EECS 195: Autonomous Systems - Spring 2020
Assignment #1
Due Date: 4/22/2020

Problem 1: my_teleop_node (20 points)

In Lecture 2 we demonstrated how to use a ROS node named "`turtlesim_teleop_key`" to control the turtle in the "`turtlesim_node`". In this problem, you will write your own ROS node named "`my_teleop_node`" that mimic the behavior of the "`turtlesim_teleop_key`" node.

Step 1: create a new workspace folder named `LastName_FirstName_ws` (replace LastName and FirstName with your last and first name)
Step 2: create a package called "`autoturtle`" inside your workspace
Step 3: create a node (under the scripts folder) named "`my_teleop_node`". This node should do the following:

- Subscribe to the topic "`/turtle1/cmd_vel`". This topic uses messages of the type "geometry_msgs/Twist" which can be imported in your Python script using:
  `from geometry_msgs.msg import Twist`
- Write Python code that takes input from the keyboard. Whenever the user hits the "w" key, the turtle should move forward. Whenever the user hits the "s" key, the turtle should move backwards. Whenever the user hits the key "a" then the turtle should rotate to the left without moving. Finally, when the user hit the key "d" the turtle should rotate to the right without moving. Below is a code snippet of how to create a message from type Twist and fill in the important data fields.

```
move_cmd = Twist()
# Linear speed in x in units/second: positive values imply
# forward, negative values == backwards
move_cmd.linear.x = 0.3     # Modify this value to change the
                            # Turtle's speed

# Turn at 0.5 radians/s
move_cmd.angular.z = 0.5    # Modify this value to cause
rotation
                            # rad/s
```

Evaluation:
The grader is going to compile your code using `catkin_make`, source your `setup.bash` file, then run your code using:

```
>> rosrun autoturtle my_teleop_node
```

It is your responsibility to make sure that no compilation errors will take place. Finally, the grader will use the keys “w” “s” “a” “d” to move the turtle.

Problem 2: swim_node (30 points)

In this problem, you are required to create a new ROS node called swim_node. This node should move the turtle in an “8 shape”.

Step 1: Create a new node in the same package `autoturtle` named `swim_node` under the scripts folder.
Step 2: Upon initialization, this node should pick some random linear velocity and some random angular velocity.
Step 3: The turtle then swims in a figure 8 shape using these random velocities.

Evaluation:
The grader is going to compile your code using `catkin_make`, source your `setup.bash` file, then run your code using:

```
>> rosrun autoturtle swim_node
```

It is your responsibility to make sure that no compilation errors will take place. Once the node starts, the turtle should continuously swim in a figure 8 shape.

Problem 3: swim_to_goal (50 points)

In this problem, you are required to create a new ROS node called `swim_to_goal`. This node should take an input from the user specifying the target (x,y) coordinate of the goal. Your node should then move the turtle to this (x,y) position.

Step 1: Create a new node in the same package `autoturtle` named `swim_to_goal` under the scripts folder.
Step 2: Upon initialization, this node will ask the user to enter two numbers called `x_goal` and `y_goal`
Step 3: Calculate the error between the turtle current position `(current_x, current_y)` and the goal `(x_goal,y_goal)`. Recall, the turtle pose can be retrieved by subscribing to `/turtle1/pose` topic. This error can be computed as follows:

- `Error_position = Euclidean distance between (current_x, current_y) and (x_goal,y_goal)`
- `Error_angle = atan2(Error_position)`

Step 4: Set the turtle velocity to be proportional to the error, i.e., when the turtle is far away from the goal it should move faster than when the turtle is near the goal, and should not move when it arrives to the goal. You can achieve this as follows:

- `Linear_velocity = K_x * Error_position`
- `Angular_velocity = K_z * Error_angle`
  where `K_x` and `K_z` are some constants that you can choose. You may use `K_x = 1.5` and `K_z = 4`. Once you calculate the velocities, you can publish them on the `/turtle1/cmd_vel` topic.

Step 5: Check if `Error_position` is smaller than 0.5, then you can stop moving the turtle. Else, go to Step 3.
Step 6: When the turtle arrives to the final goal, it should ask the user for a new `x_goal` and `y_goal`and then move the turtle accordingly.

Evaluation:
The grader is going to compile your code using `catkin_make`, source your `setup.bash` file, then run your code using:

```
>> rosrun autoturtle swim_to_goal
```

It is your responsibility to make sure that no compilation errors will take place. Once the node starts, the grader will ask the turtle to move to some random position. The grader will repeat this process three times.

Deliverable:

- One zip file that contains your workspace and all the three ROS nodes.
- Ensure, the code you are submitting does not throw any errors during the `catkin_make`. Otherwise, we won't be able to grade your assignment.