Exercise 2: Kinematics and Control of a Differential Drive Robot

Introduction

In this exercise, I programmed and implemented a closed-loop motion controller for a differential-drive robot. The main goal was to validate the kinematic model and ensure accurate motion control. This involved three tasks: computing wheel velocities for desired motion, developing a teleoperation interface, and implementing a position controller to drive the robot to a target.

Task 1: Computing Wheel Velocities

To compute the spinning speeds (ϕ_r, ϕ_l) of the robot's wheels, I implemented the function `calculateWheelSpeeds.m`. Using the given kinematic model equations:

$$v = \frac{r\phi_r + r\phi_l}{2}, \quad \omega = \frac{r\phi_r - r\phi_l}{2l},$$

I derived the wheel velocities:

$$\phi_l = \frac{v - \omega l}{r}, \quad \phi_r = \frac{v + \omega l}{r}.$$

This ensured that the robot could execute linear and angular velocities accurately. I validated this function by running the script `testCircleDrive.m`, which produced a trajectory closely matching the reference.

Task 2: Teleoperation Interface

I developed a teleoperation program, `teleoperation.m`, to allow manual control of the robot using keyboard inputs. Commands included:

- `w`: Move forward.
- `s`: Move backward.
- `a`: Rotate counterclockwise.
- `d`: Rotate clockwise.
- `q`: Stop the robot.

The program adjusted wheel speeds dynamically based on the key pressed, providing smooth operation. I ensured the simulation integration worked seamlessly by initializing connections to V-REP and setting up the required parameters.

Task 3: Position Controller

For closed-loop control, I implemented `calculateControlOutput.m`. This function computed the forward velocity (v_u) and angular velocity (ω) using proportional control:

$$v_u = K_\rho \rho, \quad \omega = K_\alpha \alpha + K_\beta \beta,$$

where ρ is the distance to the goal, α is the angle to the target, and β is the orientation error.

I further enhanced the controller to handle backward motion when $\alpha > \pi/2$ and scaled velocities to ensure constant speed. This made the robot's movement smooth and efficient.

Validation and Results

I validated the overall implementation using the script `controller.m`. By defining a target position and orientation, the robot followed the trajectory accurately. The simulated trajectory closely matched the reference trajectory, confirming the correctness of the control algorithm.

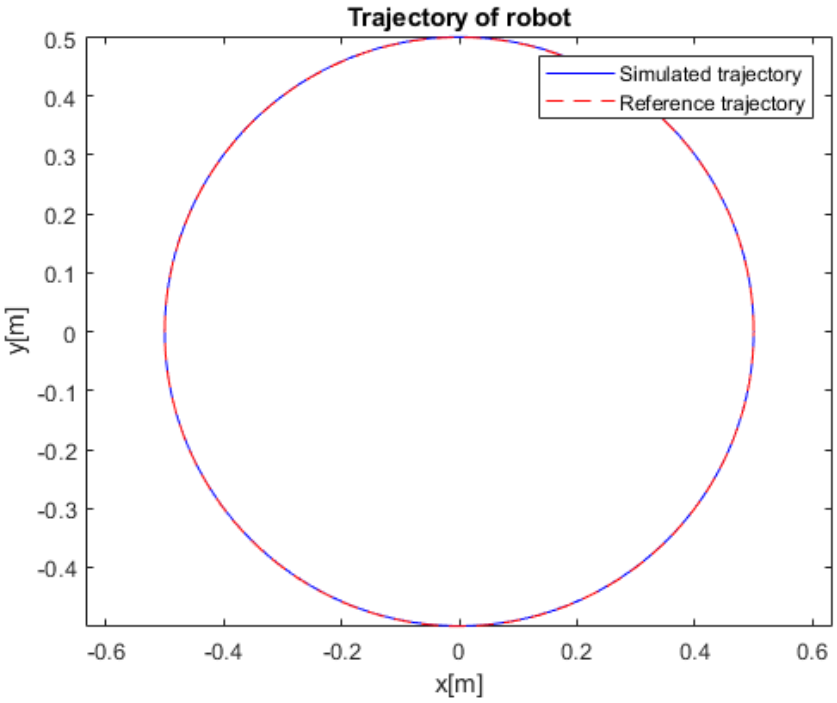

Figure 1: Simulated trajectory compared to the reference trajectory.

The figure above shows the robot's trajectory during the validation, where the blue line represents the simulated trajectory, and the red dashed line is the reference.

Conclusion

Through this exercise, I successfully implemented a kinematic controller for a differential-drive robot, including wheel velocity computation, teleoperation, and closed-loop position

control. Each task was validated within the V-REP simulation, and the results demonstrate the effectiveness of the algorithms.