IE4060 - Robotics and Intelligent Systems

Assignment 01: Intelligent Differential Drive Robot Simulation

Name: Dilshan J M H
Student ID: IT22266828

1. Introduction

A differential drive robot (DDR) is a simple and common type of mobile robot with two independent wheels on the same axis and a third small wheel for balance. By changing the speed of each wheel, the robot can move forward, turn, or follow curves. This design makes it easy to maneuver and is widely used in applications like vacuum cleaners and warehouse robots.

But to move on its own, the robot needs a control system. The controller translates high-level goals like "go to point (x, y)" into motor commands. It does this by checking the robot's current position and orientation, comparing it with the target, and then adjusting wheel speeds to reduce the difference (error). This report focuses on designing and implementing such a controller for a simulated DDR.

2. Derivations

2.1. Kinematic Equations

The forward kinematics of a DDR describe how the robot's overall motion arises from its individual wheel speeds. Let's define the robot's parameters:

- vL and vR : Velocities of the left and right wheels, respectively.
- r: Radius of each wheel.
- L: Distance between the centers of the two wheels (the axle length).
- v: Linear velocity of the robot's center.
- ω: Angular velocity (rate of rotation) of the robot.

The linear velocity of the robot is the average of the two-wheel velocities:

$$v = \frac{(v_R + v_L)}{2} \quad \text{or} \quad v = \frac{r(\omega_R + \omega_L)}{2}$$

The angular velocity is determined by the difference in wheel velocities and the distance between them:

$$\omega = \frac{(v_R - v_L)}{L} \qquad \text{or} \qquad \omega = \frac{r\,(\omega_R - \omega_L)}{L}$$

The robot's state, or "pose," in a 2D environment is defined by its coordinates (x,y) and its orientation angle θ.

Rate of change in x : $\dot{x} = v\,cos\theta$
Rate of change in y : $\dot{y} = v\,sin\theta$
Rate of change in θ : $\dot{\theta} = \omega$

To update the pose over a small time step Δt , New state: $(x_{k+1},\ y_{k+1}, \theta_{k+1})$ at time step $k + 1$ based on the state at time step k using Euler integration.

$$x_{k+1} = x_k + v_k \cos(\theta_k)\,\Delta t$$
$$y_{k+1} = y_k + v_k \sin(\theta_k)\,\Delta t$$
$$\theta_{k+1} = \theta_k + \omega_k \Delta t$$

These three equations are the core of the robot's simulation, allowing us to update its position and orientation based on the control outputs $v_k$and ω_k.

2.2. Error Terms

To guide the robot, we must first quantify how "off" it is from the target. Let the robot's current position be (x, y) and the target's(goal's) position be $(x_g\ ,\ y_g)$.

1. Distance Error (ρ): This is the straight-line Euclidean distance to the target. It is calculated using the Pythagorean theorem. Our goal is to drive this error to zero.
$$\rho = \sqrt{(x_g - x)^2 + (y_g - y)^2}$$
2. Heading Error (α): This is the angular difference between the robot's current orientation (θ) and the angle to the target.

   First, we calculate the angle to the target (θ_g), using the arctangent function.

$$\theta_g = a\tan 2\,(y_g - y\ ,\ x_g - x)$$

   Now get the difference of current orientation and the angle to the target

$$\alpha = \theta_g - \theta$$

   Note: It's important to normalize this angle to the range $[-\pi\ ,\ \pi]$ to ensure the robot always takes the shortest turn.

2.3. Control Laws

The control law uses the error terms to compute the desired linear velocity(v) and angular velocity (ω). We propose three controllers of increasing complexity.

1. **P (Proportional) Controller**: This is the simplest form of feedback control. The control output is directly proportional to the measured error.
   Linear Velocity: The forward speed is proportional to the distance error.
$$v = K_{p,\rho}\ .\ \rho$$

Angular Velocity: The turning speed is proportional to the heading error.

$$\omega = K_{p,\alpha} \,.\, \alpha$$

Reasoning: This is intuitive. If the distance error (ρ) is large, move faster. If the heading error (α) is large, turn faster. $K_{p,\rho}$ and $K_{p,\alpha}$ are positive constants (gains) that determine the strength of the response.

Expected Behavior: The robot will move towards the target but will likely **overshoot** and **oscillate** around it. As it gets closer, its speed and turn rate decrease, but it might not have enough "braking" force to stop precisely, leading to wobbly or circling behavior near the goal. It may also have a **steady-state error** (stopping slightly short of the target).

2. **PD (Proportional-Derivative) Controller**: This controller adds a derivative term, which considers the rate of change of the error. This acts as a damping force, helping to reduce overshoot.

   Linear Velocity: We add a derivative term to the linear velocity control.

$$v = K_{p,\rho} \,.\, \rho \,+\, K_{d,\rho} \,.\, \frac{d\rho}{dt}$$

   Angular Velocity: We will keep this as a P controller for now.

$$\omega = K_{p,\alpha} \,.\, \alpha \,+\, K_{d,\alpha} \,.\, \frac{d\alpha}{dt}$$

   In a discrete simulation, the derivative term $\frac{derror}{dt}$ is approximated as $\frac{error_k - error_{k-1}}{\Delta t}$

   Reasoning: The D-term acts like a brake. If the error is decreasing rapidly (i.e., the robot is rushing towards the target), the derivative term becomes negative, reducing the control output to **prevent overshoot**. It anticipates future error based on the current rate of change.

   Expected Behavior: The PD controller provides a much **smoother and more stable** response. It will reach the target faster than a P controller, with significantly less oscillation and overshoot. It is often sufficient for this type of navigation task.

3. **PID (Proportional-Integral-Derivative) Controller**: This controller adds an integral term, which accumulates past errors. This helps eliminate small, steady-state errors that the P or PD controllers might not correct. We apply this to the heading control for better accuracy.

Linear Velocity:

$$v = K_{p,\rho} \cdot \rho + K_{i,\rho} \int_0^t \rho(\tau)d\tau + K_{d,\rho} \cdot \frac{d\rho}{dt}$$

Angular Velocity:

$$\omega = K_{p,\alpha} \cdot \alpha + K_{i,\alpha} \int_0^t \alpha(\tau)d\tau + K_{d,\alpha} \cdot \frac{d\alpha}{dt}$$

In a discrete simulation, the integral term $\int error\ dt$ is approximated as $\sum (error_i \cdot \Delta t)$

Reasoning: The I-term is designed to eliminate **steady-state error**. If forces like friction or slight model inaccuracies cause the robot to stop just short of the target, the P and D terms might become zero. However, the small, persistent error will cause the integral term to grow over time, eventually producing a control output strong enough to overcome the friction and move the robot to the exact target position.

Expected Behavior: The PID controller offers the most precise control, capable of eliminating steady-state error for very high-accuracy positioning. However, it is the most complex to tune. A poorly tuned integral term can lead to "integral windup," causing large overshoots, or can make the system's response sluggish. For many robotic applications, a well-tuned PD controller is often preferred for its simplicity and excellent performance.

4. **PI Controller (Proportional-Integral)**

This controller adds an **integral (I) term** to the P controller. The integral term accumulates past errors to correct for long-term, persistent inaccuracies.

Linear Velocity: $v = K_{p,\rho} \cdot \rho + K_{i,\rho} \int_0^t \rho(\tau)d\tau$

Angular Velocity: $\omega = K_{p,\alpha} \cdot \alpha + K_{i,\alpha} \int_0^t \alpha(\tau)d\tau$

In a discrete simulation, the integral term $\int error\ dt$ is approximated as $\sum (error_i \cdot \Delta t)$

Reasoning: The PI controller combines the immediate response of the P-term with the error-correcting power of the I-term. If a P-only controller stops just short of the target due to friction or other small, constant forces (leaving a **steady-state error**), the integral term will slowly build up over time. This

accumulated value will eventually create a control command strong enough to overcome the friction and push the robot to the exact target.

Expected Behavior: A PI controller will successfully **eliminate the steady-state error** that a P-only controller might have, leading to better final accuracy. However, because it lacks the **damping** effect of a derivative (D) term, it is very **prone to overshoot**. The integral term can "wind up" as the robot approaches the goal, causing it to fly past the target and then oscillate before settling. Its settling time may be longer than a well-tuned PD or PID controller.

3. Implementation

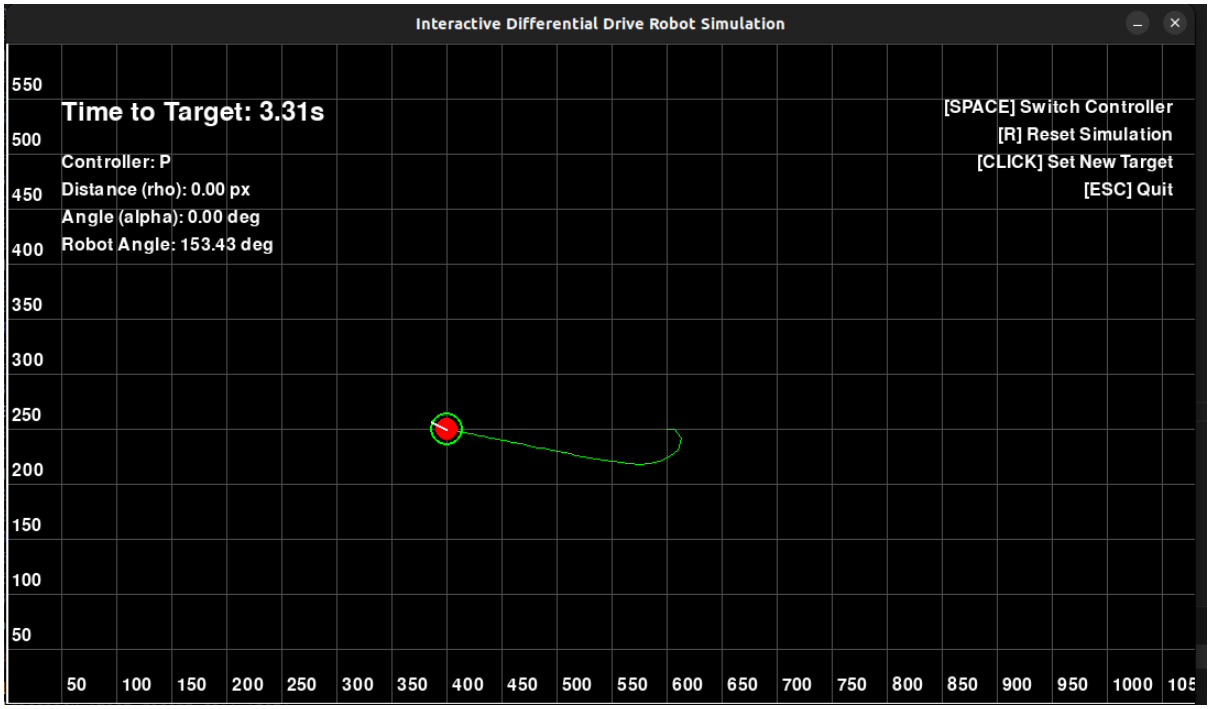
3.1. Pygame Setup

The simulation was developed in Python using the Pygame library for visualization. The environment is a simple 2D window.

- **Main Loop**: The core of the program is a while loop that runs continuously. In each iteration, it handles user input, updates the robot's state based on the control laws, and redraws all elements on the screen.
- **Robot Representation**: The robot is drawn as a green circle with a line extending from its center to indicate its current heading (θ).
- **Target**: The target is represented by a red circle.
- **Trajectory Visualization**: A list of the robot's past (x,y) positions is maintained. In each frame, lines are drawn connecting these stored points, visually rendering the path the robot has taken. A stopping condition is implemented to end the simulation when the distance error (ρ) falls below a small threshold.

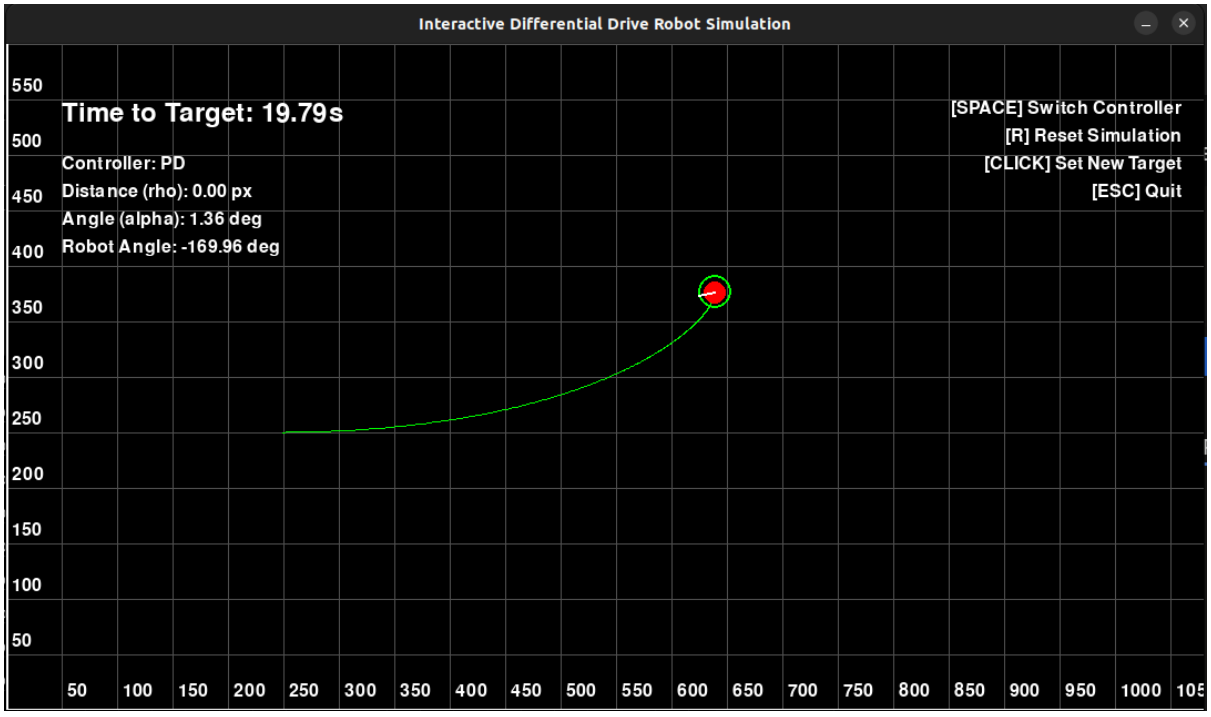
3.2. Simulation Screenshots

Figure 1: P Controller Simulation



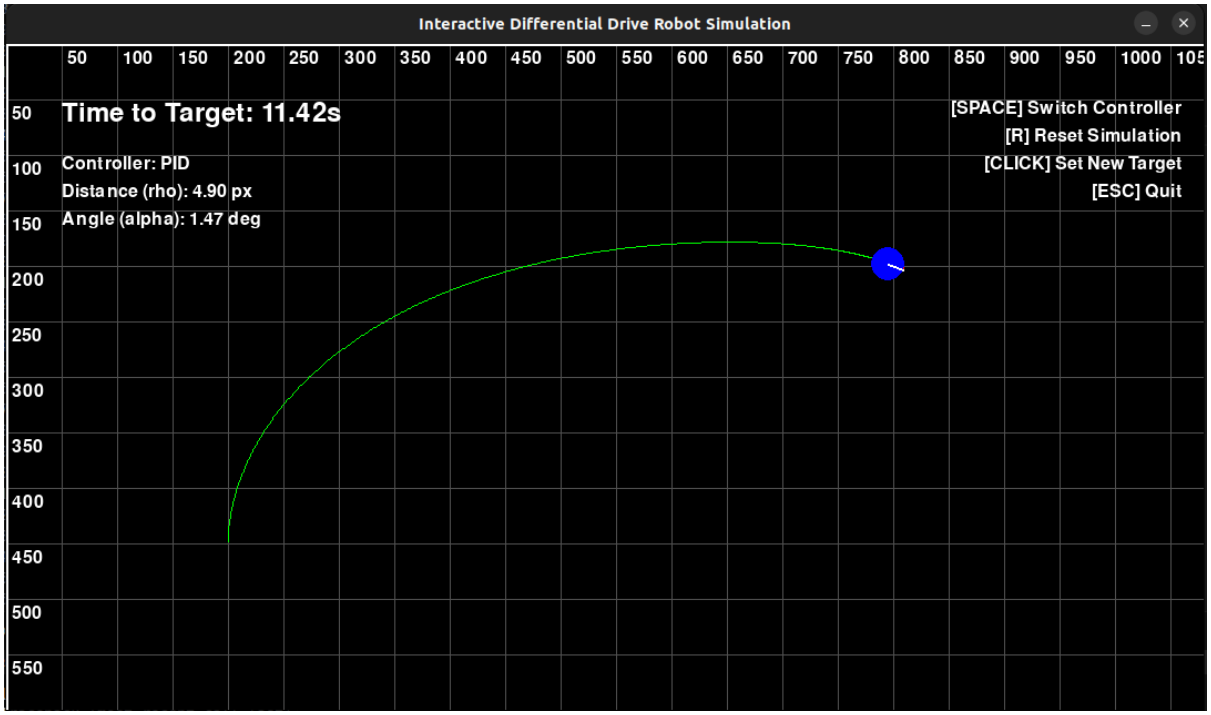
The path shows a little angular overshot atthe target, followed by a correction, demonstrating the aggressive nature of the proportional-only control. (with $K_{p,\rho}$=3, $K_{p,\alpha}$ =25)

Figure 2: PD Controller Simulation



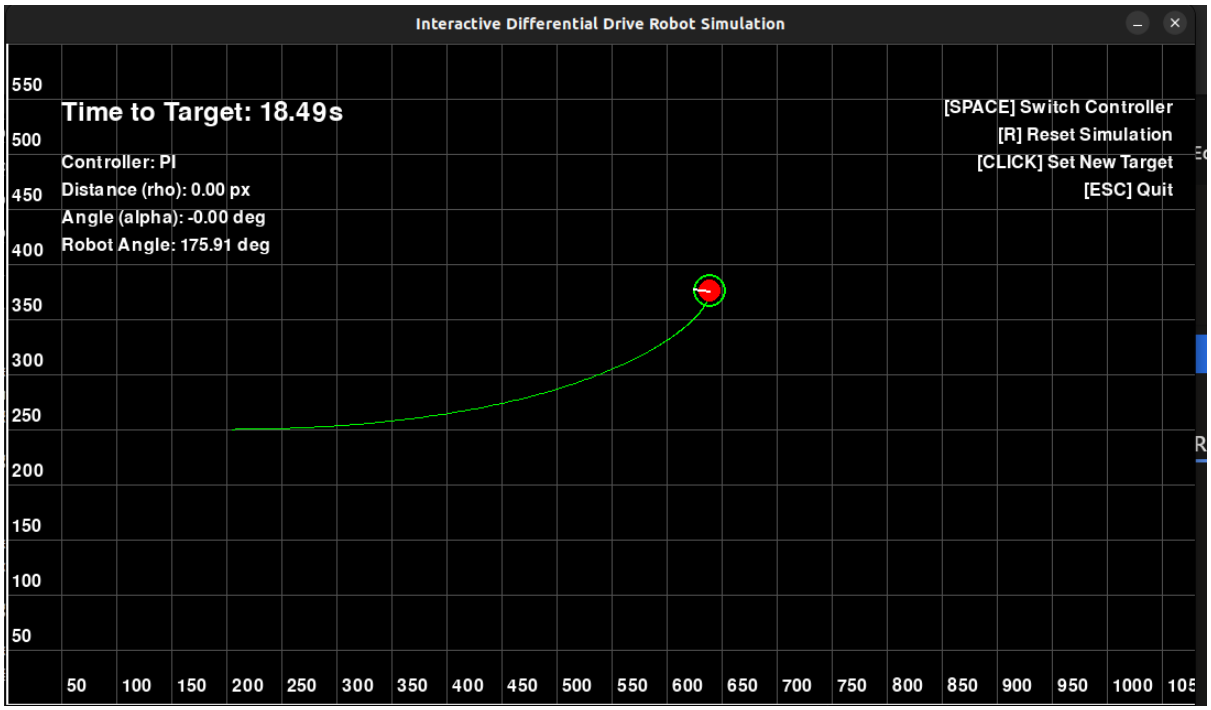
The path is much smoother, approaching the target directly with minimal or no overshot, showcasing the damping effect of the derivative term.

Figure 3: PID Controller Simulation



The path is smooth, and the robot's final orientation (indicated by the line) is perfectly aligned with the target, demonstrating the high accuracy of the PID controller. **But with limitation thresholds.**

Figure 4: PI Controller Simulation



4. Results and Discussion

4.1. Comparison of P, PD, and PID Control

The three controllers exhibited distinct behaviors as predicted by control theory.

- **P Controller**: This controller was effective at getting the robot to the general vicinity of the target but performed poorly in terms of precision. It consistently overshot the target because its speed is only dependent on distance, not on its rate of approach. This resulted in an oscillating path as it repeatedly tried to correct itself.
- **PD Controller**: The addition of the derivative term for linear velocity control marked a significant improvement. The robot decelerated as it neared the target, resulting in a smooth stop with little to no overshoot. This behavior is far more desirable for a real-world robot, as it is more efficient and predictable.
- **PID Controller**: Implementing PID for heading control provided the highest level of performance. While the PD controller stopped accurately at the target position, the final orientation could still be slightly off. The integral term in the PID controller worked to eliminate this small, steady-state heading error, ensuring the robot was not only at the target but also perfectly facing it. But had to use limitations as this is hard to tune.

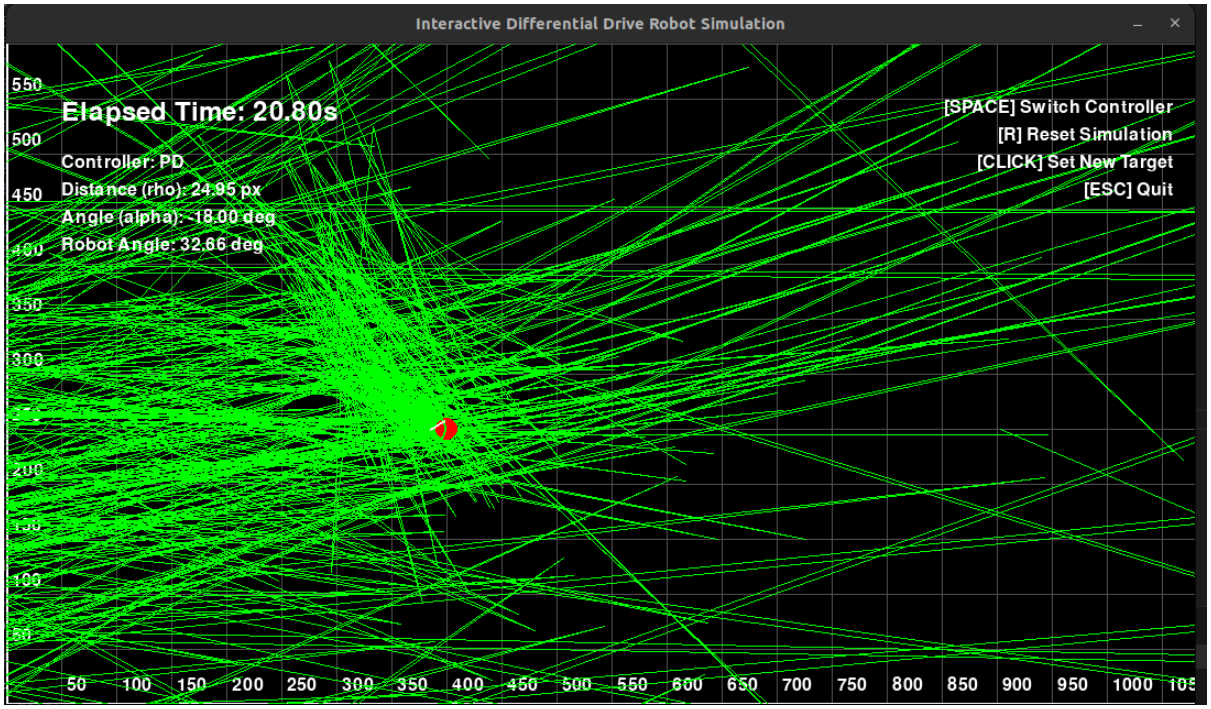
4.2. Tuning of Gains

The performance of each controller is critically dependent on the values of its gains (Kp ,Kd ,Ki).

- Kp **(Proportional Gain)**: If set too low, the robot's response is sluggish and it may take a very long time to reach the target. If set too high, the system becomes unstable and oscillates wildly.
- Kd **(Derivative Gain)**: This gain controls the damping. If too low, it won't be effective at preventing overshoot. If too high, it can overly dampen the system, causing the robot to move very slowly as it approaches the target.
- Ki (Integral Gain): This gain must be tuned carefully. If too low, it won't be effective at correcting steady-state error. If too high, it can lead to "integral windup," where the accumulated error causes a large overshoot.
  The process of tuning was iterative, involving running the simulation repeatedly with different gain values to find a combination that resulted in a fast, stable, and accurate response.

4.3. Difficulties Encountered

The primary challenge was the practical implementation of the derivative and integral terms, which are sensitive to the time step Δt of the simulation loop. A fluctuating Delta t can introduce noise into the derivative calculation. Additionally, normalizing the heading error angle to the $[-\pi,\pi]$ range was a crucial step that, if missed, would cause the robot to take unnecessarily long turns.



Controller PD , $K_{p,\rho}$ =1.2, $K_{p,\alpha}$=8, $K_{d,\alpha}$=1.5 , $K_{d,\rho}$ = 10

5. Conclusion

This assignment provided a practical and insightful bridge between the theoretical concepts of robot kinematics and control theory and their implementation in a working simulation. I successfully derived the mathematical model of a differential drive robot and implemented P, PD, and PID controllers to guide it. The simulation visually demonstrated the strengths and weaknesses of each control strategy: the simplicity but imprecision of P control, the smooth damping of PD control, and the high accuracy of PID control.

A key takeaway is that system performance is not just about choosing the right type of controller, but also about the careful tuning of its parameters. For future work, this simulation could be extended in several ways. The most immediate improvement would be the implementation of **obstacle avoidance** algorithms, using sensors like virtual LIDAR to navigate more complex environments. Another enhancement would be to implement **trajectory following**, where the robot is tasked with following a predefined path rather than just moving to a single endpoint.