Tutorial Report

Maryo Botros
mbotros@oxy.edu
Occidental College

1 Methods and Evaluation

The tutorial I used aimed to teach Python using Mindstorms. The first part of the tutorial uses the robot called "Tricky." This robot has two wheels and each wheel is controlled by a motor. This is important because this is how the robot navigates. If both wheels are moving forward, the robot will move forward, if both are moving backward, the robot will be moving backward. If the left wheel is moving forward and the right wheel is moving backward, the robot will be turning right. Finally, if the left wheel is moving backward and the right wheel is moving forward, the robot will be turning left. It uses distance sensors on the front, which will be used to help the robot avoid objects.

The code will be written in the Mindstorms software. In order for Python to interface with the Lego specific methods and functions, it needs important classes to be imported, such as functions that control the motor or use information read by the sensors. Starting off, in the objects section, I have defined an object and in the objects section called "hub" and in the programming section I told the object to dot into the speaker and then dot into the beep() method. This is like telling the object to look for the speaker and within the speaker, look for the .beep() method. When the code is run, the robot makes a faint beeping noise out of its speaker.

```
from mindstorms import MShub, Motor,
Motorpair, ColorSensor,
DistcanceSensor, App
from mindstroms.control import
wait_for_seconds, wait_until, Timer
from mindstorms.operator import
greater_than, greater_than_or_equal_to, l
ess_than, less_than_or_equal_to,
equal_to, not_equal_to
import math

# Create your objects here
hub = MSHub()

# Write your program here
hub.speaker.beep()
```

Next, I will get the robot to move. I begin by creating a movement_motors object in the objects section and setting this equal to a MotorPair('A', 'B'), where A and B represent the left and right wheel on the robot. To get the robot to move, we use the move() function on the movement_motors object and give it three parameters. The first parameter is a number value that represents how far we want to move, which will be 20. The second parameter represents the unit we want to use, which will be centimeters. The third parameter represents how much steering we want, which we will use 0. When this code is run, the robot moves forward 20 centimeters.

```
# Create your objects here
hub = MSHub()
movement_motors = MotorPair('A', 'B')

# Write your program here
hub.speaker.beep()
movement_motors.move(20, 'cm', )
```

Now we want to see if we can get the robot to move forward and then stop if it detects an obstacle in front of it, so that it is a bit smarter. To do this, we can tell the motors to start moving using the start() method. We want it to stop once it detects an obstacle coming closer to the distance sensor. In order to use the distance sensor, we will need to create an object for it in the objects section. This object will be called distance_sensor and it will be set equal to DistanceSensor('D'). Next in the programming section, we can now use this distance sensor to tell it to wait until it is closer than 10 centimeters using the wait_for_distance_closer_than(10, 'cm') function. After this, we can tell the robot to stop. After running this code, the robot will use the movement motors to move 20 centimeters and once it senses that an object is closer than 0 centimeters, using the distance sensor, it will use the movement motors to come to a complete stop.

```
# Create your objects here
hub = MSHub()
movement_motors = MotorPair('A', 'B')
distance_sensor = DistanceSensor('D')

# Write your program here
hub.speaker.beep()
```

```
# movement_motors.move(20, ’cm’, )
movement_motors.start()
distance_sensor.
wait_for_distance_closer_than(10, ’cm’)
movement_motors.stop()
```

For the second part of the tutorial, I learned a little bit more about motor control. Tricky will still be the robot used for this part, with the upgrade of the lifting arms and sensor package. This includes a motor that controls an arm as well as a color sensor on the back of the robot.

The first step for using these new modifications to the robot is to define the lifting arm in the objects section. We can define it as lifting_arm = Motor(‘C’) because ‘C’ is where the lifting arm is connected to on the robot. To get the arm to move to a position, we can use the run.to.position method on the lifting_arm object, with three different arguments. The first argument is the degrees that it is running to. The second argument is how you want to run to that position, with ‘shortest path’ as the default. There are also other substitutes for ‘shortest path’ such as ‘clockwise’ and ‘counterclockwise’. The final argument is the speed, which can take on any value from 1 to 100. When the code below is run, the robot moves the newly attached arm to position 270 or 270 degrees, using the shortest path or the direction in a circle that gets it to position 270 the quickest, at a speed of 10. It then moves to 0 degrees 0 or the lowest position, using the shortest path, at a speed of 10.

```
# Create your objects here
hub = MSHub()
movement_motors = MotorPair(’A’, ’B’)
distance_sensor = DistanceSensor(’D’)
lifting_arm = Motor(’C’)


# Write your program here
hub.speaker.beep()

# You can replace shortest path with
’clockwise’ or ’counterclockwise’
lifting_arm.run_to_posotion(270,
’shortest path’, 10)
lifting_arm.run_to_posotion(0,
’shortest path’, 10)
```

If we wanted to get the robot to complete a task, such as picking up a ball, we can combine the use of the movement motors, distance sensor and the arms. The way this could be executed is by using the run_to_position method first to make sure the arms start out at the lowest point. We can then use the start() method on the movement_motors to get them to start moving. We can then use the wait_for_distance_closer_than method on the distance_sensor and pass in 7 centimeters as a parameter. After the distance sensors reach closer than 7 centimeters to the object, we can use the stop method and then the run_to_position method and get the arms to move to 270 degrees. To set the robot up, we can place a ball in front of the robot on a straight path, a few inches away. When the code below is run, the robot lowers its arms to the lowest position, it then begins to move forward until it is 7 centimeters away from the ball, stops, and then lifts its arms, but does not pick up the ball, because the movement_motors move too fast and end up pushing the ball away from the robot too quickly.

```
# Create your objects here
hub = MSHub()
movement_motors = MotorPair(’A’, ’B’)
distance_sensor = DistanceSensor(’D’)
lifting_arm = Motor(’C’)


# Write your program here
hub.speaker.beep()

lifting_arm.run_to_posotion(0, ’shortest path’, 10)
movement_motors.start()
distance_sensor.wait_for_distance_closer_than(7, ’c
movement_motors.stop()
lifting_arm.run_to_posotion(270, ’shortest path’, 1
```

Luckily, there is a way to change the speed of the movement moreos. Before the motors are started, we can use the set_default_speed method on the movement_motors. The below code will reduce the speed of the movement motors down to a speed of 30, which is slower than the default speed. This way, the movement motors don’t move the robot too quickly, causing it to knock the ball away. When the line of code below is added before the movement motors are started, the motors will move slower.

```
movement_motors.set_default_speed(30)
```

Another motor function that is also useful is being able to use the move() function on the movement_motors object to turn the robot. This function takes three parameters, a length to move, the unit of measurement, and the amount of degrees to turn. When the code below is executed, the robot will move 20 centimeters, turning 60 degrees to the right.

```
# Turn to the right
movement_motors.move(20, ’cm’, 60)
```

2 Discussion

This tutorial offered a lot of insight into how the Mindstorms Spark robot functions and how it can be programmed to complete different tasks. The robot has the ability to move straight, turn right or left, use its distance sensor to detect how close it is to objects in front of it, and it can use an arm to pick up items. The most useful part of this tutorial is learning the methods that allow the movement_motors to make the robot go forward and turn left and right. This project will most likely not require the use of arms since I will not need to pick up items, but it is still handy to understand the motor methods and the types of arguments they take. Although this tutorial only showed a few different capabilities of a mindstorms robot, it provided the most essential information for building and programming a robot that can use wheels to navigate its environment.

Going forward, to have the root execute the plan created by the hierarchical task network planner, I will only need to know how to get the robot to move forward, backward left and right, and this tutorial explained how to make the robot move in all directions, by calling the start() and move() methods on the robot's object and passing in the proper arguments. For the part of the project where I will incorporate sensors, I will use the distance sensor, but I will most likely use multiple. Distance sensors, potentially three: one in the front, one on the right, and one on the left. This array of sensors will be most useful for receiving information on, while traversing a maze with straight edges and 90 degree corners.

3 Software documentation

3.1 Part1

```
# Robot object
hub = MSHub()

# Wheels object
movement_motors = MotorPair('A', 'B')

# Distance sensor object
distance_sensor = DistanceSensor('D')

# Lifting arm object
lifting_arm = Motor('C')

# Make robot beep
hub.speaker.beep()

# Move robot 20 centimeters forward
movement_motors.move(20, 'cm', )

# Turn robot 60 degrees to the right# Turn to
the right
movement_motors.move(20, 'cm', 60)


# Start moving robot
movement_motors.start()

# Stop moving robot
movement_motors.stop()

# Detect a distance closer than 10 centimeters
distance_sensor.
wait_for_distance_closer_than(10, 'cm')

# Move lifting arm to position 270
lifting_arm.
run_to_posotion(270, 'shortest path', 10)
```