**Assignment #2 Report**

CPSC 481 - Artificial Intelligence

**A) Team Members:**

|  |  |  |
| --- | --- | --- |
| **Names** | **Email** | **Percentage** |
| Rocio Salguero | [salgueroroci@csu.fullerton.edu](mailto:salguero.roci@csu.fullerton.edu) | 19.5% |
| Thao Nguyen | [tnguyen29@csu.fullerton.edu](mailto:tnguyen29@csu.fullerton.edu) | 22% |
| Sharayu Shetty | [sharayu333@csu.fullerton.edu](mailto:sharayu333@csu.fullerton.edu) | [19.5%](mailto:sharayu333@csu.fullerton.edu) |
| Long Nguyen | [long261vn@csu.fullerton.edu](mailto:long261vn@csu.fullerton.edu) | 19.5% |
| Sanika Deshpande | sanikadesh198@csu.fullerton.edu | 19.5% |

**B) Strategy (or heuristic):**

Following the tutorial given in O’Kane’s “A Gentle Introduction to ROS”, our team decided to write a simple C++ publisher program which generates and publishes velocity commands to a turtlesim turtle, causing it to move in the desired rectangular path.

The strategy we used to make the turtle move in a rectangular path is to break down the movements. In particular, we only need to have the turtle move in a straight line in two different amount of times to simulate a long line and a shorter one, and the turtle to move in an angular motion to rotate. As such, we only need 2 functions, move() and rotate(). The move() function takes speed and distance as its parameters, and it publishes linear velocity message to turtlesim to make the turtle move in a straight line at the desired speed until it reaches the desired distance. Likewise, the rotate() function takes angular speed and relative angle as parameters, and it publishes angular velocity message to turtlesim to make the turtle rotate at the desired angular speed until the desired relative angle is reached. Note that the relative angle is the angle from where the turtle is at to the desired orientation. For example, if the turtle is located at 90 degrees, a relative angle of 90 degrees will make it rotate until it is oriented at 180 degrees. We also added a degreesToRadians() function to convert an angle in degrees to its radian value, since it is easier to input parameters in degrees, whereas the angular distance is calculated using radian values.

After creating the move() and rotate() function, we call them in our main program, passing the desired values to make it move horizontally to the right, then rotate, move upwards, then rotate, then move to the left, then rotate, and finally move downwards back to its original position.

**Pseudocode:**

// This function moves the turtle with a certain linear velocity for a certain distance

def move(speed, distance):

Create the message vel\_msg using geometry\_msgs package

Initialize vel\_msg; set the linear velocity in the x-axis to the input speed

Set the linear velocity to 0 in y- and z- axis

Set the angular velocity to 0 in all axis

Create and set the initial time, t0, to current time, using ros::Time package

Create and set current\_distance to 0

Create and set rate to 100, using ros::Rate package

Loop until current\_distance = distance

Publish vel\_msg to turtlesim

Create a new time, t1, set it to current time

Calculate current distance, which is equal to speed \* time

Return control back to ros

Delay the program

End loop

Reset linear velocity in the x axis to 0

Publish vel\_msg to turtlesim

// This function makes the turtle turn with a specified angular velocity

def rotate (angular\_speed, relative\_angle):

Create the message vel\_msg using geometry\_msgs package

Initialize vel\_msg; set the linear velocity to 0 in all 3 axises

Set the angular velocity to 0 in the x- and y-axis

Set angular velocity in the z-axis to the input angular\_speed

Create and set the initial time, t0, to current time, using ros::Time package

Create and set current\_angle to 0

Create and set rate to 100, using ros::Rate package

Loop until current\_angle = relative\_angle

Publish vel\_msg to turtlesim

Create a new time, t1, set it to current time

Calculate current\_angle, which is equal to angular\_speed \* time

Return control back to ros

Delay the program

End loop

Reset angular velocity in the x axis to 0

Publish vel\_msg to turtlesim

// This function converts angles from degrees to radians

def degreesToRadians(angle\_in\_degrees):

angle\_in\_radians = angle\_in\_degrees \* PI /180.0

// main program

def main(argc, argv):

Initialize the ROS system

Become a node n

Create a publisher object velocity\_publisher

Initialize the message type to geometry\_msgs::Twist

Initialize topic name to "turtle1/cmd\_vel", and the queue size to 1000

Set the rate at 2Hz until the node is shut down

Call move() to move turtle to the right at speed 1 and distance 5

Call rotate() to rotate turtle 90 degrees anticlockwise at speed 10

Call move() to move turtle upwards at speed 1 and distance 2

Call rotate() to rotate turtle 90 degrees anticlockwise at speed 10

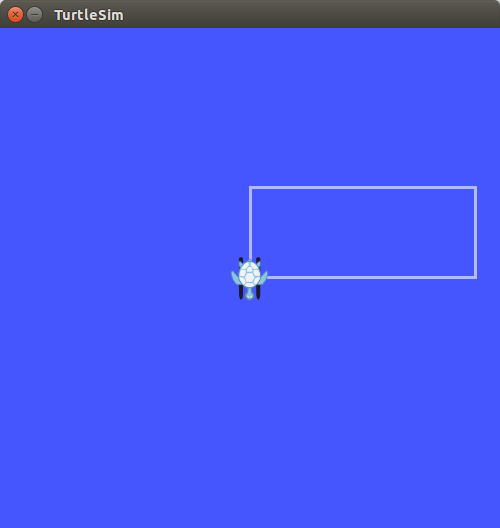
Call move() to move turtle to the left at speed 1 and distance 5

Call rotate() to rotate turtle 90 degrees anticlockwise at speed 10

Call move() to move turtle downwards at speed 1 and distance 2

return

**C) Snapshot of runtime output of “boxturtle.cpp”**

****

**D) References:**

<https://github.com/zshn25/turtlesim_cleaner/tree/master/src>

<http://wiki.ros.org/turtlesim/Tutorials>