Design of a PID Control to Perform Vehicle Trajectory Tracking

1st Marcus Vinícius Leal de Carvalho
Post Graduate Program in Electrical Engineering (PPGEE)
Polytechnic School of the University of São Paulo (USP)
São Paulo, Brazil
https://orcid.org/0000-0002-1298-8517

2nd Leopoldo Rideki Yoshioka
Post Graduate Program in Electrical Engineering (PPGEE))
Polytechnic School of the University of São Paulo (USP)
São Paulo, Brazil
https://orcid.org/0000-0003-2222-1237

3rd Bruno Augusto Angélico
Post Graduate Program in Electrical Engineering (PPGEE))
Polytechnic School of the University of São Paulo (USP)
São Paulo, Brazil
https://orcid.org/0000-0002-2748-5365

***Abstract*—Digital control systems have many applications in the field of automation. In particular, they are essential in developing autonomous vehicles (AVs). This article presents a PID control algorithm to perform trajectory control of an autonomous vehicle. The control acts on steering, acceleration, and braking. The autonomous vehicle must follow a route with a predetermined speed and orientation calculated by the path planner as a Route Definition File (RDF). Furthermore, the control system receives feedback from the position and speed sensors. The design of the control algorithm is evaluated using a simulator. The controller's effectiveness is measured through performance metrics such as the Mean Square Error(MSE) of position and velocity deviations.**

The performance of the controller is sufficient to track the trajectory. The simulation results show the effectiveness of the implemented controller. The lateral error is minimized after applying the tuning process and increasing the number of iterations.

***Index Terms*—CARLA, PID, Lateral Control, C++, Velocity**

I. Introduction

Technological advances and innovations have allowed more significant over the years, the implementation of semi-autonomous vehicles, with systems that assist the driver in perception and vehicle control, such as Electronic Stability Control (ESC), Active Cruise Control (ACC), and other active or passive systems delivered by the Advanced Driving Assistance Systems (ADAS). These systems have significantly reduced accidents and made driving easier for older people, people with attention and sleep disorders and drivers in general [3].

The final intent of the assemblers and researchers in the field is to achieve total automation, vehicle level 5 (SAE). Several Operation Design Domains (ODDs) must be covered to achieve this. To fulfill the ODDs, particular advances need to be implemented in sense, plan and act layers. This work proposes a PID control to cover the last layer of this integration. The control strategy implemented aims to address the AV's lateral and longitudinal motion, ensuring vehicle stability and comfort to the passenger.

The paper has the following organization: In Section II, the environment description, the vehicle model, and its constraints are outlined. Finally, the initial model of the mesh to be controlled is presented, with the definition of the inputs, the plant, and the outputs, as well as result metrics, which are proposed. Section III the PID controller algorithm and strategies to address the problem. Section IV outlines the obtained results of AV's trajectory tracking and persecution. Plots of lateral and longitudinal errors and off-PID effectiveness are exhibited. In Section V, the implemented PID control strategy is evaluated.

II. Contextualization

To address the motion control for an autonomous vehicle (AV), a feedback control system is proposed. The components of this system are shown in fig 1

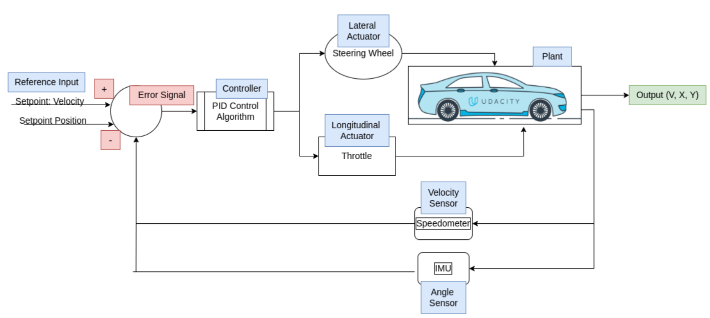

Fig. 1. Block diagram of the feedback control system. Source: [10]

During the vehicle's motion, it receives x and y targets from an array of waypoints calculated by the trajectory planning algorithm. The speed profile at which the vehicle must reach each waypoint is also calculated by the motion planner and sent to another array. Therefore the velocity profile and position are the inputs received by the AV. The PID control algorithm sends proper signals to the actuators (steering wheel and accelerator pedal) to stabilize the vehicle, compensate, and reduce the error. The errors are calculated by the sensors from the analogical data of the plant's behavior. Its value is the difference between the desired velocity and orientation (setpoints) subtracted by the sensor's measurement. It is reduced in a continuous feedback loop until an acceptable threshold value is reached.

A. Problem Statement

A vehicle has both lateral and longitudinal displacement. In this case study, the vehicle employed will be a car that follows the non-holonomic constraints, i.e., it has reduced in space dimension for possible differential movements. This car-like robot follows the bicycle model. The vehicle's lateral dynamic is shown in figure 2

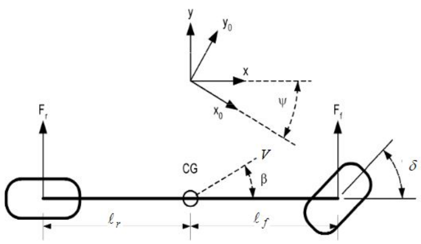

Fig. 2. vehicle lateral dynamics Source: [12]

The vehicle will have to move from point A, in one lane of the road, to point B, on the same side of the lane where it starts moving. However, three static obstacles impede its movement, forcing it to change its route. The trajectory planner calculates the route, which will define the positional (x,y) and speed (v) waypoints the car needs to reach to comply with this calculated route. In summary, the vehicle will progressively change states, going from a current state (S_0) to a final state (S_f) during the mission time $\int_0^T dt$.

$$S_O(t) = x, y, \psi, v \xrightarrow{Actuators} S_F(t + d_t) = x_f, y_f, \psi_f, v_f \tag{1}$$

The following kinematics equations define the members of expression 1

$$x_{t+1} = x_t + v_t cos(\psi) * d_t \tag{2}$$

$$y_{t+1} = y_t + v_t sin(\psi) * d_t \tag{3}$$

$$\psi_{t+1} = \psi_t + v_t / L_f * \delta * d_t \tag{4}$$

$$v_{t+1} = v_t + a_t * d_t \tag{5}$$

At each state transition, the vehicle has errors in the execution of the trajectory:

- Cross Track Error (cte): The distance (x,y) of the vehicle from the trajectory;
- Angle Difference: Difference of the vehicle orientation (ψ) and the trajectory orientation.

Considering the problem's parameters, it is intended to minimize the aforementioned errors through the PID controller, assigning suitable gains to ensure the smooth and safest motion.

B. Environment

The environment in which the plant is submitted is a road race emulated by the CARLA simulator. The AV must track and persecute the trajectory over the road, respecting a margin of lateral error and speed band.

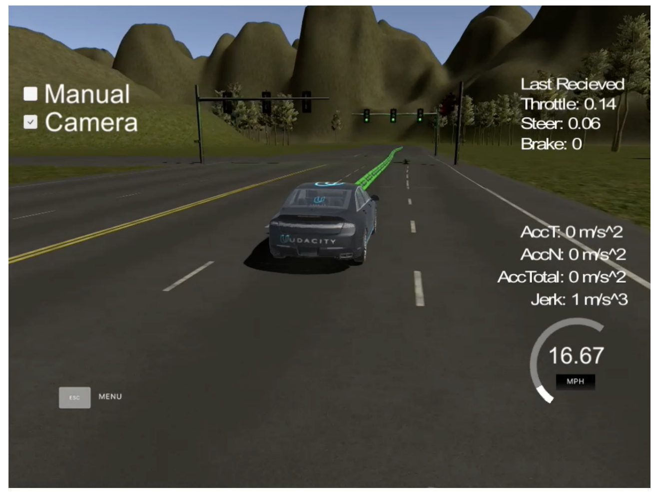

Fig. 3. Highway Environment Source: [12]

C. Vehicle Model

A dynamic vehicle model was selected rather than the kinematic to get a more realistic behavior. In this model, the following parameters are considered:

- Air Resistance and Drag Forces

$$F_{aero} = drag_{coef} * v^2 \tag{6}$$

- Gravitational Force

$$F_{grav} = m * g * sin(\psi) \tag{7}$$

- Rolling Resistance

$$R_{tire} = fric_{coef} * v \tag{8}$$

- Load Force (Resistance force, sum of all above)

$$F_{load} = F_{aero} + R_x + F_g \tag{9}$$

The dynamic model suffers from longitudinal and lateral forces while the vehicle is moving [1]. These forces are computed over the four tires:

$$F_x = f_{x1} + f_{x2} + f_{x3} + f_{x4} \tag{10}$$

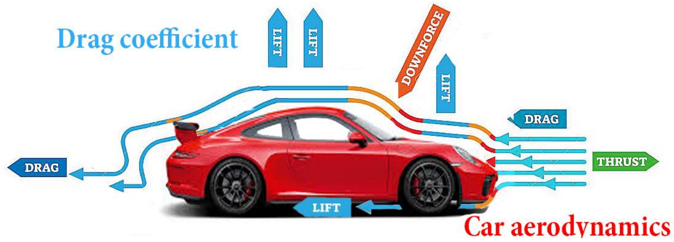

Fig. 4. Vehicle Aerodynamics Force Source: [12]

$$F_y = f_{y1} + f_{y2} + f_{y3} + f_{y4} \tag{11}$$

Finally the Slip angle and Slip Ratio, essential for the lateral and longitudinal displacement, respectively, are also included:

- Slip Angle: The angle of the wheel's velocity vector and the wheel's orientation.

$$\alpha = arctan(wheel_{Vlat}/wheel_{Vlong}) \tag{12}$$

  The force generated by this angle is responsible for turning the vehicle.
- Slip Rate: Mismatch of the vehicle wheel and the expected velocity.

$$slip_{ratio} = wheel_{radius} * w_r/v_{long} * 100 \tag{13}$$

  The slip ratio is required to generate longitudinal force, analogously to the slip angle for the lateral move.

The graphical representation of this modeling is shown in figure 5

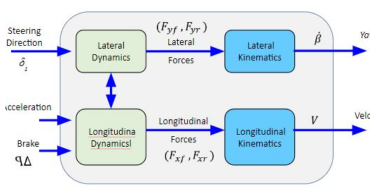

Fig. 5. Vehicle side and longitudinal control model Source: [13]

These parameters were considered reasonable enough to model the controller. However, additional parameters, such as the chassis suspension responsiveness, can be added in future works to get more realistic, in a trade-off, of increasing the complexity of the controller. More sophisticated models are implemented on [6, 8] This dynamic model of the vehicle in C++ code format can be inspected in the following link: Vehicle_Model

III. Methodology

To reach the primary goal of this project, which is following the trajectory properly in a stable manner, a subset of goals are defined below:

- Design of PID controller in C++ for lateral and longitudinal control;
- Simulate the controller output in the Carla simulator using an autonomous vehicle as the plant;
- Based on the plant's output and the AV's behavior, tune the gains and the algorithm based on Digital Control's lectures to minimize the steering and throttle errors.
- Select evaluation metrics, based on literature, to asses the controller performance;
- Plot the relevant data and make conclusions.

A. *Control Model*

The Proportion Integral Derivative (PID) controller was selected to work on steering and throttle actuators to control the dynamic model described in section II-C. This model operates by the product of the coefficients with their respective PID errors that are constantly calculated along the path. The tendency is that after a desired N number of iterations, the error will decrease and stabilize within the tolerance designed for the controller.

$$C_{out} = K_p * P_{err} + K_d * Diff_{err} + K_i * I_{err} \tag{14}$$

The errors are updated at each timestamp for both throttle and steering. The error's arithmetics are written in the following code snippet: UpdateError A boundary limit was set for both of them to ensure that the control outputs did not exceed the physical limit of the throttle and steering actuators. The steering angle is set to a maximum of 35°as recommended by [2].

B. *Errors Definition*

A measurement of how much the AV's controller is tracking and following the trajectory can be done by calculating the errors:

1) Cross-Track Error: The error between the path planner's line and the vehicle's position is the cross-track error (cte). The successor state after time t is defined as:

$$cte_{t+1} = cte_t + v_t * sin(e\psi) * dt \tag{15}$$

Assuming the reference line is a first-order polynomial f, $f(x_t)$ is the reference line, cte at the current state is defined as:

$$cte_t = f(x_t) - y_t \tag{16}$$

Substituting 16 into 15:

$$cte_{t+1} = f(x_t) - y_t + (v_t * sin(e\psi) * dt \tag{17}$$

where
$f(x_t)$-y_t is the current cte, and
v_t*sin(eψ)*dt is the change in error caused by the AV's movement.

2) Orientation Error: The subsequent difference between the actual AV's yaw angle and the desired heading is given by:

$$e\psi_{t+1} = e\psi + v_t/L_f * \delta * dt \tag{18}$$

$e\psi_t$ is the desired orientation subtracted from the current orientation:

$$e\psi_t = \psi_t - \psi des_t \quad (19)$$

Replacing 19 in 18:

$$e\psi_{t+1} = \psi_t - \psi dest_t + (v_t / L_f * \delta_t * dt) \quad (20)$$

where
ψ_t - ψdes_t is the current orientation error
v_t / L_f * δ_t * dt is the change in error caused by AV's movement.

C. Implementation

The PID controller algorithm was implemented in C++ once its performance is vital for a code running on real-time systems, emulating a real-driving scenario. [4] As stated in II-A, this PID controller aims to track and follow a planned trajectory and minimize the cte III-B1 and steering error III-B2. Figure 6 depicts the workflow of the implemented PID controller algorithm.

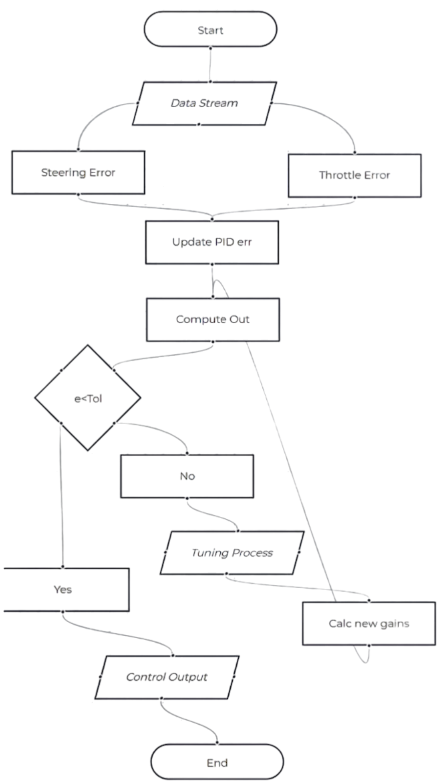

Fig. 6. Workflow of implemented controller code
Source: Own authorship

The pseudo-code of this project is contained in Appendix A. The repository with all the code can be checked at **Marcus'_Github**. The core of PID controller codes implemented is on **pid_controller.cpp**, **pid_controller.h** and **main.cpp** links.

D. Control Model Parameters Tuning

An optimal controller will reduce the cross-track and orientation errors from the vehicle pose and the desired trajectory. Adjusting the coefficients (Kp, Kd, Ki) to fit the actual trajectory to the reference trajectory (setpoint) is time-consuming. Due to the circumstances of a short deadline and the complexity of implementation, a tuning algorithm still needed to be developed. Therefore the gains were adjusted iteratively, approximating the executed trajectory to the expected trajectory.

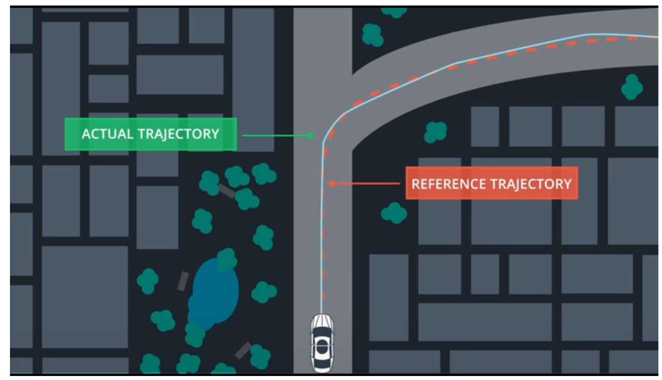

Fig. 7. Minimizing Errors: The Tuning Process Source: [12]

E. Automatically Tune the PID

There are several methods to tune a PID controller to find an optimal value for this controller's gains. OpenAI recommends the following methods:

- Ziegler-Nichols
- Relay Feedback
- Tyreus-Luyben
- Genetic Algorithm
- Model Predictive Control (MPC)
  Evaluating these methods, I would select the MPC, as it can generate the process response in the future. As an autonomous vehicle is already able to predict some maneuvers (thanks to the predictor module), it is reasonable to get proper inputs to feed this sort of controller, optimizing the controller's output to fit the setpoint (steering, throttle, brake) in an efficient, stable and quick manner.

F. Dynamic Model vs Model Free

According to [5], the Model-Free based controllers can be implemented in an easier and reduced time once they don't require a detailed process model. In this way, the controller engineer can eliminate the usage of complex algebra and equations, setting up the controller using simple methods. In addition, the controller presents robustness to noise and changes in the process features. They can adapt to changes in the process dynamics and maintain good control performance even with uncertainties in the process model [7].

The drawback is the obtainment of less acute output [5], resulting in suboptimal control performance, especially in applications with complex or nonlinear processes [7]. fliess2013model states that this sort of controller is also sensible in abrupt changes in velocity signal. Another disadvantage is that if the system changes, additional calibrations are required to maintain the same PID performance. [9] highlight that the algebraic estimation method (Model Free) requires high computational efforts, requiring more processing power and increasing hardware price.

To summarize, designing a model-based controller requires expertise from the plant to be controlled and awareness of the variables, gains, and parameters which influence it. Depending on the application, this can be time-consuming and costly, making a Model Free implementation more feasible. On the other hand, a model-based controller can guarantee more accuracy and more clearance of the System understanding however, if not properly designed can be sensitive to change in the model.

G. *Improving the PID Controller*

To improve the PID controller for an autonomous vehic application, a Machine Learning assignment for the gai (K, P, I) could guarantee optimal performance. In this wa for a specific application, I would define specific desire metrics, such as **trajectory smoothness, passenger comfor distance to the obstacle**, etc. Then I would use Softwa in the Loop (SIL) and Hardware in the Loop (HIL) to te my Machine Learning (ML) algorithm, which would feed th KPI gains automatically with random values in simulatio After X time or Y iterations, select the range of gains valu that best returned good values for smoothness of trajector passenger's comfort, short and safest distance to the obstacl etc. Therefore I would get this range of acceptable KPIs use by my ML algorithm and refeed them into the simulation (SIL-HIL) to tune these gains more and more.

IV. Results

At the first steps of the implemented controller III-C, th vehicle did not present acceptable behavior and collided wi the front leading vehicle. While tuning the steering control, th AV incremented the turn and hit the wall, as can be checke

Fig. 8. Bad Controller's Output. Source: Own Authorship

The gains for these cases were set to low values, expecting the AV to correct its position for reasonable values once the proportional, derivative, and integral errors were constantly updated. However, applying appropriate coefficients was crucial to getting satisfactory results. The coefficients were tuned empirically, changing the values as the vehicle's behavior in the simulator approached an ideal condition. It was required to increase the proportional gain (Kp) significantly compared to the Kd and Ki. This gain is responsible for steering the car toward the center line of the trajectory (against the cross-track error). The differential portion (Kd) was needed to counteract the proportional trend to overshoot the trajectory's line to the other side, avoiding the back-and-forth movement that causes motion sickness. Finally, the integral portion tries to eliminate a possible bias that could prevent the error from being eliminated. As the AV in the simulator had no bias, the integral gain (Ki) was set close to zero. The throttle profile and steering profile can be checked respectively in charts 9 and 10.

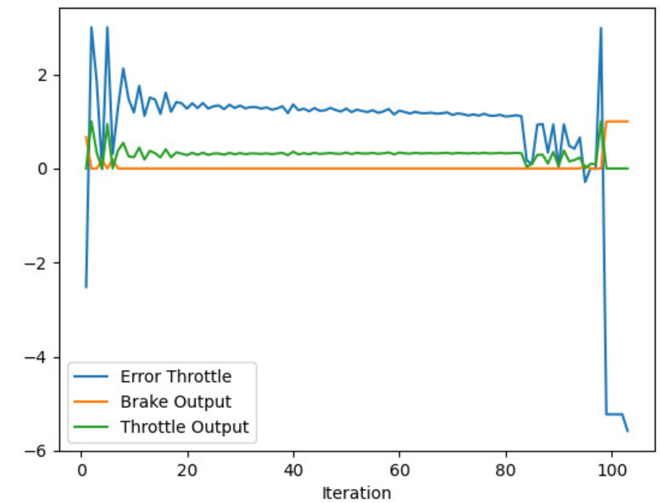

Fig. 9. Throttle Output and error. Source: Own Authorship

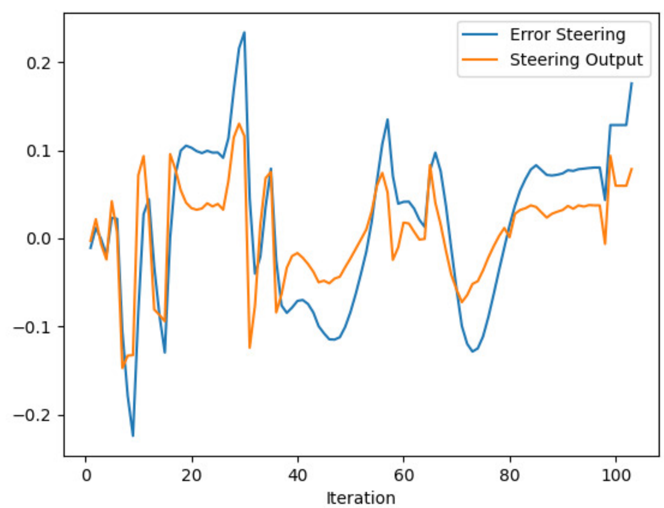

Fig. 10. Steering Output and error. Source: Own Authorship

The throttle error was minimized from the beginning of the vehicle's movements, demonstrating the efficiency of PID control for speed. Nevertheless, the steering control is not considerably minimized from the beginning of the motion. The changes in the steering angle error are more random, with more significant variations in peaks and valleys. A video from the simulation and the performance of the implemented pid controller can be visualized at **Simulation Video**: https://www.youtube.com/watch?v=uBeim-YSiV8

V. Conclusion

In this work, a PID controller was implemented to achieve efficient trajectory tracking. The controller actuated on the vehicle's lateral and longitudinal movements. The algorithm approached the actual car path within the reference path with reduced cte and orientation errors as the number of iterations increased. Due to the short time available for the execution of the project, there was not enough time to implement the optimization algorithm for the control coefficients (Kp, Kd, Ki), called gradient decedent or twiddle. Furthermore, the author has not found enough time to apply other control methods, using transfer functions to obtain lower error rates and achieve more sophisticated control models. These challenges will be implemented in future works. A benchmark of different controllers [11] is also intended.

Acknowledgment

Acknowledgment to FUNDEP for the Ph.D. scholarship, professor Augusto Angélico for the good lectures provided during this discipline, and my advisor, professor Leopoldo Yoshioka for the assistance during this project's development.

References

[1] Egbert Bakker, Lars Nyborg, and Hans B Pacejka. "Tyre modelling for use in vehicle dynamics studies". In: *SAE Transactions* (1987), pp. 190–204.

[2] Dainis Berjoza et al. "Research in kinematics of turn for vehicles and semitrailers". In: *Proceedings of 7th International Scientific Conference on Engineering for Rural Development*. 2008, pp. 29–30.

[3] Marcus Vinıcius Leal de Carvalho et al. "A review of ROS based autonomous driving platforms to carry out automated driving functions". In: (2022).

[4] *Five Skills Self-Driving Companies Need*. https://medium.com/hackernoon/five-skills-self-driving-companies-need-8546d2aba7c1. Accessed: 2022-12-27.

[5] Michel Fliess and Cédric Join. "Model-free control". In: *International Journal of Control* 86.12 (2013), pp. 2228–2252.

[6] Andrew Jacob Gray. *An Integrated Framework for Planning and Control of Semi-Autonomous Vehicles*. University of California, Berkeley, 2013.

[7] *Introducing ChatGPT research release*. https://openai.com/blog/chatgpt/. Accessed: 2022-01-31.

[8] Jason Kong et al. "Kinematic and dynamic vehicle models for autonomous driving control design". In: *2015 IEEE intelligent vehicles symposium (IV)*. IEEE. 2015, pp. 1094–1099.

[9] Matheus Schwalb Moraes. "Algebraic derivative estimation applied to nonlinear control of magnetic levitation." PhD thesis. Universidade de São Paulo, 2015.

[10] Rajesh Rajamani. "Lateral vehicle dynamics". In: *Vehicle Dynamics and control*. Springer, 2012, pp. 15–46.

[11] Chinmay Vilas Samak, Tanmay Vilas Samak, and Sivanathan Kandhasamy. "Proximally Optimal Predictive Control Algorithm for Path Tracking of Self-Driving Cars". In: *Advances in Robotics-5th International Conference of The Robotics Society*. 2021, pp. 1–5.

[12] *Self-Driving Car Engineer*. https://www.udacity.com/course/self-driving-car-engineer-nanodegree--nd0013. Accessed: 2022-12-27.

[13] *Veículos Autônomos Modelamento cinemático e dinâmico*. http://tiny.cc/9rk2vz. Accessed: 2022-12-28.

APPENDIX

PseudoCode

```
# PID Controller - Pseudocode
import libs;
#### Variables and Functions Initialization
#Initialize the gains and assign values
vector P = [Kp, Kd, Ki];
#Set Control Constraints (min and max values)
Set lim_s = [Steer_min, Steer_max];
Set lim_t = [Throttle_min, Throttle_max];
#Get Ego Vehicle State
Get state_pose = ego_vehicle.location.pose (x,y,z);
Get state_orientation = ego_vehicle.location.orientation (yaw);
Get state_velocity = ego_vehicle.speed (v);
#Get Path Planner Trajectory (Waypoints)
Get wp_x, wp_y, wp_v, wp_yaw, goal_wp = motion_planner.generate_path()
#Get Motion Planner Actions (Velocity Profile)
Get des_speed, behavior, lead_car_state = velocity_profile.generate_trajectory
#Set Obstacle spot by use of sensor's data
Set occupied_grid = set_obst();
#Calculate the PID errors (proportional, differential, integral)
# which feeds the controller: Reference_value - Plant's output
p_err, diff_err, int_err = pid.error()
#---------------------------------------------------------------
#Main Control Loop
int main ()
{
"Starting Server. Streaming Data to client"
"Data in the Loop"
while data:

#Get current time, store the previous time for differential term
    t = newupdate.time(dt);
    delta_t = t - prev_t;
    prev_t = t;
#-----------------------Steering------------------------------
#Compute Steering Error = Desired Heading - Actual Heading
    st_err = atan2(pose_f, pose_o) - yaw;
        #atan2(pitch) = reference    # yaw = plant's output
#Compute Steering sent by PID steering control
    st_out = kp*cte + kd*diff_err + Ki*int_err;
#-----------------------Throttle------------------------------
#Compute Throttle Error = Last waypoint velocity - Current speed
    thr_err = v_l - v; # v_l = reference, v = plant's output
#Compute Throttle sent by PID throttle control
    thr_out = kp*cte + kd*diff_err + Ki*int_err;
#check if acceleration or deceleration
    if throttle > 0:
        thr_out = thr;
        brk_out = 0;
    else
        thr_out = 0;
        brk_out = -thr;
}
```

Fig. 11. PID controller Pseudocode Source: Own Authorship