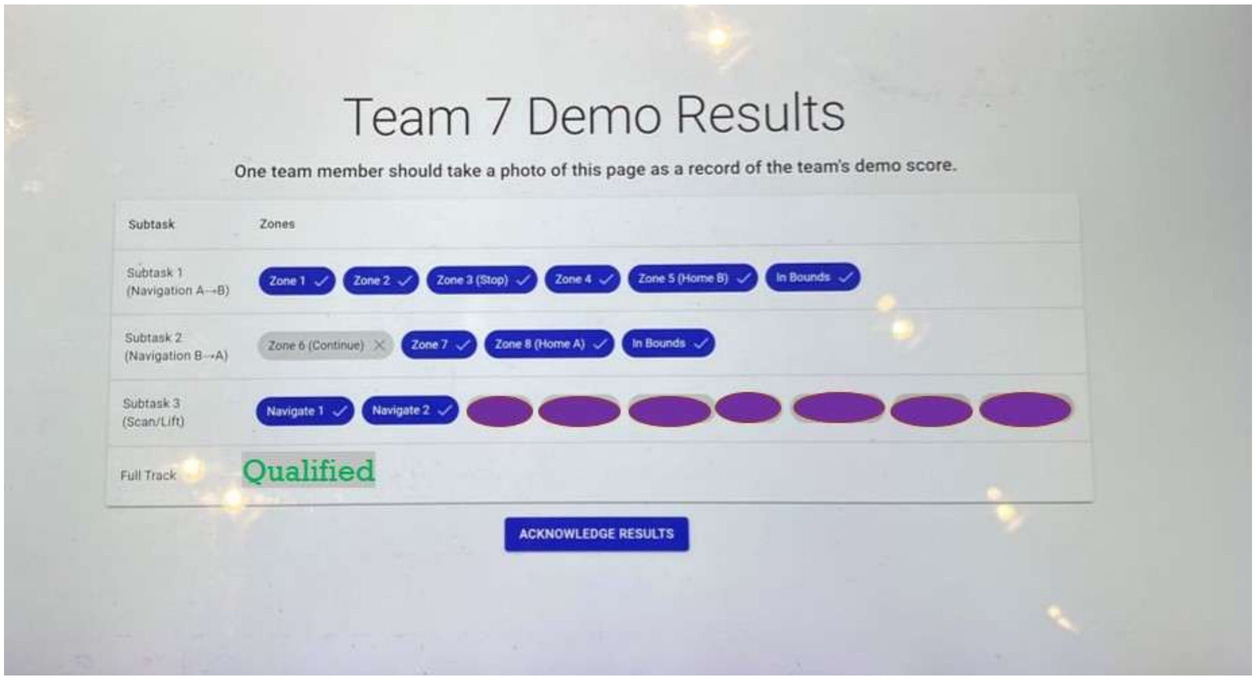


```python
def pick_up_box_final(box):
    move_straight(-2.5)
    lifting_arm.on_for_rotations(10, -0.75)
    time.sleep(3)
    move_straight(2.5)
    lifting_arm.on_for_rotations(10, 0.75)
    time.sleep(3)
    move_straight(-2)
    if (box <= 6):
        turn_left(90)
    else:
        turn_right(90)

def exit_aisle(aisle, box):
    if (aisle == "A1" or aisle == "A2" or aisle == "C1" or aisle == "C2"):
         move_straight(box_ref[box] + 6)
    else:
        move_straight(box_ref[box] + 54)


# shelf as string, box as int, barcode as list, fullfillment_area as string
def final_track(shelf, box, barcode, fulfillment_area):
    current_pos = go_to_inventory_location(shelf, box)
    if (subtask_3(box, barcode)):
        pick_up_box_final(box)
        exit_aisle(shelf, box)
    else:
        print("the box doesn't match, but I will pick it up anyway")
        pick_up_box_final(box)
        exit_aisle(shelf, box)

    current_pos[0] = 0
    dropoff_and_go_home(fulfillment_area, current_pos)

# final_track("B2", 5 , [1, 0, 1, 0], "B")
# lifting_arm.on_for_rotations(10, 1)

gyro_sensor.calibrate()
current_pos = go_to_inventory_location("B2", 5)
turn_left(180)
dropoff_and_go_home("B", current_pos)
```

Axiom I - On the Demo



During the demo, we were able to complete all the subtasks with somewhat ease. In subtasks 1 and 2, we were tasked to move forward, rotate the robot and return back to the home base. For subtasks 3 and 4, the robot was supposed to locate the box, scan the barcode, lift the box, and take the box to the desired location. We had some hiccups in accomplishing tasks 3 and 4 but were solved after some slight modifications to the code and grip mechanism on the forklift.



Team 7 Demo Results

One team member should take a photo of this page as a record of the team's demo score.

Subtask	Zones
Subtask 1 (Navigation A→B)	Zone 1 ✓ Zone 2 ✓ Zone 3 (Stop) ✓ Zone 4 ✓ Zone 5 (Home B) ✓ In Bounds ✓
Subtask 2 (Navigation B→A)	Zone 6 (Continue) ✕ Zone 7 ✓ Zone 8 (Home A) ✓ In Bounds ✓
Subtask 3 (Scan/Lift)	Navigate 1 ✓ Navigate 2 ✓ [redacted] [redacted] [redacted] [redacted] [redacted] [redacted] [redacted]
Full Track	Qualified

Future Enhancements

1. Square Wheelbase: To ensure that the APR can turn on the spot
2. More stable lifting mechanism: To decrease the weight dragging on the front
3. Three wheels on each side instead of two to alleviate axle strain and increase contact with the ground and hence torque.
4. The components used to design the robot such as motors can be enhanced for improved performance.
5. The sensors used in the APR can be upgraded to improve their accuracy and reliability.