symmetry

MDPI

Article

Kinematic Fuzzy Logic-Based Controller for Trajectory Tracking of Wheeled Mobile Robots in Virtual Environments

José G. Pérez-Juárez [1], José R. García-Martínez [2,*], Alejandro Medina Santiago [3], Edson E. Cruz-Miguel [2], Luis F. Olmedo-García [1], Omar A. Barra-Vázquez [1] and Miguel A. Rojas-Hernández [1]

[1] Laboratorio de Control y Robótica, Facultad de Ingeniería en Electrónica y Comunicaciones, Universidad Veracruzana, Poza Rica 93390, Mexico; joseperez07@uv.mx (J.G.P.-J.); lolmedo@uv.mx (L.F.O.-G.); omabarra@uv.mx (O.A.B.-V.); mrojas@uv.mx (M.A.R.-H.)
[2] Análisis de Sistemas y Tecnologías Emergentes, Facultad de Ingeniería en Electrónica y Comunicaciones, Universidad Veracruzana, Poza Rica 93390, Mexico; edsoncruz@uv.mx
[3] SECIHTI-National Institute for Astrophysics, Optics and Electronics, Computer Science Coordination, Puebla 72840, Mexico; amedina@inaoep.mx
* Correspondence: romangarcia@uv.mx

Abstract: Mobile robots represent one of the most relevant areas of study within robotics due to their potential for designing and developing new nonlinear control structures that can be implemented in simulations and applications in specific environments. In this work, a fuzzy steering controller with a symmetric distribution of fuzzy numbers is proposed and designed for implementation in the kinematic model of a non-holonomic mobile robot. The symmetry in the distribution of triangular fuzzy numbers contributes to a balanced response to disturbances and minimizes systematic errors in direction estimation. Additionally, it improves the system's adaptability to various reference paths, ensuring accurate tracking and optimized performance in robot navigation. Furthermore, this fuzzy logic-based controller emulates the behavior of a classic PID controller by offering a robust and flexible alternative to traditional methods. A virtual environment was also developed using the UNITY platform to evaluate the performance of the fuzzy controller. The results were evaluated by considering the average tracking error, maximum error, steady-state error, settling time, and total distance traveled, emphasizing the trajectory error. The circular trajectory showed high accuracy with an average error of 0.0089 m, while the cross trajectory presented 0.01814 m, reflecting slight deviations in the turns. The point-to-point trajectory registered a more significant error of 0.9531 m due to abrupt transitions, although with effective corrections in a steady state. The simulation results validate the robustness of the proposed fuzzy controller, providing quantitative insights into its precision and efficiency in a virtual environment, and demonstrating the effectiveness of the proposal.

Keywords: fuzzy controller; mobile robot; UNITY; kinematics control

Academic Editor: Hsien-Chung Wu

Received: 29 January 2025
Revised: 10 February 2025
Accepted: 14 February 2025
Published: 17 February 2025

Citation: Pérez-Juárez, J.G.; García-Martínez, J.R.; Medina Santiago, A.; Cruz-Miguel, E.E.; Olmedo-García, L.F.; Barra-Vázquez, O.A.; Rojas-Hernández, M.A. Kinematic Fuzzy Logic-Based Controller for Trajectory Tracking of Wheeled Mobile Robots in Virtual Environments. *Symmetry* **2025**, *17*, 301. https://doi.org/10.3390/sym17020301

1. Introduction

Mobile robots have gained remarkable popularity in recent years, extending their presence to several areas, such as industrial, professional, and educational settings [1]. In the latter, students actively participate in developing robots for school projects, contributing to the continuous growth of mobile robotics. This growth has generated the need to locate mobile robots at any point in space, which requires obtaining a kinematic model that allows one to determine in real time the robot coordinates and its orientation for a fixed reference frame [2].

Within the field of mobile robotics, there is a wide variety of models with different configurations, depending on their structural type or physical characteristics, resulting in various types of movement and, therefore, different kinematics in each case. Ackerman-type mobile robots are characterized because their movement is comparable to that of a car [3]. For this reason, its kinematic model allows for applications such as autonomous ground vehicles and exploration robots [4]; this same model, characterized by the inherent turning restrictions of its directional system, poses significant challenges, such as designing controllers that allow precise tracking of the trajectory.

The design of kinematic controllers is essential for ensuring that mobile robots follow trajectories accurately and efficiently [5]. Among the most common and prominent techniques is the use of proportional–integral–derivative (PID) controllers, whose simplicity and effectiveness have always been characterized. However, for complex scenarios, integrating fuzzy logic (FL) in the controller design offers a more robust strategy capable of adapting to variable conditions [6]. However, implementing these controllers often requires advanced navigation sensors that accurately measure the robot's movement and orientation in real applications. These sensors are essential to address environmental challenges and ensure reliable performance.

Detailed simulations are crucial before implementing kinematic control systems in a physical environment. These allow for the evaluation of the system's behavior under different conditions and the adjustment of parameters to optimize the design [7]. Today, multiple virtual environments offer advanced tools for designing and testing controllers, facilitating the development of more robust and secure solutions for mobile robots. Unity is a 3D development engine widely used in video game development. However, its applications are not limited to that; its great versatility has made it a key tool in numerous fields, such as education, simulation, and autonomous system design [8,9]. Unity is a flexible and powerful tool that allows one to model physical systems and replicate real-world scenarios with great fidelity [10].

This article offers two significant contributions to the state of the art:

- The design of a fuzzy kinematic controller for an Ackermann-type robotic structure offers an efficient and flexible solution for regulating robot motions, standing out for its ability to adaptively adjust gains in real-time. Unlike traditional methods, this controller does not require an exact mathematical model of the system, which simplifies its implementation and improves robustness against uncertainties and variations in the environment. The main innovation lies in the use of a fuzzy approach to determine PID-type gains, allowing for greater precision and stability in tracking specific trajectories. This dynamic adaptation capability makes the controller especially useful in highly variable scenarios where conventional methods could fail to maintain optimal system performance.
- A methodology for implementing the controller in a virtual environment developed in UNITY, which accurately simulates the robot's behavior. In this environment, control data are received and applied to the virtual robot, allowing the robot to move along predefined trajectories. This implementation validates the controller's performance and provides a clear and detailed view of the robot's behavior in different scenarios, facilitating the evaluation of its performance and the continuous improvement of the system. Real-time visualization within the virtual environment allows the controller to be efficiently tuned and optimized before its implementation on physical hardware.

This paper is structured as follows: Section 2 briefly discusses related works to our proposed research. Section 3 presents the developed methodology, and is divided into three main parts: the derivation of the kinematic model for the Ackermann-type robot, the design of the fuzzy PID controller, and the development of the 3D simulation in Unity. Section 4

presents the simulation results, considering different trajectories. Section 5 discusses the performance of the kinematic fuzzy-logic-based controller for trajectory tracking. Finally, Section 6 provides the conclusions and references.

2. Related Works

Trajectory tracking is one of the critical challenges in mobile robotics as it requires advanced control techniques that are capable of handling uncertainty and nonlinearity [11]. To better understand the current landscape on this subject, we performed a bibliometric analysis that visualized development trends and emerging areas of interest within the literature, as shown in Figure 1. Our study revealed that traditional control methods, such as PID and adaptive control laws, are highly cited. However, recent research shows a growing interest in optimal control, reinforcement learning, fuzzy logic-based controllers (FLCs), and other artificial intelligence techniques. Classical PID controllers are used due to their simplicity and effectiveness in maintaining stability; however, their performance deteriorates in highly nonlinear systems, requiring parameter tuning and model linearization as observed in [12]. Shojaei et al. [13] applied an adaptive control law to a differential-type robot; in this, the kinematic model was obtained, and from it, the linearization of the system was carried out using the feedback technique to obtain a model in which a control system can be applied. Although this technique presents robust results and is applied to a physical model, it is necessary to have a linear model to implement or resort to linearization techniques to approximate a model. To address these limitations, optimal control methods such as time-varying linear quadratic control (TVLQ) have been proposed by Caran et al. [14]. Their optimal control-based algorithm is founded upon the direct numerical calculation of the time-varying elements of the control law matrix, avoiding formalism of the set of linear matrix inequalities (LMIs) and the approximation of the solution of the Riccati partial differential matrix equation. On the other hand, Veselov et al. [15] took into account the nonlinear properties of a mobile tracked robot and employed an analytical design of aggregated regulators (ADARs), a method derived from synergetic control theory (SCT) that allows the synthesis of control laws for complex nonlinear systems without using linearization procedures. Unlike model-based approaches, which require precise system identification, reinforcement learning methods have been explored. Xie et al. [16] solved the trajectory tracking problem using an optimized reward reinforcement learning (ORRL) algorithm based on the Q-learning framework. Similarly, Ha et al. [17] evaluated the performance of an intelligent controller built on a deep Q-network (DQN) algorithm. While these approaches demonstrate good adaptive ability in complex driving environments, they demand significant computational resources and mathematical formulation, which may limit their real-time applicability. In contrast, fuzzy-logic-based controllers offer a viable alternative in such situations, as they dynamically adjust gains based on system behavior, making them particularly well-suited for scenarios where uncertainties or variations may arise. Several researchers have evaluated the effectiveness of FLCs in mobile robotics. Khesrani et al. [18] employed FL to auto-tune the endogenous feedback controller parameters, demonstrating good tracking of the desired tasks despite uncertainties. Meanwhile, Thuong et al. [19] implemented a proportional–derivative FL controller for trajectory following, while Mai et al. [20] combined a backstepping and adaptive fuzzy PID approach for a nonholonomic mobile robot. These results demonstrated good performance in terms of small distance errors, fast responses, and accuracy. Recent works have further explored fuzzy controllers integrated with genetic algorithms [21] and neural networks [22] to enhance system performance. A comprehensive description of fuzzy hybridization with other artificial intelligence techniques, as well as current fuzzy logic approaches, is reviewed in [23,24]. In order to evaluate the performances of different

control strategies without the costs and risks associated with real-world implementations, simulation platforms play a crucial role in the development of mobile robots. The bibliometric map in Figure 1 highlights the use of simulation environments for validating control strategies. Bai et al. [25] simulated a path-tracking controller based on the dynamic prediction model using MATLAB/Simulink, allowing for rapid prototyping and control system analysis. In addition, Shamshiri et al. [26] carried out a numerical simulation for an H_∞ robust adaptive controller by integrating MATLAB/Simulink and CoppeliaSim. Using a dedicated simulation environment highlights MATLAB's limitations in 3D graphics and interactive simulations. Mobile robots and their dynamic environment have been simulated in Gazebo, an open-source 3D robotics simulator that can be synchronized with ROS (a robot operating system). Molina-Leal et al. [27] employed Gazebo to evaluate the performance of a long short-term memory (LSTM) neural network for trajectory planning, whereas Zhang et al. [28] conducted simulations of mobile robots for obstacle avoidance in environments with dynamic obstacles. More recently, Unity3D has gained popularity due to its advanced rendering capabilities, physics engines, and accessibility. In [29], a Unity3D-based simulation environment was used to test unmanned ground vehicle (UGV) motion planning algorithms, demonstrating the viability of game engines for realistic robotic simulations. Integrating physics engines and building information modeling (BIM) enhances simulation realism. In addition, Liu et al. [30] proposed a mixed reality simulation as they used sensor information from the real world to project into a virtual environment created in Unity. Finally, it has been utilized for teleoperation, where mobile robots are remotely controlled through virtual reality (VR) in immersive virtual environments [31,32]. Despite advances in this field, certain gaps remain. While FLCs have demonstrated adaptation capabilities in variable scenarios, few works have explored them in virtual environments like Unity3D. By addressing the limitations of traditional control methods, the complexity of advanced algorithms, and the enhancement of the simulation framework, this research contributes to the development of efficient and scalable mobile robot designs.

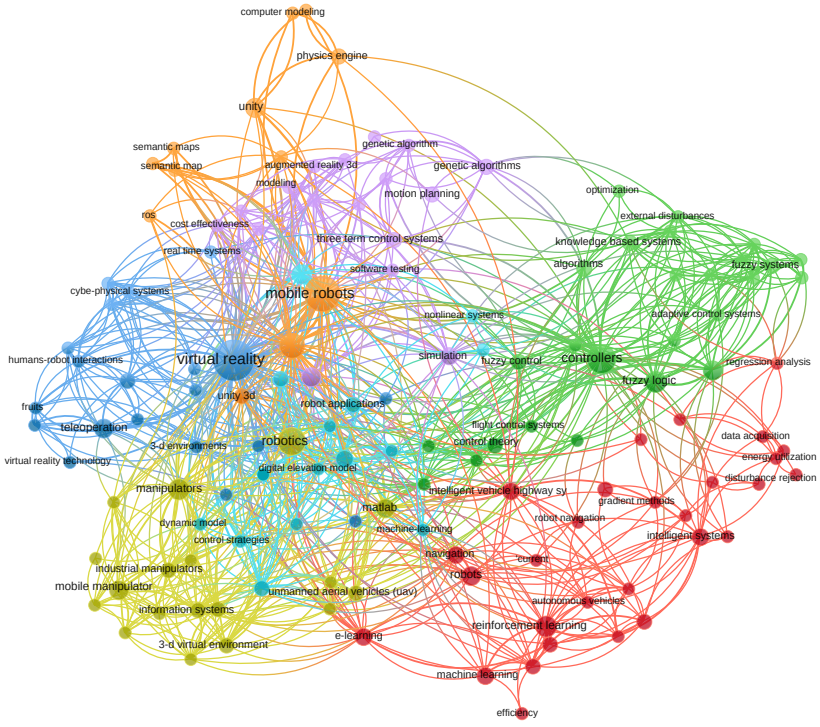


Figure 1. Bibliometric map of related works.

3. Materials and Methods

3.1. Mathematical Model

Within the group of four-wheeled mobile robots, the Ackerman-type configuration consists of two groups of two wheels: the group of rear drive wheels that are responsible for providing traction to the vehicle and the group of front wheels that are responsible for giving direction to the robot. In both groups, there is a single actuator in charge of this action, that is, traction and direction are combined. The movement of the Ackerman robot is characterized by how it addresses turns. Like other robot models, the robot can move in a straight line without a problem but cannot turn instantly since it requires a certain minimum radius to act. In addition, the drive wheels will experience some slippage during this action.

The representation of the Ackerman robot movement can be visualized in Figure 2; from there, it is possible to obtain the mathematical model that describes the kinematics of its movement; this kinematic model (KM) will be used to carry out the simulation. In Table 1, it is possible to observe the description of each of the elements.

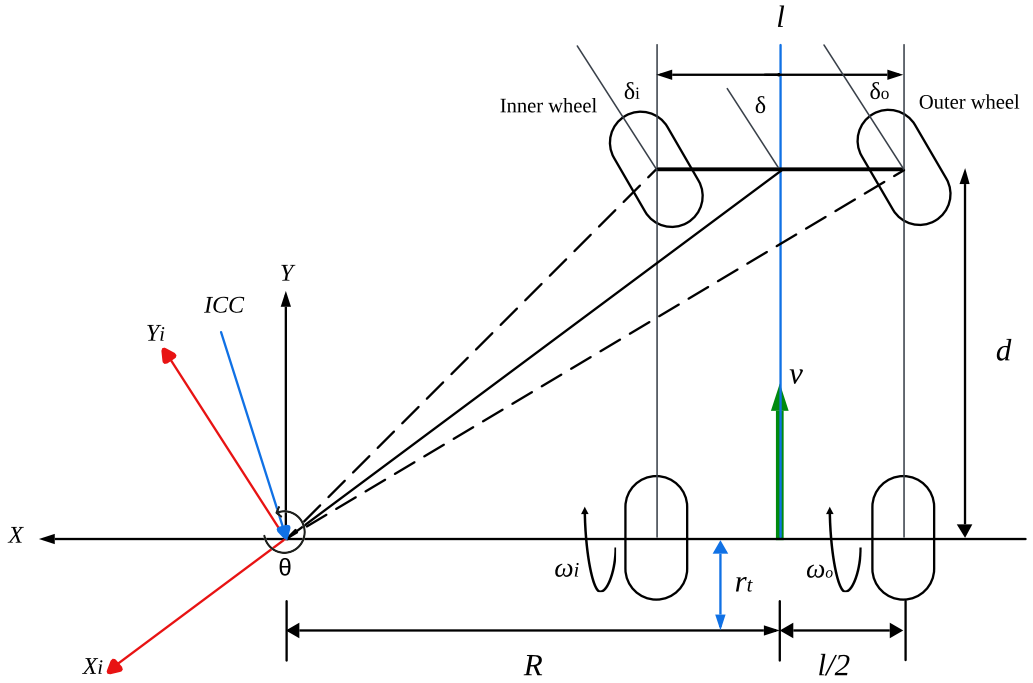


Figure 2. Kinematic model of the Ackerman-type robot.

Table 1. Elements of the Ackermann-type robot's kinematic model.

Variable	Description
ICC	Instantaneous center of curvature
l	Separation distance between front wheels
d	Separation distance between side wheels
δ	Front wheels turning angle
δ_i	Inner wheel turning angle
δ_o	Outer wheel turning angle
v	Linear speed of the robot
R	Radius of gyration (radius of curvature of the path)
ω_i	Inner wheel angular velocity
ω_o	Outer wheel angular velocity
θ	Turning angle

Analyzing the Ackerman configuration displayed in Figure 2, we observe how the turn occurs in this configuration. This model is distinctive in that it cannot rotate around its own reference system. The Ackerman configuration, when turning, does so through a circumference with a central point known as the instantaneous center of curvature (ICC). The steering angles of the front wheels determine this rotation center. The ICC is located precisely at the intersection of the extended axes of all the wheels, and the robot will rotate around this point. The ICC is not a fixed point; it can change at any moment with the slightest adjustment in the steering angle of the front wheels. Consequently, the dimensions of the turning circle will also be affected. The ICC does not depend on the number of wheels but rather on the number of constraints. In the case of the Ackerman configuration, the ICC is determined by two kinematic constraints: 1. The front wheels produce one constraint. Despite having two wheels, they generate the same set of constraints. 2. The rear axle provides the second constraint, which is fixed and cannot rotate [33]. In short, when the robot moves its front wheels, its entire body will turn around an instantaneous circle of radius R. This angle can be estimated by Equation (1):

$$R = \frac{d}{\tan \delta} \tag{1}$$

this equation describes how the curvature of the turning radius of the Ackerman robot is inversely proportional to the tangent of the steering angle, δ, and the distance between the lateral axles, d. When R tends to infinity, the movement is straight. Additionally, it is possible to observe how the physical characteristics and movements of the robot affect its trajectory.

The Ackerman robot's mechanism allows movement so the rear wheels do not produce a steering angle. Based on this, it is possible to determine the orientations of each wheel, with each wheel producing a different angle concerning the ICC; the inner wheel must be oriented at an angle, δ_i, greater than the outer wheel, δ_o; this allows the robot to turn around the midpoint between the rear wheel axes. The resulting angle at the midpoint corresponds to δ. The steering angles of the front wheels are determined by Equations (2) and (3).

$$\delta_o = \frac{\pi}{2} + \arctan \frac{R + \frac{l}{2}}{d} \tag{2}$$

$$\delta_i = \frac{\pi}{2} + \arctan \frac{R - \frac{l}{2}}{d} \tag{3}$$

Consequently,the inner wheel must move at a lower speed than the outer wheel, that is, $v_i < v_o$. Due to this configuration and the fact that only one actuator is responsible for the rotation of the rear wheels, both wheels rotate at the same angular speed, denoted as ω, around ICC. Therefore, using Equations (4) and (5), it is possible to calculate the linear velocities of each of the rear wheels:

$$v_o = \omega \left(R + \frac{l}{2} \right) \tag{4}$$

$$v_i = \omega \left(R - \frac{l}{2} \right) \tag{5}$$

To obtain the direct KM (DKM) of this configuration, it must be assumed that the steering angles of the front wheels are equal (as if there were only one steering wheel). This is because the steering angle around ICC is taken at the center of this axis, resulting in an average of both angles denoted by Equation (6):

$$\delta = \frac{\pi}{2} + \arctan \left(\frac{R}{d} \right) \tag{6}$$

then, the DKM for an inertial reference system is constituted by Equation (7).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{7}$$

This model describes linear velocities for the values of $\dot{x}$ and $\dot{y}$, and angular velocities for the value of $\dot{\theta}$. In this model, it can be observed that the linear velocities $\dot{x}$ and $\dot{y}$ depend on the system's rotation angle θ, while the rotational velocity $\dot{\theta}$ strictly depends on the angular velocity of the rear wheels, ω, since they are identical. This velocity can be determined using Equation (8):

$$\omega = \frac{v}{R} \tag{8}$$

It is assumed that $\dot{\theta}$ can be expressed in terms of the linear speed (v) and the radius of ICC (R). Still, it is also possible to express it in terms of the linear speed (v), the turning angle of the front wheels (δ), and the distance between the lateral axles (d); therefore, $\dot{\theta}$ can be expressed as shown in Equation (9):

$$\dot{\theta} = \frac{v}{R} = \frac{v}{d}\tan\delta \tag{9}$$

This equation can be used for the simulation process since ω corresponds to a physical measurement. By substituting Equation (9) into Equation (7), the following DKM of the Ackerman robot is obtained, as presented in Equation (10):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v}{d}\tan\delta \end{bmatrix} \tag{10}$$

Equation (10) and the modification in Equation (9) are implemented for the simulation. Since these equations are expressed in the continuous domain, it is necessary to apply temporal discretization. To achieve this, we use the forward finite difference method. The formulas obtained from this process are presented in Equations (11)–(13):

$$x[k] = x[k-1] + \Delta t \cdot v\cos(\theta[k-1]) \tag{11}$$

$$y[k] = y[k-1] + \Delta t \cdot v\sin(\theta[k-1]) \tag{12}$$

$$\theta[k] = \theta[k-1] + \frac{v\cdot\Delta t}{d}\tan(\delta) \tag{13}$$

The discrete model presented in Equations (11) and (12) does not consider possible changes in the robot orientation during time intervals. Therefore, it can produce errors when curved trajectories are performed or when Δt is not small enough. Consequently, corrections to the basic model based on numerical integration were added to obtain more precise approximations. This adjustment in the model can be observed in Equations (14) and (15):

$$x[k] = x[k-1] + \Delta t \cdot v\cos\left(\theta[k-1] + \left(\frac{v\cdot\Delta t}{2d}\right)\right) \tag{14}$$

$$y[k] = y[k-1] + \Delta t \cdot v\sin\left(\theta[k-1] + \left(\frac{v\cdot\Delta t}{2d}\right)\right) \tag{15}$$

The term $\frac{v\cdot\Delta t}{2d}$ represents the midpoint of the angular change; this refinement significantly enhances the accuracy for non-linear trajectories, resulting in more precise approximations.

Equations (13)–(15) will be used later in the design of the controller and for 3D simulations within Unity.

3.2. Controller Proposal

The implemented controller corresponds to a PID control with adjustable gains using FL. PID controllers are widely recognized and used for their efficiency and simplicity. However, as mentioned before, their performance can be limited in systems with non-linear dynamics or high uncertainty, factors that represent significant challenges in their implementation [34]. It is precisely in this context that FL demonstrates its usefulness. A fuzzy PID controller can adjust its gains, adapting to changing environments dynamically. This provides greater flexibility to the system and significantly improves the controller's performance [35]. The mathematical model of the PID controller is represented by Equation (16).

$$\delta(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{d}{dt} e(t) \tag{16}$$

where $e(t)$ is the error, and K_p, K_i, and K_d are the proportional, integral, and derivative gains, respectively. $\delta(t)$ is the orientation of the KM of the mobile robot. These are the values that FL will adjust. The control diagram of the system can be seen in Figure 3.

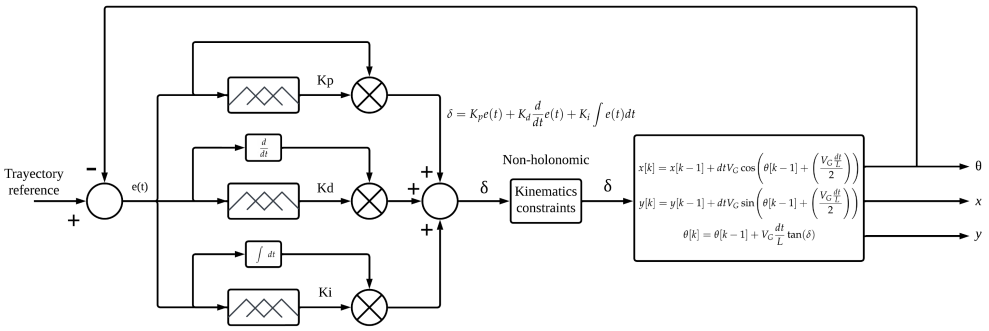


Figure 3. Control structure proposed.

According to the control scheme presented in Figure 3, it can be observed that the fuzzy PID control focuses on controlling the robot's steering angle $\delta(t)$. The input to the controller is a reference trajectory, from which the trajectory generated by the simulation is subtracted to determine the directional error that can be visualized in Equation (17):

$$e(t) = \theta_{des} - \theta(t) \tag{17}$$

Considering the trajectory data, $\theta(t)$ is determined at each trajectory moment and compared with the results obtained from the simulation. θ_{des} is the desired orientation. This comparison is made within the kinematic limitations of the model; in this study, $\delta(t)$ is restricted to a range from $\frac{\pi}{4}$ to $-\frac{\pi}{4}$ *rad*.

Subsequently, $\delta(t)$ is utilized in the discrete DKM through Equations (13)–(15), yielding the positions x and y, as well as the ICC angle $\theta(t)$. The obtained values of x and y are employed to generate the trajectory for simulation, after which, the process returns to the start for error comparison. On the other hand, to demonstrate the system's stability, the closed-loop transfer function of the orientation of the Ackermann structure must be found [36]. First, the steering angle Equation (6) is substituted into Equation (17), corresponding to the PID controller. This substitution yields Equation (17), describing the behavior of the controller in terms of the variables $\delta(t)$ and $\dot{\delta(t)}$:

$$\dot{\theta}(t) = \frac{v(t)}{d} \tan\left(K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}\right) \tag{18}$$

This equation exhibits non-linear behavior due to the function $\tan(\cdot)$, which introduces complexities in its analysis and a direct solution using conventional analytical techniques. However, under the assumption of small deviations in the steering angle ($\delta(t) \approx 0$), applying a linearization by first-order approximation is possible, considering that $\tan(\delta(t)) \approx \delta(t)$. This simplification reduces the complexity of the model, allowing the system to be represented by linear equations that facilitate both the stability analysis and the design of controllers using classical linear control methods, as shown in Equation (19):

$$\dot{\theta}(t) \approx \frac{v(t)}{d}\left(K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}\right) \tag{19}$$

If the velocity is assumed to be constant $v(t) = V$, Equation (20) can be obtained:

$$\dot{e}(t) + \frac{V}{d}K_p e(t) + \frac{V}{d}K_i \int e(t)dt + \frac{V}{d}K_d \frac{de(t)}{dt} = 0 \tag{20}$$

The Laplace transform with zero initial conditions is applied to Equation (20) to obtain Equation (21), which represents the behavior of the PID controller in the complex domain:

$$\left(s + \frac{V}{d}K_d s + \frac{V}{d}K_p + \frac{V}{d}\frac{K_i}{s}\right)E(s) = 0 \tag{21}$$

By applying the Laplace transform to Equation (17) and substituting the result into Equation (21), followed by rearranging the terms, Equation (22) is derived:

$$G_{cl}(s) = \frac{\theta(s)}{\theta_{ref}(s)} = \frac{\frac{V}{d}(K_d s^2 + K_p s + K_i)}{\left(1 + \frac{V}{d}K_d\right)s^2 + \frac{V}{d}K_p s + \frac{V}{d}K_i} \tag{22}$$

The poles are evaluated by solving the characteristic equation derived from the transfer function presented in Equation (22) to assess the stability of the system. This characteristic equation corresponds to a second-order system, whose general form allows the identification of the coefficients that determine the location of the poles in the complex plane. The poles of the system, which define its dynamic behavior, are expressed by Equation (23). The location of these poles is essential as it determines whether the system is stable, unstable, or oscillatory. For the system to be stable, both poles must have a negative real part, thus ensuring that the oscillations decrease over time and the system tends to a steady state:

$$s = \frac{-\frac{V}{d}K_p \pm \sqrt{\left(\frac{V^2}{d^2}K_p^2\right) - 4\frac{V}{d}K_i\left(1 + \frac{V}{d}K_d\right)}}{2\left(1 + \frac{V}{d}K_d\right)} \tag{23}$$

If $K_p^2 - 4(1 + K_d)K_i > 0$, the poles of the system are real and negative whenever $K_p > 0$, which ensures that the system will be stable and free of oscillations. This condition is especially suitable for robot orientation control since a non-oscillatory response ensures more precise and smoother movements, avoiding unwanted overshoots that could affect the robot's trajectory. Based on this assumption, the fuzzy controller is designed to adjust the control parameters dynamically to maintain stability under different operating conditions. This ensures that the poles remain in the left half-plane and that the system responds stably and efficiently. For the steering angle of an Ackermann robot, the linguistic variable error, error derivative, and error integral are employed to replicate the structure of a classical PID adapted to nonlinear systems. The error indicates the angular deviation from the desired path, the error derivative anticipates abrupt changes in orientation to improve dynamic response, and the error integral corrects persistent deviations, ensuring accurate

tracking. Three fuzzy systems are designed to tune the controller gains, one for each control gain (K_p, K_i, K_d), using the error, error integral, and error derivative as linguistic variables, respectively. Linguistic values are described by the following descriptors: *negative-large (NB)*, *negative-small (NS)*, *zero (ZE)*, *positive-small (PS)*, and *positive-large (PL)*, allowing continuous adaptation of the gains to optimize steering angle control while respecting the holonomic constraints of the system.

For the gain K_p, five linguistic values are defined in Table 2, distributed symmetrically to ensure an equitable adjustment in calculating the control gains. This symmetry is essential to avoid biases in the controller response, ensuring that the corrections are homogeneous when faced with deviations in both directions. Furthermore, it allows for a smooth transition between the different gain levels, improving the accuracy of the system. Triangular membership functions are used to represent each of the linguistic values. This type of function is selected due to its practicality in situations with a clearly defined central value or inflection point. Furthermore, they stand out for their mathematical simplicity and computational efficiency, significantly reducing the load during the inference process. This feature makes them an ideal choice for systems where performance is necessary. Thanks to their ease of implementation and low computational costs, triangular membership functions are particularly useful in simulation applications, where a balance between accuracy and efficiency is required. The triangular membership functions corresponding to these linguistic values, designed with a symmetrical distribution, are illustrated in Figure 4, highlighting their impact on the linearity of the adjustment.

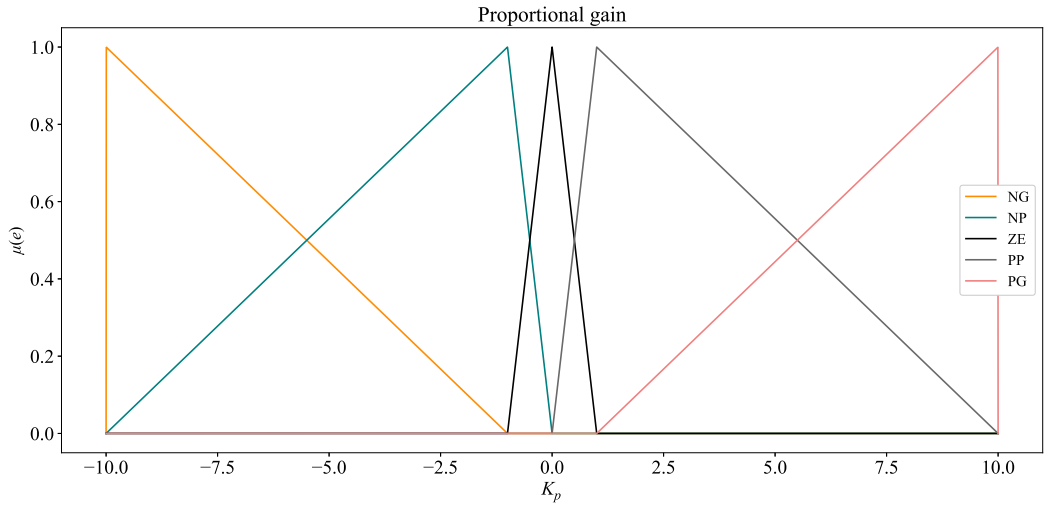


Figure 4. Error linguistic variable distribution.

Table 2. Error linguistic variable.

Linguistic Value	Positions
NL	[−10,−10,−1]
NS	[−10,−1,0]
ZE	[−1,0,1]
PS	[0,1,10]
PL	[1,10,10]

For the output, singleton functions were chosen to reduce the computational costs; the descriptions of the output values can be seen in Table 3. Only positive values were selected for singletons because controller gains were restricted to positive values to ensure stability. Therefore, three linguistic values, big (B), medium (M), and small (S), were selected to compute the gain K_p.

Table 3. Singleton distribution for K_p computing.

Linguistic Value	Positions P_i
B	10.0
M	4.0
S	2.5

A similar methodology determines the linguistic values for K_d and K_i. Table 4 provides a description of the linguistic values for K_d, and Figure 5 illustrates its membership functions.

Table 4. Error derivative linguistic variables.

Linguistic Value	Positions
dNL	[−10,−10,−1]
dNS	[−10,−1,0]
dZE	[−1,0,1]
dPS	[0,1,10]
dPL	[1,10,10]

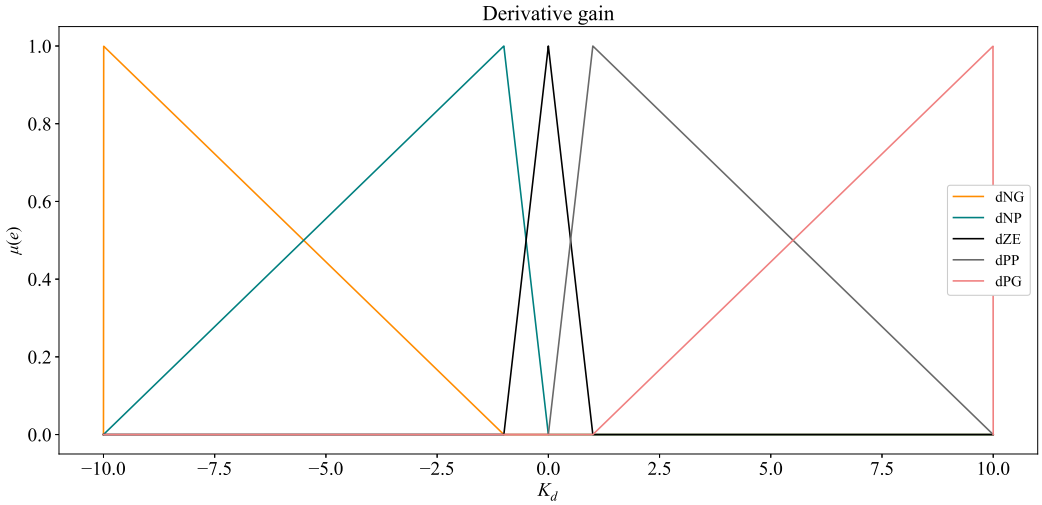


Figure 5. Error derivative linguistic variable distribution.

The values for the system output for the derivative gain, K_d, are shown in Table 5.

Table 5. Singleton distribution for K_d computing.

Linguistic Value	Positions P_i
dB	0.9
dM	0.6
dS	0.0

For the integral gain, K_i, the input linguistic values are displayed in Table 6, and the output values are displayed in Table 7. The membership functions are shown in Figure 6.

Table 6. Integral of error linguistic variable.

Linguistic Value	Positions
iNL	[−10,−10,−2]
iNS	[−10,−2,−0.5]
iZE	[−0.5,0,0.5]
iPS	[0.5,2,10]
iPL	[2,10,10]

Table 7. Singleton distribution for K_i computing.

Linguistic Value	Positions P_i
iB	8.0
iM	4.0
iS	0.0

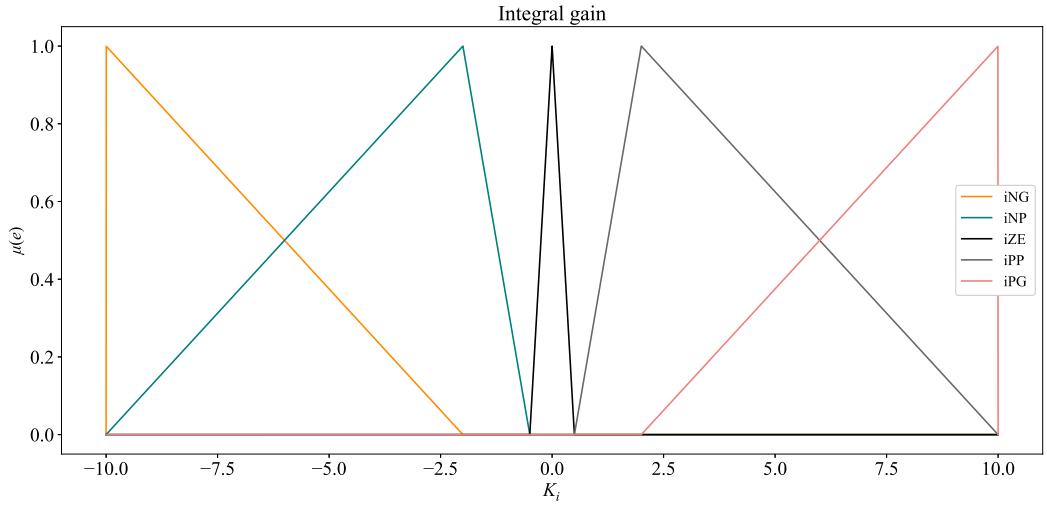


Figure 6. Integral of error linguistic variable distribution.

A one-to-one relationship is established with the input values and five corresponding output values, allowing for direct fuzzification. Once this symmetric relationship is defined, the output of the fuzzy system is determined using the centroid method. This approach provides a smooth and gradual output, which is particularly useful in motor applications. The centroid defuzzification method is derived from Equation (24):

$$k_{gains} = \frac{\sum \mu(P_i) \cdot P_i}{\sum \mu(P_i)} \tag{24}$$

In this context, P_i denotes the position of the i-th singleton, while $\mu(P_i)$ represents a fuzzy number. The rules for the controller are outlined in Table 8. These rules allow the fuzzy PID controller to adapt to different system conditions. For large errors or rapid changes, the controller increases gains to apply stronger corrections, while under stable conditions, it reduces gains to minimize overfitting and maintain system stability. This is critical for Ackermann-type robots, where rotational constraints require precise, dynamic adjustments to the control response.

Table 8. Fuzzy rules for the mobile robot.

K_p	K_d	K_i
If e(t) is NL, then K_p is B	If $de(t)$ is dNL, then K_d is dB	If $ie(t)$ is iNL, then K_i is iB
If $e(t)$ is NS, then K_p is M	If $de(t)$ is dNS, then K_d is dM	If $ie(t)$ is iNS, then K_i is iM
If $e(t)$ is ZE, then K_p is S	If $de(t)$ is dZE, then K_d is dS	If $ie(t)$ is iZE, then K_i is iS
If $e(t)$ is PS, then K_p is M	If $de(t)$ is dPS, then K_d is dM	If $ie(t)$ is iPS, then K_i is iM
If $e(t)$ is PL, then K_p is B	If $de(t)$ is dPL, then K_d is dB	If $ie(t)$ is iPL, then K_i is iB

3.3. 3D Simulation Environment Design

The controller simulation consists of two stages: first, a Python script calculates and sends the route coordinates, and second, a C# script in Unity receives these data for processing and simulation. Figure 7 illustrates the block diagram of the procedure. It is important to note that this simulation method is chosen to prevent overloading processes in Unity.

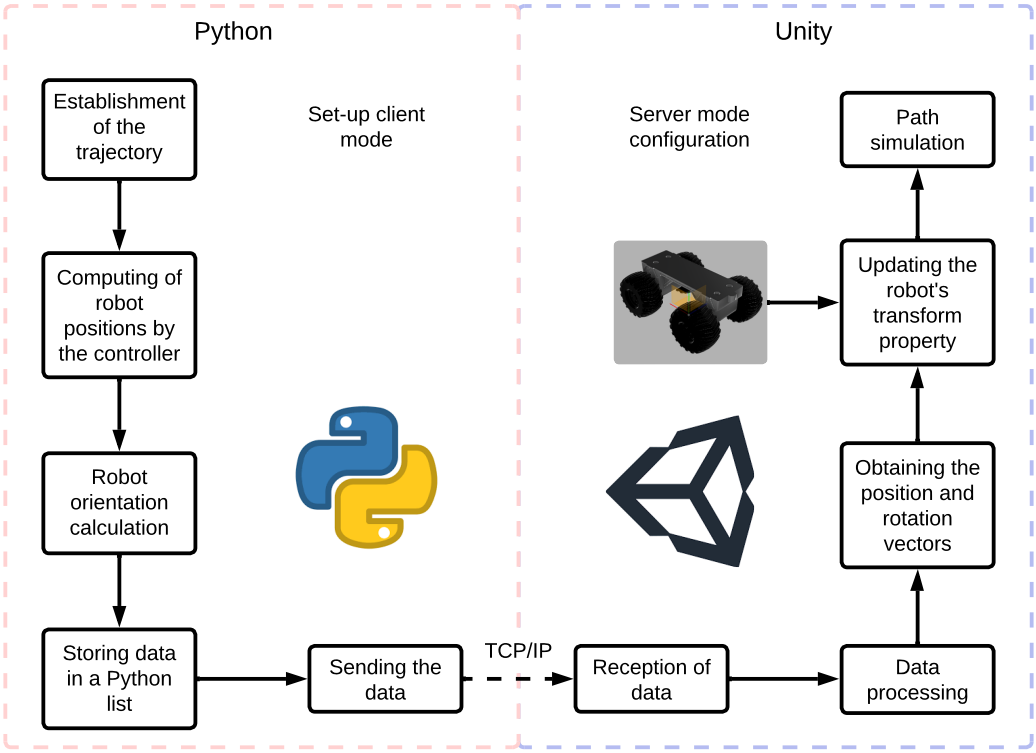


Figure 7. Virtual environment block diagram.

The fuzzy controller described in Section 3.2, programmed in Python 3.11.5, is implemented in the first stage. Various trajectories to be followed are proposed, and, using the controlled model, the positions of x, y, and the angle θ are obtained. As previously discussed in the kinematic model presented in Section 3.1, for non-holonomic robots, θ represents the instantaneous curvature angle generated by the robot relative to its ICC. Therefore, data are still required to perform the 3D simulation. In order to implement such data in the simulation in Unity, it is necessary to analyze the parameters from a three-dimensional point of view (x, y, z). Conventional KMs, such as the one reviewed in Section 3.1, are represented in a two-dimensional environment (x, y) and do not analyze the robot's orientation, an essential detail to be able to perform a 3D simulation. This problem can be seen reflected in Figure 8.

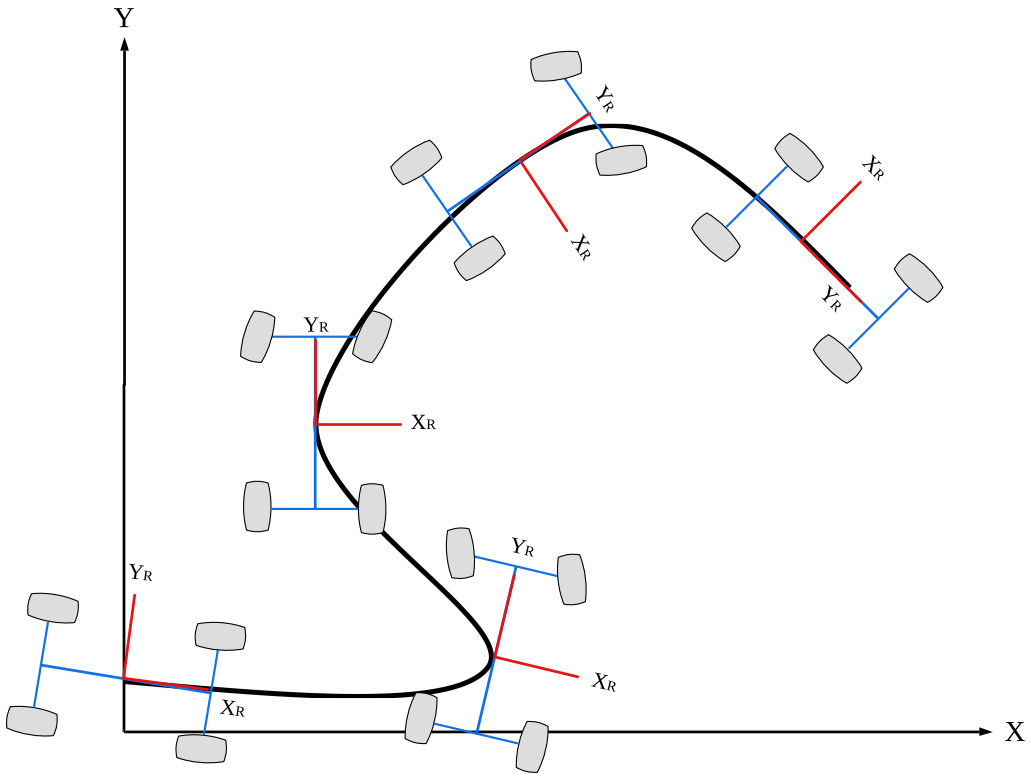


Figure 8. Determination of the yaw angle.

As illustrated in Figure 8, the steering mechanism does not consider how the object's rotation varies as it moves along the path. In 3D environments like Unity, each object has its own reference frame independent of the global frame. This local frame allows the object to not only move within the global frame (changing its x and y positions in a 2D view) but also to orient or rotate around its own axis, reflecting more complex 3D dynamics. Within Unity, these rotations correspond to Euler angles (*pitch, roll, and yaw*), which are angles also used in robotics to identify the inclinations of robots. *Yaw* is the angle responsible for determining orientation along the z-axis, representing the robot's self-rotation; this is the missing element needed to perform the 3D simulation.

From the analysis in Figure 8 and taking the orientation calculations used in SLAM (simultaneous localization and mapping) navigation systems as a reference, it can be observed that the angle *yaw* represents the rate of change of the position in the plane. Graphically, this angle corresponds to the slope of the path followed by the system. The angle *yaw* can be determined as the tangent of the angle formed between two consecutive points on the path, which allows it to be calculated using Equation (25). In it, *arctan2* is used to account for the signs of the subtractions:

$$\psi = \arctan 2\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right) \tag{25}$$

The Python script also calculates Equation (25). Once the method for obtaining all necessary values is defined, these values are computed and stored in a list that organizes the data in the following format $(x, 0, y, 0, \psi, 0)$. This structure is used due to the way Unity interprets positions. Subsequently, all the data stored in the list are sent sequentially by a thread through the TCP/IP protocol, establishing a local communication where the script acts as a client of the connection. The complete development of the program can be observed in Figure 9.

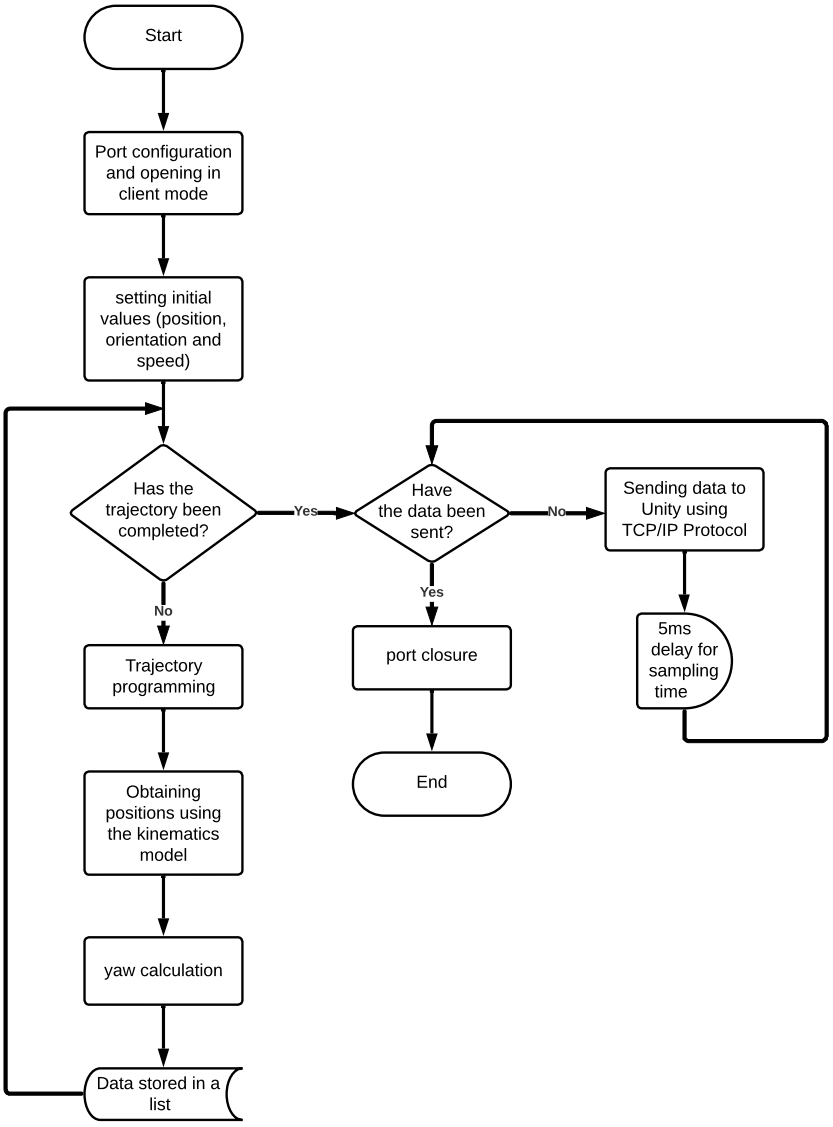


Figure 9. Flowchart of the first stage.

The second stage takes place in Unity, where the 3D simulation of the path taken by the robot is carried out. This process uses a script in C#, Unity's native language. This script receives the reception of data sent from Python via the TCP/IP protocol. Since the Python script is configured in client mode, Unity is the server that listens to all incoming connections.

Once the connection is established, the data are read sequentially. As they are received, the corresponding elements are interpreted and processed. The data arrive in the form of a text string (*string*) with the format $(x, 0, y, 0, \theta, 0)$. This is attributed to how Unity manages the positions and rotations of objects. In Unity, every object has two motion vectors: one for the position coordinates, represented as $(left/right, up/down, forward/backward)$, and one for the rotation values, expressed as $(pitch, yaw, roll)$. In this way, when receiving the data, it is only necessary to split them into two vectors: one for the position and one for the rotation. The splitting process begins by checking that a string containing precisely six comma-separated values is received. The six elements are extracted and stored in an array if this condition is met. A variable of the *Vector*3 type is then created to assign the first three data points corresponding to the position. The remaining three data points, representing the rotation values, are stored in a variable of the *Quaternion* type. Before being stored, each datum is cast from string to float. Once each piece of data is processed, the robot's

position and orientation are updated in the *Update* method of the script. To trace the robot's path, within this method, each time the position is updated and it is detected that it is different from the previous one, a new *GameObject* is generated, leaving a mark on the position traveled by the robot. This process allows the complete path of the robot to be visualized during the simulation. Figure 10 shows the flow diagram corresponding to the described process.

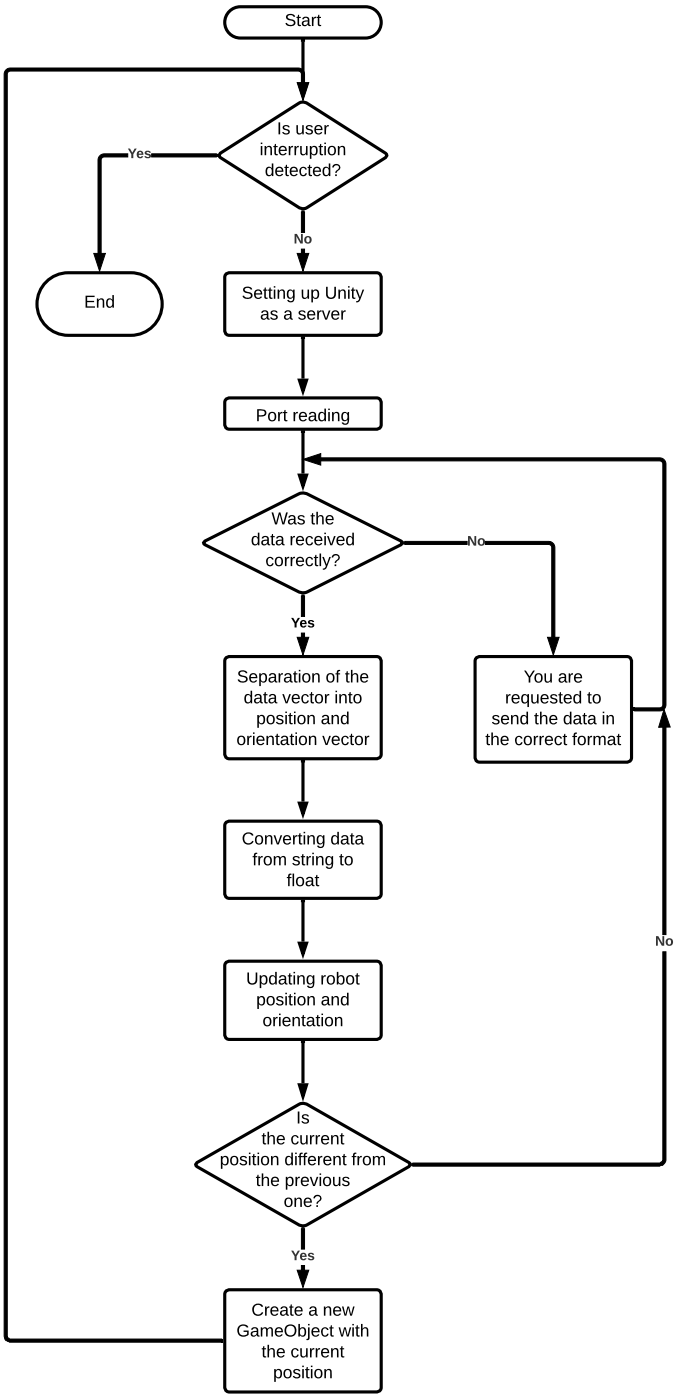


Figure 10. Flowchart of the second stage.

The simulation starts by initializing the scene in Unity, which includes opening the port and waiting for the data to be received. The Python script then calculates all the data corresponding to the robot's path. Once this calculation is completed, the data are sent

to Unity via the TCP/IP protocol. As the data are received, it is possible to visualize the robot's path in real time within the simulated environment.

To visualize the objects in the 3D environment, a scene is designed to allow the robot's path to be observed. This scene includes a base terrain, the *GameObject* representing the robot, and other fundamental elements provided by the Unity environment to complete the simulation. The design of the scene ensures a clear and understandable representation of the path. The layout of this environment can be seen in Figure 11.

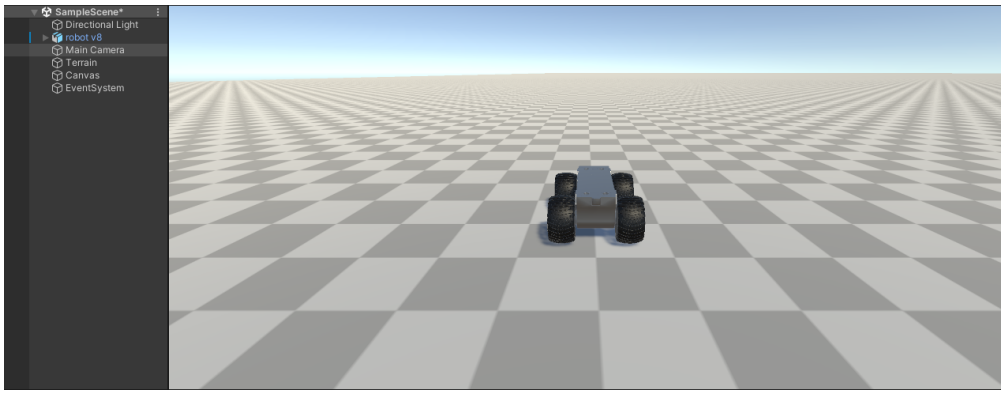


Figure 11. Virtual environment scene.

The 3D model of the robot was designed using Fusion 360 software, with dimensions of 174 × 73 × 126 mm. This model was then scaled within Unity to fit the requirements of each test. The model was exported in *obj* format. It was important to ensure that it was oriented according to the Unity coordinate system, where the y-axis represents the height, as opposed to classic formats where this corresponds to the z-axis. Likewise, the reference system of the model had to be centered on the robot to ensure that the rotations were accurate and corresponded to those expected. Figure 12 shows the 3D design of the robot used in the simulation. Table 9 depicts the geometry of the 3D model of the robot.

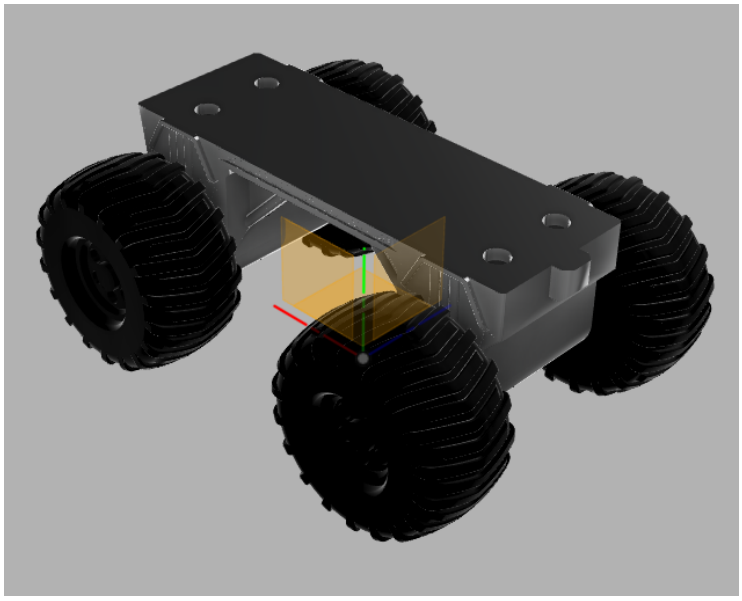


Figure 12. A 3D model of the robot in Fusion 360.

Table 9. Characteristics of the 3D model of the robot.

Description	Measures
Width	73 mm
Long	174 mm
Height	12.6 mm
Distance between the axles	107.6 mm
Wheel separation	91.75 mm
Wheel radius	33.34 mm

4. Results

Three different path-tracking scenarios are analyzed. In the first scenario, the following circular reference path is considered, which allows for the evaluation of the system's ability to maintain a smooth and continuous movement. In the second scenario, the tracking of discrete points distributed on a map is proposed, simulating trajectories composed of specific positions to be reached. Finally, in the third scenario, the following of crossed paths is addressed, which implies more abrupt changes in direction and greater complexity in control. In each of these cases, different initial configurations and variations in the dimensions of the robot are analyzed, with the aim of evaluating the performance of the system under different operating conditions. An ideal surface is always assumed for the simulation process, which implies that rotations in *pitch* and *roll* are not considered.

4.1. Circular Trajectory

For this test, a constant linear speed of 2 m/s and a spacing between axes of 0.25 m are used. The proposed path is circular with a radius of 10 m. Initially, the robot is located at the coordinates (5,5) as a reference point, oriented at an angle of $-\frac{\pi}{2}$ rad. From these initial parameters, the successive positions of the robot are determined during the path tracking. Figure 13 illustrates both the proposed reference path and the one followed by the robot, highlighting its performance in the simulation.

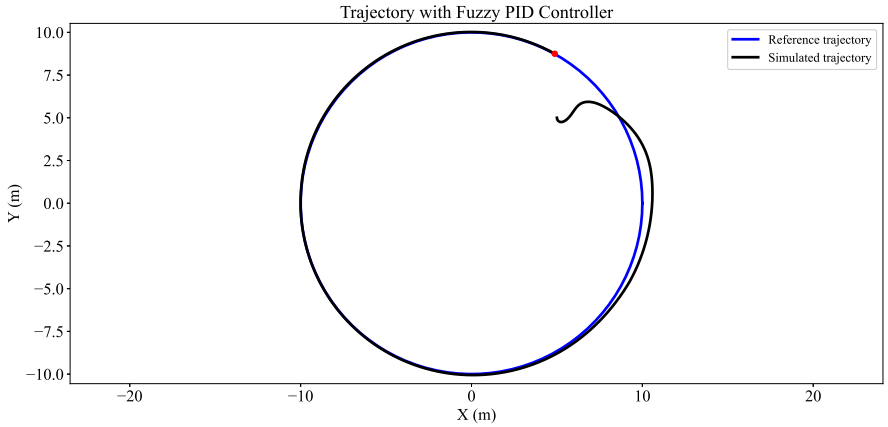


Figure 13. First test: simulate circular path in Python.

The controller identifies the initial position of the robot and, based on the proposed restrictions, begins the movement to correct its position continuously. Figure 14 shows how the error evolves as the robot moves. In this case, it can be seen that the error never reaches zero, which is consistent with the nature of the circular path. The angle δ must remain constant to maintain the turn. If the angle δ were equal to zero, it would imply that the robot would follow a straight-line path, which would be incompatible with the proposed circular movement.

Figure 15 illustrates the applied control signal, which shows its correlation with the error. The signal is consistently directed to maintain the desired trajectory. The data presented in the graph are confined to angles of $\frac{\pi}{4}$ and $-\frac{\pi}{4}$ rad due to the constraints established in the model.

Figure 16, shows the simulation of the path in the Unity environment. In these images, it can be observed how the positions generated along the route were marked, showing a high coincidence with the trajectory previously obtained in Python. This correspondence between both simulations validates the accuracy of the implemented model.

It is also noteworthy that the orientation of the robot in Unity is precisely aligned with the direction of the path, which reflects the coherent and realistic behavior of the control system. This detail allows us to appreciate not only the movement of the robot but also its ability to dynamically adapt to the expected path. The combination of positional accuracy and correct orientation demonstrates that the simulation process offers a faithful and visually realistic representation of the robot's performance in the virtual environment.

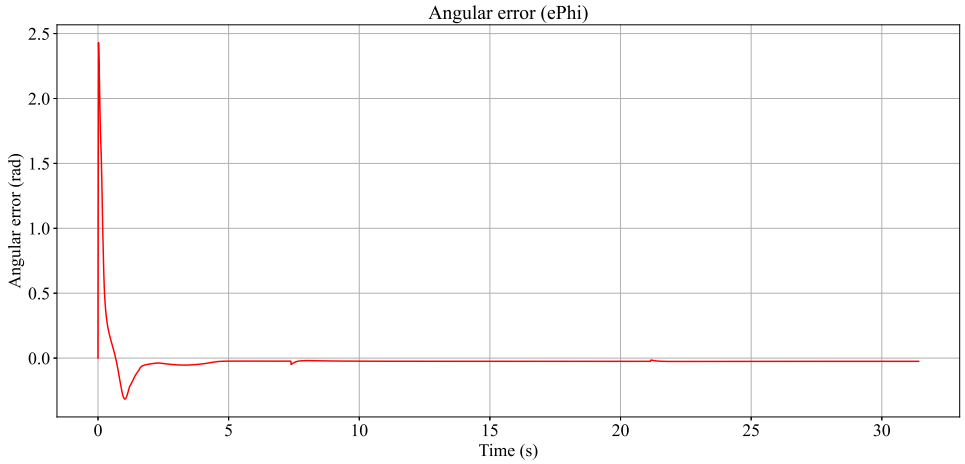


Figure 14. Angular error for circular path.

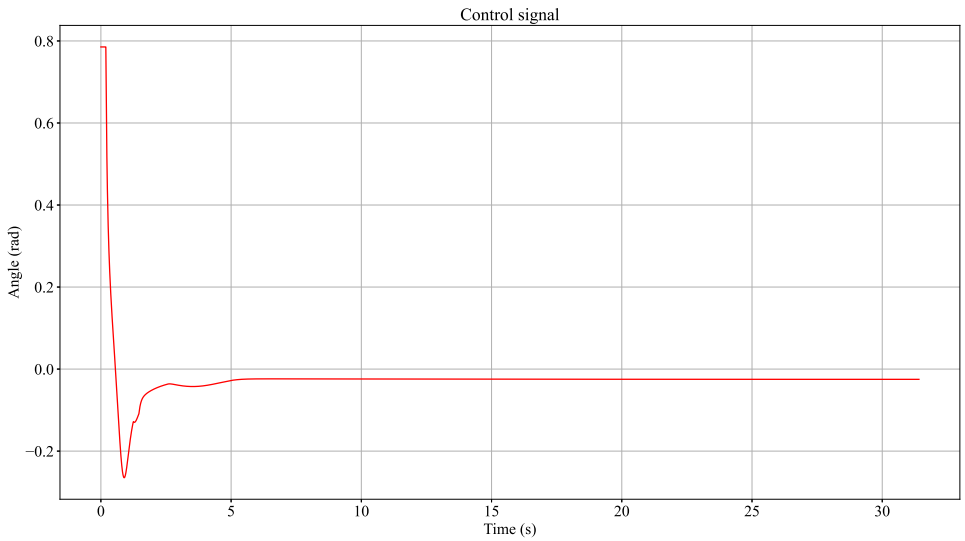
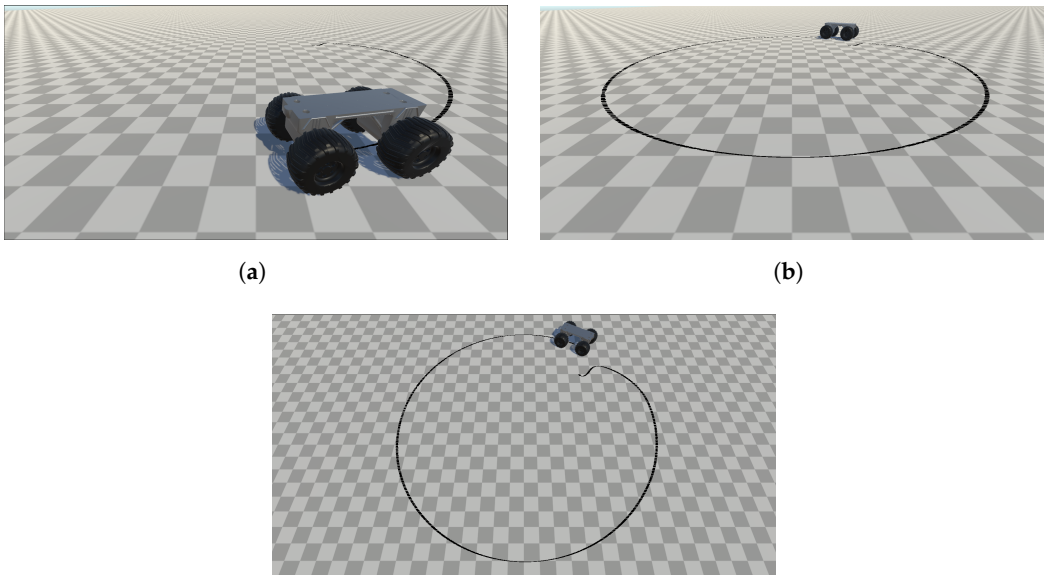


Figure 15. Circular path control signal



(**a**) (**b**)

(**c**)

Figure 16. Simulation of the circular trajectory made in Unity. (**a**) View of the robot orientation for the path. (**b**) Front view of the circular path simulation. (**c**) Aerial view of the circular path simulation.

4.2. Point-to-Point Trajectory

In this second case, multiple points distributed along the map were defined, and straight lines were drawn from them to connect the points and form the desired path. This test used a constant linear speed of 2 m/s and a separation between axes of 0.5 m. The robot was initially positioned at the coordinates $(2, 1)$ with an orientation of $\frac{3\pi}{5}$ rad.

The result obtained in this simulation is shown in Figure 17, where the tracking performed by the robot can be observed. This analysis allows it to evaluate the capacity of the system to maintain a precise and continuous movement between the points of the path, considering the geometric and dynamic restrictions established for the model.

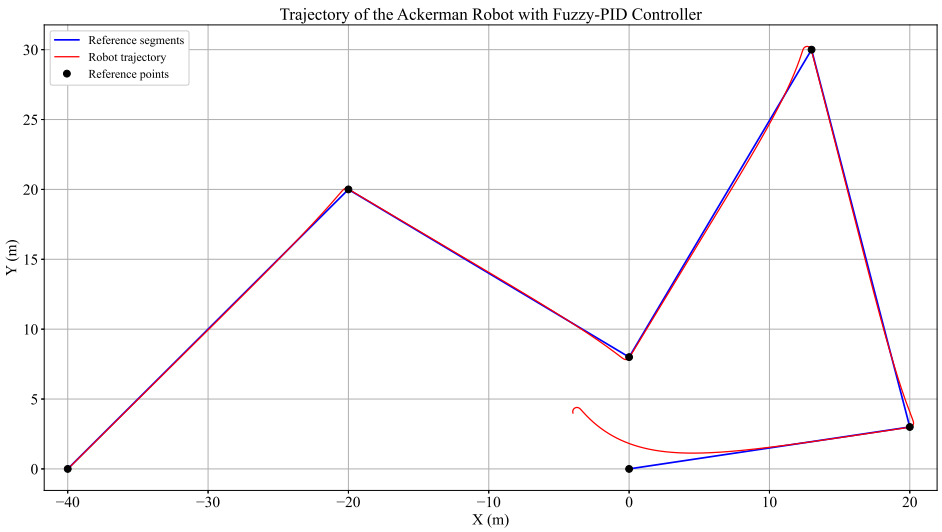


Figure 17. Second trajectory test: point-to-point case.

Figure 18 shows the error graph corresponding to the test performed in this section. The changes that occur when the robot reaches one of the objectives and generates a zero error can be observed. As mentioned above, this behavior indicates that the robot continues moving in a straight line after correcting its trajectory.

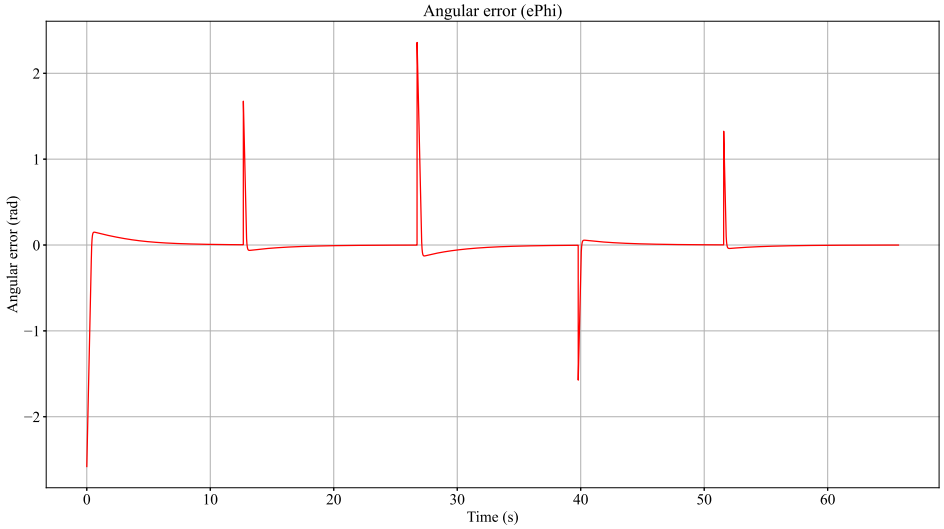


Figure 18. Angular error for point tracking path.

Figure 19 presents the control signal graph for the current case, highlighting the overshoots that occur when reaching specific points along the path. These overshoots are inherent consequences of the kinematic configuration of the Ackermann-type robot, requiring turning maneuvers at reference points before continuing its movement. Since the Ackermann robot has a geometry with a single steering axis, when reaching a turning point, an adjustment in the steering angle is necessary to compensate for the path error, which generates a small oscillation or overshoot in the control signal. In addition, a fast and efficient response to the control action is observed, suggesting that the system possesses a high capacity for making quick adjustments in the robot's direction. This responsiveness is critical for minimizing stabilization time after overshoots and ensuring that the robot returns to its desired path as accurately as possible.

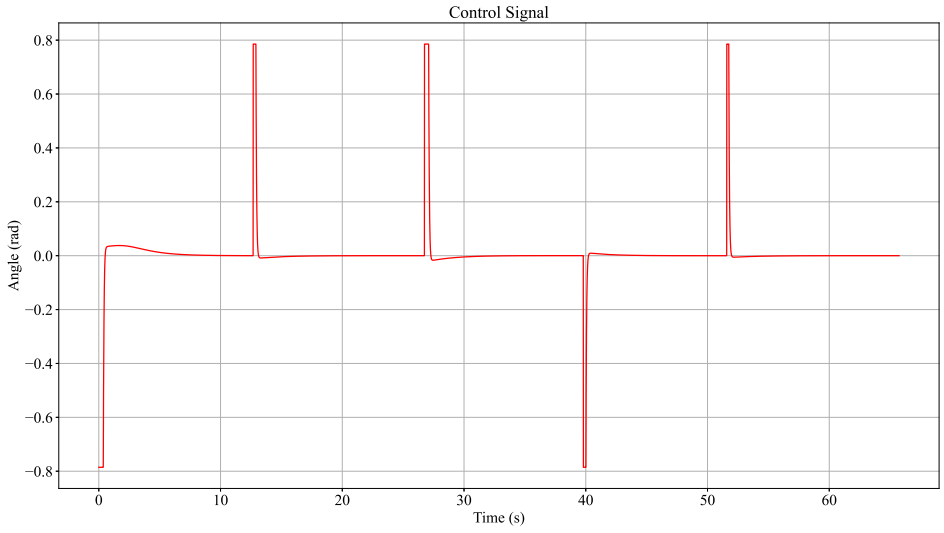
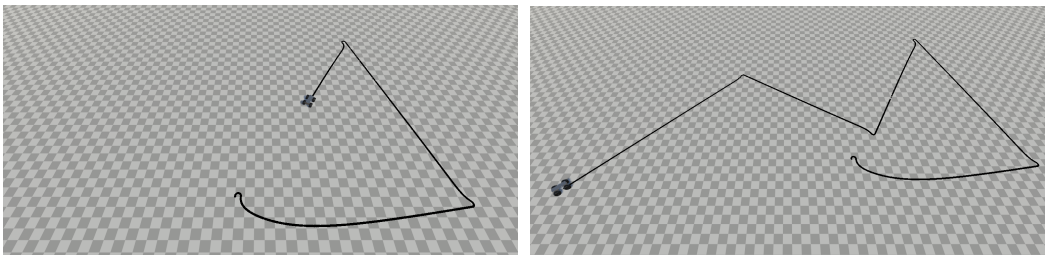


Figure 19. Control signal for point-to-point trajectory.

The simulation in Unity is displayed in Figure 20, where it can be seen how the robot's path was carried out.



(**a**) (**b**)

Figure 20. Point tracking path simulation done in Unity. (**a**) Partial path of the simulated robot in Unity. (**b**) Complete robot path in Unity.

4.3. Crossed Trajectory

Finally, a case where the trajectory crosses the same area is proposed; the test configurations are as follows: linear velocity of 3 m/s, a distance between the axes of 0.1 m, an initial position at coordinates $(8, 3.5)$, and an initial orientation of 0° rad. The results of this test can be observed in Figure 21, demonstrating how the robot quickly integrates into the trajectory and follows it almost perfectly.

Figure 22 shows the error graph, where it can be observed that the error is remarkably small. This behavior is mainly attributed to the geometry of the trajectory, which includes sections with linear segments. The controller effectively manages these straight sections, allowing the error to remain low at all times. The controller's ability to adjust the orientation and maintain precision during linear sections contributes to the stability of the system and minimizes the necessary corrections. On the other hand, in sections where the trajectory presents greater complexity, the error tends to increase slightly. However, it remains low, highlighting the controller's efficiency in maintaining performance under various geometric conditions.

Figure 23 presents the control signal generated for the third evaluated trajectory. Given the geometry of this route, it can be observed that the required control actions are significantly reduced, especially in the sections characterized by sharper curves. This is due to the inherent smoothness of the trajectory, which limits the need for constant adjustments in the robot's orientation. Additionally, the control signal shows high stability in most sections, indicating efficient performance of the control system by minimizing oscillations and abrupt corrections even in scenarios with geometric variations.

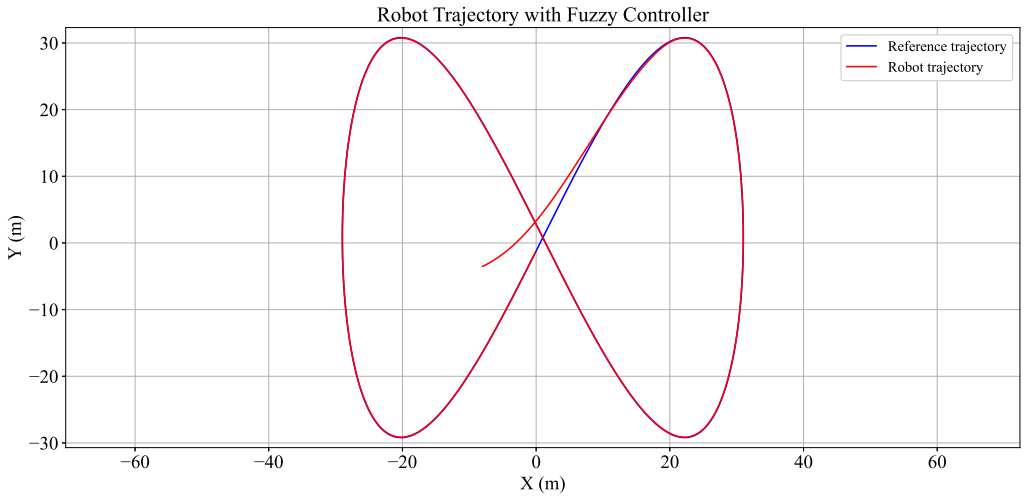


Figure 21. Third case of trajectories: movement always in change.

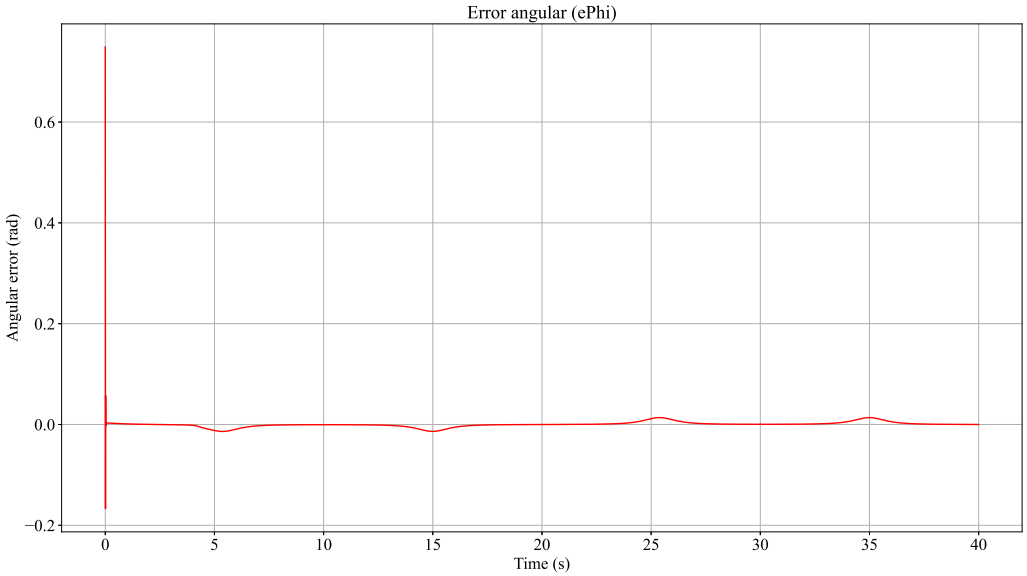


Figure 22. Angular error of rotation δ for the crossed trajectory.

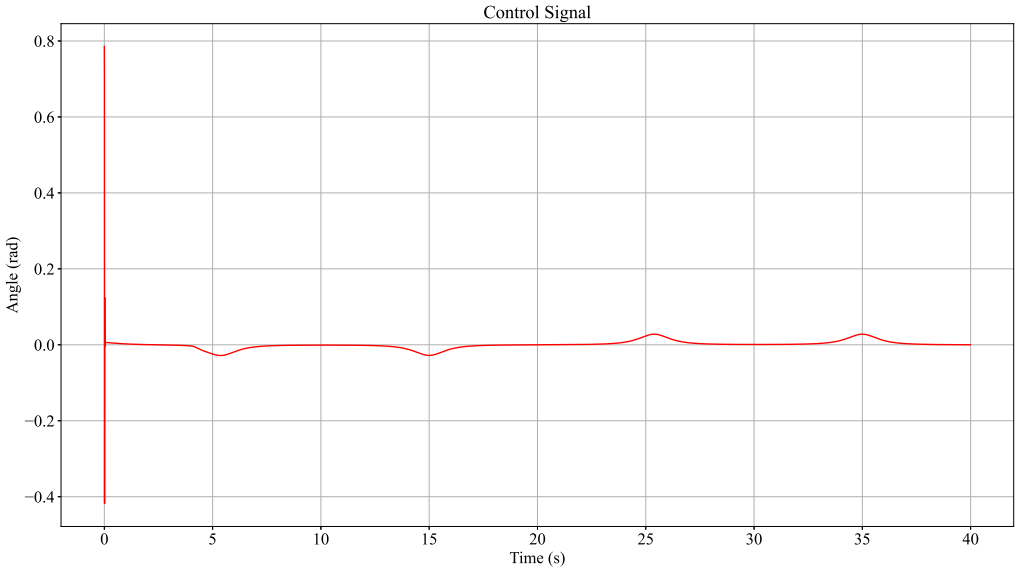
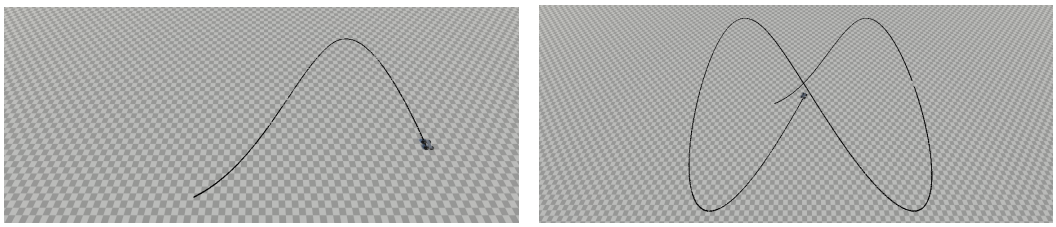


Figure 23. Control signal for the cross path.

Finally, Figure 24 shows the simulation of the last trajectory, which runs correctly.

The simulation was carried out on a laptop equipped with an Intel Core i3-1115G4 processor with a frequency of 3.00 GHz, 12 GB of RAM, and integrated Intel(R) UHD graphics. Different load levels were recorded on the system components during the simulation process. On average, the CPU operated at 75% of its capacity, while the RAM remained at 50% usage. On the other hand, the GPU was at 100% for most of the simulation, although in the final stage, its consumption decreased to approximately 80%.

In the second case, a significantly larger volume of data was generated for processing due to a higher sampling time, considerably impacting system performance. In this scenario, the processor constantly worked at 100% of its capacity, the RAM reached an average usage of 85%, and the GPU remained operating at 100% throughout the simulation time. This behavior shows the increasing computational demands associated with more intensive data processing and highlights the importance of optimizing resources for more complex scenarios.



(**a**) (**b**)

Figure 24. Simulation of the crossing trajectory implemented in Unity. (**a**) Partial robot path of the crossed trajectory. (**b**) Full robot path of the cross trajectory.

5. Discussion

Table 10 shows the trajectory tracking metrics of the mobile robot controlled using a fuzzy PID. The trajectories analyzed include a circular pattern, a point-to-point displacement, and a cross path. Each of these trajectories presents different control challenges, allowing the robustness and adaptability of the controller to be assessed. The metrics considered include the average tracking error, maximum error, steady-state error, settling time, average orientation error in rad, total time, and total distance traveled. These indicators provide a comprehensive view of the system's performance in various scenarios. Also, it is important to note that the average tracking error, the steady-state error, and the average orientation error in rad are measured for times greater than the settling time in order to validate the controller's performance. The maximum error tends to be large because the error exists from the robot's position to the reference at $t = 0$.

For the circular path, the controller demonstrates highly accurate performance. The average tracking error is 0.0089 m, while the maximum error reaches 3.7 m, which is attributed to the robot starting at a point outside the path. This initial deviation highlights the controller's ability to correct significant initial errors and quickly converge toward the desired reference. The steady-state error is only 0.0012 m, demonstrating the system's ability to stabilize near the desired reference once initial perturbations are mitigated. The settling time of 7.246 s reflects the period required for the robot to stabilize within acceptable error margins, demonstrating the controller's robustness under adverse initial conditions. The average orientation error is 0.015 rad, highlighting the effectiveness of the steering angle control in maintaining proper orientation throughout the path. Finally, the total distance traveled is 62.8 m, with a total time of 32.5 s—values that agree with the perimeter expected for a circular path of 10 m radius, thus validating the overall accuracy of the system.

On the point-to-point trajectory, the results show significantly different behavior. The average tracking error is considerably higher, at 0.9531 m, and the maximum error reaches 4.56 m. These deviations are mainly due to the robot's need to correct its initial position outside the trajectory, demonstrating the controller's ability to handle abrupt transitions and correct significant errors. Despite these initial deviations, the steady-state error is low, at 0.001543 m, indicating that the robot is able to arrive with high precision at the endpoint. The settling time is 5.784 s, reflecting how quickly the robot stabilizes its trajectory once the initial errors are corrected. The average orientation error is the largest among the analyzed trajectories, at 0.9743 rad, indicating that the steering angle control faces greater challenges in direct displacements. The total distance traveled is 75.4 m, and the total time taken is 67 s, reflecting the length of the straight path plus the necessary corrections.

The cross-track path exhibits intermediate performance in terms of accuracy and stability. The average tracking error is 0.01814 m, and the maximum error is 8.96 m, which are significant deviations attributable to the complexity of the path and the robot's initial position outside the reference. However, the controller shows robustness by reducing the steady-state error to 0.00232 m, evidencing its ability to correct complex deviations. The

settling time is 9.940 s, indicating the time needed to stabilize after overcoming the abrupt changes in direction that are characteristic of this path. The average orientation error is 0.021 rad, suggesting effective control of the steering angle despite the complexity of the path. The total distance traveled is 58.3 m, and the total time is 40 s, which reflects efficiency in navigating complex paths.

Overall, these findings highlight the effectiveness and reliability of the fuzzy PID controller in tracking various trajectories. Despite starting at points outside the trajectory, the controller is able to correct significant initial errors, reach the desired reference with high accuracy, and maintain stable tracking under diverse conditions. The variation in settling time and maximum errors highlights the system's ability to adapt to control challenges, validating its applicability in dynamic and complex scenarios.

Table 10. Trajectory tracking metrics.

Trajectory	Average Tracking Error (m)	Maximum Error (m)	Steady-State Error (m)	Settling Time (s)	Average δ Error (rad)	Total Time (s)	Total Distance Traveled (m)
Circular	0.0089	3.7	0.0012	7.246	0.015	32.5	62.8
Point-to-point	0.9531	5.784	0.001543	4.564	0.9743	67	75.4
Cross	0.01814	8.96	0.00232	9.940	0.021	40	58.3

A FL-based kinematic controller designed to guide a mobile robot in a virtual environment shares fundamental principles with control systems used in autonomous vehicles. When comparing this technique with others, such as the one presented in [26], where a comparison is made between a traditional PID and an H∞-enhanced PID controller based on robust adaptive learning, the work mainly focuses on accuracy when giving a curve using the given references. In the results of the traditional PID, a turn with a minimum radius of 3 m was obtained, but it was necessary to place more reference points to be able to re-enter the trajectory. When using the H∞ improvement, they obtained precise trajectory control, resulting in a radius of 2.9 m when turning, which improved stability and accuracy. Despite these factors, these controllers present their details. For the traditional PID, the disadvantages were previously discussed: it requires a well-identified plant and performs inaccurately in the presence of non-linearities; in addition, when uncertainties arise, its performance is not better. As for the adaptive controller based on H_∞, although it offers improvements in stability and precision, it entails a more exhaustive mathematical analysis, which increases the computational cost. This greater complexity can represent a limitation in applications where computational resources are restricted or computation time efficiency is critical for its implementation. The work presented in [37] proposed a control scheme based on adaptive dynamic programming combined with a critical neural network. The results demonstrated accurate responses of the controller, achieving reduced errors with RMSE values of 0.21 m in e_x, 0.07 m in e_y, and 0.01 rad in e_δ. However, the authors themselves recognized several disadvantages associated with this methodology. Adaptive dynamic programming is a technique that has still not been explored, which limits the availability of references and complementary studies in the literature. In addition, using a critical neural network significantly increases the computational cost, which represents a challenge for real-time applications. The network training process also requires considerable computational resources, which forces the use of high-performance equipment to carry out this task efficiently. As a final case, the work proposed by [38] presented path-tracking algorithms based on predictive control models, focusing on execution times and tracking accuracy. The three algorithms developed are conflict-based search (CBS), spatiotemporal hybrid A* conflict-based search (STH-CBS), and weighted spatiotemporal hybrid A*

conflict-based search (WSTH-CBS), achieving success rates of 85.71%, 100%, and 100%, respectively, in their best tests. Despite these promising results, an important limitation of this technique lies in the mathematical complexity inherent in predictive controls, which are susceptible to modeling errors. This can affect the robustness of the system in scenarios where the dynamics of the environment or the system itself are not completely defined or present uncertainties.

The ability of FL to handle uncertainty and imprecision, inherent characteristics of real environments, makes it a valuable tool for precise trajectory tracking. Furthermore, this technique is relatively easy for any designer to implement, as it allows for omitting complex mathematical analyses that are indispensable in other methodologies. Unlike learning-based approaches, it does not require training processes for its operation, representing a significant advantage in simplicity and development times. This feature also contributes to reducing computational costs, facilitating its implementation on devices with low processing capacity and making it a viable option for real-time applications or in environments with limited resources. This technology can be applied to various systems in urban mobility, from adaptive cruise control to fully autonomous driving. By enabling smoother and more adaptive decision-making, fuzzy controllers contribute to safer and more efficient driving, especially in complex and dynamic environments.

6. Conclusions

This paper presents a significant advancement in mobile robotics using a kinematic controller based on FL for path following. The fuzzy controller overcomes the limitations of traditional PID controllers, especially in dynamic and uncertain environments, offering greater adaptability and precision. Simulations carried out within the virtual environment developed in Unity demonstrated the effectiveness of the control system, recording an average error of 0.0089 m in circular trajectories, 0.01814 m in cross trajectories, and 0.9531 m in point-to-point trajectories, where the most significant error is associated with the system's difficulties in maintaining precision during sudden changes in direction. However, it manages to correct these deviations with a low steady-state error. A key contribution of this work is the controller's adaptability, which is crucial for autonomous mobile robots navigating dynamic and unpredictable environments. This controller not only emulates the responses of a classic PID but also improves the flexibility and performance of the system, consolidating itself as a robust alternative. In addition, using Unity facilitates real-time visualization and parameter adjustments, reducing physical testing risks and costs and speeding up robotic development. Practical implications include applications such as autonomous vehicles and robotic exploration, where precision and adaptability are crucial. In the future, incorporating multi-sensory feedback and hybrid strategies with machine learning could enhance their capabilities, improving interpretation and response to complex environments.

Author Contributions: Conceptualization, J.G.P.-J. and J.R.G.-M.; methodology, J.G.P.-J., A.M.S. and J.R.G.-M.; software, J.G.P.-J. and J.R.G.-M.; validation, A.M.S., O.A.B.-V. and E.E.C.-M.; formal analysis, J.R.G.-M. and A.M.S.; investigation, J.R.G.-M. and L.F.O.-G.; resources, E.E.C.-M., O.A.B.-V. and M.A.R.-H.; data curation, A.M.S., O.A.B.-V., J.R.G.-M., L.F.O.-G. and M.A.R.-H.; writing—original draft preparation, J.R.G.-M., A.M.S., J.G.P.-J. and L.F.O.-G.; writing—review and editing, O.A.B.-V., A.M.S., E.E.C.-M. and J.R.G.-M.; visualization, J.G.P.-J. and M.A.R.-H.; supervision, J.R.G.-M. and A.M.S.; project administration, J.R.G.-M. and A.M.S.; funding acquisition, J.R.G.-M., L.F.O.-G., A.M.S., E.E.C.-M., O.A.B.-V. and M.A.R.-H. All authors have read and agreed to the published version of the manuscript.

Funding: SECIHTI funded this research under the scholarship 2006985.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: The first author thanks SECIHTI for the scholarship awarded for being a student in the Master's Degree program in Engineering Sciences at the Faculty of Mechanical and Electrical Engineering, Universidad Veracruzana, Poza Rica—Tuxpan Region. Grammarly was used to enhance the paragraphs in this work.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PID	proportional–integral–derivative
FL	fuzzy logic
FLC	fuzzy-logic-based controller
TVLQ	time-varying linear quadratic
LMIs	linear matrix inequalities
ADAR	analytical design of aggregated regulators
SCT	synergetic control theory
ORRL	optimized reward reinforcement learning
DQN	deep Q-network
ROS	robot operating system
LSTM	long short-term memory
UGV	unmanned ground vehicle
BIM	building information modeling
VR	virtual reality
ICC	instantaneous center of curvature
KM	kinematic model
DKM	direct kinematic model
NB	negative-large
NS	negative-small
ZE	zero
PS	positive-small
PL	positive-large
B	big
M	medium
S	small
CBS	conflict-based search
STH-CBS	spatiotemporal hybrid A* conflict-based search
WSTH-CBS	weighted spatiotemporal hybrid A* conflict-based search

References

1. Atchade-Adelomou, P.; Alonso-Linaje, G.; Albo-Canals, J.; Casado-Fauli, D. qRobot: A Quantum Computing Approach in Mobile Robot Order Picking and Batching Problem Solver Optimization. *Algorithms* **2021**, *14*, 194. [CrossRef]
2. Amer, N.H.; Zamzuri, H.; Hudha, K.; Kadir, Z.A. Modelling and Control Strategies in Path Tracking Control for Autonomous Ground Vehicles: A Review of State of the Art and Challenges. *Intell. Robot. Syst.* **2017**, *86*, 225–254. [CrossRef]
3. Zhang, X.; Xie, B.; Yang, Y.; Liu, Y.; Jiang, P. Research on Running Performance Optimization of Four-Wheel-Driving Ackerman Chassis by the Combining Method of Quantitative Experiment with Dynamic Simulation. *Machines* **2024**, *12*, 198. [CrossRef]
4. Cao, Y.; Li, B.; Deng, Z.; Guo, X. Parking Trajectory Planning for Autonomous Vehicles Under Narrow Terminal Constraints. *Electronics* **2024**, *13*, 5041. [CrossRef]
5. Kang, Y.; Xue, B.; Zeng, R. Self-Adaptive Path Tracking Control for Mobile Robots under Slippage Conditions Based on an RBF Neural Network. *Algorithms* **2021**, *14*, 196. [CrossRef]
6. Ju, J.; Zhao, Y.; Zhang, C.; Liu, Y. Vibration Suppression of a Flexible-Joint Robot Based on Parameter Identification and Fuzzy PID Control. *Algorithms* **2018**, *11*, 189. [CrossRef]

7. Grosset, J.; Oukacha, O.; Fougères, A.-J.; Djoko-Kouam, M.; Bonnin, J.-M. Fuzzy Multi-Agent Simulation for Collective Energy Management of Autonomous Industrial Vehicle Fleets. *Algorithms* **2024**, *17*, 484. [CrossRef]
8. Hussein, A.; García, F.; Olaverri-Monreal, C. ROS and Unity Based Framework for Intelligent Vehicles Control and Simulation. In Proceedings of the 2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES), Madrid, Spain, 12–14 September 2018; pp. 1–6.
9. Eisenträger, K.; Haubner, J.; Brade, J.; Einhäuser, W.; Bendixen, A.; Winkler, S.; Klimant, P.; Jahn, G. Evaluating the Effects of Virtual Reality Environment Learning on Subsequent Robot Teleoperation in an Unfamiliar Building. *IEEE Trans. Vis. Comput. Graph.* **2023**, *29*, 2220–2229. [CrossRef] [PubMed]
10. Haces-Garcia, A.; Zhu, W. Building an Accessible and Flexible Multi-User Robotic Simulation Framework with Unity-MATLAB Bridge. *Computers* **2024**, *13*, 282. [CrossRef]
11. Pham, T.T.; Le, M.T.; Nguyen, C.N. Omnidirectional Mobile Robot Trajectory Tracking Control with Diversity of Inputs. *Int. J. Mech. Eng. Robot. Res.* **2021**, *10*, 639–644. [CrossRef]
12. Carmona, R.R.; Sung, H.G.; Kim, Y.S.; Vazquez, H.A. Stable PID Control for Mobile Robots. In Proceedings of the 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), Singapore, 18–21 November 2018; IEEE: New York, NY, USA, 2018; pp. 1891–1896.
13. Shojaei, K.; Shahri, A.M.; Tarakameh, A.; Tabibian, B. Adaptive Trajectory Tracking Control of a Differential Drive Wheeled Mobile Robot. *Robotica* **2011**, *29*, 391–402. [CrossRef]
14. Ćaran, B.; Milić, V.; Švaco, M.; Jerbić, B. Optimal Control-Based Algorithm Design and Application for Trajectory Tracking of a Mobile Robot with Four Independently Steered and Four Independently Actuated Wheels. *Actuators* **2024**, *13*, 279. [CrossRef]
15. Veselov, G.; Sklyarov, A.; Chávez, J.V. Non-linear Control of a Tracked Robot. In Proceedings of the 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 22–25 July 2019; pp. 641–646.
16. Xie, H.; Ma, X.; Qin, Q.; Sun, X. Trajectory Tracking Control for Ackerman Vehicle Based on Improved Reward Function. In Proceedings of the 2024 43rd Chinese Control Conference (CCC), Kunming, China, 26–28 July 2024; pp. 519–524.
17. Ha, T.N.; Tu, T.N.; Dung, N.T.; Mien, T.L.; Thuy, C.T. Deep Q-Network (DQN) Approach for Automatic Vehicles Applied in the Intelligent Transportation System (ITS). In Proceedings of the 2023 International Conference on System Science and Engineering (ICSSE), Ho Chi Minh, Vietnam, 27–28 July 2023; pp. 527–532.
18. Khesrani, S.; Hassam, A.; Boutalbi, O.; Boubezoula, M. Motion Planning and Control of Nonholonomic Mobile Robot Using Flatness and Fuzzy Logic Concepts. *Int. J. Dyn. Control* **2021**, *9*, 1660–1671. [CrossRef]
19. Thuong, T.T.; Ha, V.T.; Truc, L.N. Intelligent Control for Mobile Robots Based on Fuzzy Logic Controller. In Proceedings of the International Conference on Intelligent Systems & Networks, Hanoi, Vietnam, 18–19 March 2023; Springer Nature: Singapore, 2023; pp. 566–573.
20. Mai, T.A.; Dang, T.S.; Duong, D.T.; Le, V.C.; Banerjee, S. A combined backstepping and adaptive fuzzy PID approach for trajectory tracking of autonomous mobile robots. *J. Braz. Soc. Mech. Sci. Eng.* **2021**, *43*, 1–13. [CrossRef]
21. Tolossa, T.D.; Gunasekaran, M.; Halder, K.; Verma, H.K.; Parswal, S.S.; Jorwal, N.; Maria Joseph, F.O.; Hote, Y.V. Trajectory tracking control of a mobile robot using fuzzy logic controller with optimal parameters. *Robotica* **2024**, *42*, 2801–2824. [CrossRef]
22. Zhao, T.; Qin, P.; Zhong, Y. Trajectory Tracking Control Method for Omnidirectional Mobile Robot Based on Self-Organizing Fuzzy Neural Network and Preview Strategy. *Entropy* **2023**, *25*, 248. [CrossRef]
23. Tang, H.H.; Ahmad, N.S. Fuzzy logic approach for controlling uncertain and nonlinear systems: A comprehensive review of applications and advances. *Syst. Sci. Control Eng.* **2024**, *12*, 2394429. [CrossRef]
24. Hentout, A.; Maoudj, A.; Aouache, M. A review of the literature on fuzzy-logic approaches for collision-free path planning of manipulator robots. *Artif. Intell. Rev.* **2023**, *56*, 3369–3444. [CrossRef]
25. Bai, G.; Liu, L.; Meng, Y.; Luo, W.; Gu, Q.; Wang, J. Path Tracking of Wheeled Mobile Robots Based on Dynamic Prediction Model. *IEEE Access* **2019**, *7*, 39690–39701. [CrossRef]
26. Shamshiri, R.R.; Azimi, A.; Behjati, M.; Ghasemzadeh, A.; Dworak, V.; Weltzien, C.; Karydis, K.; Cheein, F.A.A. Online path tracking with an integrated H_∞ robust adaptive controller for a double-Ackermann steering robot for orchard waypoint navigation. In *International Journal of Intelligent Robotics and Applications*; Springer: Berlin/Heidelberg, Germany, 2024; pp. 1–21.
27. Molina-Leal, A.; Gómez-Espinosa, A.; Escobedo Cabello, J.A.; Cuan-Urquizo, E.; Cruz-Ramírez, S.R. Trajectory Planning for a Mobile Robot in a Dynamic Environment Using an LSTM Neural Network. *Appl. Sci.* **2021**, *11*, 10689. [CrossRef]
28. Zhang, Q.; Ma, W.; Zheng, Q.; Zhai, X.; Zhang, W.; Zhang, T.; Wang, S. Path Planning of Mobile Robot in Dynamic Obstacle Avoidance Environment Based on Deep Reinforcement Learning. *IEEE Access* **2024**, *12*, 189136–189152. [CrossRef]
29. Chen, Z.; Chen, K.; Song, C.; Zhang, X.; Cheng, J.C.P.; Li, D. Global Path Planning Based on BIM and Physics Engine for UGVs in Indoor Environments. *Autom. Constr.* **2022**, *139*, 104263. [CrossRef]
30. Liu, Y.; Novotny, G.; Smirnov, N.; Morales-Alvarez, W.; Olaverri-Monreal, C. Mobile Delivery Robots: Mixed Reality-Based Simulation Relying on ROS and Unity 3D. In Proceedings of the 2020 IEEE Intelligent Vehicles Symposium (IV), Las Vegas, NV, USA, 19–21 October 2020; pp. 15–20.

31. Pantusin, F.J.; Carvajal, C.P.; Ortiz, J.S.; Andaluz, V.H. Virtual Teleoperation System for Mobile Manipulator Robots Focused on Object Transport and Manipulation. *Technologies* **2024**, *12*, 146. [CrossRef]
32. Galarza, B.R.; Ayala, P.; Manzano, S.; Garcia, M.V. Virtual Reality Teleoperation System for Mobile Robot Manipulation. *Robotics* **2023**, *12*, 163. [CrossRef]
33. Figueroa-Olmedo, J.R.; Montalvo-Lopez, W.M.; Bayas-Sampedro, M.M. *Cinemática y Dinámica de Robots Móviles con Ruedas*, 1st ed.; CILADI: Guayaquil, Ecuador, 2023; pp. 136–138.
34. Carlucho, I.; De Paula, M.; Acosta, G.G. An Adaptive Deep Reinforcement Learning Approach for MIMO PID Control of Mobile Robots. *ISA Trans.* **2020**, *102*, 280–294. [CrossRef]
35. Saidi, S.M.; Mellah, R.; Fekik, A.; Azar, A.T. Real-Time Fuzzy-PID for Mobile Robot Control and Vision-Based Obstacle Avoidance. *Int. J. Serv. Sci. Manag. Eng. Technol.* **2022**, *13*, 1–32. [CrossRef]
36. Klancar, G.; Zdesar, A.; Blazic, S.; Skrjanc, I. *Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems*; Butterworth-Heinemann: Oxford, UK, 2017.
37. Azimi, A.; Shamshiri, R.R.; Ghasemzadeh, A. Adaptive Dynamic Programming for Robust Path Tracking in an Agricultural Robot Using Critic Neural Networks. *Agric. Eng. EU* **2025**, *80*, 1.
38. Shen, Z.; Du, H.; Yu, L.; Zhu, W.; Zhu, M. A Path Planning and Tracking Control Algorithm for Multi-Autonomous Mobile Robot Systems Based on Spatiotemporal Conflicts and Nonholonomic Constraints. *Actuators* **2024**, *13*, 399. [CrossRef]