Lab 3: Open Loop Control

Yvon Kim: 862331596

Lab Section: Wednesday

GITHUB FOR CODE: https://github.com/ykim336/ee106_ros

+50%: Explaining My Approach

```
#!/usr/bin/env python3
import rospy
from geometry_msgs.msg import Twist
from math import pi
```

(1) I import all the necessary libraries, including the near accurate pi.

```
LINEAR_SPEED  = 0.1 # this determines the speed
ANGULAR_SPEED = pi / 8 # this determines angular velocity
HZ            = 20
SIDE_LENGTH   = 1 #this determines the distance
TURN_ANGLE    = pi / 2 # this is 90 degrees
```

(2) I define the important parameters. In this case, I determine the rate of the speed and rotation of the robot. In this case, I set the distance to 1 and the angle to be 90 degrees (eventually making it a square).

```
class Turtlebot:
        def __init__(self):
        rospy.init_node("turtlebot_simple_square", anonymous=True)
        self.vel_pub = rospy.Publisher("cmd_vel", Twist, queue_size=10)
        rospy.sleep(0.5)
        self.rate = rospy.Rate(HZ)
        rospy.loginfo("TurtleBot Simple Square: Node Started. Driving a square.")
        self.run()
        rospy.signal_shutdown("Square complete.")
```

(3) Firstly, I create a class and make an initialization statement. I programmed it such that when Turtlebot is initialized, then it will initialize a node and call a publisher. Then, the sleep function (which is important or it won't work) allows some time to connect. Then, I determine the rate/clock of the model. Finally, I use the loginfo function to print out the statement in the terminal so I know what's going on. This is important for convenience and debugging. Finally, I call the run function – which has yet to be explained – which will create the *imperfect square movement. When it is done, it will put a signal out to indicate completion.

```
def _publish_for_duration(self, twist_msg, duration_seconds):
        ticks = int(duration_seconds * HZ)
        for _ in range(ticks):
                if rospy.is_shutdown():
                return False
                self.vel_pub.publish(twist_msg)
                self.rate.sleep()
        return True
```

(4) I define the clock such that the model is on point. It also checks if the server is alive. This will continuously check if the model is finished or not. It serves as an overhead of the entire operation.

```
def run(self):
        move_cmd = Twist()
        move_cmd.linear.x = LINEAR_SPEED

        turn_cmd = Twist()
        turn_cmd.angular.z = ANGULAR_SPEED

        stop_cmd = Twist()

        linear_duration = SIDE_LENGTH / LINEAR_SPEED # determines the m/s
        angular_duration = TURN_ANGLE / ANGULAR_SPEED

        for i in range(4):
                rospy.loginfo(f"Side {i+1}: Moving forward.")
                if not self._publish_for_duration(move_cmd, linear_duration): return
                if not self._publish_for_duration(stop_cmd, 0.2): return # Brief stop

                if i < 3: # Turn for the first 3 corners
                rospy.loginfo(f"Side {i+1}: Turning.")
                if not self._publish_for_duration(turn_cmd, angular_duration): return
                if not self._publish_for_duration(stop_cmd, 0.2): return # Brief stop

        # Final stop
        for _ in range(HZ // 2): # Stop for 0.5 seconds
                if rospy.is_shutdown(): break
                self.vel_pub.publish(stop_cmd)
                self.rate.sleep()
```

(5) Time for the big boy code. Basically, it first calls a twist which basically holds the movement information. Then it executes the linear and angular motion. It also calculates the duration of each type of movement. Then to create a square, I made a for loop which displays the info in the terminal, checks if the server is alive, and then moves in the respective order. Finally, it stops for 0.5 seconds and then sleeps after publishing a stop command. It's pretty cool!

```
if __name__ == "__main__":
        try:
        Turtlebot()
        except rospy.ROSInterruptException:
        rospy.loginfo("Execution interrupted.")
        except Exception as e:
        rospy.logerr(f"An unexpected error occurred: {e}")
```

(6) The final piece of code actually runs the thing. We created a class, so we need to declare it and actually use it. Here, we also made a try and except instance to make sure everything moves with certainty.

+40%: Robot Visits All Four Vertices, +10%: Exits Gracefully

```
VIDEO_DEMONSTRATION = 'https://youtu.be/bkRA4x7zVFE'
```

(7) Here, I actually run the code! I uploaded it on youtube so you can watch it since I'll be turning this in as a PDF report. Note: the robot isn't perfect, but due to the lack of closed loop integration, it is off. However, it is good enough.