CS 7639 001 Project 1: Robotarium and Feedback Control

Modeling

1) a. Relationship between wheel rpm, ω_{wheel}, and forward velocity of robot, v, and radius of wheel, r.

   v = forward velocity m/s

   r = 0.015 m (radius of wheel 1.5 cm)

   ω_{wheel} = wheel angular velocity rad/s = $2\pi/60$ * rpm rpm needs to convert to rad/s

   $v = r * \omega_{wheel} = 0.015 * 2\pi/60 * rpm$

   Or

   $v = r * (\omega_{wheel}(rad/s) * 60/2\pi)$ # rad/s to rpm

   b. Relationship between translational velocity of wheels, v_{wheel},and angular velocity of robot, ω.

   v_R = right wheel speed

   v_L = left wheel speed

   d = distance between two wheels

   $\omega = (v_R - v_L)/D$

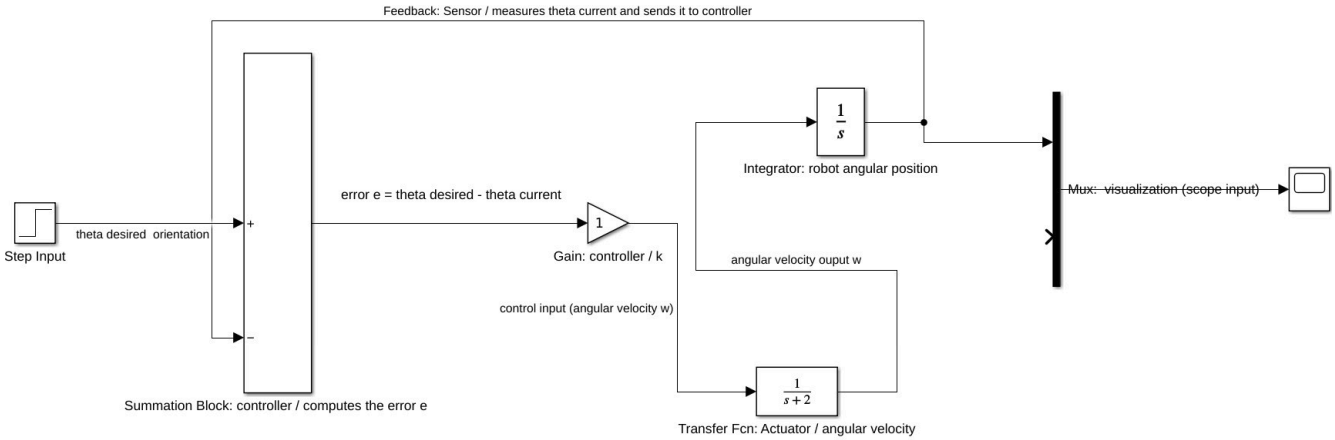
   Robot spinning at its center, left and right wheel spin opposite directions but same speed

   $v_R = +v_{wheel}$, $v_L = -v_{wheel}$

   $v_R - v_L = (+v_{wheel}) - (--v_{wheel}) = 2v_{wheel}$

   $\omega = (v_R - v_L)/d = (2v_{wheel})/d$

2)



3) **Sensor**: measures robot's real time position and updates it to controller with feedback
Finds out the robot's current orientation (theta current) and provides feedback to the controller. In the code, it's p = r.get_poses() which is getting the real-time position and orientation of the robot, which then sends this feedback to the controller to adjust the motion continuously. In the block diagram, it's the integration block 1/s to get theta current and sends it back to controller using line between integrator block and summation block

Actuator: Applies correction to robot, converts control input (angular velocity ω) into the robot's physical movement, adjusting its trajectory. In the code, it would be u = np.array([velocity], [omega]]) which sends computed control input to the robot, executing movement based on velocity and angular ω. In the block diagram, it's the line between transfer function block and integrator block.

Controller: Measures how much correction is needed to reach target orientation
It computes the error e = theta desired - theta current, generates control input angular velocity ω, and also adjusts movement using feedback from the sensor. In the code, it's the function feedback_control(p, target_pose) which computes angular error and uses gain k to correct robot orientation. In block diagram, it's the line between Gain controller k and summation block

Theoretical Number of of Iterations needed in Simulation Task 3

v = 8 cm/s = 0.08 m/s
time_step = 0.033 sec (30Hz)
d = ? distance between target 1 and target 2

```
target1 = np.array([[.5],[0],[np.pi]])
target2 = np.array([[-0.5],[0],[np.pi]])
```

Robot's movement is along the X-axis
d = target2 - target1 = |x2 - x1| = -0.5 - 0.5 = 1 m

```
def num_of_iterations_target2(distance, speed):             #find total
number of iterations needed to reach a given distance
    return int(np.ceil(distance / (speed * time_step)))
```

speed = v
iterations (i) = d/(v*time_step) = 1 m / 0.08 m/s * 0.033 s) = 1 / 0.00264 = 378.78.
Roughly 379 iterations

Data Analysis

1.

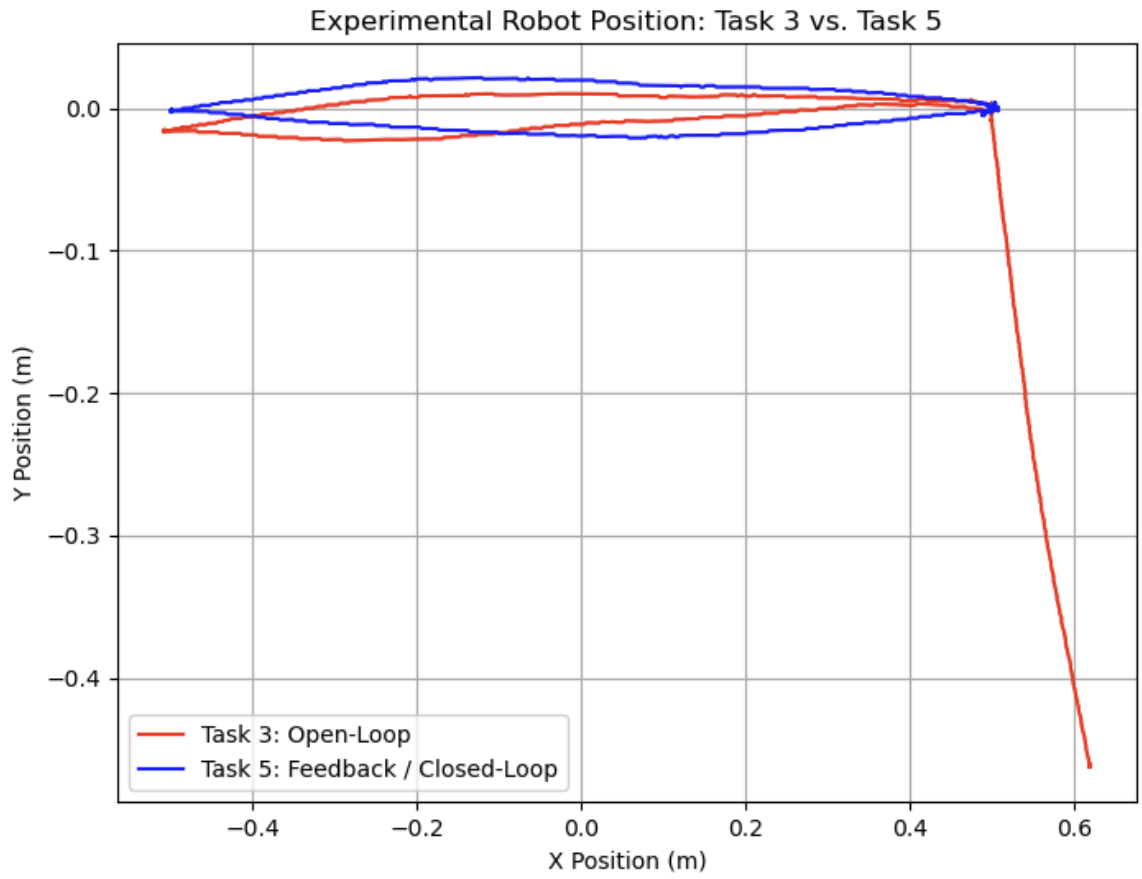


Figure 1: Experimental Position Data of Task 3 and 5

3a.

In Figure 1, Task 3 shows deviation from the target 2 path at -0.5 x position. The robot follows a predicted path but then drifts away as seen in the red line trajectory shifting downwards. If there isn't feedback, the robot can make errors over time, which leads to deviation.

Task 5 reaches the correct positions for target 2 and 1. When the robot initially starts from target 1, it drives off and feedback control fixes that deviation, keeping it closer to the path to target 2.

Based on these findings, open-loop control results in more errors and larger positional drifts if there aren't any corrections.

Feedback closed-loop control creates these corrections when deviation occurs. It will fix the offset by adjusting the robot's orientation and position, resulting in less errors and remaining closer to the path.

2.

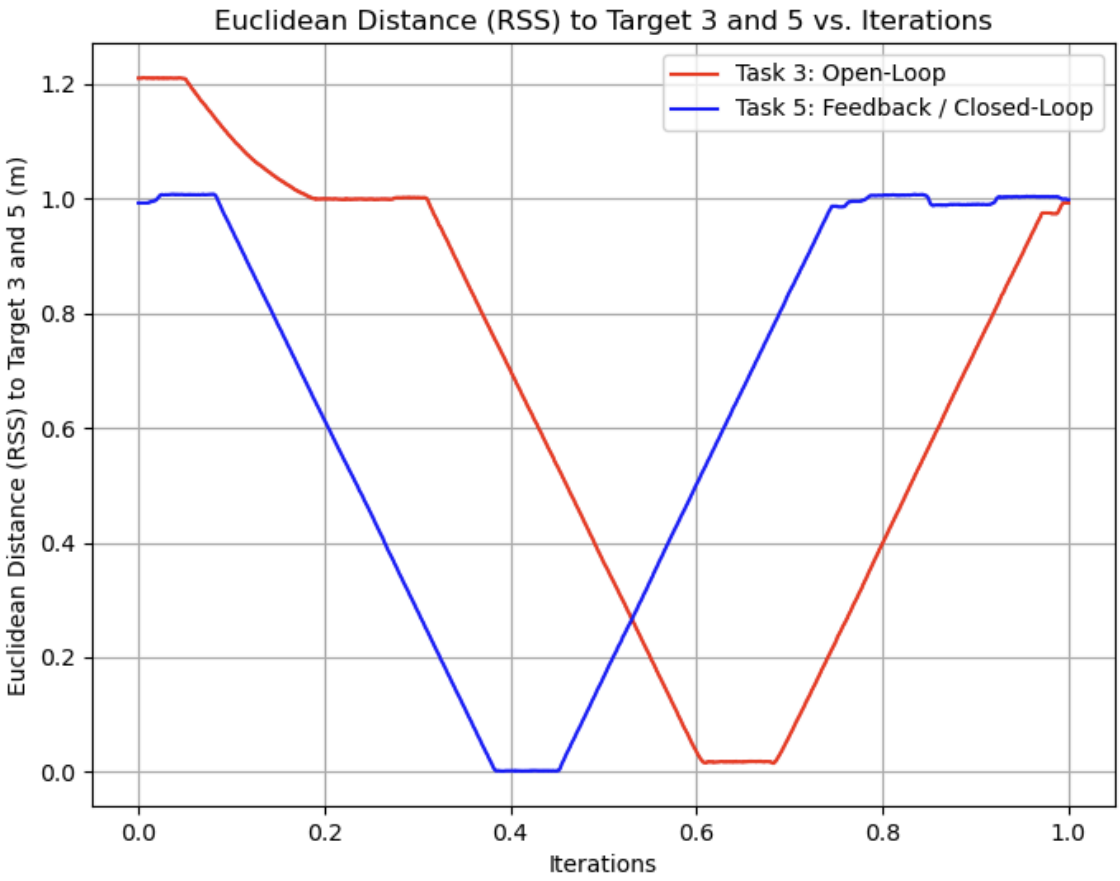


Figure 2: Distance to Target 3 and 5 against Iterations

3b.

In Figure 2, Task 3 distance is gradually decreasing as the robot moves towards target 2. Once it reaches target2, it transitions to the next movement state. There are higher iterations utilized when the robot goes to target 2, which is due to build up of errors. Task 5 distance uses less iterations compared to Task 3 due to error corrections over time, and it's a smoother line. When the robot goes back to target 1, it corrects itself even more unlike Task 3.

Based on these findings, Task 3 robot moves forward but takes awhile to reach target 2, resulting in fluctuation due to the deviations. Task 5 corrects the robot's orientation and position and shows a more smooth and direct path when it reaches to target. It's more precise when it returns to target 1