IE4060
Computer Systems Administration
4th Year, 1st Semester

Assignment

Intelligent differential drive robot simulation

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

10/1/2025

Declaration

I certify that this report does not incorporate without acknowledgement, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief it does not contain any material previously published or written by another person, except where due reference is made in text.

Registration Number : IT22069122

Name : K.B.H.M.C.C Bandaranayake

Table of Contents

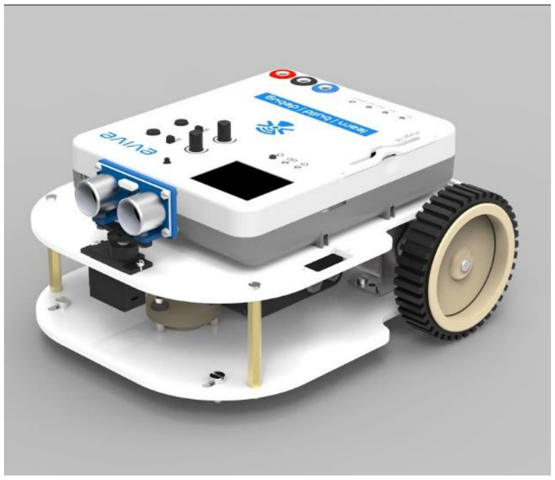
1. Introduction 4
   1.1 What is a differential drive robot? 4
   1.2 Why is control necessary? 4
2. Derivations 5
   2.1 Kinematic Equations 5
   2.2 Distance and Heading Error 5
   2.3 Control Laws 5
3. Implementation 6
   3.1 Pygame Setup 6
   3.2 Python Classes 6
   3.3 Controls 6
   3.4 Screenshots 7
4. Results and Discussion 10
   4.1 Comparison of Controllers 10
   4.2 Gain Tuning 10
   4.3 Difficulties Encountered 10
5. Conclusion 11
6. References 12

List of Figures

Figure 1: P controller 7
Figure 2: PD controller 8
Figure 3: PID controller 9

1. Introduction

1.1 What is a differential drive robot?



- A differential drive robot is a mobile robot with two independently driven wheels mounted on the same axis. By controlling the speed and direction of each wheel, the robot can move forward, backward, and rotate in place. This configuration is simple, widely used in research and robotics education, and suitable for tasks requiring navigation in 2D space

1.2 Why is control necessary?

- Control is essential to ensure the robot reaches a target accurately and efficiently. Without proper control, the robot may overshoot, drift off course, or follow inefficient paths. Controllers such as P, PD, and PID adjust the robot's linear and angular velocities based on real-time error measurements, allowing precise navigation towards a target.

2. Derivations

2.1 Kinematic Equations

- The motion of a differential drive robot in 2D can be described by the following kinematic equations:

$$\dot{x} = v\cos(\theta)\,, \quad \dot{y} = v\sin(\theta)\,, \quad \dot{\theta} = \omega$$

- Where are the robot's position coordinates, is its heading, is the linear velocity, and is the angular velocity.
- set-velocity function calculates Right-wheel and left-wheel speeds simultaneously for a give v and omega.

$$\omega_r = \frac{2v + \omega L}{2R}, \omega_l = \frac{2v - \omega L}{2R}$$

- To update the robot's position over a small-time step , Euler integration is used.

  Euler integration:

$$x_{new} = x + \dot{x} \cdot dt, \quad y_{new} = y + \dot{y} \cdot dt, \quad \theta_{new} = \theta + \dot{\theta} \cdot dt$$

2.2 Distance and Heading Error

- To navigate towards a target (x_t , y_t) , the robot calculates:
  - **Distance error:**

$$e_d = \sqrt{(x_t - x)^2 + (y_t - y)^2}$$

  - **Heading error:**

$$e_\theta = \text{wrap\_to\_pi}(\text{atan2}(y_t - y, x_t - x) - \theta)$$

2.3 Control Laws

- Controllers compute linear and angular velocities based on errors.
  - **P Controller:**

$$v = K_p e_d, \quad \omega = K_{p\theta} e_\theta$$

  - **PD Controller:**

$$v = K_p e_d + K_d \dot{e_d}, \quad \omega = K_{p\theta} e_\theta$$

  - **PID Controller:**

$$v = K_p e_d + K_d \dot{e_d}, \quad \omega = K_{p\theta} e_\theta + K_{i\theta} \int e_\theta \, dt + K_{d\theta} \dot{e_\theta}$$

3. Implementation

3.1 Pygame Setup

- The simulation uses Pygame with a 1000×700 pixel window. The robot is visualized as a triangle, the target as a red cross, and the trajectory as a green line. A clock updates the simulation at 60 FPS

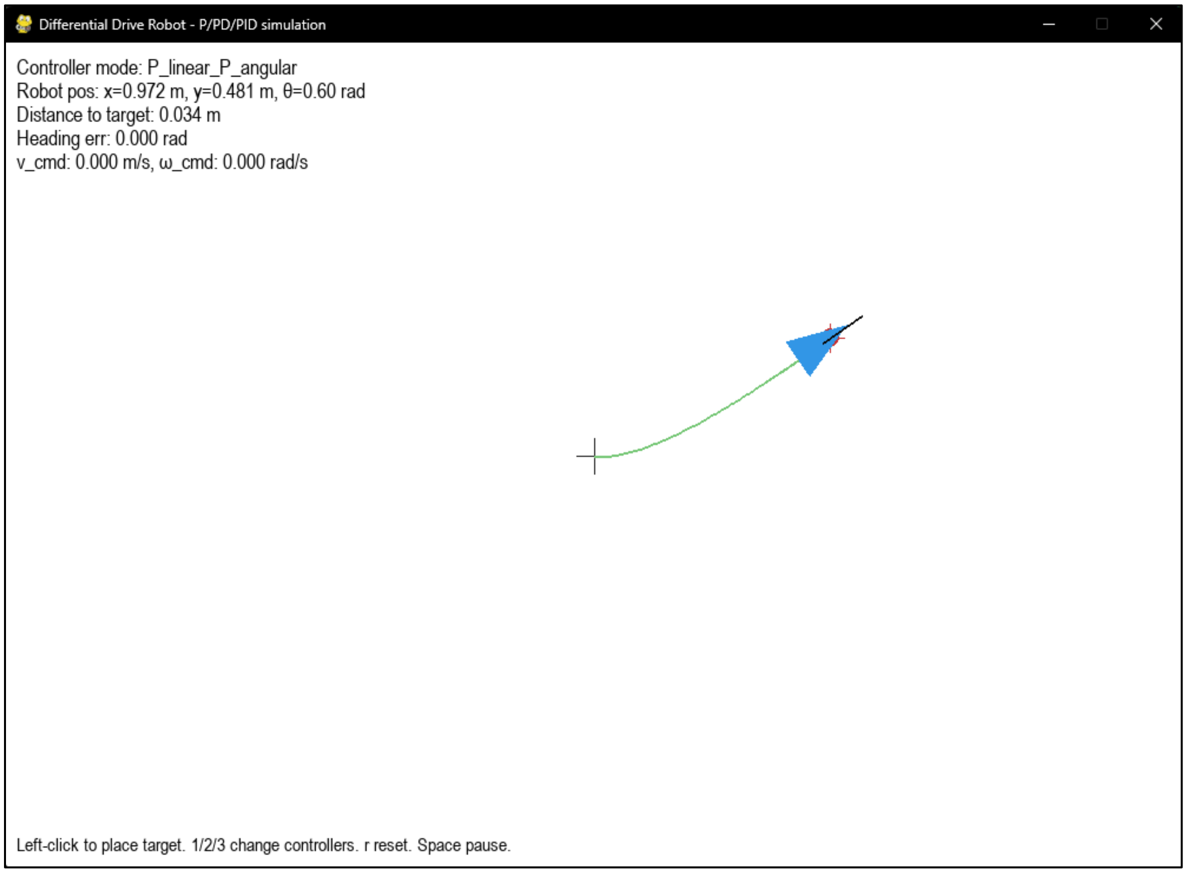
3.2 Python Classes

- **DifferentialDriveRobot**: Tracks the robot's position, heading, linear and angular velocities, and updates its state using Euler integration.
- **Controller**: Computes the robot’s linear and angular velocity commands using P, PD, or PID control laws

3.3 Controls

Key	**Function**
Left-click	Set target
1	P controller
2	PD controller
3	PID controller
r	Reset robot
Space	Pause / resume

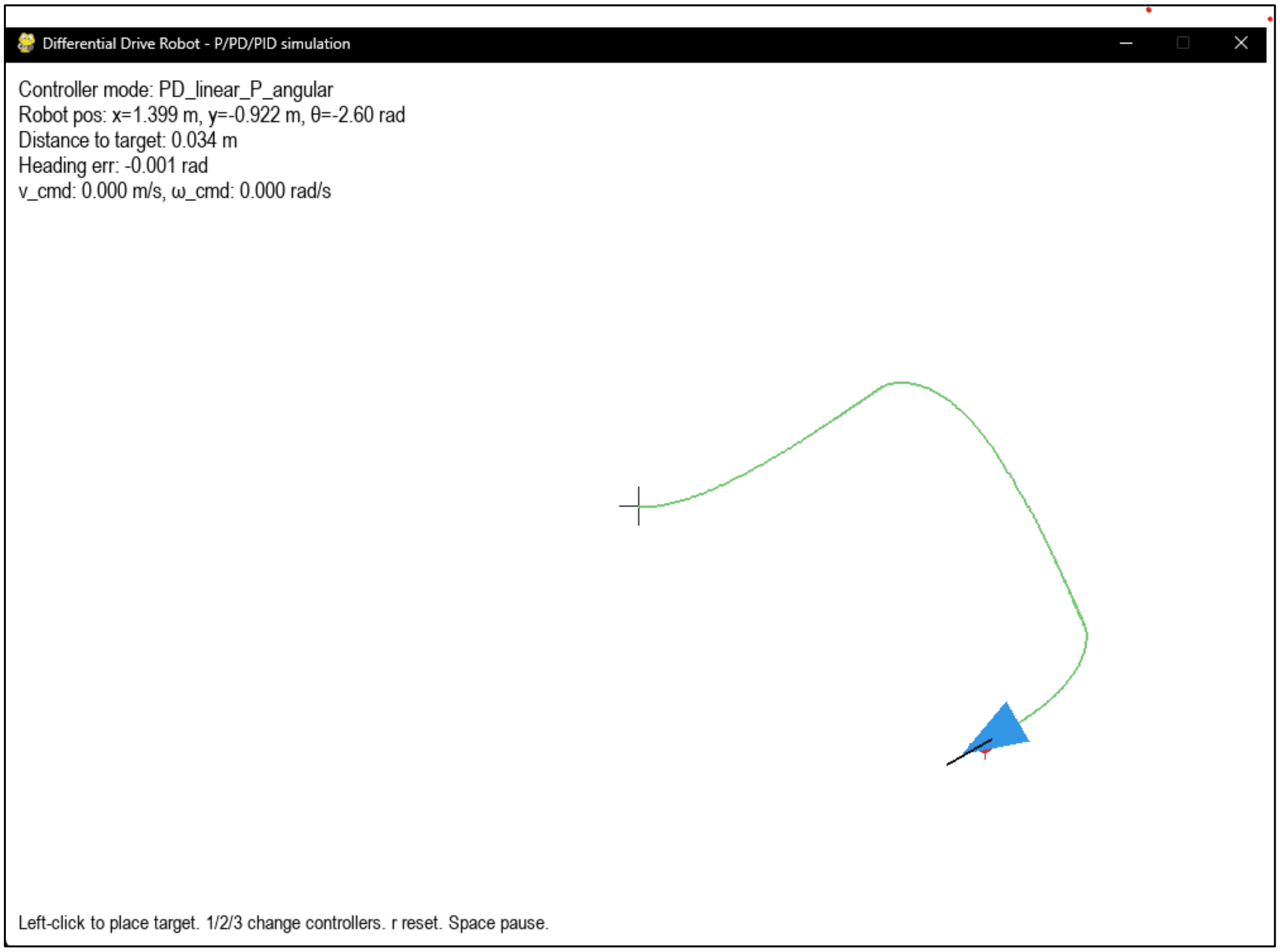
3.4 Screenshots


Differential Drive Robot - P/PD/PID simulation

```
Controller mode: P_linear_P_angular
Robot pos: x=0.972 m, y=0.481 m, θ=0.60 rad
Distance to target: 0.034 m
Heading err: 0.000 rad
v_cmd: 0.000 m/s, ω_cmd: 0.000 rad/s
```

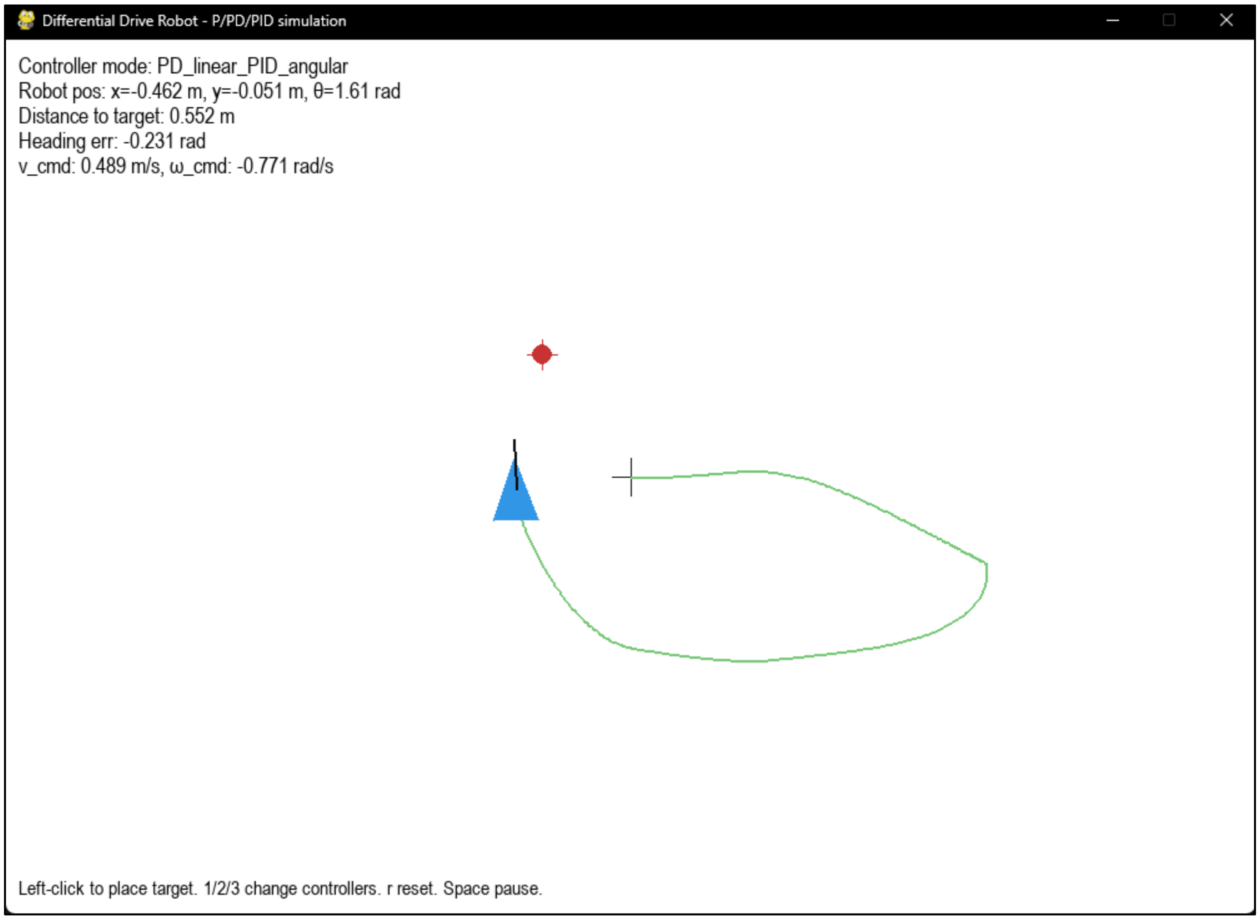
Left-click to place target. 1/2/3 change controllers. r reset. Space pause.

Figure 1: P controller


Controller mode: PD_linear_P_angular
Robot pos: x=1.399 m, y=-0.922 m, θ=-2.60 rad
Distance to target: 0.034 m
Heading err: -0.001 rad
v_cmd: 0.000 m/s, ω_cmd: 0.000 rad/s

Left-click to place target. 1/2/3 change controllers. r reset. Space pause.

Figure 2: PD controller



Differential Drive Robot - P/PD/PID simulation

```
Controller mode: PD_linear_PID_angular
Robot pos: x=-0.462 m, y=-0.051 m, θ=1.61 rad
Distance to target: 0.552 m
Heading err: -0.231 rad
v_cmd: 0.489 m/s, ω_cmd: -0.771 rad/s
```

```
Left-click to place target. 1/2/3 change controllers. r reset. Space pause.
```

Figure 3: PID controller

4. Results and Discussion

4.1 Comparison of Controllers

- **P Controller:** Fast but prone to overshooting; simple implementation.
- **PD Controller:** Reduces oscillation; reacts to changes in error faster than P controller.
- **PID Controller:** Smoothest trajectory; corrects steady-state error and aligns heading precisely

4.2 Gain Tuning

- **Kp**: Increases robot speed toward the target.
- **Kd**: Reduces overshoot and oscillations.
- **Ki**: Eliminates steady-state error, especially in PID angular control

4.3 Difficulties Encountered

- Selecting appropriate time step dt to balance stability and smoothness.
- Handling large initial heading errors, which can cause corner cutting.
- Maintaining stability at higher speeds.

5. Conclusion

- The simulation demonstrated the kinematics of a differential drive robot and the effects of P, PD, and PID controllers. PID control provided the most accurate and smooth path to the target.
- Possible Improvements:
  - Implement obstacle avoidance.
  - Smooth acceleration/deceleration for realistic motion.
  - Path planning with multiple waypoints

6. References

- Siciliano, B. et al., Robotics: Modelling, Planning and Control, Springer, 2010.
- Pygame Documentation: https://www.pygame.org/docs/