Michigan Technological University

MEEM 4707: Autonomous system

Spring 2024

Lab – 3

By

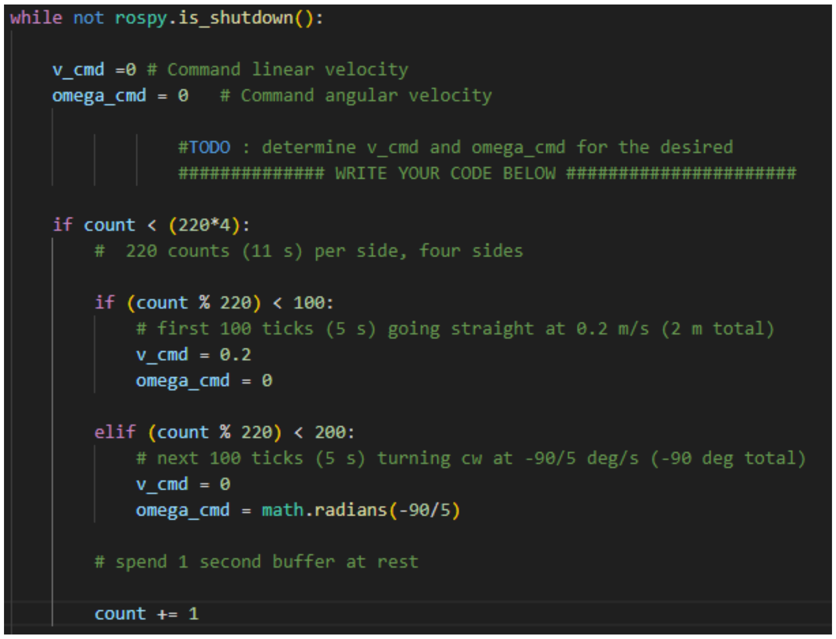
Colton Kreischer
Amanda West

Name	Logic Build	Coding	Report Writing	Total
A	50%	50%	50%	150%
B	50%	50%	50%	150%

Problem 1

1. Set up a timing-based controller to draw a square of dimension 0.5 m, counterclockwise, once.
2. Set up a timing-based controller to draw a square of dimension 1 m, clockwise, once.

Capture your trajectories in the real world and include them in your report. (You may not be able to follow the squares exactly, so you don't need to try to have exact square routes.)



```
while not rospy.is_shutdown():

    v_cmd =0 # Command linear velocity
    omega_cmd = 0   # Command angular velocity

            #TODO : determine v_cmd and omega_cmd for the desired
            ############# WRITE YOUR CODE BELOW #####################

    if count < (220*4):
        #  220 counts (11 s) per side, four sides

        if (count % 220) < 100:
            # first 100 ticks (5 s) going straight at 0.2 m/s (2 m total)
            v_cmd = 0.2
            omega_cmd = 0

        elif (count % 220) < 200:
            # next 100 ticks (5 s) turning cw at -90/5 deg/s (-90 deg total)
            v_cmd = 0
            omega_cmd = math.radians(-90/5)

        # spend 1 second buffer at rest

        count += 1
```

Figure 1: Original code used to move the robot in a square based solely on timing.

The code in Figure 1 was used to generate the paths in Figure 2 on the real robot, with variable values changed for directionality and side length. This was clearly imprecise without feedback or some form of tuning.

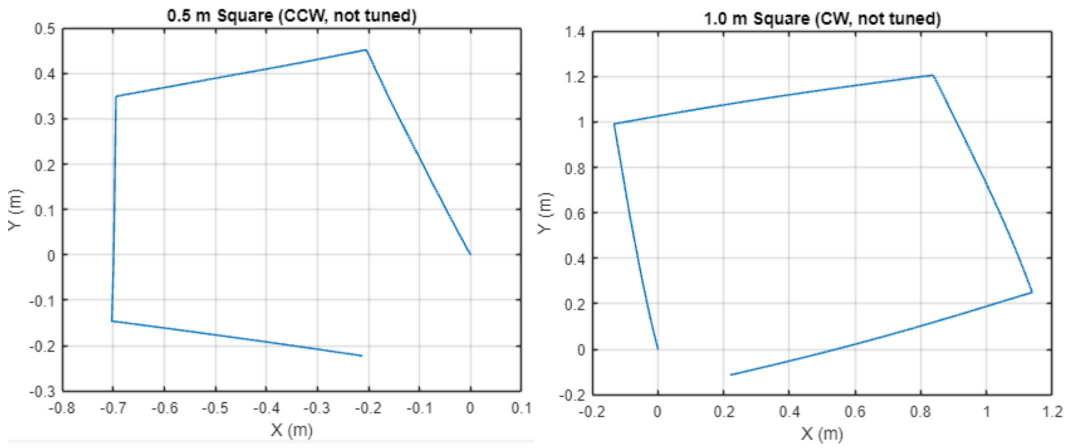
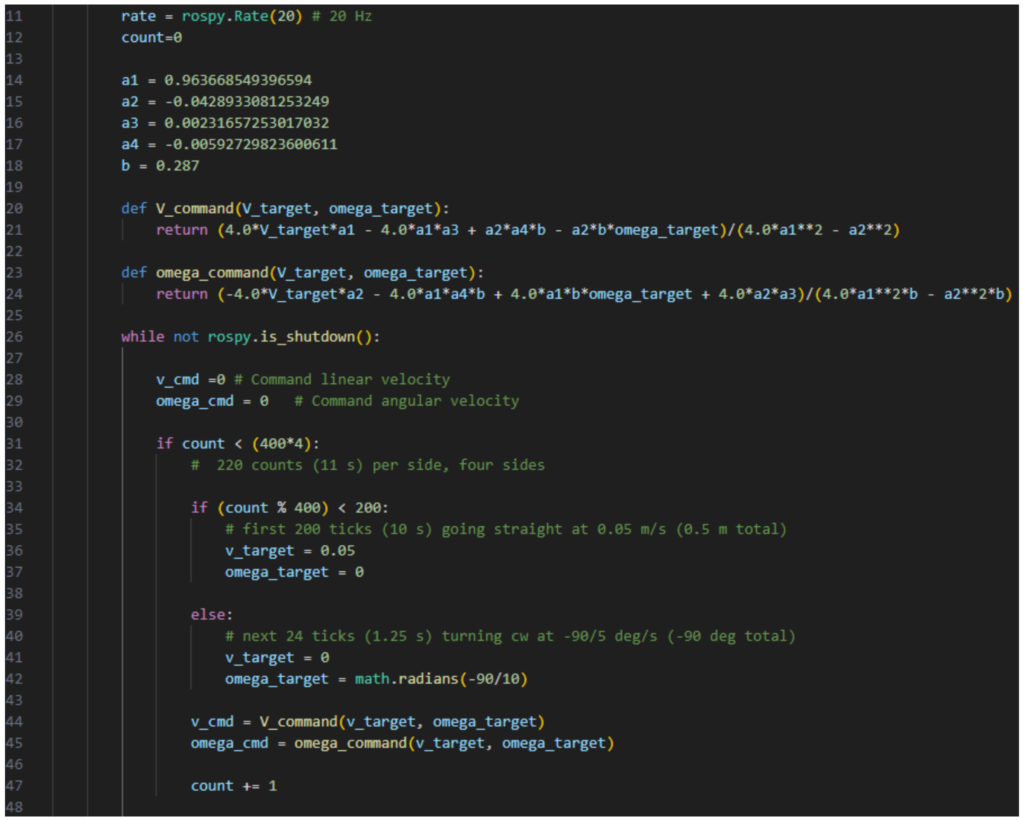


Figure 2: Squares drawn by the robot in real life, shown as timeless paths.

Problem 2. In pre-lab, you have calculated tunning parameters. During the lab session, use the tuned parameters ($a1$, $a2$, $a3$, and $a4$) **in Gazebo** to let the robot follow a 0.5-meter square counterclockwise once. **Capture your desired trajectory and the robot's estimated (recorded) trajectory in Gazebo and include them in your report.**

$$V_{actual} = a_1 \times V_{command} + 0.25 \times a_2 \times b \times \omega_{command} + a_3$$
$$\omega_{actual} = \frac{a_2}{b} \times V_{command} + a_1 \times \omega_{command} + a_4$$

The code shown in Figure 3 was used to generate the path shown in Figure 4. After inverting the above equations (shown on lines 20-24 in Figure 3), and using parameters tuned at the same speeds, the program was able to draw the square much more accurately.



```
11          rate = rospy.Rate(20) # 20 Hz
12          count=0
13
14          a1 = 0.963668549396594
15          a2 = -0.0428933081253249
16          a3 = 0.00231657253017032
17          a4 = -0.00592729823600611
18          b = 0.287
19
20          def V_command(V_target, omega_target):
21              return (4.0*V_target*a1 - 4.0*a1*a3 + a2*a4*b - a2*b*omega_target)/(4.0*a1**2 - a2**2)
22
23          def omega_command(V_target, omega_target):
24              return (-4.0*V_target*a2 - 4.0*a1*a4*b + 4.0*a1*b*omega_target + 4.0*a2*a3)/(4.0*a1**2*b - a2**2*b)
25
26          while not rospy.is_shutdown():
27
28              v_cmd =0 # Command linear velocity
29              omega_cmd = 0   # Command angular velocity
30
31              if count < (400*4):
32                  #  220 counts (11 s) per side, four sides
33
34                  if (count % 400) < 200:
35                      # first 200 ticks (10 s) going straight at 0.05 m/s (0.5 m total)
36                      v_target = 0.05
37                      omega_target = 0
38
39                  else:
40                      # next 24 ticks (1.25 s) turning cw at -90/5 deg/s (-90 deg total)
41                      v_target = 0
42                      omega_target = math.radians(-90/10)
43
44                  v_cmd = V_command(v_target, omega_target)
45                  omega_cmd = omega_command(v_target, omega_target)
46
47                  count += 1
48
```

Figure 3: *New code used for moving in a square, taking into account the previously found a_1-a_4 coefficients.*

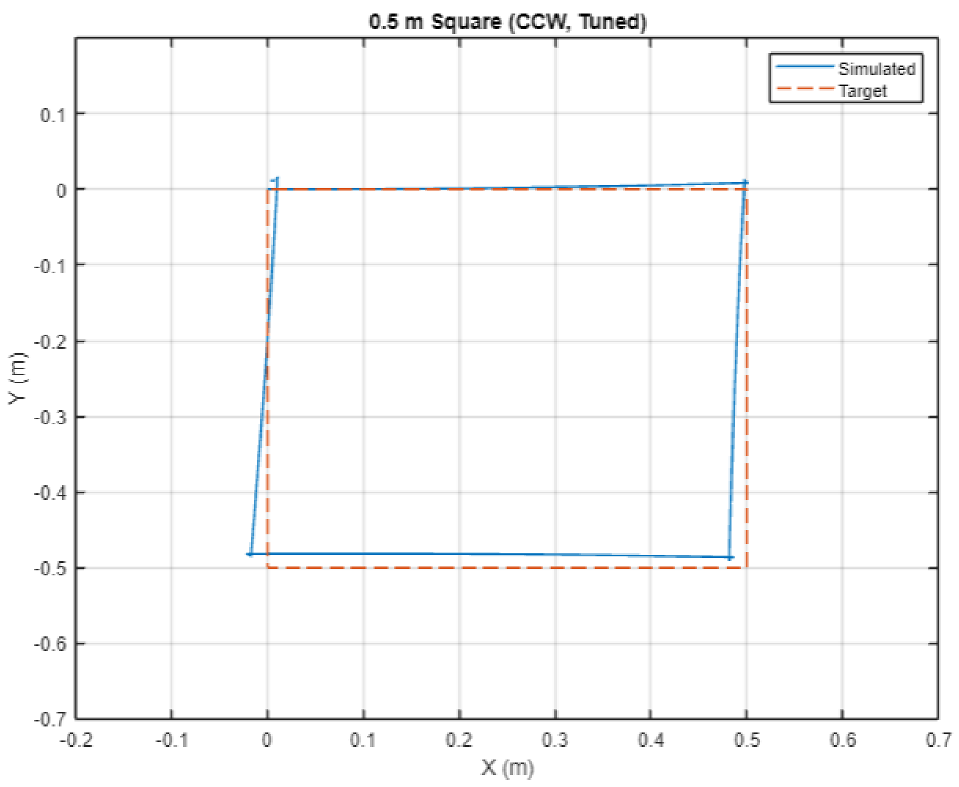


Figure 4: *Square path simulated and recorded using the tuned a_1-a_4 coefficients.*

Discussion from Lab3

- Objective

The objective of this lab is to practice tuning open-loop dead reckoning control, as well as to become familiar with the error caused by open-loop control. Using 1:1 analysis between simulated and experimental trajectories, participants should be able to improve the accuracy of the open-loop control.

- Approach to achieve the Objective

To get the original data, a ROS node would be used to repeatedly command linear velocity and angular velocity to draw a square. Using this data with the equations above, the coefficients a_1-a_4 could then be solved for. Finally, the inverse of the equations may be used (where command and actual values are flipped) to attempt to negate the disparities between the target and actual values.

- Challenges faced and countermeasures taken

Once the inverted equations were in place, there was still significant (if not worse) disparity from the target path. The speeds were reduced to manage this (10 s linear, 10 s angular), however there was still disparity that the tuned controller did not resolve. Re-tuning the coefficients at the new speed, however, did resolve it to a high degree of accuracy. This likely indicates that the coefficients are directly related to the originally run speed, meaning that future tuning must be done on a case-to-case basis.

• The difference in strategy: Pre-lab vs. Lab strategy

While I originally expected the tuned coefficients to be linked only to the system itself, representing mechanical properties that would be valid under any combination of speeds, it appears that the tuned dead-reckoning controller is not so robust. Minor changes led to tuned parameters no longer being valid, so our strategy had to be adjusted accordingly (re-tune for every different speed/distance).

• Observations and Learnings

Though the controller is still not closed-loop, it performed much better than I expected it to using only 4 tuned parameters. While future labs will likely incorporate closed-loop control, this lab has shown me that tuned open-loop controllers have plenty of practical applications for consistent systems which do not warrant the addition of a sensor for feedback.