ME 425 Lab#2 Post Lab Report

Arda Gencer
34124

February 11, 2026

1 Introduction

In lab # 2, The odometric estimation code that was developed in Lab # 1 was developed further and a motion control system that performs a parking motion was developed and integrated into the previous code. Then, this algorithm was tested and tuned with different parameter values and also tested for different starting pose values.

2 Procedure

First of all, the setup with the Turtlebot was made as indicated in the previous lab documents, then the same odometry tracking algorithm was implemented inside the MATLAB code so that the robot will be able to track its own position and adjust the wheel velocities to make itself go where the user wants it to park.

Before going on to the next step, how the control law will be implemented inside MATLAB should be discussed:
From the old odometry program, the following quantities will be used:

$x(k), y(k), \theta(k)$: x, y positions and θ value in k th time step (estimated by the odometry algorithm)

1. By using the polar coordinate transformation that is shown in class and also given in the lab document, the new parameters that is shown on the Figure 1 are defined to represent the pose of the robot:

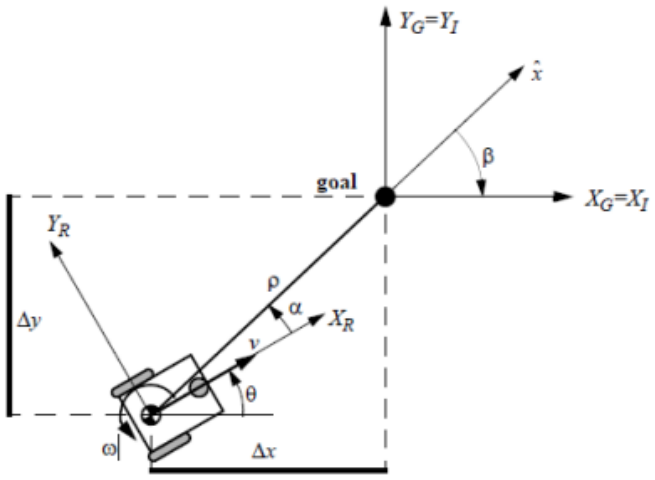

Figure 1: Robot Kinematics after Polar Transformation

$$\Delta x = x_g - x(k) \text{ and } \Delta y = y_g - y(k)$$
$$\rho = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$
$$\alpha = -\theta + \text{atan2}(\Delta y, \Delta x)$$
$$\beta = -\theta - \alpha$$

2. After obtaining the parameters in polar coordinates, the control law that is also derived in class and given in the lab document was applied:

$$v = k_\rho \rho$$

$$\omega = k_\alpha \alpha + k_\beta \beta$$

where k_ρ, k_α and k_β are the controller parameters that will be tuned manually before running the program.

Now, in the MATLAB script, at the start of the program, and also for each iteration inside the while loop, these operations will be performed after the odometric estimation. That means that after each iteration, depending on the pose of the robot at that iteration, a new v and ω value will be determined and sent to the robot as a ROS2 message. Also, during each iteration, the x, y, θ and ρ values are recorded into different vectors for the plotting and analysis of this data for later.

After the program is done, the data that was received and saved during the runtime of the program is plotted using the code that was given in the lab document. Each od these plots for each different run with the robot that was done during the lab can be found in the Results section of the report.

After the controller was completed, the script was run several times by changing the controller parameters and starting pose (location and orientation) of the robot and the effect of these changes was tested and observed. The results of these tests will be discussed further in the Discussion section of this report.

3 Results

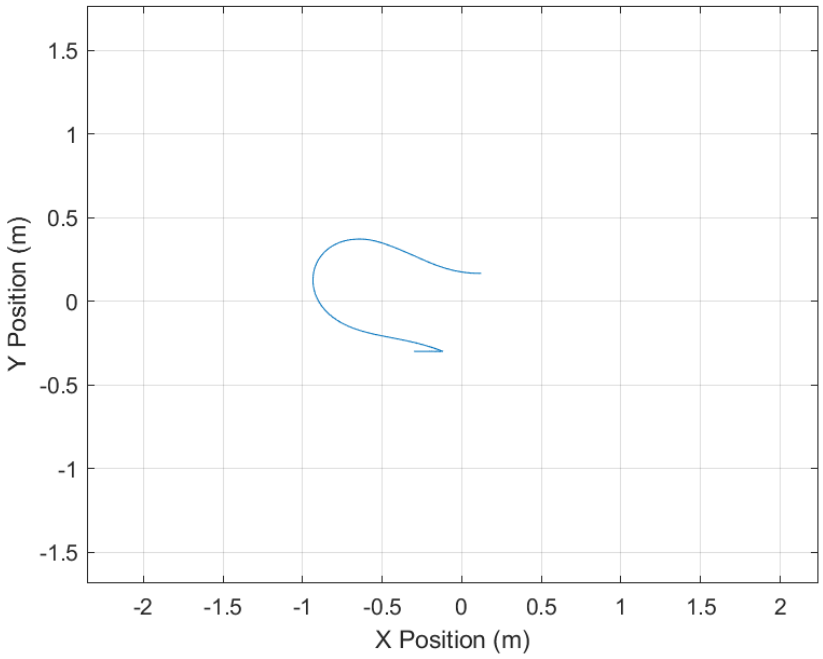

Figure 2: The trajectory taken for parameters: $k_p = 0.25, k_\alpha = 20, k_\beta = 20$

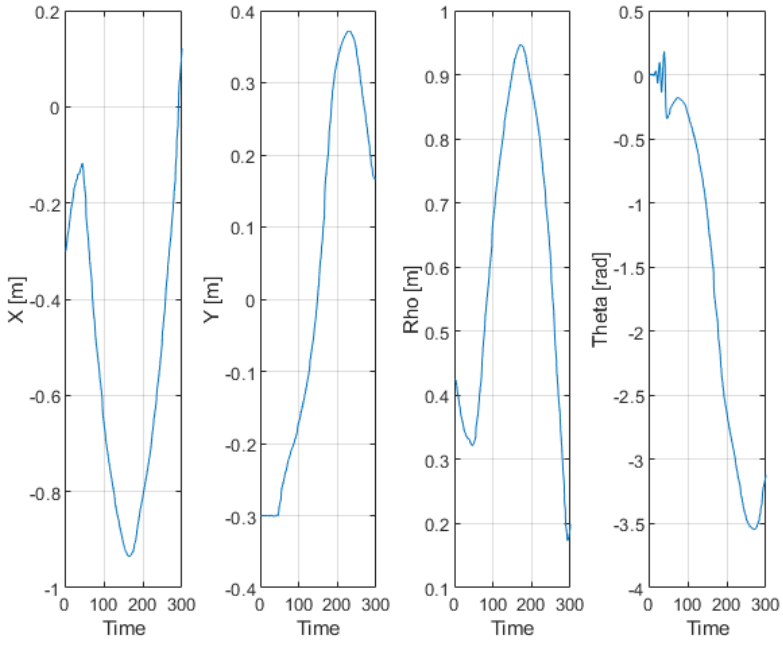

Figure 3: The X, Y, ρ and θ values for parameters: $k_p = 0.25, k_\alpha = 20, k_\beta = 20$

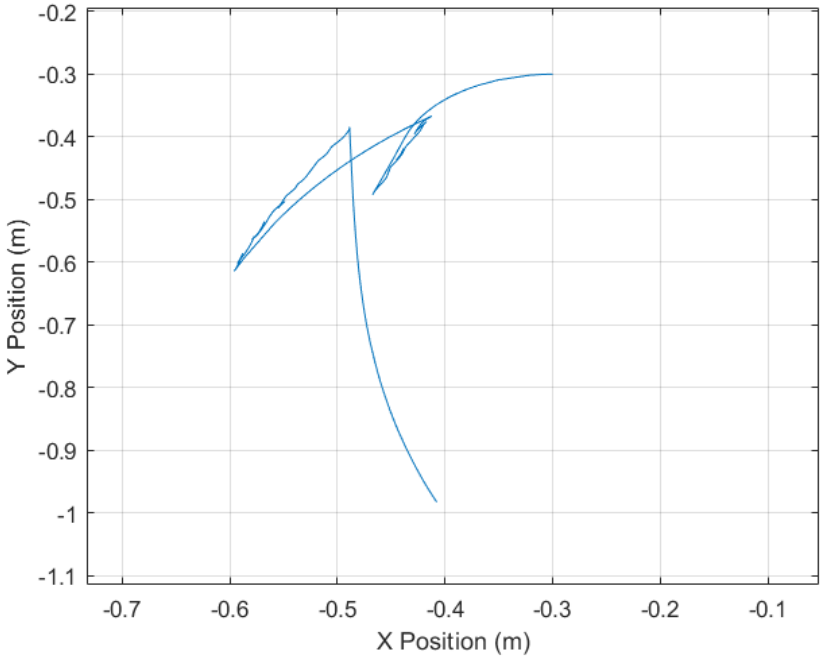

Figure 4: The trajectory taken for parameters: $k_p = 0.15, k_\alpha = 20, k_\beta = -10$

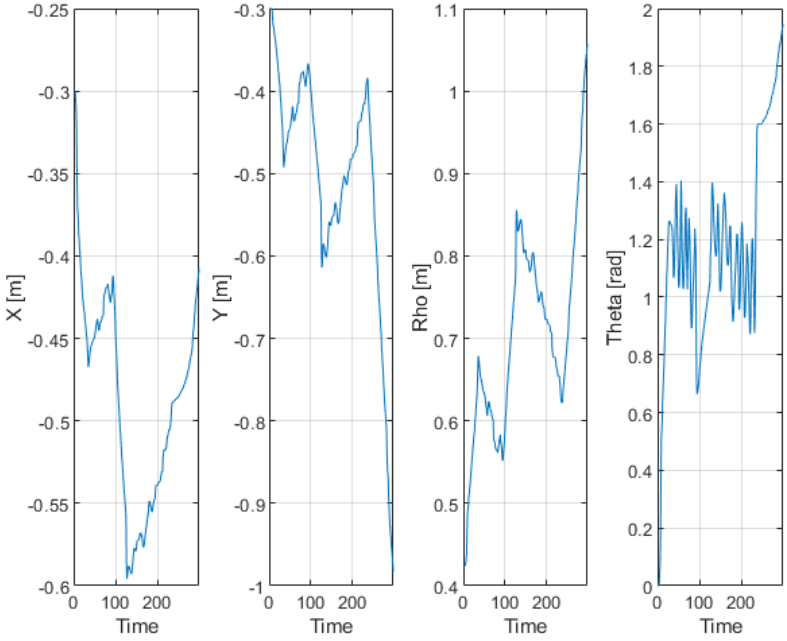

Figure 5: The X, Y, ρ and θ values for parameters: $k_p = 0.15, k_\alpha = 20, k_\beta = -10$

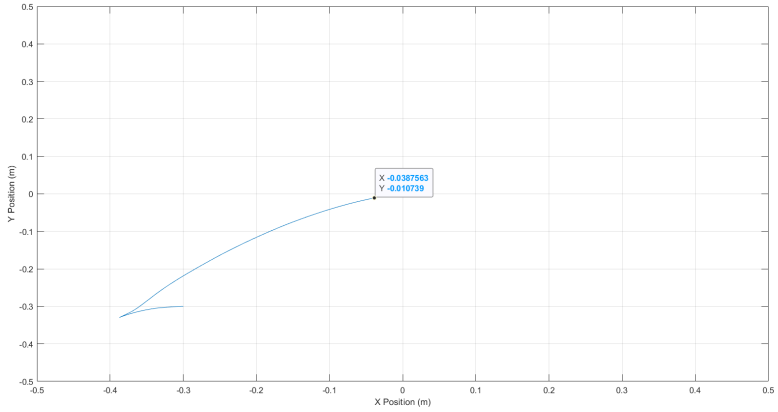

Figure 6: The trajectory taken for parameters: $k_p = 0.25, k_\alpha = 3, k_\beta = -1$

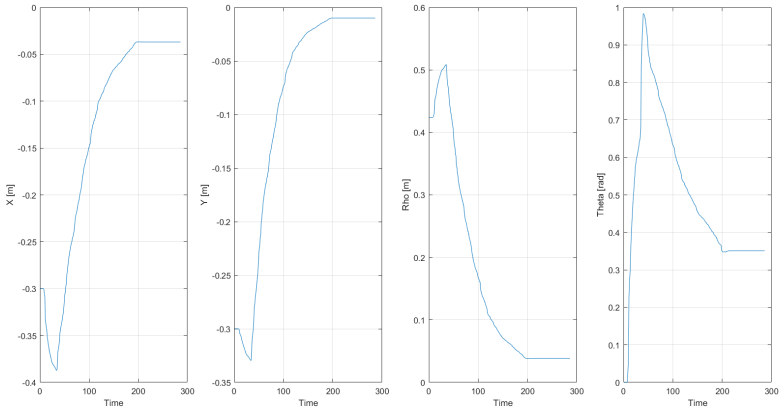

Figure 7: The X, Y, ρ and θ values for parameters: $k_p = 0.25, k_\alpha = 3, k_\beta = -1$

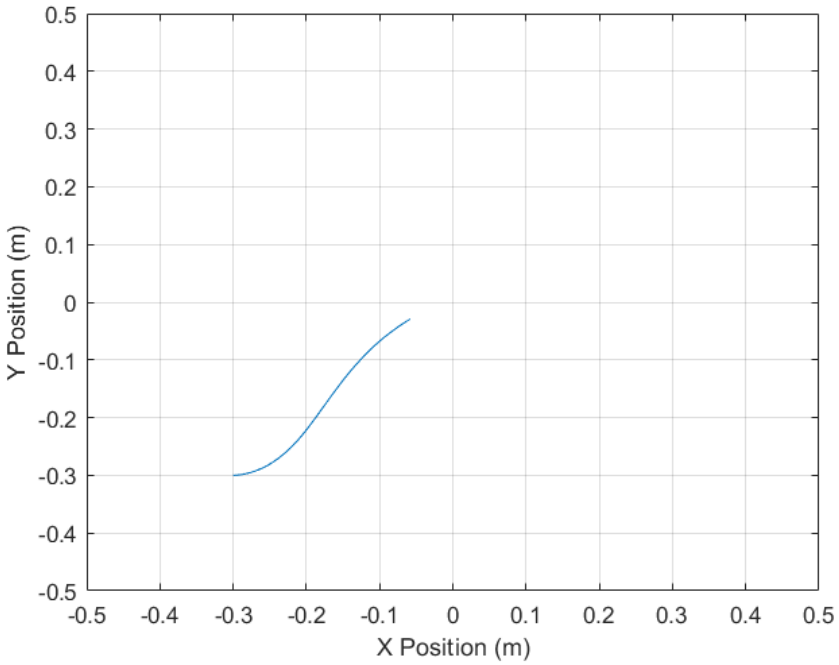

Figure 8: The trajectory taken for starting conditions: $X = -0.3, Y = -0.3, \theta = 0$

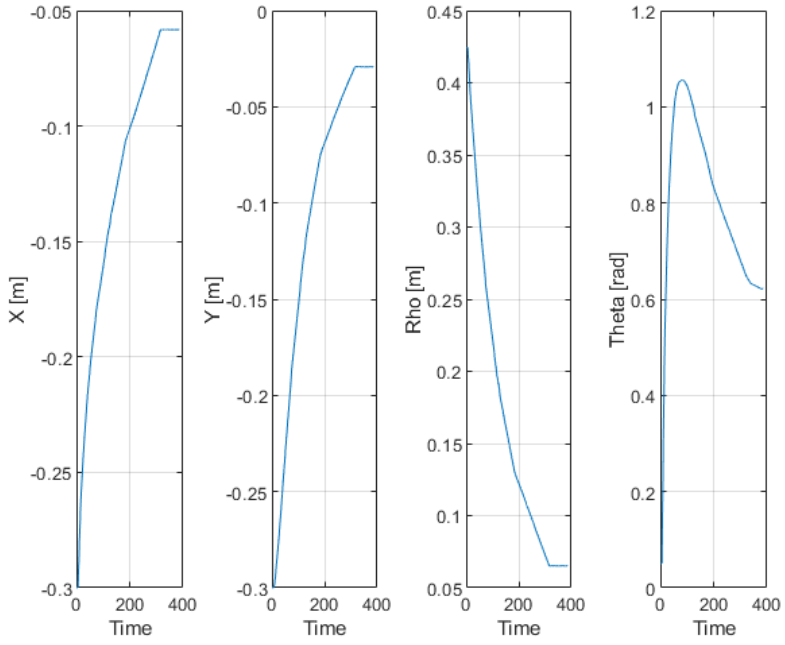

Figure 9: The X, Y, ρ and θ values for starting conditions: $X = -0.3, Y = -0.3, \theta = 0$

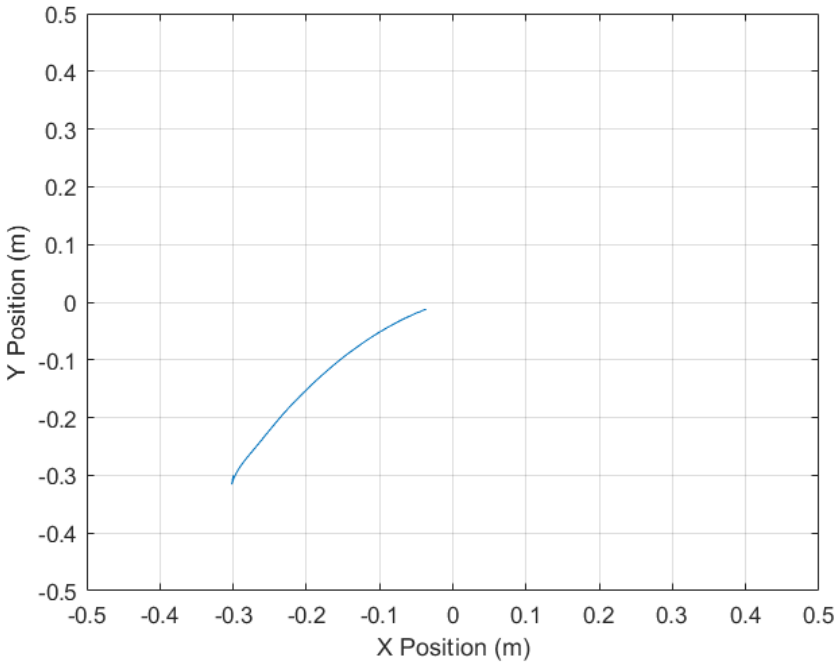

Figure 10: The trajectory taken for starting conditions: $X = -0.3, Y = -0.3, \theta = 1.5\ rad$

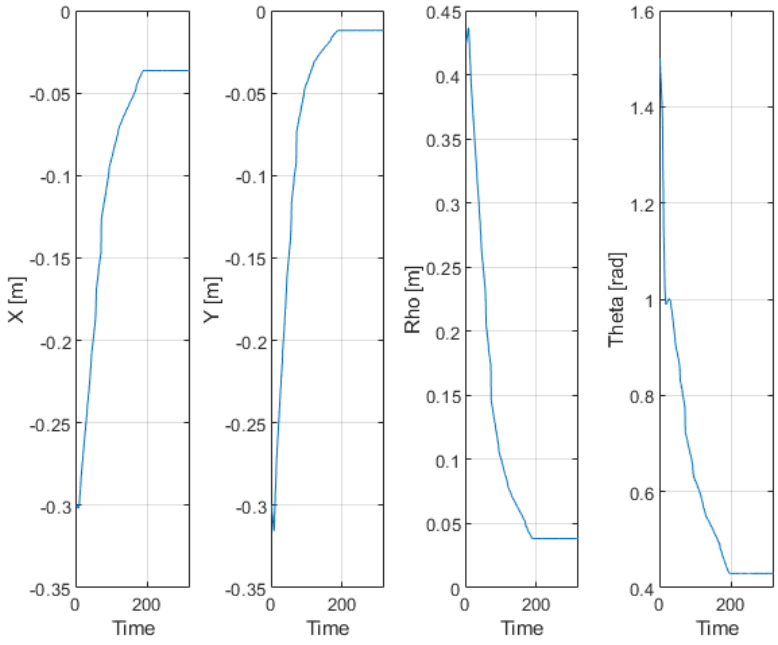

Figure 11: The X, Y, ρ and θ values for starting conditions: $X = -0.3, Y = -0.3, \theta = 1.5\ rad$

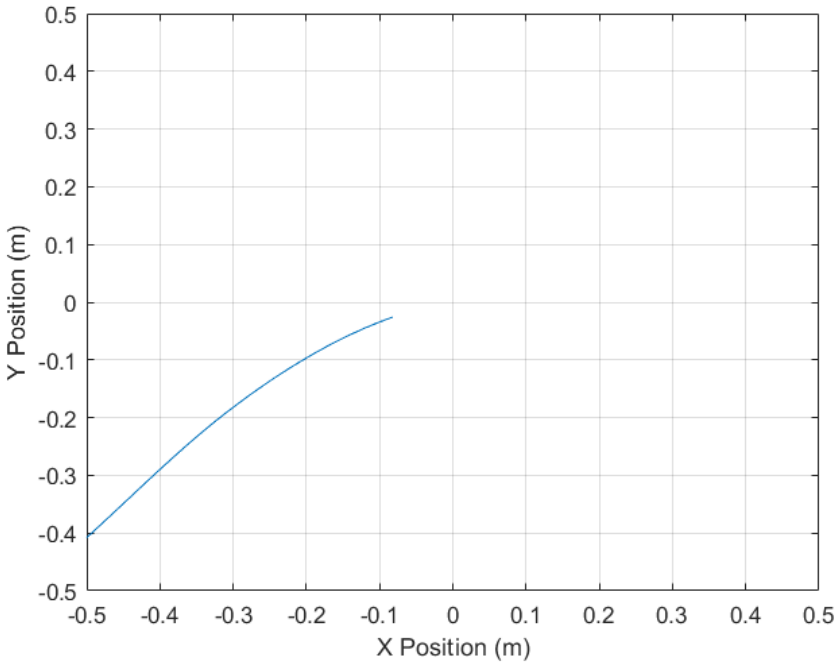

Figure 12: The trajectory taken for starting conditions: $X = -0.8, Y = -0.6, \theta = 0$

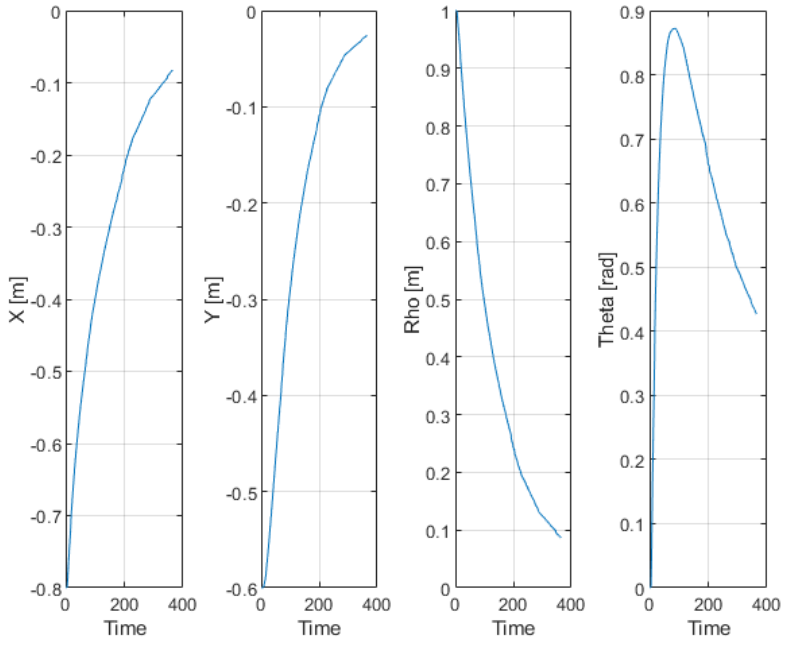

Figure 13: The X, Y, ρ and θ values for starting conditions: $X = -0.8, Y = -0.6, \theta = 0$

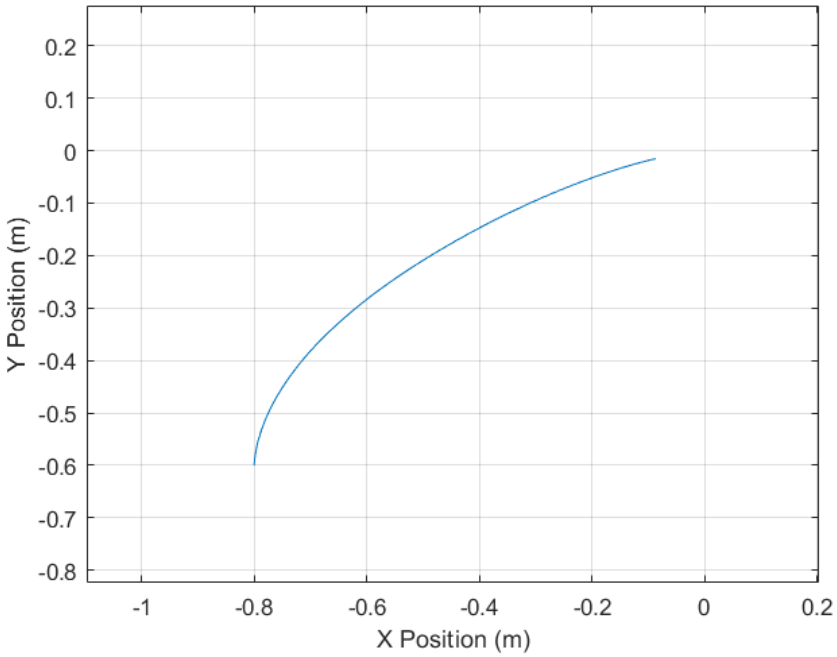

Figure 14: The trajectory taken for starting conditions: $X = -0.8, Y = -0.6, \theta = 1.5\ rad$

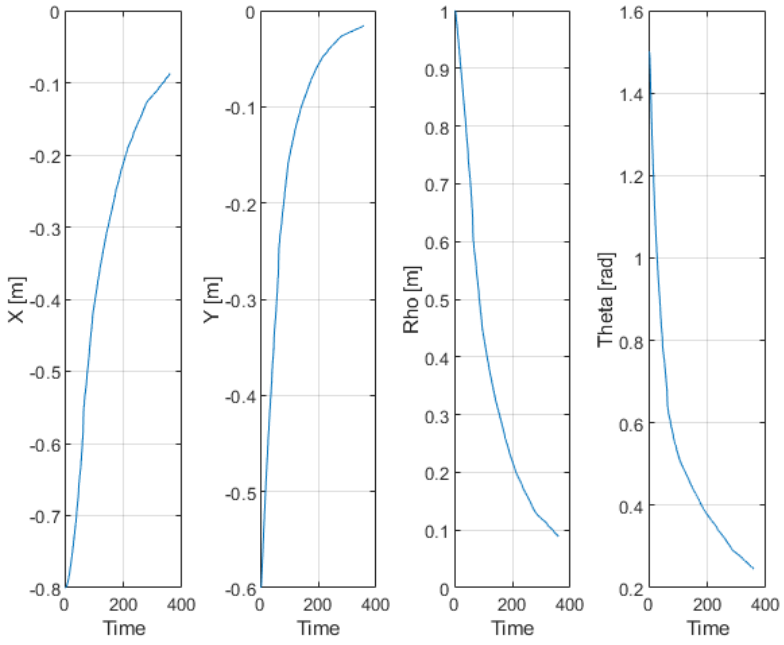

Figure 15: The X, Y, ρ and θ values for starting conditions: $X = -0.8, Y = -0.6, \theta = 1.5\ rad$

4 Conclusion

At the end of the lab, the previous knowledge on odometry was combined with the new knowledge of motion control to develop a controller that could perform several parking maneuvers with different starting positions and orientations of the robot. How to design the theoretical controller in practice and how to select suitable parameters for it were also practiced. How changing each these parameters affected the system in which way was also observed.

5 Discussion

1. How did you adjust the control gains k_ρ, k_α and k_β? How do they affect the robot's motion?

   At first, the control parameters were set as $k_\rho = 0.25, k_\alpha = 20, k_\beta = 20$. This resulted in robot taking a big turn and ending up in a slightly offset position from the origin.

   Then they were modified as $k_\rho = 0.15, k_\alpha = 20, k_\beta = -10$. This made the trajectory of the robot quite chaotic and it ended up rotating in various patterns and ended up even farther away from the goal than at the start.

   Finally, they were set as $k_\rho = 0.25, k_\alpha = 3, k_\beta = -1$. This configuration of parameters gave the best result. Although the robot initially went the opposite way for a while. However, since the rotation parameters were lowered, the robot didn't rotate so erratically this time. It ended up very close to the goal (with there being about $\%1 - \%3$) error.

2. Choose $\alpha \in (-\pi/2, \pi/2]$ and test the robot's parking performance for various initial positions and orientations. Do ρ and θ converge to zero? Discuss the resulting motion trajectories.

   For these alpha values, 2 different runs were made. One was with the exact starting pose as before ($X = -0.3, Y = -0.3, \theta = 0$) and it yielded similar results except that the robot didn't go in the opposite direction this time. The other one was with the starting pose of $X = -0.8, Y = -0.6, \theta = 0$. This resulted in the robot taking a less curvier path than when starting conditions were chosen as $X = -0.3, Y = -0.3, \theta = 0$. For both runs, ρ converged almost to zero, with its final value being smaller than 0.05. For θ however, the same can't be said. Its final value were about 0.4 *rad* for both runs approximately.

3. Choose $\alpha \in (-\pi, -\pi/2] \cup (\pi/2, \pi]$ and test the robot's parking performance for various initial positions and orientations. Do ρ and θ converge to zero? Discuss the resulting motion trajectories.

   For larger α values like this, two runs were made: one with $X = -0.3, Y = -0.3, \theta = 1.5$ and one with $X = -0.8, Y = -0.6, \theta = 1.5$. The parameters that were used before started not working. The robot would rotate a bit and then just start going in the opposite direction. So, the parameters were modified as $k_\rho = 0.15, k_\alpha = 0.8, k_\beta = -0.25$ (these parameters were reached experimentally by playing with them.) When the control parameters were set as such, the robot got very close to the goal, with there being about 0.1 m of error approximately. As for the ρ and θ values, a similar behaviour that was seen at the small α values were observed, with the ρ value being aroun 0.05-0.1 m and θ value being around 0.4 and 0.2 rad for each test.

6 Appendix (All MATLAB Scripts That Were Used):

Listing 1: motionController.m, Used To Estimate The Trajectory Of The Turtlebot and then controls the motion of the robot to perform a parking maneuver using the control law

```
clear; close all; clc; clear node;
wrap = @(ang) atan2(sin(ang), cos(ang));
%% 1 Environment Setup
setenv('ROS_DOMAIN_ID','45'); % Set to your robot's ROS_DOMAIN_ID
setenv('ROS_LOCALHOST_ONLY', '0'); % 0 implies multi-host communication
setenv('RMW_IMPLEMENTATION','rmw_fastrtps_cpp'); % Middleware

node = ros2node('motionController');  % Create node
%% 2 Publishers & Subscribers
velPub = ros2publisher(node,"/cmd_vel","geometry_msgs/Twist", ...
"Reliability","reliable","Durability","volatile","Depth",10);

encSub = ros2subscriber(node,"/joint_states","sensor_msgs/JointState", ...
"Reliability","besteffort","Durability","volatile","Depth",10);
%% 3 Initial Encoder Reading
encMsg = receive(encSub,10);
jointNames = string(encMsg.name);
idxL = find(jointNames=="wheel_left_joint" | jointNames=="left_wheel",1);
idxR = find(jointNames=="wheel_right_joint" | jointNames=="right_wheel",1);
oldLeftWheelPos = encMsg.position(idxL);
oldRightWheelPos = encMsg.position(idxR);

%% 4 Parameters
x = -0.3; % Your Starting x [m]
y = -0.3; % Your Starting y [m]
theta = 0.05; % Your Starting theta [rad]
x_goal = 0.0;
y_goal = 0.0;
R = 0.033; % Wheel Radius [m]
L = 0.287; % Axle Distance [m]

%% 5 Variables Initialization
%dataMatrix = [];
x_values = [x];
y_values = [y];
theta_values = [theta];
rho_values = [sqrt((x_goal - x)^2+(y_goal - y)^2)];

velMsg = ros2message(velPub);
velMsg.linear.y = 0;
velMsg.linear.z = 0;
velMsg.angular.x = 0;
velMsg.angular.y = 0;

deltaX = x_goal - x;
deltaY = y_goal - y;
ro = sqrt(deltaX^2+deltaY^2);
alpha = -theta + atan2(deltaY,deltaX);
beta = -theta - alpha;
kp = 0.15;
ka = 0.8;
kb = -0.25;
v = kp*ro;
omega = ka*alpha+kb*beta;
v = max(min(v, 0.22), -0.22);
omega = max(min(omega, 2.84), -2.84);

velMsg.linear.x = v;
velMsg.angular.z = omega;
send(velPub, velMsg);

exeTime = 20; period = 0.01; t0 = tic;
iteration = 2;
while toc(t0) < exeTime

```

```
    %% 6.1 Read Encoders
     encMsg = receive(encSub,10);
     leftWheelPosition = encMsg.position(idxL);
     rightWheelPosition = encMsg.position(idxR);

     %% 6.2 Odometric Estimation
     Dl = R * (leftWheelPosition - oldLeftWheelPos);
     Dr = R * (rightWheelPosition - oldRightWheelPos);
     Dc = (Dr + Dl) / 2;
     dTheta = (Dr - Dl) / L;

     %% 6.3 Values Recording (x,y,theta)
     x = x + Dc * cos(theta + dTheta/2);
     y = y + Dc * sin(theta + dTheta/2);
     theta = theta + dTheta;

     x_values(iteration) = x;
     y_values(iteration) = y;
     theta_values(iteration) = theta;


     %% 6.4 Motion Controller
     deltaX = x_goal - x;
     deltaY = y_goal - y;
     ro = sqrt(deltaX^2+deltaY^2);
     rho_values(iteration) = ro;
     alpha = -theta + atan2(deltaY,deltaX);
     beta = -theta - alpha;

     v = kp*ro;
     omega = ka*alpha+kb*beta;

     velMsg.linear.x = v;
     velMsg.angular.z = omega;
     send(velPub, velMsg);

     oldLeftWheelPos = leftWheelPosition;
     oldRightWheelPos = rightWheelPosition;
     iteration = iteration + 1;

     pause(period);
end

% Final stop
velMsg.linear.x = 0;
velMsg.angular.z = 0;
send(velPub, velMsg);


%% 7 Plotting Results
figure; plot(x_values,y_values);
xlabel("X Position (m)"); ylabel("Y Position (m)");
xlim([-0.5 0.5]); ylim([-0.5 0.5]); grid on;
figure; subplot(1,4,1); plot(x_values);
xlabel("Time"); ylabel("X [m]"); grid on;
subplot(1,4,2); plot(y_values);
xlabel("Time"); ylabel("Y [m]"); grid on;
subplot(1,4,3); plot(rho_values);
xlabel("Time"); ylabel("Rho [m]"); grid on;
subplot(1,4,4); plot(theta_values);
xlabel("Time"); ylabel("Theta [rad]"); grid on;
```