CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Book Chapter

RosDrive: An Open-source ROS-Based Vehicular Simulator for STEM Control Systems Classes Tutorial

Enio Filho*
Jones Yudi
Mohamed Abdelkader
Anis Koubâa*
Eduardo Tovar*

*CISTER Research Centre

CISTER-TR-220201

2022

RosDrive: An Open-source ROS-Based Vehicular Simulator for STEM Control Systems Classes Tutorial

Enio Filho*, Jones Yudi, Mohamed Abdelkader, Anis Koubâa*, Eduardo Tovar*

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: enpvf@isep.ipp.pt, jonesyudi@unb.br, mohamed.abdelkader@systemtrio.com, aska@isep.ipp.pt, emt@isep.ipp.pt

https://www.cister-labs.pt

Abstract

The study of control systems in the engineering courses is quite complex, given the difficulty of some teachers in exemplifying and allowing the student to understand how such systems affect the environment. In this context, the STEM methodologies aim to fill this gap between the traditional classes and the student comprehension of the topic through the active learning process. Realistic open-source simulators can be interpreted as one solution for this STEM implementation, allowing students to test, modify and create different configurations and sensors with a low-cost environment.

This work presents a flexible open-source 3D simulation framework, based on ROS, of a line follower vehicle, using an embedded PID controller, a camera for processing and detecting lines, and sonars for detecting and avoiding obstacles. This simulator integrates several controller systems, allowing the student to build consistent skills in control and related areas, analyze the impacts of models configurations, and extends its knowledge to new techniques.

RosDrive: An Open-Source ROS-Based Vehicular Simulator for STEM Control Systems Classes Tutorial

Enio Vasconcelos Filho, Jones Yudi, Mohamed Abdelkader, Anis Koubaa, and Eduardo Tovar

Abstract The study of control systems in the engineering courses is quite complex, given the difficulty of some teachers in exemplifying and allowing the student to understand how such systems affect the environment. In this context, the STEM methodologies aim to fill this gap between the traditional classes and the student comprehension of the topic through the active learning process. Realistic open-source simulators can be interpreted as one solution for this STEM implementation, allowing students to test, modify and create different configurations and sensors with a low-cost environment. This work presents a flexible open-source 3D simulation framework, based on ROS, of a line follower vehicle, using an embedded PID controller, a camera for processing and detecting lines, and sonars for detecting and avoiding obstacles. This simulator integrates several controller systems, allowing the student to build consistent skills in control and related areas, analyze the impacts of models configurations, and extends its knowledge to new techniques.

Keywords Control systems · Education · Simulator · STEM · Open-Source · ROS

E. Vasconcelos Filho (✉) · A. Koubaa · E. Tovar
ISEP, CISTER Research Centre, Rua Alfredo Allen 535, 4200-135 Porto, Portugal
e-mail: enpvf@isep.ipp.pt

A. Koubaa
e-mail: akoubaa@psu.edu.sa

E. Tovar
e-mail: emt@isep.ipp.pt

J. Yudi
Automation and Control Group, University of Brasilia, Brasilia, Brazil
e-mail: jonesyudi@unb.br

M. Abdelkader · A. Koubaa
Prince Sultan University, Riyadh, Saudi Arabia
e-mail: mabdelkader@psu.edu.sa; mohamed.abdelkader@systemtrio.com

M. Abdelkader
Systemtrio Electronics L.L.C, Abu Dhabi, United Arab Emirates

A. Koubaa (ed.), *Robot Operating System (ROS)*,
Studies in Computational Intelligence 1051,
https://doi.org/10.1007/978-3-031-09062-2_5

1 Introduction

Control system techniques are one of the most significant challenges in several engineering courses. Since it requires extensive mathematical background, a theoretical load is quite extensive, requiring effort to learn by students and teachers. Moreover, there is still an inherent difficulty in transporting the studied theory to practice, making it challenging to retain learning [1]. Thus, alternative teaching techniques [2] can facilitate knowledge production and construction of the skills expected by the agents involved. In addition, [3] concludes in their work that the student's perception of applicability and the ability to construct different solutions is a motivator for the search for more knowledge.

This line of education development puts the student as a producer of dynamic and practical knowledge. It should be encouraged to take an active and autonomous attitude and not necessarily follow pre-established models [4]. So, the student can go further and propose new solutions to existing problems and even create different issues. Active student engagement in the learning process also helps to keep the motivation to research and learn [5], using Active Learning techniques. Thus, integrating different areas of knowledge, experimentation, and implementation allows the student to retain more excellent expertise and develop new skills. This integration of knowledge is called STEM—Science, Technology, Engineering, and Mathematics [6].

A standard solution in many universities is using pre-defined laboratory sessions, using commercial kits such as [7–9]. Although such solutions are attractive, efficient, and robust, they are often expensive and not flexible for experimenting and developing different solutions. Nevertheless, using Arduino development kits has shown promising results as a learning tool [10]. This study suggests the development of kits that can be used throughout the semesters, gradually increasing the project's difficulty [11] and even in specific dynamics and control systems projects [12].

In the same line of knowledge integration, other low-cost projects have been developed and implemented, giving students greater flexibility in experimenting with techniques and knowledge. For example, in [13], the authors proposed an educational line-following robot based on Arduino, allowing the implementation of low-level control techniques. A similar application is presented in [14], proposing an even lower cost robot with less flexibility. The increased complexity of possible control algorithms is achieved on other platforms, such as those seen in [15, 16]. However, such applications imply a significant increase in project costs. Thus, although Arduino-based solutions integrate the theoretical model and practice regarding control aspects, they present a limitation regarding the complexity of the algorithms, given the restriction of processing capacity and design flexibility, due to the need to purchase different sensors.

A solution that combines a low-cost implementation with flexibility and allows knowledge retention through experimentation and active learning is based on realistic open-source simulators. In STEM, a simulator represents a crucial stage of development and education, reducing the time to produce prototypes. Thus, emulating

a real scenario with physical interactions allows the development of safety tests in different environments and situations. As a result, it is possible to experiment with techniques, analyze results and propose solutions flexibly, with great speed and less cost.

The work done in [17] presents some of these tools, comparing simulators such as Webots [18], Gazebo, and ROS [19], using criteria such as supported operating systems, programming languages, documentation, tutorials, among others. In addition to these tools, others have been developed over time, such as the one presented in [20], where a virtual laboratory is designed so that students can experiment with models of line-following robots for competitions. However, such a simulator does not allow the 3D visualization of the models, allowing only the testing of the proposed algorithms. Another interesting simulator is proposed in [21], which presents CARLA, an open-source simulator aimed at autonomous-driving research in this work. It is a very realistic simulator with many items, with several physical interactions between the components. However, despite being an extensible platform for new developments, its vehicle control methods are limited to artificial intelligence learning models without control models.

The authors of [22] present a simulator that uses a competition model to teach robotics based on ROS. An autonomous robot capable of traveling a path is used in this simulator, following directions on the track. Such a simulator showed promising results when crossing the designated paths but presented the limitation of not using a realistic vehicle model or even different control models. It has also been used in competition simulation, which increases students' comprehension and stimulates self-learning [23, 24].

Seeking to use the advantages of a simulator capable of emulating realistic vehicles, RosDrive is presented. A flexible platform based on ROS and the 3D simulator, Gazebo, for studying different models of vehicle control. RosDrive uses an electric vehicle model [25], with several sensors capable of covering different routes and avoiding obstacles. Thus, the student will be able to implement additional control strategies, analyze the system's responses, and visualize the impacts of theoretical models and their variables on the simulated scenarios. In addition, the tool allows the use of different strategies in different vehicles, allowing the comparison between the adopted models. For instance, a line follower controller mode with obstacle avoidance will be implemented to exhibit the simulator results. The tool's flexibility allowed its extension for the study of communication models [26], the development of hardware in the loop (HIL) simulation [27], and the implementation of the same control model on a testbed platform [28]. As it is an open-source tool, the full code access is provided in Sect. 5, for general use, with all the necessary steps for its installation. The general simulator environment is illustrated in Fig. 1. The concrete contributions of this paper are presented below:

– Present RosDrive, a realistic open-source 3D vehicle simulator that allows the students to apply different control models techniques, improve their practical knowledge in the control area, and create new strategies to perform vehicle movements;

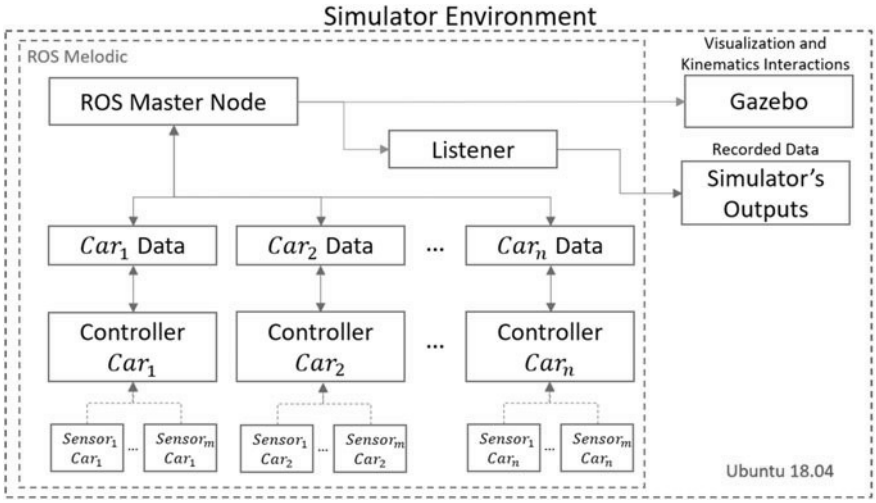


Fig. 1 General simulator architecture

– Introduce a flexible simulator architecture, that allows modules exchange, parameters configurations, and system response analysis and visualization;
– Demonstrate the simulator flexibility due to the use of a camera and sonar sensors to perform a Line Follower algorithm, and an Obstacle Detection and Avoidance strategy integrated with a PID Cruise Controller model;

This paper is organized as follows: Sect. 2 shows the Simulator Architecture, the leading technologies, and the minimal system requirements. The control and perception modules including the speed control algorithm, the image processing module with the heading controller, and the collision avoidance module are introduced in Sect. 3. The results of designed scenarios are presented in Sect. 4, while the main conclusions are drawn in Sect. 5. Finally, the software installation and execution instructions are shown in Sect. 5.

2 Simulator Architecture

This section will introduce the simulator tools and their general architecture, providing details about the vehicle model and data analysis.

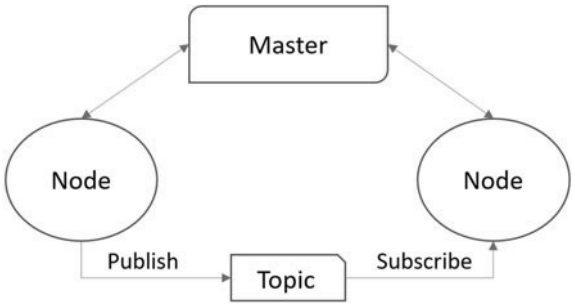


Fig. 2 Publish/subscribe model

2.1 Robot Operating System (ROS)

ROS is an open software developed by Open Source Robotics Foundation. It is a robotic middleware with many software frameworks for robot software development. It provides hardware abstraction, enabling users to avoid low-level problems, with profound device control, communication between nodes and processes, and packet management. ROS-based functions are realized in nodes that may post, receive and reproduce control features, sensor data, state of the node, or general messages. ROS is not a real-time framework or a Real-time Operating System (RTOS). This project will be used in ROS Melodic distribution.

The basic concepts of ROS are nodes, Master, messages, and topics. The Master node works as a central node of the system, storing data and information regarding the ROS Nodes. Nodes inform their registration information to the Master and then can receive data from other nodes. The Master is also responsible for reporting the nodes, using Callbacks, if new information or connections are made. The nodes exchanges messages using the publish/subscribe method, as described in Fig. 2.

Due to its flexibility, ROS has been used in several vehicular applications, such as ground [29], aerial [30], and water [31] and many other robotic platforms. As a consolidated open-source community, several new libraries are available and supported, at the same time that it is highly portable between platforms, including embedded platforms [32]. The extensive material allows a quick learning curve for the student, enabling a simple familiarization with the commands and interfaces and quickly creating new modules.

2.2 Gazebo

One of the critical aspects of a learning-oriented system is its ability to present the results of user interactions intuitively. Thus, the high capacity of ROS to integrate with other platforms shows itself to be a competitive advantage since its functionalities can be extended, expanding the experimentation horizon. For example, integration with a robotic simulation tool helps to visualize the iterations between objects simply,

aiding in learning [33]. One of the most used tools for robotic simulation in ROS is Gazebo. The Gazebo is an open-source 3D robotics simulator, for indoor and outdoor environments, with multi-robot support that allows a complete implementation of dynamic and kinematic physics and a pluggable physics engine. Furthermore, it provides a realistic rendering of backgrounds, including high-quality lighting, shadows, and textures. In addition, it can model sensors that "see" the simulated environment, such as laser range finders, cameras (including wide-angle), Kinect style sensors, among others.

The Gazebo present the same message interface as the rest of the ROS ecosystem. So, the development of ROS nodes is compatible with simulation, logged data, and hardware. Many projects integrate ROS with Gazebo, such as the QuadRotor presented in [34], the Humanoid implementation in [35], and the Ground Vehicle in [29]. As a powerful and very visual tool, Gazebo has also been used as the simulation environment for several technology challenges and competitions, such as NASA Space Robotics Challenge (SRC) [36], Agile Robotics for Industrial Automation Competition (ARIAC) [37], and Toyota Prius Challenge [38].

Gazebo is responsible for realistically mimicking the system's fundamental dynamics, representing physical issues such as mass, inertia moment, friction, and even collisions. To ensure better representation, Gazebo supports four engines: Simbody [39], Bullet Physics [40], ODE [41], and DART [42]. Such engines guarantee a wide range of representations, bringing simulations closer to reality, offering the student a greater possibility of representing theoretical concepts practically.

Although some Gazebo components show some lag with new technologies, its overview still has more advantages than the alternatives presented. For example, Unity [43] has similarities in the implemented physics, but its integration with ROS is still complex. Furthermore, the Webbots recently developed a ROS integration but still do not have the same flexibility in implementing different physical models. Finally, the Coppelia [44] does not have the same rendering quality [45] as Gazebo, although it has similar flexibility and quality in physical representation.

2.3 Scenario and 3D Vehicle Model

The Gazebo allows the construction of several different scenarios, including as many objects as desired. Those objects can be static or dynamic and controlled in the simulation. For illustration, this work introduces the track presented in Fig. 3 with and without obstacles. Those obstacles can be removed or added by the user.

ROS applications have a launch file that allows the easy start of several applications with previously saved scenarios and desired configurations. In this project, the file *car_demo.launch* is responsible for starting the track, and the *cars.launch* defines the vehicle's initial coordinates and model. The simulator flexibility allows different car models, including or removing sensors, modifying their positions and configurations. The sensors data can be real-time observed though the *RViz* software.

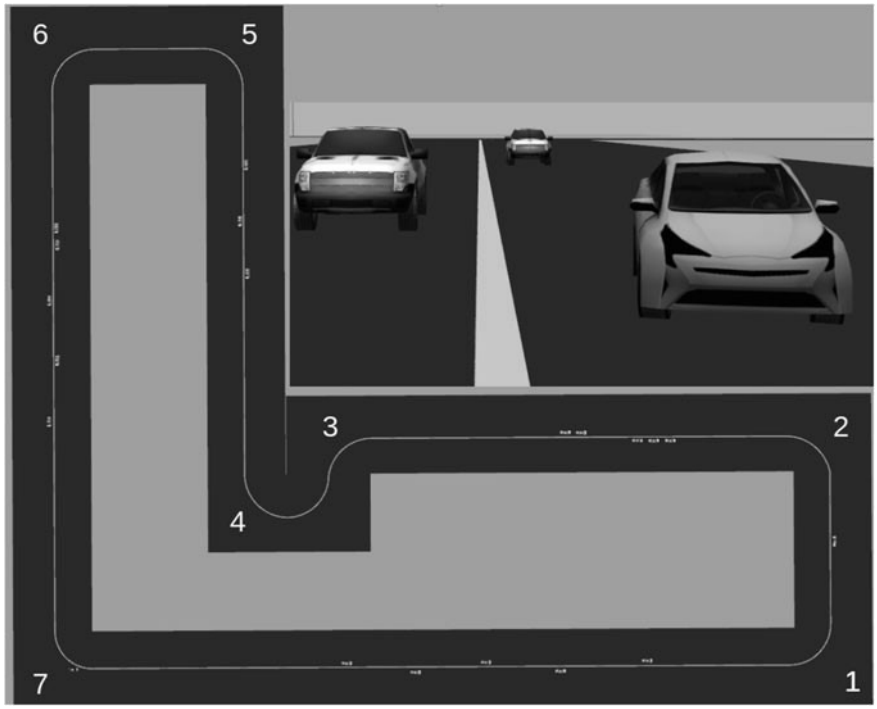
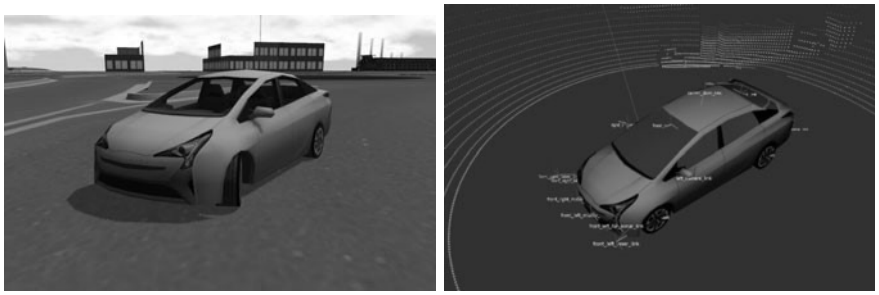


Fig. 3 Track model



Simulated 3D Prius front view Prius Sensors Position

Fig. 4 Prius Gazebo model [25]

The 3D car model used in this work was presented in [25]. Figure 4 illustrates the Hybrid Prius 3D model's main details. Its fundamental dynamics are contained in the node *PriusHybridPlugin.cpp*, and the model's characteristics can be edited in *prius.urdf*.

The primary vehicle controllers, such as throttle, brake, steering, and gear, can be actuated by publishing to a ROS topic. Thus, the vehicle Powertrain will control

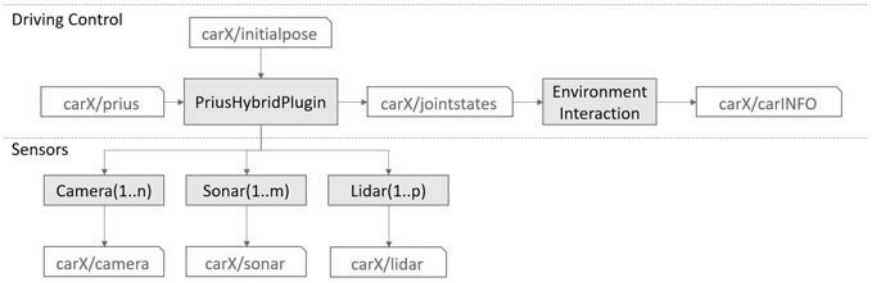


Fig. 5 Prius information structure

the gear control in this simulation. The simulated vehicle also has multiple sensors: 16-beam LIDAR on the roof, eight ultrasonic sensors, four cameras, and two planar LIDARs. However, adding or removing sensors is a simple task that allows adjustments, as necessary. Furthermore, implementing the vehicle with all the kinematics and basic controls enables the study of other project aspects, such as motion control, platooning, stability, and detection and avoidance models.

The vehicle information flowchart is illustrated in Fig. 5. All the vehicles have the same model, and the simulation is composed of $n \in \mathbb{N}$ vehicles. The full set of cars can be defined as $car_n = \{i \in \mathbb{N} | 0 \leq i \leq n\}$. The information provided by each module/node is:

- $car_i/initialPose$: defines the vehicle's initial position
- $car_i/prius$: new vehicle settings—throttle, break, steering
- $car_i/jointstates$: conditions of each vehicle component—wheels and steering
- $car_i/carINFO$: vehicle's current state—throttle, brake, speed, latitude, longitude, steering, heading, etc.
- $car_i/camera$: vehicle's onboard cameras info
- $car_i/sonar$: vehicle's onboard sonars info
- $car_i/lidar$: vehicle's onboard LIDARs info
- PriusHybridPlugin: dynamics and vehicle model
- Environment Interaction: Gazebo calculation about interactions
- Camera $(1 \ldots n)$, Sonar $(1 \ldots m)$ and Lidar $(1 \ldots p)$: sensor nodes

All the sensors can be added, removed, or modified in the file *prius.urdf*. The vehicle control is managed through the data sent to $car_i/prius$ topic, which works as the vehicle's input center, receiving throttle, brake, and steering. The throttle and brake have a limit from 0 to 1, and the steering has a range from $-30°$ to $+30°$. Its format is defined in the "Control.msg". To better understand the text, the rest of this text will refer to a generic simulated vehicle identified by the "i" index, unless in cases where some differentiation is necessary.

2.4 System's Outputs

As a simulator for learning purposes, the system's outputs are essential. Moreover, as a STEM application, with many details, several analysis must be performed using a mathematical approach. The output data will allow the study and comparison of each simulation, allowing the student to evaluate the impact of slight differences in the system's response in each experiment. The system's outputs are provided in *.csv* files generated during the simulation. The module *listener.py* is responsible for collecting the desired vehicle's data in the related topics and exporting that to a *.csv* file.

During the simulation, the topic $car_i/carINFO$ can be used to perform a Real-Time system evaluation, showing the vehicle's most important information, like coordinates, heading, speed, throttle, and brake conditions. The *listener.py* collects this data and adds some information to the simulation's output file, triggered by the car's movement or spent time. The output file contains the timestamp, coordinates, speed, speed error, throttle and brake percentage, heading, heading error, and sonar information, in this version.

3 Control Algorithms

This section will introduce the controller models used in this simulator. Then, it will discuss the Cruise Controller (CC), the Line Follower characteristics, and the Obstacle Avoidance Strategy. The Prius model simulates sensors that publish to the $car_i/carINFO$ topic. This topic contains the main data about the vehicle, like latitude, longitude, altitude, heading, speed, direction, steering angle, acceleration pedal percentage, and brake pedal percentage. All the data is updated every 0.01 s.

3.1 Vehicle Model

The vehicle model used in this work is based on the two-degree-of-freedom bicycle system, as shown in Fig. 6. This model considers the car's rotation around the z-axis (θ) and its lateral velocity. Assuming x and y as the vehicle's frame coordinates, respectively, and θ its rotation in the z-axis, X, Y, and Θ are their absolute equivalents in the global frame. Thus, the vehicle frame can be expressed using the rotation angle θ in the global frame ($\Theta = \theta$). Finally, The steering angle, expressed in the vehicle's frame, is defined as δ and admits that both wheels turn the same value. By applying Euler–Newton equations [46], it is possible to simplify the vehicle's dynamics in the plane as:

$$m\ddot{x} = m\dot{y}\dot{\theta} + F_{x_F} + F_{x_R}, \tag{1a}$$

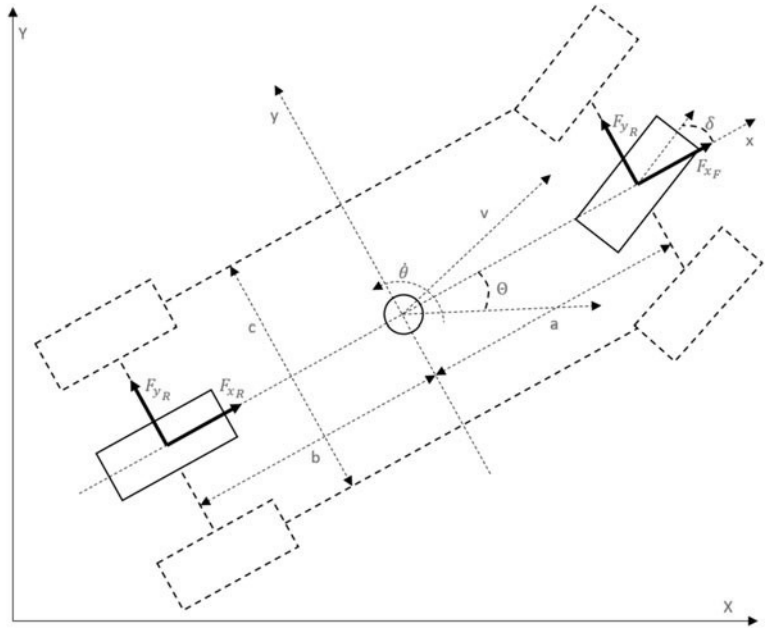


Fig. 6 Vehicle 2D model and coordinates

$$m\ddot{y} = -m\dot{x}\dot{\theta} + F_{y_F} + F_{y_R}, \tag{1b}$$

$$I\ddot{\theta} = aF_{y_F} - bF_{y_R} + c(-F_{x_{F,l}} + F_{x_{F,r}} - F_{x_{R,l}} + F_{x_{R,r}}), \tag{1c}$$

where I is the inertia moment, m is the vehicle mass, and F_x and F_y are the forces in x and y directions, and the subscriptions r and l indicates the force direction compound. Finally, the kinematic model, translated to X and Y coordinates, can be described as:

$$\dot{X} = \dot{x}\cos\Theta - \dot{y}\sin\Theta, \tag{2a}$$

$$\dot{Y} = \dot{x}\sin\Theta - \dot{y}\cos\Theta, \tag{2b}$$

$$\dot{\Theta} = \dot{\theta}. \tag{2c}$$

3.2 Cruise Controller (CC)

Like an actual vehicle, the Prius model does not allow direct speed control but only throttle and brake adjustments. So, this simulator adopts a Proportional Integral

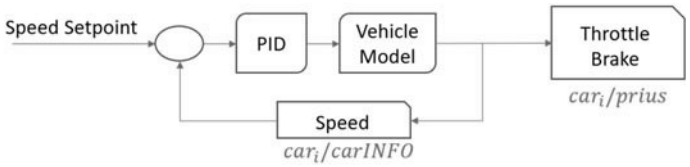


Fig. 7 PID CC

Derivative (PID) strategy to the vehicle speed controller. Although this controller is quite simple, it will help the student to develop basic control skills and move towards other implementations, including several autopilot strategies and tuning models [47]. In this way, the Cruise Controller (CC) will be responsible for keeping the vehicle constant speed during the vehicle's movement and adjusting it when necessary, changing the brake and the throttle pedals, through the $car_i/prius$ topic. The PID equation is defined as follows:

$$\alpha(t) = K_P * \varepsilon^{\sigma}(t) + K_I * \int \varepsilon^{\sigma}(t)dt + K_D * \frac{\Delta\varepsilon^{\sigma}(t)}{dt}, \tag{3}$$

where K_P, K_I, and K_D denote the Proportional, Integral, and Derivative gain constants respectively, $\varepsilon^{\sigma}(t)$ is the speed error, measured by the difference between the current speed value and the desired one and α is the desired system acceleration. The α is then normalized to a value between $-1 \ldots 1$, representing the Throttle and Brake pedals usage. A positive value indicates that the Throttle pedal has been used while the Brake is free. Conversely, the Brake is pressed for a α negative value, and the Throttle pedal is free. The complete controller is illustrated in Fig. 7, where it is assumed that the time constant of the actuator is much bigger than the motor one, and the CC algorithm is summarized in Algorithm 1.

Algorithm 1 Cruise Controller Algorithm

Input: Speed Set Point, Current Speed
Output: Throttle and Brake percentage

$\varepsilon^{\sigma} \leftarrow speed\_set\_point - current\_speed$
$\alpha \leftarrow PID(\varepsilon^{\sigma})$
$\alpha_{control} \leftarrow Normalized[-1 \ldots 1](\alpha)$
if $\alpha_{control} \geq 0$ **then**
    $throttle \leftarrow \alpha_{control}$
    $brake \leftarrow 0$
else
    $throttle \leftarrow 0$
    $brake \leftarrow \alpha_{control}$
end if

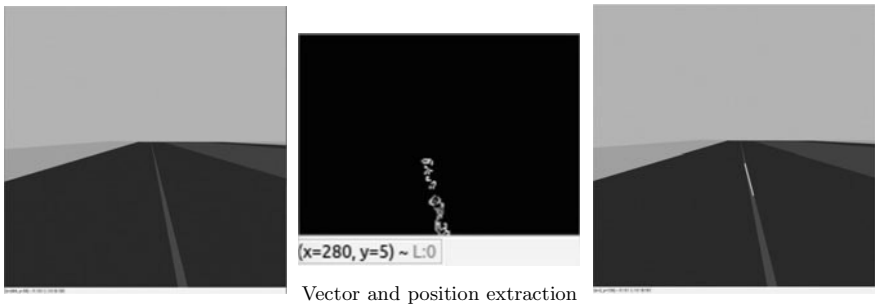
3.3 Line Follower

In this work, the vehicle will simulate a standard trajectory path following method, using a road line [48]. The simulated car has several cameras, and one of them is used to identify the road line and follow it with real-time detection. The Line Follower (LF) algorithm processes the captured image and delivers information regarding the line position to the controller. The vehicle's controller will keep its center over the line with a second PID controller. The implemented algorithm is similar to the one proposed in [49]. Nevertheless, as Gazebo provides a realistic camera view, it is possible to implement algorithms without a real one, changing the image coordinates, frame rate, data size, among other image capture characteristics, and evaluate the changes' impact over the controller.

An OpenCV node was implemented to read the data from the onboard front camera. This node subscribes to the topic $car_i/front\_camera$ and virtually receives all the images from the camera in an 800×800 pixels frame. Then, the LF algorithm filters the image to find a vertical line in the track, and the detection is performed using the Progressive Probabilistic Hough Transform (HT) [50]. This method is commonly used in image processing and can help detect any shape if it can be represented in mathematical form.

The Line Detection (LD) algorithm is illustrated in three frames of Fig. 8. The first one, in Fig. 8a shows the vehicle camera simulated view. The LD algorithm applies a mask over this image to filter it, highlighting a particular color. This color can be adjusted following the Red Green Blue (RGB) model. The filtered image is then converted to a greyscale picture, as presented in Fig. 8b, allowing the edges detection using the Canny Edge detection [51], using vectors with Cartesian coordinates. Finally, these edges are integrated with the HT, defining a *most probably* line to be followed, as demonstrated in Fig. 8c.

The line coordinates are published in $car_i/line\_data$ topic and can be used by the LF controller module. This module is called *controller* and is responsible for the vehicle's motion controller. The vehicle heading error (ε^{θ}) related to the line reference



Vehicle front camera view | Vector and position extraction | Line Detection - Final View

Fig. 8 Line detection process

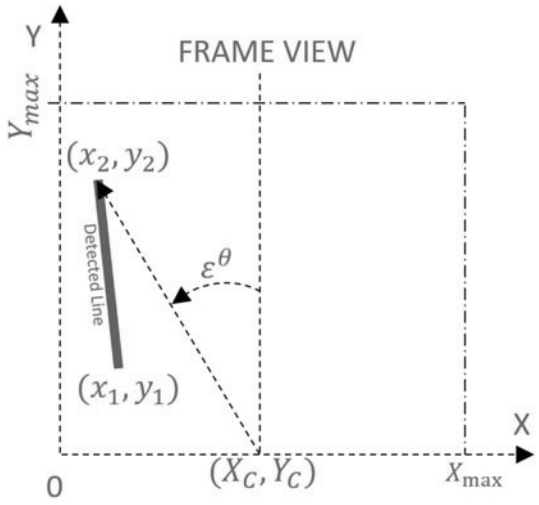


Fig. 9 Vehicle heading error (ε^{θ})

is defined as a relative measurement, using the center of the image frame as illustrated in Fig. 9. In this figure, the *Detected Line* is the output of the LD algorithm, with (x_1, y_1) and (x_2, y_2) respectively the initial and the final coordinates. The X_{max} and Y_{max} represent the frame limits and X_C is the frame center point in X axis. The ε^{θ} is defined as the angular difference between the X_C and the (x_2, y_2) coordinates, given by Eq. 4. Finally, the *controller* calculates the car's *Steering Wheel Angle*, using the PID control action presented in Eq. 5.

$$\varepsilon^{\theta}(t) = \arcsin \frac{X_C - x_2}{y_2 - Y_C} \tag{4}$$

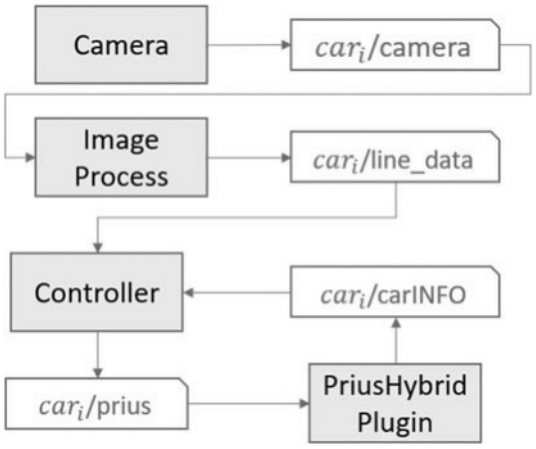


Fig. 10 Line detection and driving controller

$$\theta_{wheels}(t) = K_P^{\theta} * \varepsilon^{\theta}(t) + K_I^{\theta} * \int \varepsilon^{\theta} dt + K_D^{\theta} * \frac{\Delta\varepsilon^{\theta}(t)}{dt}, \tag{5}$$

where K_P^{θ}, K_I^{θ} and K_D^{θ} denote the Proportional, Integrator, and Derivative gain constants, and θ_{wheels} is the *Steering Wheel Angle* to be applied to the vehicle. Figure 10 shows the general LF flowchart, including the controller action, while the complete LF algorithm can be observed in Algorithm 2.

Algorithm 2 Line Follower Algorithm

Input: Image Frame
Output: Steering Angle
1: Mask image to find Vertical Lines
2: Filter image to obtain Data Vectors
3: $Line\_Vectors \leftarrow Hough\_Line\_Transform$
4: $Line\_Coordinates \leftarrow MERGE(Line\_Vectors)$
5: $\varepsilon^{\theta} \leftarrow Eq.4$
6: $\theta_{wheels} \leftarrow PID(\varepsilon^{\theta})$

3.4 Obstacle Detection and Avoidance

Obstacle detection and avoidance is one of the most common autonomous vehicular application, given the demanded safety conditions. So, in this simulator demonstration, sonars are used to detect and avoid unpredicted obstacles and help the vehicles to keep the LF algorithm. The car_i will use six sonars: four in the car's front and one on each side of it, as seen in Fig. 11. The simulator allows the user to change the sonar's positions and ranges and add or remove them in *prius.urdf* file. This simulator assumes that the obstacles are positioned near the reference line and have the same lateral size as the vehicles.

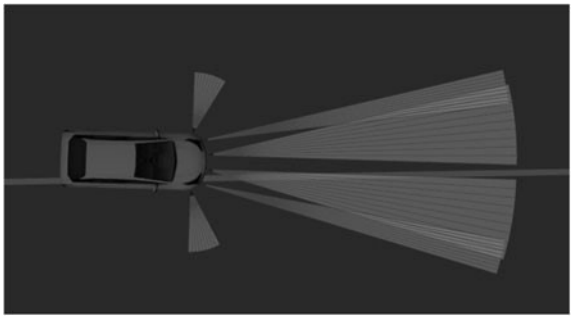


Fig. 11 Sonar visualization

Algorithm 3 Line Follower with Detection and Avoidance Algorithm

Input: Sonars Info, Image Frame
Output: Steering Angle

```
while Object_Detected do
    if Right_Object_Detected then
        θ_wheels ← Left_Deviation
    else if Left_Object_Detected then
        θ_wheels ← Right_Deviation
    end if
end while
Line_Follower_Algorithm (Algorithm 2)
```

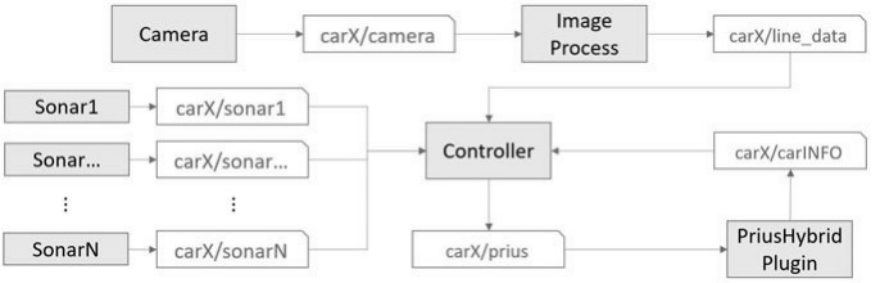


Fig. 12 General vehicle architecture, with line detection and detection and avoidance modules

The sonars are used together with the LF algorithm. However, as the Detection and Avoidance (DA) algorithm has priority over the LF, it assumes the vehicle controller until the obstacle is out of view and the LF is reactivated. So, the algorithm 3 is an extension of the LF algorithm. When the sonars detect an obstacle, the DA controller turns the vehicle in the opposite direction, within a fixed θ_{wheels} value. This heading adjustment is continued until the four front sonars stop detecting the obstacle. Then, the lateral sonars avoid the vehicle trying to return to the line before it overtakes the obstacle. Finally, the LF algorithm uses the last information about the detected line to return to the desired trajectory. The DA block diagram is presented in Fig. 12.

4 Experimental Validation

A control simulation environment should present several controller tools to the student. This section will introduce three main tools developed in RosDrive that allow the student to analyze the vehicle's controller performance and elaborate on different strategies to guarantee its safety. The vehicle's controller performance can be defined in several ways, including fuel consumption, final speed, acceleration, among others. In this chapter, the performance is measured by the vehicle's capacity to track the setpoint, both in speed and heading adjustments.

Table 1 Cruise Controller PID Settings

CC	K_P	K_I	K_D
$PID1$	10.8	0.0	0.0
$PID2$	10.8	2.16	0.270
$PID3$	10.8	4.32	0.135
$PID4$	10.8	2.16	0.135

4.1 Cruise Controller Implementation

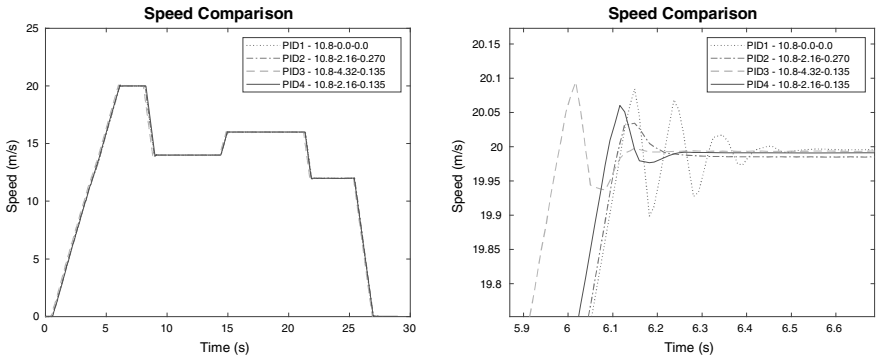
The CC was developed as an independent module. So, it works as a black box implementation, where the inputs are the setpoint and current vehicle's speed, and the outputs are the throttle and brake percentage, while the controller parameters are adjusted inside the module. This architecture choice increases the simulator's flexibility, allowing the user to replace the controller and adjust its parameters.

Taking into account Fig. 3, the straight line between the points 7 and 1, without obstacles, was used to evaluate the CC and check how does the vehicle behaves with several accelerations and decelerations. In this scenario, the vehicle speed setpoint was changed from 20.0, to 14.0, 16.0, 12.0 m/s and finally 0.0 m/s. All the speed settings are defined in the *controller.py* file in the parameters section. They are related to the vehicle's current position on the track.

The controller parameters K_P, K_I, and K_D were defined with the Ziegler Nichols (ZN) empirical method [52]. The vehicle was accelerated from a rest position until it reached the first setpoint speed in the proposed scenario. Increasing K_P until the system oscillation limit, it was possible to determine the ultimate gain (K_u), at 18, with a period of 0.1 ms. These values show how the vehicle's actuator has a rapid response since the oscillation period is fast. In this test, the Ziegler Nichols Tuning parameters are $K_P = 10.8$, $K_I = 2.16$, and $K_D = 0.135$. The system's response is presented in Fig. 13.

As described above, the RosDrive was designed so that the student can change the system's characteristics and observe the impact on the vehicle's response. In addition to changing the controller model, the change of control parameters already implies different responses to be analyzed, providing the user with a practical study of the characteristics of each one of them. Three variations of the parameters obtained with ZN are proposed to exemplify their impacts on the vehicle's control action. These parameters are shown on the Table 1, where $PID1$ is a proportional-only controller, $PID2$ increases the derivative component, $PID3$ enforces the integrator component, and finally $PID4$ shows the parameters obtained by the method ZN.

Figure 13a and b show in detail the impact of controller changes on the system response, and in Fig. 13b it is possible to observe that the control proportional-only ($PID1$) presents a more significant oscillation and that the increase of the derivative component ($PID2$) makes the response slower, but with a smaller overlap. On the other hand, the increase in the integrator component ($PID3$) makes the system faster



Vehicle's Speed Response Vehicle's Speed Detail Response

Fig. 13 Vehicle speed response to different PID parameters

but with a greater overshoot on the setpoint. In this scenario, the parameters obtained by ZN show a better response as they are at an intermediate point of response time and overshoot. In all models presented, the controller error was minimal with a maximum steady-state value lower than 0.006 m/s.

The PID CC example shows the student the basic vehicle controller models. It translates the conceptual controller view to a practical application, reducing the gap between the theoretical aspects and the implementation one, allowing the development of active skills and opening the doors to the student's creativity. A shortly CC RosDrive demonstrator is presented in https://youtu.be/QFVwgFyhaF4. All the video demonstrators links are presented in Sect. 5.

4.2 Line Follower (LF) Controller

The LF controller is responsible for the vehicle's heading adjustment, performed by the Heading Controller (HC). This control ensures that the vehicle safely makes the circuit curves, preventing accidents. As can be seen in Fig. 3, in this scenario, tighter curves were chosen, allowing the student to analyze more complex situations, such as car skidding. Under these conditions, the vehicle's controller is adjusted in one curve and then evaluated its performance on the whole circuit.

Initially, the circuit's curves radius were analyzed to define the maximum speed that would prevent the vehicle from going off at the curve's tangent. The maximum speed (v_{out}) is given by $|v_{out}| = \sqrt{\mu \cdot |g| \cdot R}$, where μ is the friction's coefficient, $|g|$ is the gravity acceleration and R is the curvature ray. In the proposed scenario, it is defined that $\mu = 0.9$, $g = -9.8$ m/s, and $R = 18.38$ m, which means that $v_{out} = 13.34$ m/s.

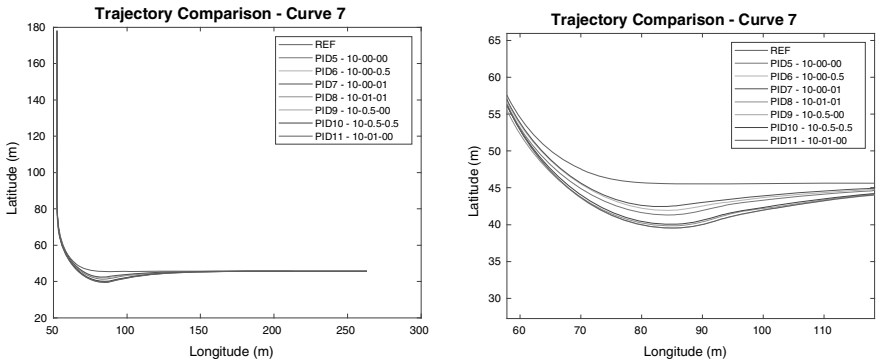
Table 2 HC PID parameters

HC	Speed (m/s)	K_P^θ	K_I^θ	K_D^θ	HC	Speed (m/s)	K_P^θ	K_I^θ	K_D^θ
REF	13	10.0	0.0	0.5	$PID8$	15	10.0	1.0	1.0
$PID5$	15	10.0	0.0	0.5	$PID9$	15	10.0	0.5	0.0
$PID6$	15	10.0	0.0	1.0	$PID10$	15	10.0	0.5	0.5
$PID7$	15	10.0	0.0	1.0	$PID11$	15	10.0	1.0	0.0

The heading controller has a different evaluation in comparison with the CC. In the CC, the setpoints are defined through a step function, while in the HC, the setpoints function follows the curve design, with a long transition phase. It means that the system's response should be evaluated after the desired heading is constant. Due to this condition, a more complex scenario is proposed to evaluate the system's response in adversarial conditions. Initially, the vehicle's trajectory was fixed with v_{out} as the heading reference. Then, the objective was to find the most suitable HC PID parameters for the system with $v = 15$ m/s. It means that the HC will suffer from skidding. In this scenario, the student's experience determining the best HC PID parameters will be necessary since the ideal conditions presented in theory are not present. Furthermore, it will increase the student's perception of the problem and stimulate creative new solutions since the vehicle's speed increase will increase the skidding, compromising the system's stability. It is also important to highlight that the user can set up any speed and check its response.

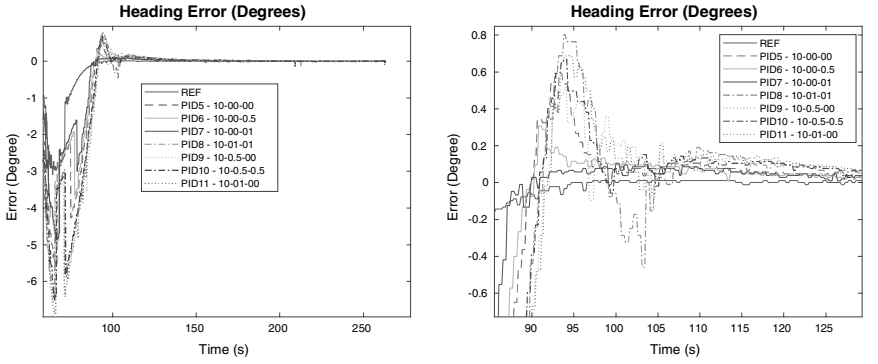
The HC PID parameters were obtained initially in curve 7 given the long straight lines before and after. The obtained parameters are presented in Table 2. Figure 14a presents the vehicle's trajectory on curve 7, while Fig. 14b perform an in-depth view of the same curve. Both figures illustrate how the reference HC has a smoother trajectory, with no skidding. As expected, there is some skidding in all the HC PID configurations, with a speed setpoint of 15 m/s. However, these figures analysis allow the identification of the best controller performance, even on these conditions. So, in the proposed scenario, the $PID7$ presents a better response due to the derivative action, avoiding extreme adjusts keeping the vehicle's trajectory near to the REF trajectory. On the other hand, the integral action presented in $PID8$ and $PID11$ configurations produces more oscillation and increases the distance between the REF and the performed trajectory due to the skidding.

Figure 15 presents the vehicle's heading error (ε^θ), due to the different HC PID parameters. As expected, while the LF reference adjusts the heading setpoint, the vehicle's heading suffers from much oscillation, trying to respond to the new conditions. However, the system's response can be better studied after the transition, when the LF algorithm sets the new line. This situation can be observed in Fig. 14b. This figure highlights the smooth response of $PID7$, with little oscillation above the REF response. Nevertheless, the systems' response with $PID8$ and $PID11$ have a considerable overshoot and take much more time until the stabilization.



Vehicle's Trajectory - Curve 7 Vehicle's Detail Trajectory - Curve 7

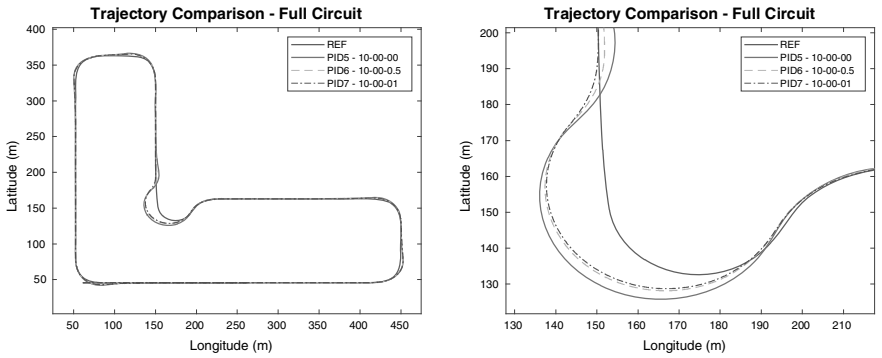
Fig. 14 Vehicle trajectory analysis (curve 7) under different HC PID settings



Vehicle's Heading Error (ε^{θ}) Vehicle's Heading Error (ε^{θ}) Detailed View

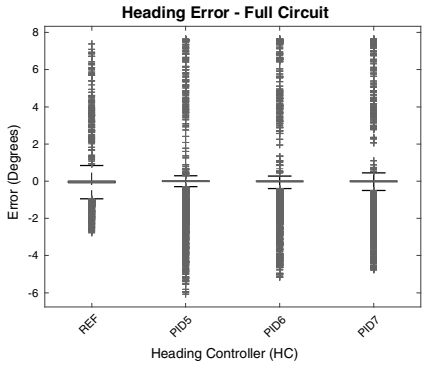
Fig. 15 Vehicle heading error(ε^{θ}) at curve 7 under different HC PID settings

The HC PID response analysis can be extended to the entire circuit. Looking at the heading error (ε^{θ}) presented in Fig. 15, the best controllers response were performed by the PD configurations, namely the $PID6$ and $PID7$. Furthermore, a full lap was performed to evaluate the vehicle's heading controller, comparing its trajectory and the general ε^{θ}. Figure 16a present the vehicle's trajectory comparison in the full lap. It shows the vehicle's skidding in all the curves and the most distinguished one in curve 4. Thus, Fig. 16b highlight the vehicle's trajectory in this curve, showing that although all the HC PID configurations suffer from high skidding on this curve, the $PID7$ configuration provides a smaller skidding and is the faster one to stabilize the system after the curve. Finally, the Fig. 16c shows a comparison between the general ε^{θ} during the full lap. It demonstrates that the REF configuration has the smallest error variation during the circuit and that the $PID7$ error response is the most approximate to it.



Vehicle's Trajectory - Full Lap

Vehicle's Detail Trajectory - Curve 4



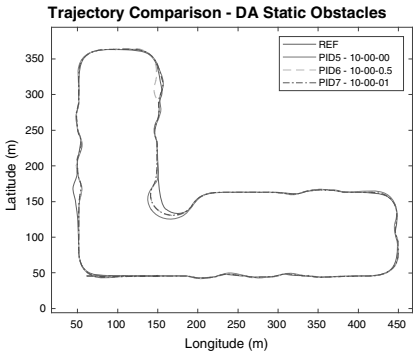
Vehicle's Heading Error (ε^{θ}) at a Full Lap under best HC PID Settings

Fig. 16 Vehicle trajectory analysis (Full Lap) under different HC PID settings

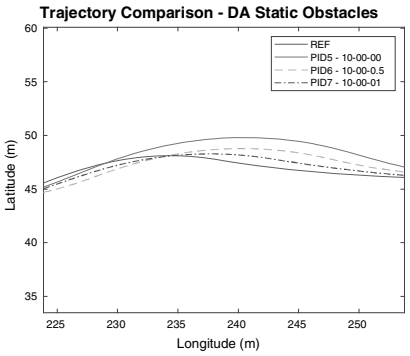
This scenario was built to illustrate the simulator's flexibility, merging the LF algorithm with the HC method under an adversarial context. In this way, the student will be able to extend its capabilities, changing the controller's parameters and checking the system's response, proposing new situations, and evaluating them. Furthermore, it will help students build and reinforce their capabilities and skills without damaging any equipment by extrapolating the commonly encountered theoretical conditions.

4.3 Obstacle Detection and Avoidance

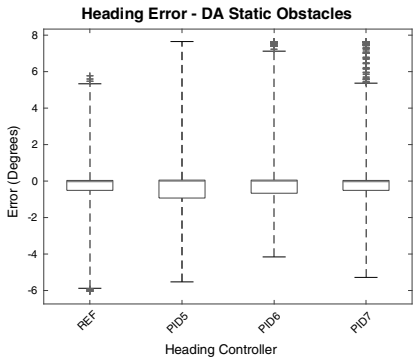
In addition to the analysis of the CC and HC controllers on the vehicle's performance in isolated scenarios, RosDrive allows the analysis of its interaction with other vehicles, whether static or dynamic.



Vehicle's Trajectory Static Obstacle Avoidance - Full Lap



Vehicle's Trajectory Detail Static Obstacle Avoidance



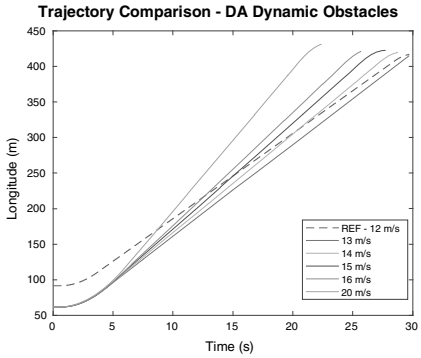
Vehicle's Heading Error (ε^{θ}) with Static Obstacle Avoidance

Fig. 17 Vehicle trajectory analysis (static obstacle avoidance) under different HC PID settings

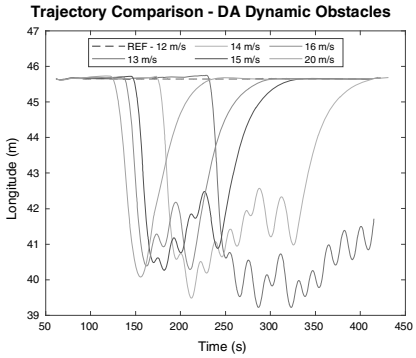
4.3.1 Static Obstacle Detection and Avoidance

The vehicle's ability to perform a trajectory avoiding several close obstacles was initially analyzed, comparing the heading error (ε^{θ}) given the HC parameters changes. The 19 static obstacles are illustrated in Fig. 3 and are modeled as Pickup vehicles. These obstacles are positioned on the straight circuit lines avoiding curves overtaking. Again, the REF vehicle running with a 13.0 m/s speed was presented against the $PID5$, $PID6$, and $PID7$ HC configuration, running at 15.0 m/s. The vehicle's trajectory is presented in Fig. 17a, illustrating that the skidding is still present, mostly in curve 4.

Furthermore, the vehicles do not necessarily follow the same trajectory to avoid obstacles. This situation is illustrated in the straight line between curves 4 and 5, where the vehicle with the HC $PID6$ avoids the last obstacle with a left turn and the others perform a right curve. This obstacle avoidance action responds to the first



Vehicle's Dynamic Obstacle Avoidance Longitude



Vehicle's Dynamic Obstacle Avoidance trajectory

Fig. 18 Vehicle trajectory analysis (dynamic obstacle avoidance)

sonar activated in the vehicle. As the vehicle's trajectories are slightly different due to the HC response, the car's angular position at that point is not the same for all the configurations, providing different sonar activation. The same situation is observed in the last obstacle between curve 7 with all the HC PID configurations compared with the REF.

To compare the HC PID's performance in a static detection and avoidance condition, Fig. 17b emphasize the vehicle's movement over the second track obstacle. It is possible to observe that the HC PID performance follows the model presented in the LF algorithm, with $PID7$ providing the best system's response in comparison with REF. However, the ε^{θ}'s variation between REF, $PID6$, and $PID7$ is similar, given the rapid heading transitions triggered by the ODA and the LF algorithms, as presented in Fig. 17c. On the other hand, the boxplots of $PID7$ ε^{θ} in both Figs. 16c and 17c show that although the maximum variation is similar, these errors appear more frequently, no longer presenting themselves as outliners but as values that are repeatedly perceived. These errors happen due to several obstacles and the constant need to adjust the vehicle's position caused by the HC action.

4.3.2 Dynamic Obstacle Detection and Avoidance

RosDrive's flexibility allows different control models, algorithms, and movement strategies. Thus, it is possible to evaluate strategies for overtaking vehicles in motion, observing how the fastest vehicle behaves and if it can perform the maneuver safely. To conduct this demonstration, the car_1 and car_2, with $v_1(t) > v_2(t)$ were defined, with $x_2(0) = x_1(0) + 30\,\text{m}$. This way, car_1 has time to reach its maximum speed before the sonar detects the presence of car_2 and only then starts the overdrive process.

The *vehicles.launch* allows the setup and launches of the necessary vehicles, with no additional development. In this file, the vehicle models, their initial positions, and the algorithms to be used are instantiated. Initially car_1 (REF) and car_2 are defined with the same CC and HC parameters as $PID4$ and $PID7$. In the same file, the car_2 sonars are deactivated, avoiding its reaction to $car1$ presence. So, $car1$ accelerates, detects the presence of car_2 and performs the overtake action. As the same ODA strategy presented in Sect. 3.4 is applied, after the obstacle detection, $car1$ will return to the line only after overtaking the obstacle, when the lateral sonars indicate that there are no more obstacles there.

In this scenario, v_2 was defined as 13.0, 14.0, 15.0, 16.0, and 20.0 m/s and $v_1 =$ 12 m/s, while the front sonar's ranges are set to 20 m and the lateral ones are 2 m.

When theoretically studying physical systems, it is common to analyze that vehicles are points in space and that overtaking, for example, is just a matter of validating the relationship between space traveled in time, having as reference the speed of the two points. However, in a realistic simulator, vehicles cannot be treated as points in space but as bodies that can collide and must avoid this to remain safe. Thus, the overtaking process begins with detecting the body ahead, followed by a diversion action and consequent movement.

This controller action of car_1 is illustrated in Fig. 18. Figure 18a illustrates a more simplified view of the system, indicating the longitudinal trajectory of car_1 and car_2 under all different velocity conditions of car_1. It is important to note that the movement of car_2 is the same in all scenarios, as its speed is constant, and its movement is not affected by car_1. In this figure, it is possible to observe the crossing point when the curve referring to $car1$ crosses the curve of car_2. Thus, it is seen that the speed of 13.0 m/s, car_1 is not able to exceed car_2 on the desired route, indicating an unsafe maneuver.

However, extending this view to a 2D dimension, as presented in Fig. 18b, it is possible to analyze how does the overtake movement is performed. Under the proposed conditions, only the vehicle with a 20 m/s speed has overtaken car_2 with no oscillatory movement. With other speeds, car_1 lateral sonar detects car_2 presence, and adjust car_1 heading position, avoiding a collision. This figure illustrates how car_1 with a 13.0 m/s speed cannot overtake car_2 in the desired time and finally, how the same movement with 14.0 m/s is dangerous since the overtaken process ends just at the limit of the desired trajectory.

In this way, the student has more information to check how the movement was performed and propose different safety strategies. In addition, this scenario allows the student to create new sonars detection algorithms, different controller strategies, and some intelligent systems to increase the system's safety.

5 Conclusions and Future Works

This chapter presents RosDrive, an open-source simulator for developing the study of vehicle control techniques in a realistic way. It presents the conditions that make this simulator able to bring together the students' theoretical lessons with their practical implementation, reducing gaps in their training. By using ROS as a development platform, RosDrive is compatible with the most diverse Linux systems and has continuous support. In addition, its wide adoption by the community allows for the rapid introduction of new sensors and components and integration with other platforms.

Scenarios were developed to evaluate vehicle speed and heading control models using systems similar to actual vehicles. These systems use acceleration and braking variation and cameras to analyze the environment and take action. Furthermore, algorithms were introduced to avoid collisions between controlled vehicles and static or dynamic obstacles, providing different analyzes and conditions for different control and security systems.

For this, RosDrive features a modular architecture based on a flexible set of easily adjustable and interchangeable control tools to assist in the extrapolations imagined by students and teachers who will use the tool. With this, we firmly believe that active learning can be reinforced in classrooms, introducing low-cost dynamic models that mimic reality, increasing the applicability of knowledge, and consolidating the skills necessary for the formation of the control engineer.

Shortly, we hope to develop communication between vehicles, enabling the exchange of information between them, simulating scenarios referring to the ITS models. We also hope to extend the configuration modules for more intuitive platforms using windows that facilitate the user's vision. Finally, we plan to introduce the use of RosDrive in a classroom context, visualizing the difficulties of students and teachers and analyzing the impacts of its adoption in teaching control systems.

Supplementary Material: Simulator Installation and Execution Instructions

This section presents the main requirements for installing and configuring RosDrive Simulator. Furthermore, it summarizes the prerequisites and indicates the current repository for downloading the necessary files.

5.1 Main Code Repository

– The complete code can be found at: https://github.com/enioprates/rosdrive

5.2 Main Requirements

- Operating System: Ubuntu 18.04
- ROS Distribution: Melodic
- Gazebo: 9 or above
- Python: 2.7 or above
- GCC: 7.3 or above

5.3 Setup Project

1. Install ROS Melodic following the instructions of:
   http://wiki.ros.org/melodic/Installation/Ubuntu.
2. Download files from Github RosDrive Simulator:
   https://github.com/enioprates/rosdrive
3. RosDrive Setup
   - **RosDrive Simulator**
     (a) Open a new terminal inside .../CISTER_car_simulator
     (b) Type: catkin_make
   - **RosDrive Simulator Controllers**
     (a) Open a new terminal inside .../CISTER_image_processing
     (b) Type: catkin_make

5.4 How to Run RosDrive Simulator

1. Run the simulator:
   (a) Open a new terminal inside .../CISTER_car_simulator
   (b) Type: source devel/setup.launch
   (c) Type: roslaunch car_demo demo_t.launch

   The GAZEBO should open on your screen with the vehicle in the position defined in *cars_t_curve_2.launch*. The vehicle will remain stopped until the Vehicle Controller is turned on.
2. PAUSE the simulation and reset the time!
3. Starting the Vehicle Controllers:

   (a) Open a new terminal inside .../CISTER_image_processing
   (b) Type: source devel/setup.launch
   (c) Type: roslaunch image_processing vehicle.launch

   This launcher file will start the vehicle's Line Follower Detection and the Vehicle's movement controller, respectively with the nodes *lane_lines_detection.py* and *simulation_connector.py*.
4. Start the Gazebo simulation

5.5 Optional Configuration

1. Speed PID Configuration
   (a) Open ...../CISTER_image_processing/src/scripts/*simulation_connector.py*
   (b) Find #*Speed PID*
   (c) Adjust $K_P \rightarrow kp\_vel$, $K_I \rightarrow ki\_vel$, and $K_D \rightarrow ki\_vel$
2. Heading PID Configuration
   (a) Open ...../CISTER_image_processing/src/scripts/*simulation_connector.py*
   (b) Find #*Steering PID*
   (c) Adjust $K_P^\theta \rightarrow kp\_steer\_l$, $K_I^\theta \rightarrow ki\_steer\_l$, and $K_D^\theta \rightarrow kd\_steer\_l$

5.6 RosDrive Video Demonstrations

1. Cruise Controller demonstrator:
   https://youtu.be/QFVwgFyhaF4.
2. Line Follower demonstrator:
   https://youtu.be/sRlXk2K1IJc
3. Static Obstacle Detection and Avoidance demonstrator:
   https://youtu.be/YEoO2CQUiKc.
4. Dynamic Obstacle Detection and Avoidance demonstrator:
   https://youtu.be/5135G3aafLw

Acknowledgements This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDP/UIDB/04234/2020); also by the FCT and the EU ECSEL JU under the H2020 Framework Programme, within project(s) ECSEL/0010/2019, JU grant nr. 876019 (ADACORSA). Disclaimer: This document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

References

1. Simonsen, B., Fairbanks, S., Briesch, A., Myers, D., Sugai, G.: Evidence-based practices in classroom management: considerations for research to practice. Educ. Treat. Child. **31**(1), 351–380 (2008)
2. Maciejewski, A.A., Chen, T.W., Byrne, Z.S., De Miranda, M.A., Mcmeeking, L.B.S., Notaros, B.M., Pezeshki, A., Roy, S., Leland, A.M., Reese, M.D., Rosales, A.H., Siller, T.J., Toftness, R.F., Notaros, O.: A holistic approach to transforming undergraduate electrical engineering education. IEEE Access **5**, 8148–8161 (2017)
3. Jamison, A., Kolmos, A., Holgaard, J.E.: Hybrid learning: an integrative approach to engineering education. J. Eng. Educ. **103**, 253–273 (2014)
4. McKenna, A.F.: Educating engineers: designing for the future of the field. J. Higher Educ. **81**, 717–719 (2010)
5. Shooter, S., Mcneill, M.: Interdisciplinary collaborative learning in mechatronics at Bucknell University. J. Eng. Educ. **91**, 339–344 (2002)
6. Xie, Y., Fang, M., Shauman, K.: STEM education. Ann. Rev. Sociol. **41**, 331–357 (2015)
7. WEG: Learning workbenches for training. Teaching Equipment. Teaching Equipment. Electric Panels. WEG - Products, May 2020. Library Catalog: https://www.weg.net/
8. Nulle, L.: Lucas Nülle - Lucas-Nuelle Training Systems for vocational training and didactic (2020)
9. Feedback PLC: Welcome to Feedback plc (2020). Library Catalog: https://fbkmed.com/
10. Galadima, A.A.: Arduino as a learning tool. In: 2014 11th International Conference on Electronics, Computer and Computation (ICECCO), pp. 1–4 (2014)
11. Chancharoen, R., Sripakagorn, A., Maneeratana, K.: An Arduino kit for learning mechatronics and its scalability in semester projects. In: 2014 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), pp. 505–510 (2014)
12. Omar, H.M.: Enhancing automatic control learning through Arduino-based projects. Eur. J. Eng. Educ. **43**, 652–663 (2018)
13. Gonçalves, J., Pinto, V.H., Costa, P.: A line follower educational mobile robot performance robustness increase using a competition as benchmark. In: 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT), pp. 934–939 (2019). ISSN: 2576-3555
14. Serrano Pérez, E., Juárez López, F.: An ultra-low cost line follower robot as educational tool for teaching programming and circuit's foundations. Comput. Appl. Eng. Educ. **27**, 288–302 (2019)
15. Cruz-Martín, A., Fernández-Madrigal, J., Galindo, C., González-Jiménez, J., Stockmans-Daou, C., Blanco-Claraco, J.: A LEGO mindstorms NXT approach for teaching at data acquisition, control systems engineering and real-time systems undergraduate courses. Comput. & Educ. **59**, 974–988 (2012)
16. Leonard, M., Morgan, J., Coffelt, J.P.: Digital systems teaching and research (dstr) robot: a flexible platform for education and applied research. In: Proceedings of the 2018 ASEE Gulf-Southwest Section Annual Conference. The University of Texas at Austin, p. 5, ASEE (2018)
17. Staranowicz, A., Mariottini, G.L.: A survey and comparison of commercial and open-source robotic simulator software. In: Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments - PETRA '11, Heraklion, Crete, Greece, p. 1. ACM Press (2011)
18. Michel, O.: Webots: professional mobile robot simulation. Int. J. Adv. Rob. Syst. **1**, 5 (2004)
19. Open Source Robotic Foundation: ROS/Introduction (2018)
20. Suwasono, S., Prihanto, D., Dwi Wahyono, I., Nafalski, A.: Virtual laboratory for line follower robot competition. Int. J. Electr. Comput. Eng. (IJECE) **7**, 2253 (2017)
21. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: An Open Urban Driving Simulator (2017). arXiv:1711.03938 [cs]
22. Costa, V., Rossetti, R., Sousa, A.: Simulator for teaching robotics, ROS and autonomous driving in a competitive mindset. Int. J. Technol. Human Interact. **13**, 19–32 (2017)

23. Grover, R., Krishnan, S., Shoup, T., Khanbaghi, M.: A competition-based approach for undergraduate mechatronics education using the arduino platform. In: Fourth Interdisciplinary Engineering Design Education Conference, pp. 78–83 (2014). ISSN: 2161-5330
24. Regueras, L.M., Verdu, E., Munoz, M.F., Perez, M.A., de Castro, J.P., Verdu, M.J.: Effects of competitive E-learning tools on higher education students: a case study. IEEE Trans. Educ. **52**, 279–285 (2009). Conference Name: IEEE Transactions on Education
25. Tully Foote: Demo of Prius in ROS/GAZEBO (2017). https://github.com/osrf/car_demo
26. Vieira, B., Severino, R., Filho, E.V., Koubaa, A., Tovar, E.: COPADRIVe - a realistic simulation framework for cooperative autonomous driving applications. In: IEEE International Conference on Connected Vehicles and Expo - ICCVE 2019, Graz, Austria, pp. 1–6. IEEE (2019)
27. Filho, E.V., Severino, R., Rodrigues, J., Gonçalves, B., Koubaa, A., Tovar, E.: CopaDrive: an integrated ROS cooperative driving test and validation framework. In: Robot Operating System (ROS), vol. 962, pp. 121–174. Springer International Publishing (2021). Series Title: Studies in Computational Intelligence
28. Filho, E.V., Guedes, N., Vieira, B., Mestre, M., Severino, R., Gonçalves, B., Koubaa, A., Tovar, E.: Towards a cooperative robotic platooning testbed. In: IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Ponta Delgada, Portugal, pp. 332–337. IEEE (2020)
29. Rivera, Z.B., De Simone, M.C., Guida, D.: Unmanned ground vehicle modelling in gazebo/ROS-based environments. Machines **7**, 42 (2019)
30. Furrer, F., Burri, M., Achtelik, M., Siegwart, R.: RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. Springer International Publishing, Cham (2016)
31. Vaz, F.C., Portugal, D., Araújo, A., Couceiro, M.S., Rocha, R.P.: A localization approach for autonomous underwater vehicles: a ros-gazebo framework (2018)
32. Kim, M.S., Delgado, R., Choi, B.-W.: Comparative study of ros on embedded system for a mobile robot. J. Autom. Mob. Robotics Intell. Syst. **12**, 61–67 (2018)
33. Kuosa, K., Distante, D., Tervakari, A., Cerulo, L., Fernández, A., Koro, J., Kailanto, M.: Interactive visualization tools to improve learning and teaching in online learning environments. Int. J. Distance Educ. Technol. **14**, 1–21 (2016)
34. Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., von Stryk, O.: Comprehensive simulation of quadrotor UAVs using ROS and gazebo. In: Simulation, Modeling, and Programming for Autonomous Robots, vol. 7628, pp. 400–411. Springer, Berlin, Heidelberg (2012)
35. Feng, H., Wong, C., Liu, C., Xiao, S.: ROS-based humanoid robot pose control system design. In: 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, pp. 4089–4093 (2018)
36. Hambuchen, K.A., Roman, M.C., Sivak, A., Herblet, A., Koenig, N., Newmyer, D., Ambrose, R.: NASA's space robotics challenge: advancing robotics for future exploration missions. In: AIAA SPACE and Astronautics Forum and Exposition, Orlando, FL. American Institute of Aeronautics and Astronautics (2017)
37. NIST: Agile Robotics for Industrial Automation Competition (2020). https://www.nist.gov/el/intelligent-systems-division-73500/agile-robotics-industrial-automation-competition. Last Modified: 2020-07-09T09:19-04:00
38. Open Source Robotics Foundation: Prius Challenge (2017)
39. Sherman, M., Eastman, P.: Simtk - simbody: Multibody physics api
40. PyBullet: Bullet real-time physics simulation
41. Smith, R.: Ode - open dynamics engine
42. Lab, G., Lab, H.R.: Dynamic animation and robotics toolkit - dart
43. Hussein, A., García, F., Olaverri-Monreal, C.: Ros and unity based framework for intelligent vehicles control and simulation. In: 2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES), pp. 1–6 (2018)
44. Robotics, C.: Coppelia robotics
45. Rizzardo, C., Katyara, S., Fernandes, M., Chen, F.: The importance and the limitations of sim2real for robotic manipulation in precision agriculture (2020)

46. Attia, R., Orjuela, R., Basset, M.: Combined longitudinal and lateral control for automated vehicle guidance. Veh. Syst. Dyn. **52**, 261–279 (2014)
47. Vartika, V., Singh, S., Das, S., Mishra, S.K., Sahu, S.S.: A review on intelligent pid controllers in autonomous vehicle. In: Reddy, M.J.B., Mohanta, D.K., Kumar, D., Ghosh, D. (eds.) Advances in Smart Grid Automation and Industry 4.0 (Singapore), pp. 391–399. Springer Singapore (2021)
48. Tesla: Transitioning to Tesla Vision (2021)
49. Kondakor, A., Torcsvari, Z., Nagy, A., Vajk, I.: A line tracking algorithm based on image processing. In: 2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, pp. 000039–000044. IEEE (2018)
50. OpenCV: Hough Line Transform (2019)
51. OpenCV: Canny Edge Detection (2022)
52. Kobatake, K., Okazaki, T., Arima, M.: Study on optimal tuning of pid autopilot for autonomous surface vehicle. IFAC-PapersOnLine **52**(21), 335–340 (2019). 12th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2019

Enio Vasconcelos Filho is currently Assistant Professor at the Federal Institute of Goias (IFG). Since 2018, he has been carrying out his Ph.D. in Electrical and Computer Engineering at the University of Porto (FEUP), Portugal. He joined the CISTER Research Unit in 2018, working with Cyber-Physical Cooperative Systems, notably in vehicular communications. He has a BSc degree in Automation and Control (2006) and an M.Sc. degree in Mechatronics Systems (2012), both from Universidade de Brasilia (UnB). His most significant interests are autonomous vehicles, communication networks, and artificial intelligence in real-time applications.

Jones Yudi is a Professor at the University of Brasília since 2010, in the area of Industrial Automation. Doctor in Electrical Engineering & Information Technology from Ruhr-Universität Bochum (2018), Master in Mechatronic Systems from the University of Brasília (2010), with a degree in Control and Automation Engineering (Mechatronics) from the same University (2007). Specialist in the design of hardware/software architectures for high-performance embedded systems, focusing on image and signal processing, instrumentation and control. Experience in distributed systems, sensor networks, ubiquitous computing, Industrial Internet of Things and Industry 4.0.

Mohamed Abdelkader is currently an Assistant Professor at Prince Sultan University, Riyadh, KSA, where he leads multiple R&D projects related to autonomous delivery drone system, and autonomous drone hunting system, in the Robotics and Internet of Things lab. Abdelkader obtained his Ph.D. in Mechanical Engineering from King Abdullah University of Science Technology (KAUST) working with real-time distributed autonomous multi-robot systems for pursuit-evasion applications. His main research works are in the development of AI-powered swarm robotic systems, and safe autonomous navigation. He is a technical team lead at Systemtrio, Abu Dhabi, UAE, developing autonomous robotics solutions including a UAV swarm system for security and surveillance, affiliated with the IEEE Robotics and Automation Society, and a reviewer in several robotics conferences and journals including ICUAS, IROS, and JINT.

Anis Koubaa is a Professor in Computer Science, Advisor to the Rector of Research Governance, Director of the Research and Initiatives Center, and Director of the Robotics and Internet of Things Research Lab, in Prince Sultan University. He is also a Senior Researcher in CISTER/INESC TEC and ISEP-IPP, Porto, Portugal. He is also a Senior Fellow of the Higher Education Academy (HEA) in the UK. He received several distinctions and awards including the Rector Research Award in 2010 at Al-Imam Mohamed bin Saud University and the Rector Teaching Award in 2016 at Prince Sultan University. He is on the list of top 2.

Eduardo Tovar received the Licentiate, M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Porto, Porto, Portugal, in 1990, 1995 and 1999, respectively. Currently he his Professor in the Computer Engineering Department at the School of Engineering (ISEP) of Polytechnic Institute of Porto (P. Porto), where he is also engaged in research on real-time distributed systems, wireless sensor networks, multiprocessor systems, cyber-physical systems and industrial communication systems. He heads the CISTER Labs, an internationally renowned research centre focusing on RTD in real-time and embedded computing systems. He is currently the Vice-chair of ACM SIGBED and is member of the Executive Committee of the IEEE Technical Committee on Real-Time Systems (TC-RTS). He is currently deeply involved in the core team setting-up a Collaborative (industry-academic) Lab on Cyber-Physical Systems and Cyber-Security Systems.