



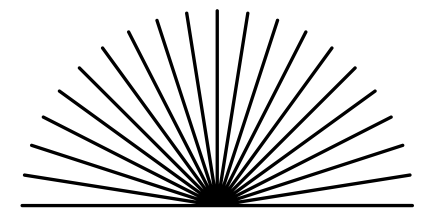
Cyber-Physical Systems Lab: Autonomous Applications
Summer Semester 2024

F1/10 CAR PID CONTROLLER

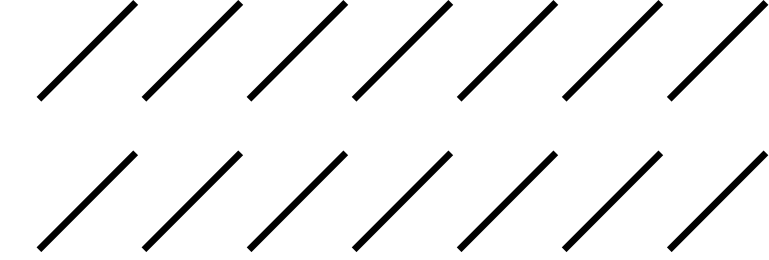
Development, Testing and Integration

PRESENTED BY:

Matheus Salomão, Matheus Takaki & Nicolas Nabrink



Agenda



03	Overview
04	Goals
05	Timeline
06	PID Controller Implementation
15	Integration with Mux Node
16	Testing and Results
20	Challenges and Solutions
21	Conclusion
22	Q&A

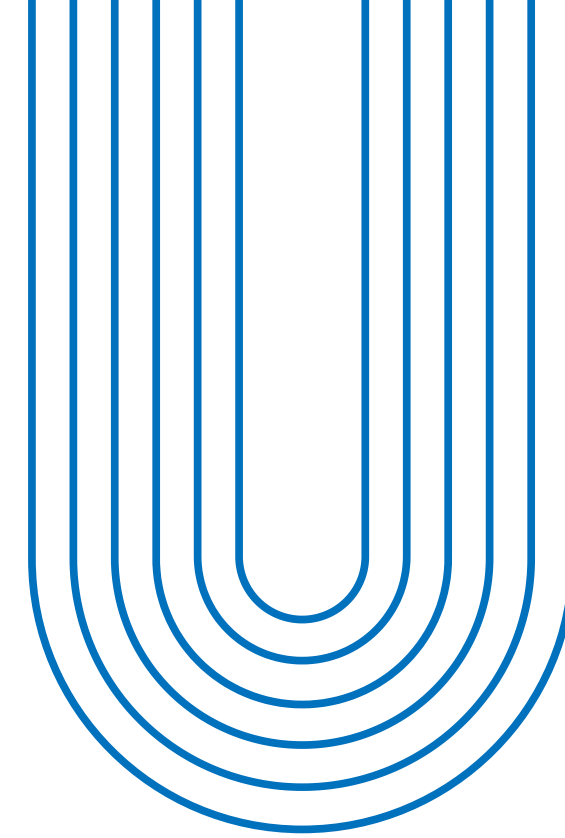


Overview

Implement a PID controller for the F1/10 car to maintain a constant distance from a wall while navigating a track.

- 01 Design and Implement PID Controller:** Develop a PID controller to maintain an ideal distance between the car and the wall, improving autonomous navigation
- 02 Integration with Existing Systems:** Connect the PID controller to the existing mux node to allow seamless switching between manual control (keyboard and gamepad) and autonomous control.
- 03 Testing and Optimization:** Validate the PID controller's performance in simulation and real-world conditions, fine-tuning parameters for optimal results.

Objectives and Goals



Implementation

Implement a PID Controller that allows the car to complete a lap.

Integration

Integrate all the working modes of the car.

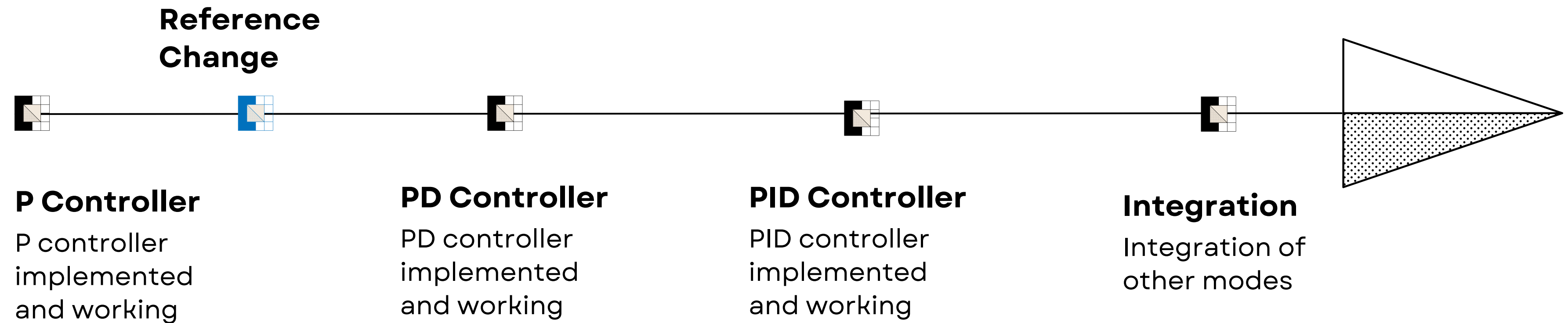
Performance

Tune the controller to perform the fastest lap with a fixed speed, minimize the error and the mean system input.

Timeline

05/22

Development timeline

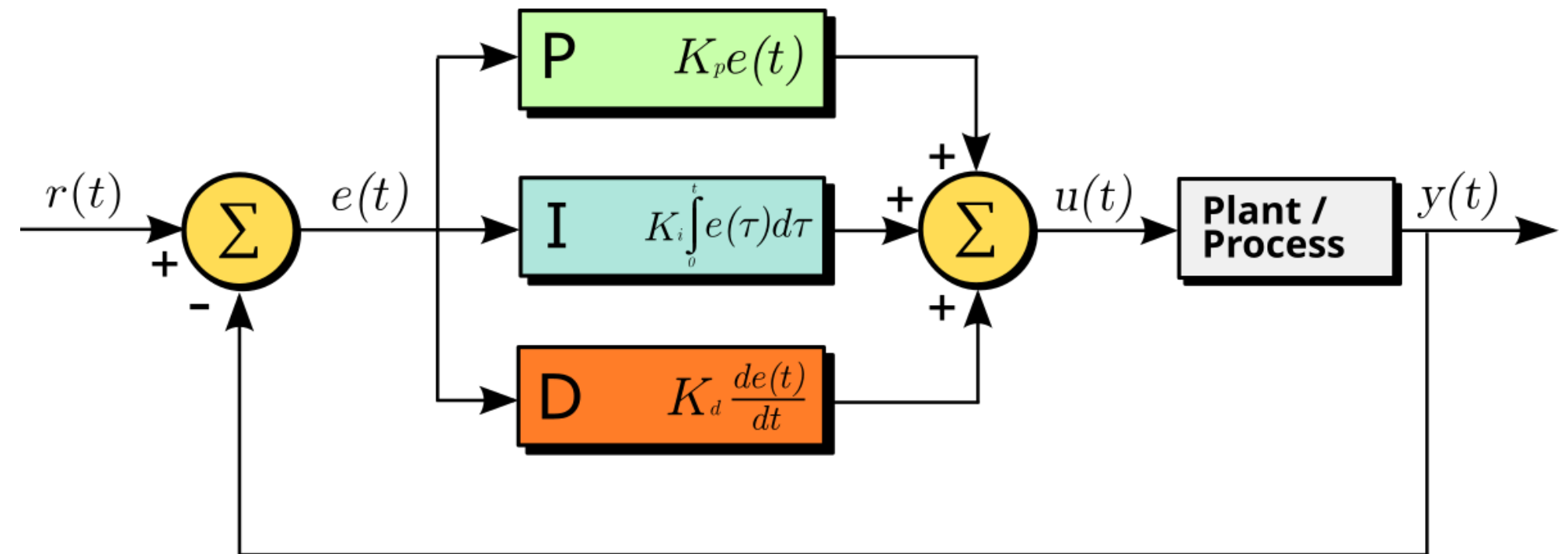


PID Controller

Proportional: Adjusts the control output based on the current error. Higher values result in a stronger response to the error, aiming to reduce it quickly.

Integral: Adjusts the control output based on the rate of change of the error. It helps anticipate future errors by reacting to the error's rate of change, improving stability and reducing overshoot.

Derivative: Adjusts the control output based on the accumulation of past errors. It helps eliminate steady-state errors by integrating the error over time.

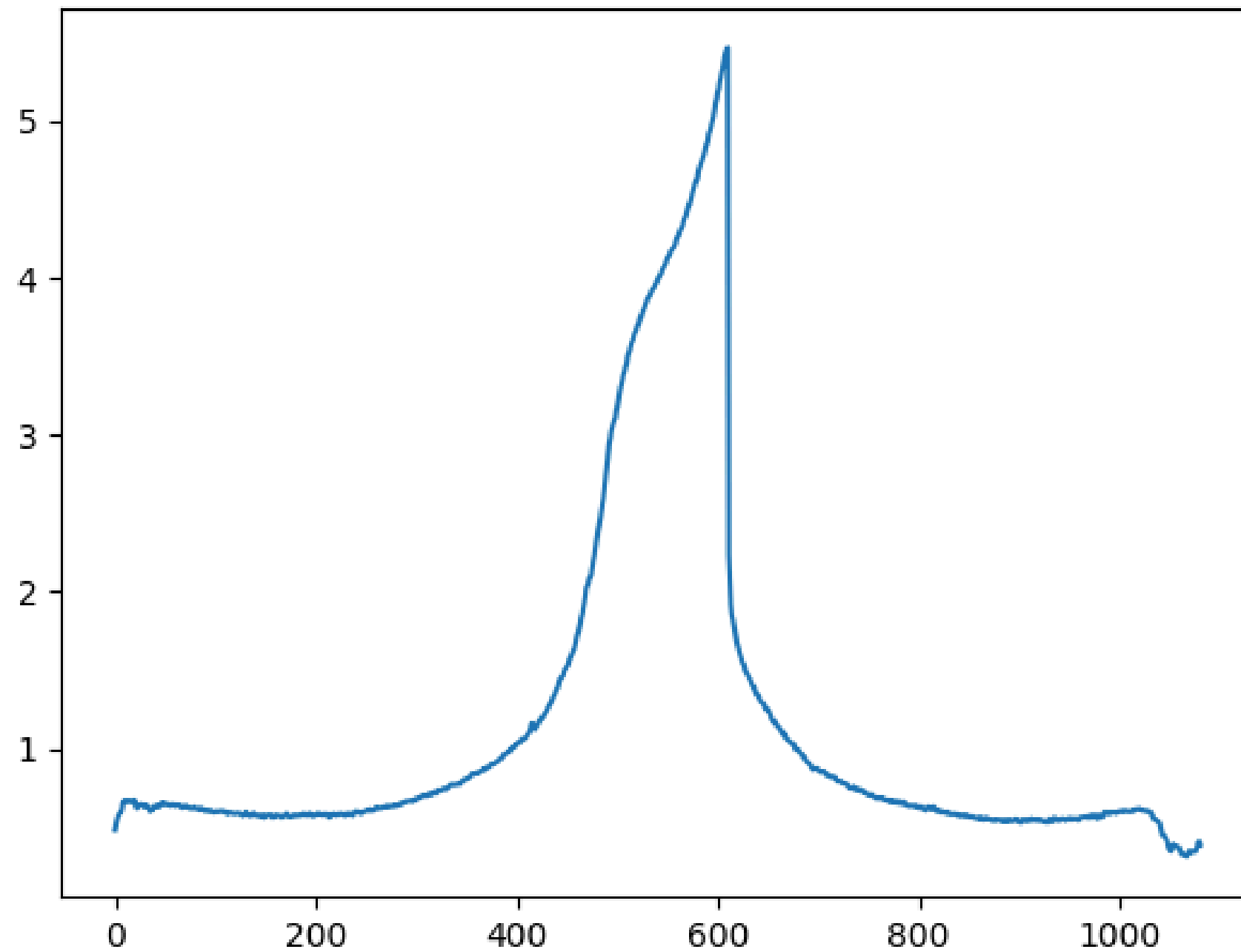


Reference Tracking

- Started by tracking a fixed distance to the wall.
- Cropped the first and last 80 data points of the Lidar.
- Applied a dynamic reference calculated as the average of the minimum distances from both walls (center of the road).

```
std::vector<float> ranges_right(full_ranges.begin()+ 80, full_ranges.begin()+540);  
std::vector<float> ranges_left(full_ranges.begin()+540, full_ranges.end()-80);  
d_objective = (min_element(ranges_right)+min_element(ranges_left))/2; //target
```

Lidar - Range Measurement



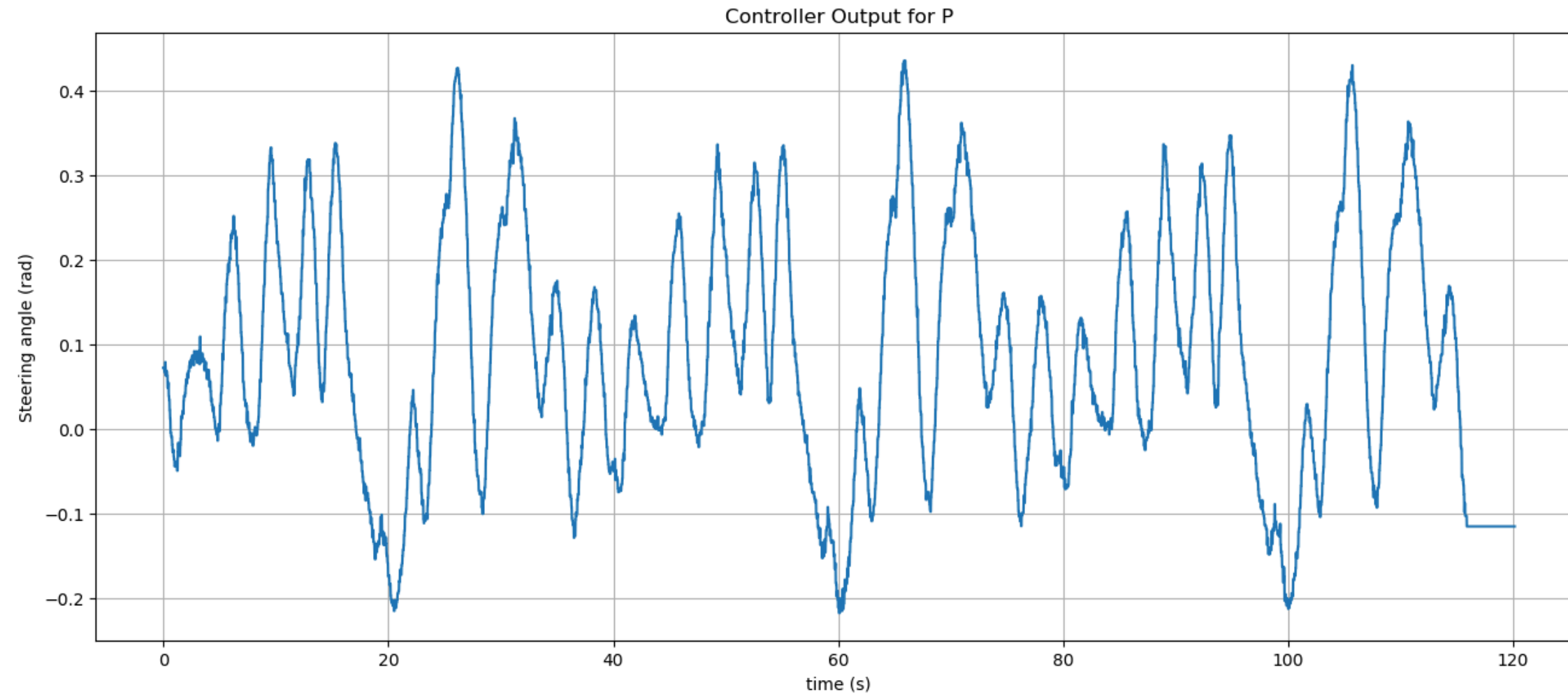
Proportional Controller

- Implemented a proportional controller based on the error from the dynamic reference and the minimum distance acquired from the Lidar.
- Tuned to perform a lap without crashing, aiming on optimizing the performance parameters.
- First implemented the normal error then changed to the projected error.

$$e(t) = d - \Delta d - d_{ref}$$

```
e = min_range - d_objective - L*sin(input_yaw);  
float K_p = (min_index > length_ranges/2)?2.5f:-2.5f;
```

Proportional Controller Result



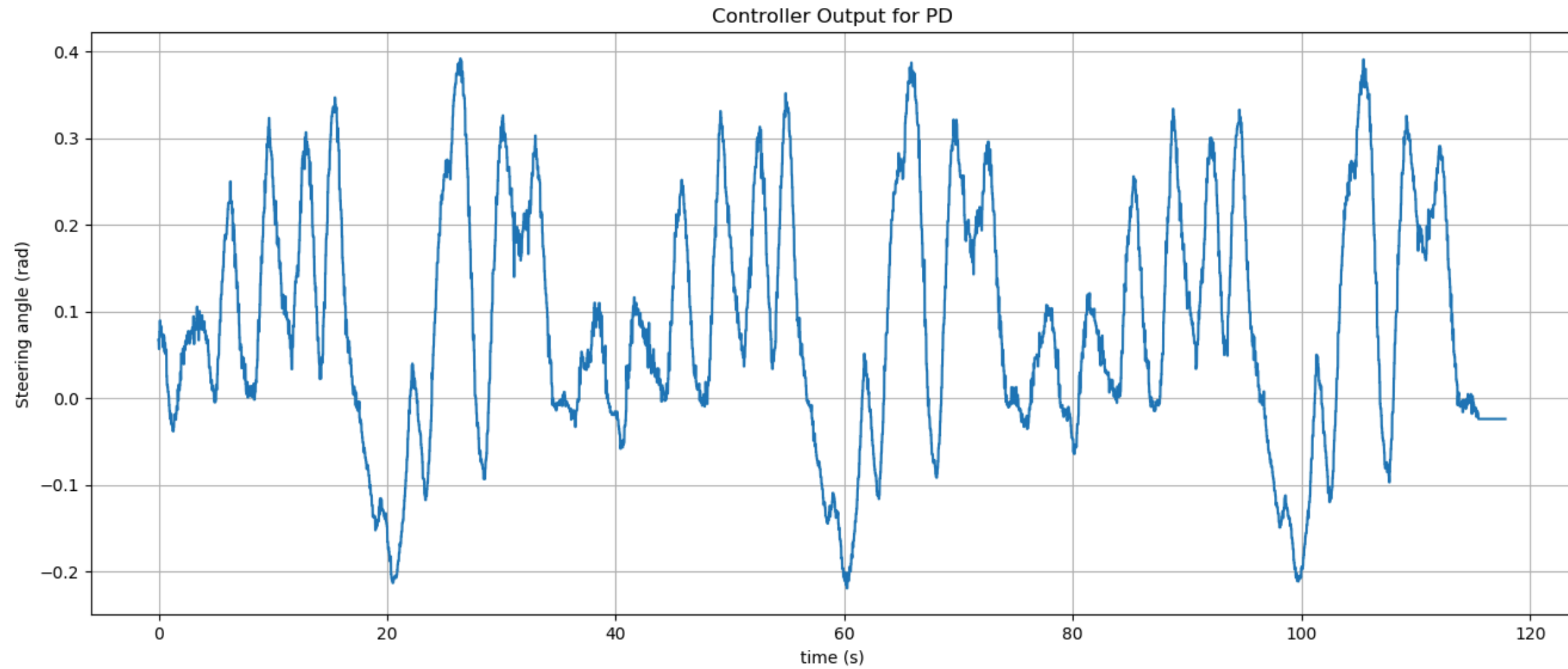
- $K_p = 2.5$
- Error = 162.8
- Mean System Input = 0.1352 rad
- Lap Time = 116 s

PD Controller

- Derivative gain was added using the current error and the last error measured.
- Tuning with Ziegler–Nichols method for classic PD was tried but unsuccessful.
- Gain was tuned based on performance and crash report.

$$input = K_p \times e(t) + K_d \times (e(t) - e(t-1))$$

PD Controller Result



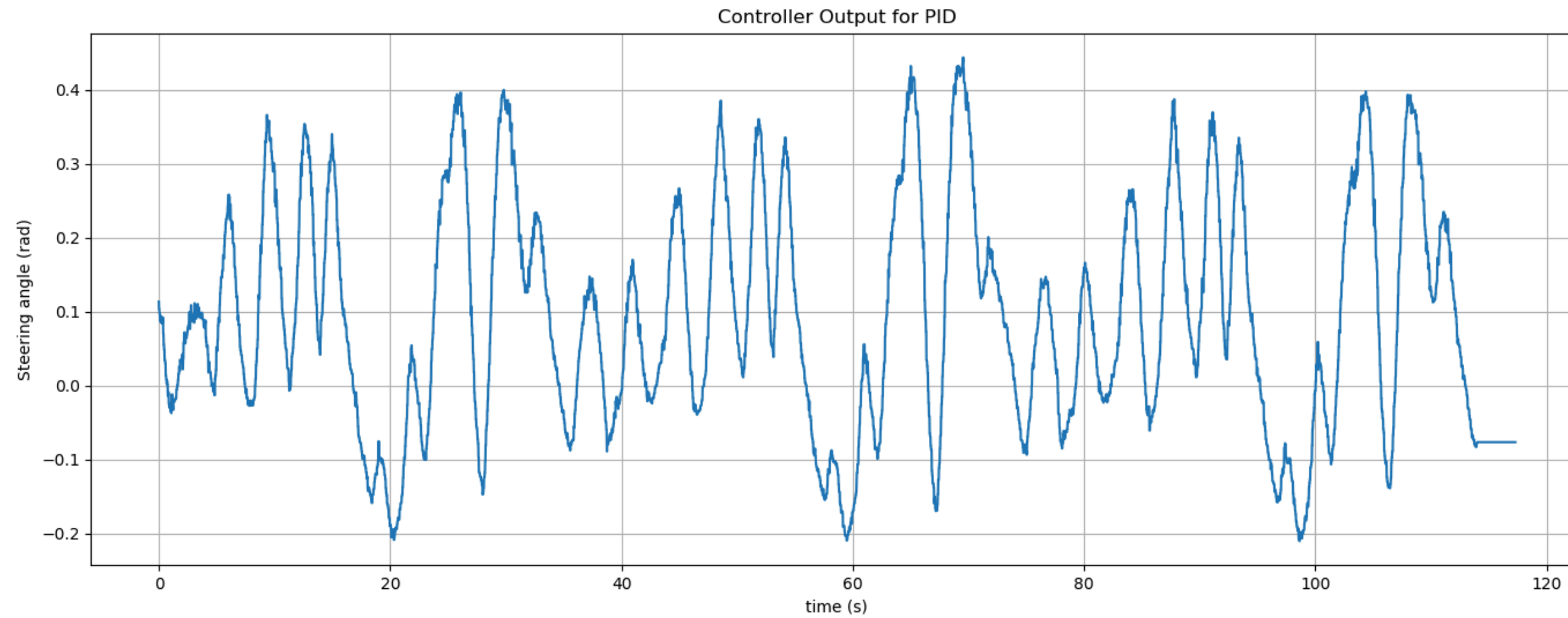
- $K_p = 2.5$ $K_d = 1.0$
- Error = 161.8
- Mean System Input = 0.1271 rad
- Lap Time = 115 s

PID Controller

- A integral part was added to the system.
- The error sum was minimized.

```
sum_err = ( min_index > length_ranges/2)? sum_err + e: sum_err -e;  
input_yaw = K_p*e + K_d*(e_old-e) + K_i * sum_err;
```

PID Controller Result



- $K_p = 2.5$ $K_d = 1.0$ $K_i = 0.01$
- Error = 11.34
- Mean System Input = 0.1423 rad
- Lap Time = 113 s

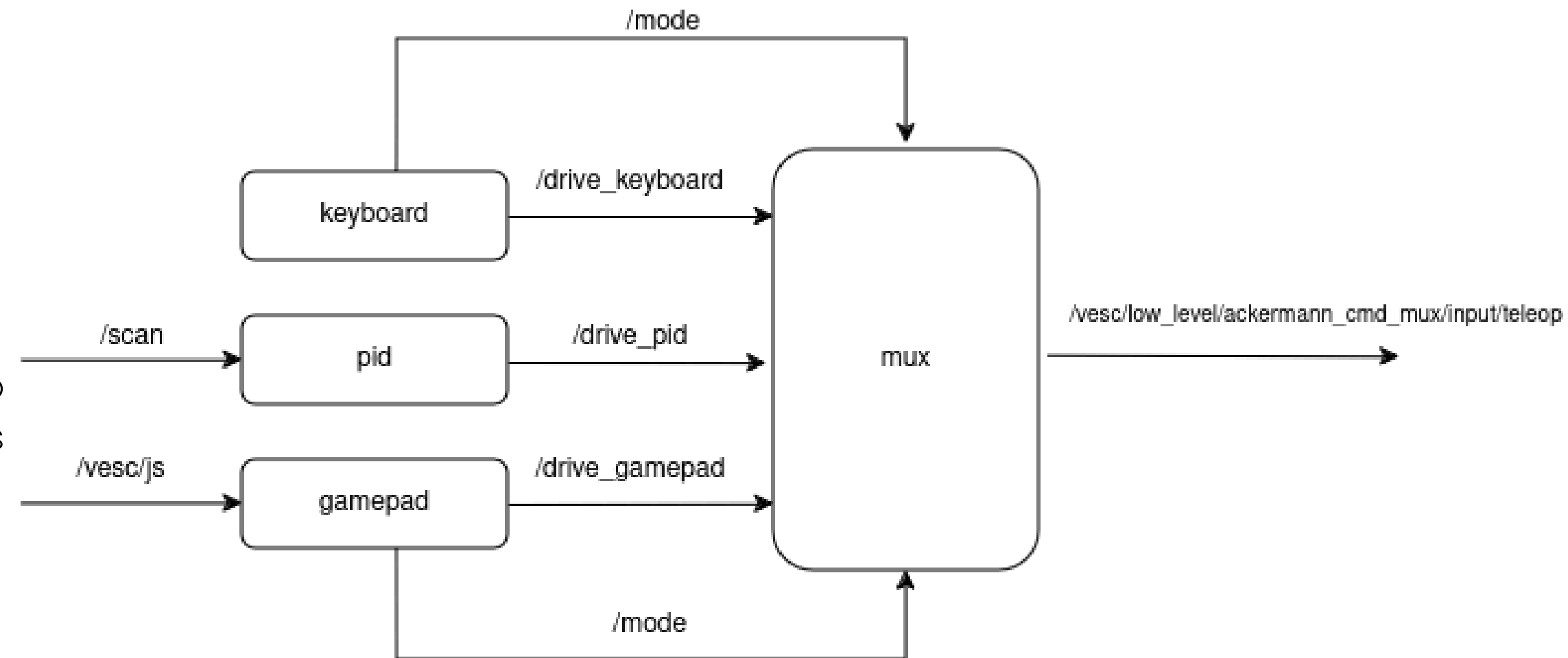
Input Multiplexing

Gamepad: This node subscribes to the joystick topic and forwards incoming commands to the mux. It also publishes on the /mode topic when receiving inputs from the joystick.

Keyboard: Publishes on the mode topic to allow selection of the control mode; publishes motion commands on its own drive topic

PID: This node contains the autonomous control logic. It subscribes to the lidar topic and publishes on its own drive topic

Mux: responsible for the selection of which motion commands to forward to the car topic, according to “/mode”

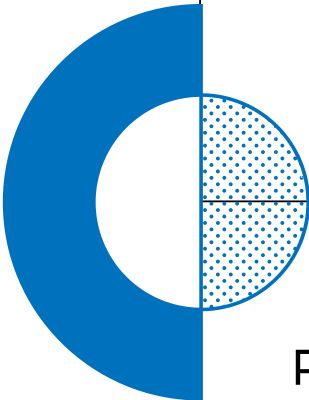


Testing and Results

Implementation: PID was successfully implemented, completing the goal of one lap

Integration: Gamepad, keyboard and autonomous mode were integrated via the mux.

Performance: The performance parameters were measured and the gains tuned.



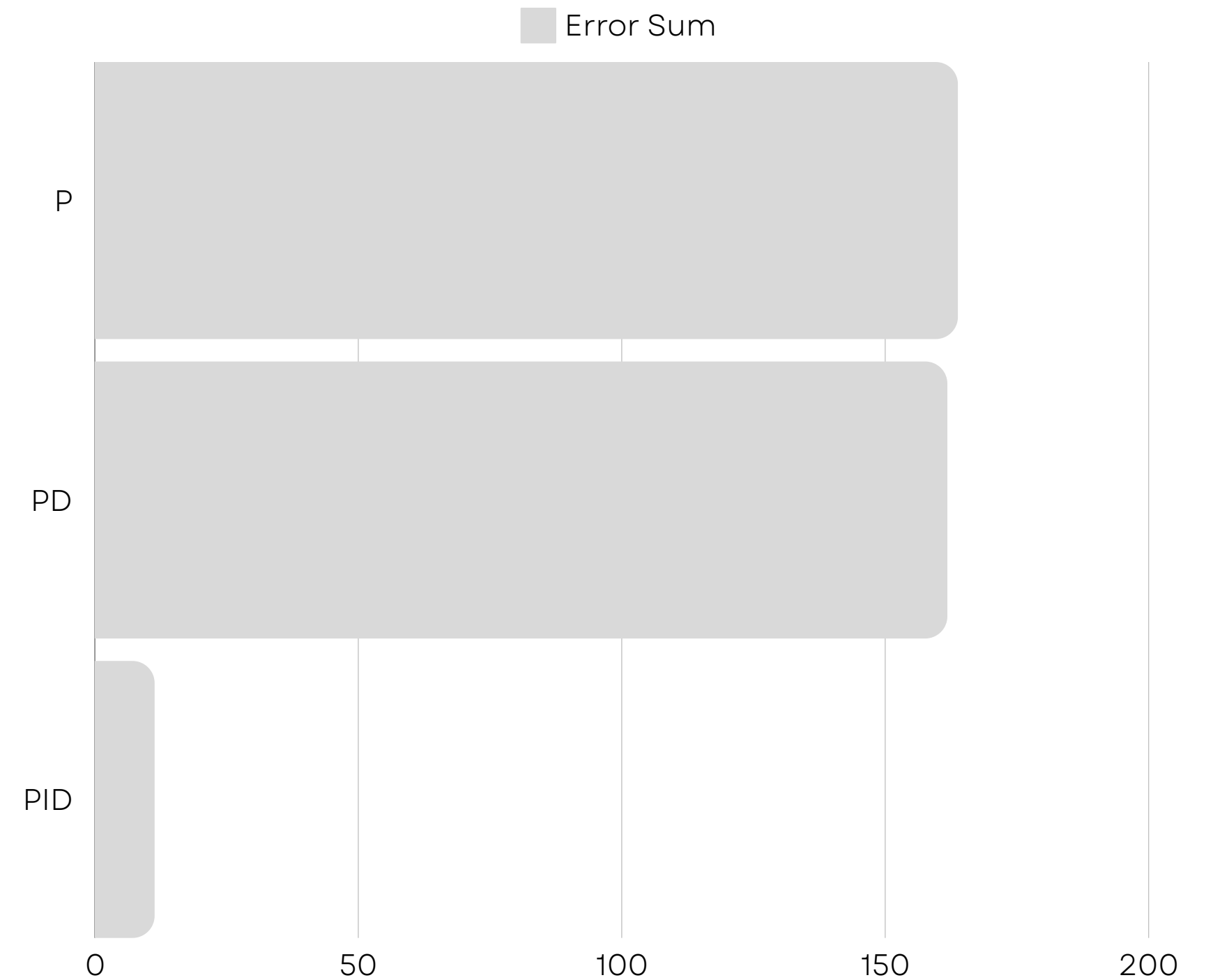
Starts waiting for the keyboard input
Press “p” for autonomous mode
Press “k” for keyboard mode
Press “g” for gamepad mode
Press any key on gamepad for kill switch when on autonomous mode

Error Comparison

P
Error Sum = 163.8

PD
Error Sum = 161.8

PID
Error Sum = 11.34



Input Comparison

P

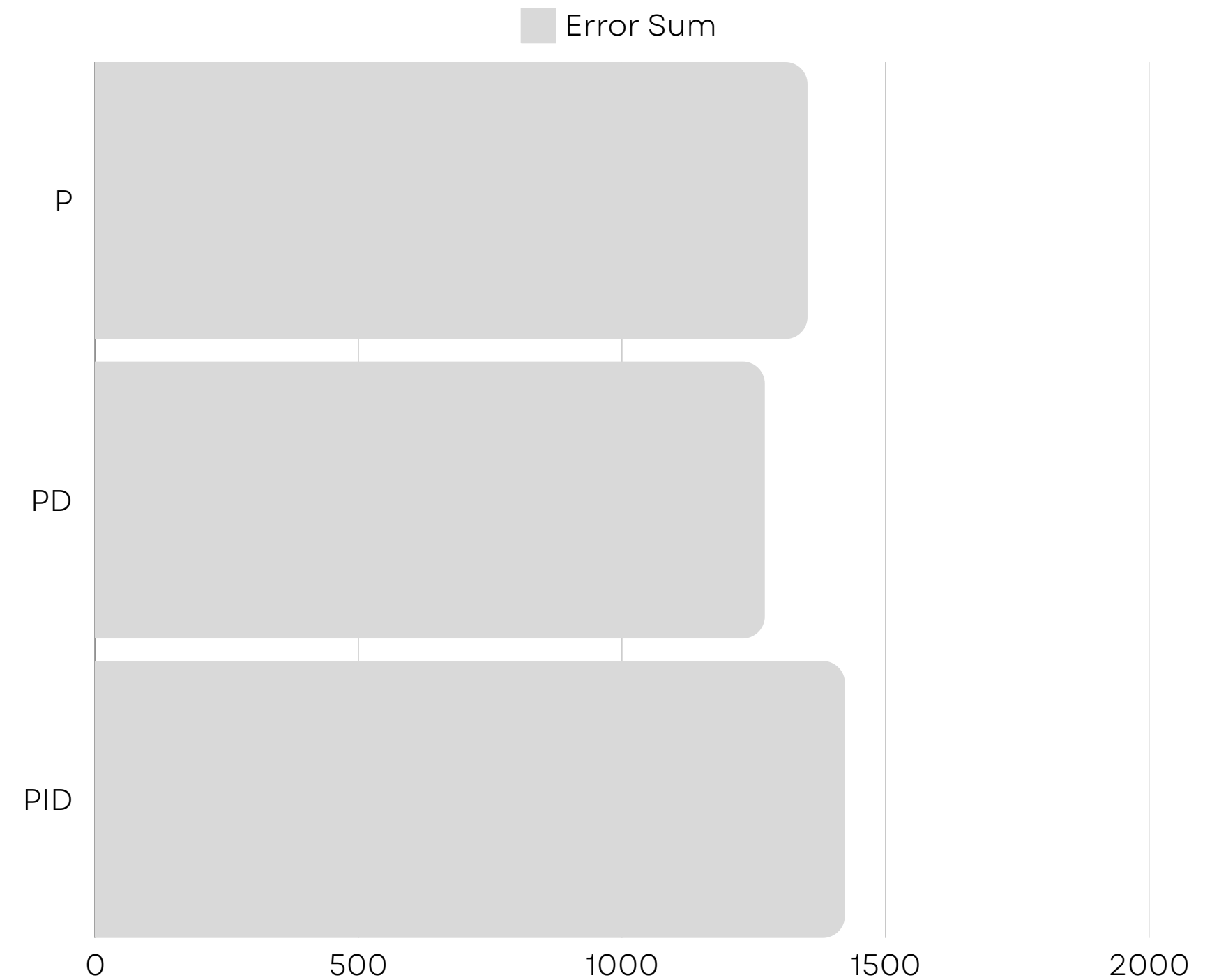
Mean System Input = 0.1352 rad

PD

Mean System Input = 0.1271 rad

PID

Mean System Input = 0.1423 rad



Time Comparison

P

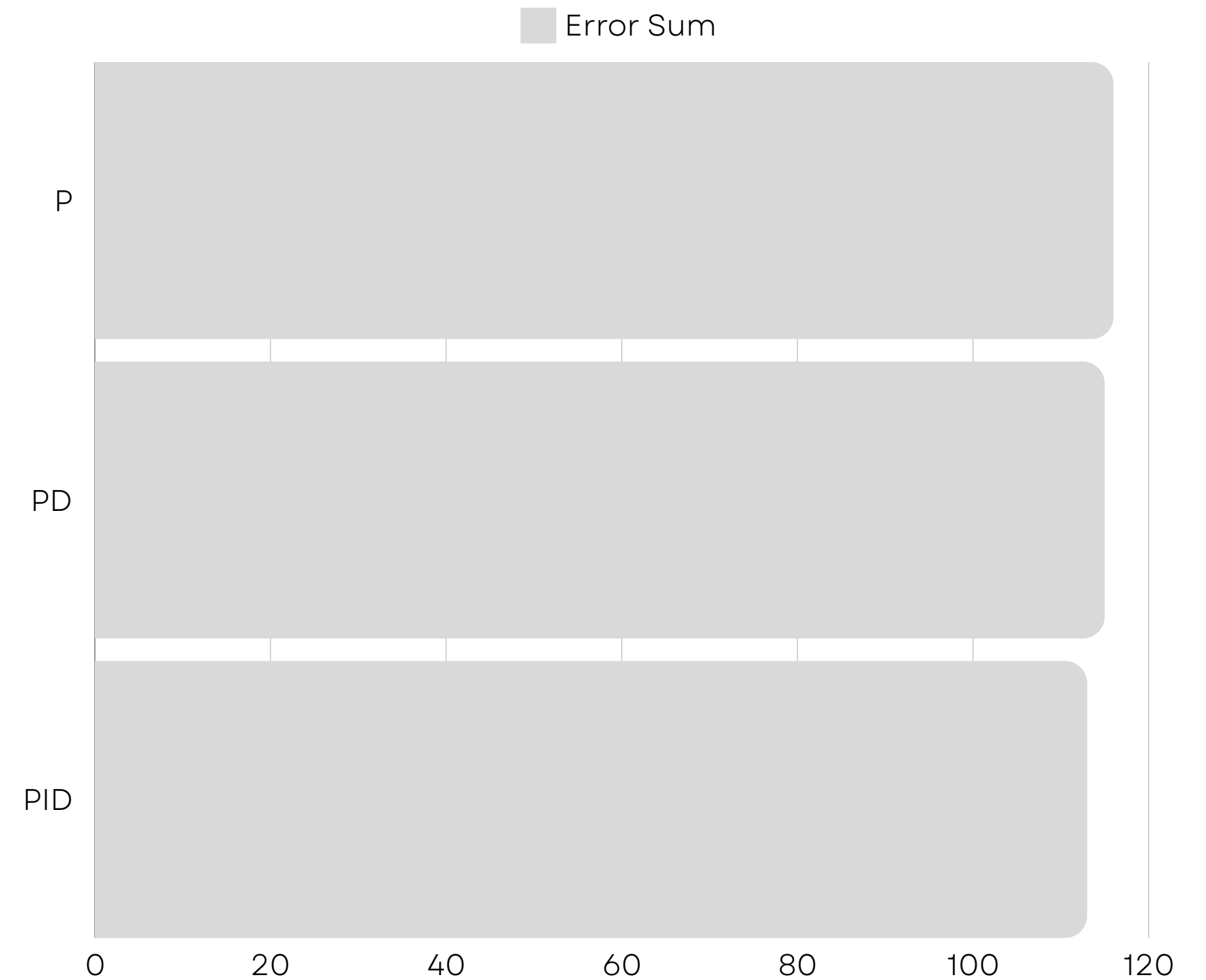
3 Laps Time = 116 s

PD

3 Laps Time = 115 s

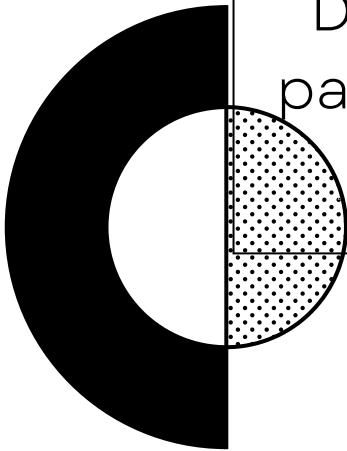
PID

3 Laps Time = 113 s



Challenges & Solutions

WHAT CAN STILL BE IMPROVED?



Challenges

Solutions

Speed was not varied	More time to tune parameters for different speeds
Controllers performed similarly	<ul style="list-style-type: none">• Diferrent road situations• Try better tuning approach
Difficult to establish a reference path (oscillations) due to the track shape	Reference filtering

Conclusion

Q&A Time