Lab 1

Zeta Group:
Justin Evaristo
Aditya Malik
Ben Poreh
Andrew Wood

Our lab was focused on learning how to connect to and control a turtlebot using ROS. The instructions were rather simple, develop code to drive the turtlebot around in a 1 meter by 1 meter square, and observe how the turtlebot behaves.

Put simply, our code is a “blind” controller, in the sense that it publishes commands and expects them to be carried out instantaneously exactly as they are given without any sort of feedback evaluating the success of the commands. This type of control scheme is better suited for a simulated environment in which the assumptions stated above about the command and execution relationship are valid. In the real world, commands are not followed instantaneously, and often they do not succeed. The result is an ineffective and error prone control system and inaccurate control of robot pose (position and orientation).

To examine this, our controller algorithm works as follows:

```
forward_time := 4                           // seconds
forward_speed := 1.0 / forward_time         // will move 1.0 meters in forward_time seconds
turning_time := 3                           // seconds
yaw_rate := PI / (2 * turning_time)         // will turn PI / 2 in turning_time seconds

while ROS system is running:
    move_forward(forward_speed, forward_time)

    rotate_about_z(yaw_rate, turning_time)
    move_forward(forward_speed, forward_time)

    rotate_about_z(yaw_rate, turning_time)
    move_forward(forward_speed, forward_time)

    rotate_about_z(yaw_rate, turning_time)
    move_forward(forward_speed, forward_time)

    rotate_about_z(yaw_rate, turning_time)
```

This algorithm will, in theory, tell the turtlebot in a perfect square and end in a pose equal to the initial pose of the turtlebot. The turtlebot will make these square paths until ROS is turned off, either when the turtlebot is powered off, the turtlebot and the machine running this algorithm loses Internet connectivity, or the user shuts down ROS.

This algorithm does not work very well in directing the turtlebot into a perfect square. Based off of the 1 square foot tiles that the floor is made of, during the first pass alone of the square maneuver, the robot performed as follows:

Square 1 linear movement (goal) (meters)	Square 1 linear movement (actual) (meters)	Square 1 linear movement (% error)
1.0	0.91	9%
1.0	1.067	6.7%
1.0	1.22	22%
1.0	1.067	6.7%

Square 1 angular movement (goal) (degrees)	Square 1 angular movement (actual) (degrees)	Square 1 angular movement (% error)
90	50	44.4%
90	65	27.8%
90	95	5.6%

As these charts show, the accuracy of our controller was abysmal, resulting in the turtlebot not tracing out a concise square. Something that is interesting to note is that when watching our video, the turtlebot does not move in a straight line, in reality it lists as is moves linearly.

This listing in fact causes the turtlebot to end up closer to the intended waypoints than we thought possible. Due to the geometry of the listing, when the robot makes a shallow turn, the listing corrects the shallow turn and overall moves the robot closer towards the intended waypoint. The only unintended side effect of listing is that the orientation of the turtlebot after the listing is now non-deterministic. We believe that the error in the behavior of the turtlebot is due to several factors:

1) Wheel slip. When the turtlebot moves, the wheels can slip just like in a car with no traction. This causes discontinuity between the controller and the actual state of the turtlebot by interrupting the timing of the current command.

2) Commands are not executed instantaneously. When commands are given, the turtlebot has to physically move. There is a delay for the turtlebot reaching the intended linear or angular velocity as physically the turtlebot cannot instantaneously start moving at the given velocity. This also causes an interrupting the timing of the current command and creates a discontinuity between the controller and the actual state of the robot.

3) Commands are not stopped instantaneously. When a command is executed (and the robot reaches the commanded velocity), the command is executed until another one replaces it. When this occurs, the robot still moves in the direction of the previous command until the robot reaches a complete stop. This is not accounted for in the functionality of the controller, which creates a discontinuity between the controller and the actual state of the robot.

Overall, the controller implemented is not an effective controller of the turtlebot. Our code can be found at https://github.com/aew61/eecs376 and a video of our code controlling the turtlebot can be found at https://www.youtube.com/watch?v=jIA8FkwBnL4&feature=youtu.be