

Article

Hierarchical Autonomous Navigation for Differential-Drive Mobile Robots Using Deep Learning, Reinforcement Learning, and Lyapunov-Based Trajectory Control

Ramón Jaramillo-Martínez ^{1,2} , Ernesto Chavero-Navarrete ^{3,*}  and Teodoro Ibarra-Pérez ^{2,*} 

¹ Posgrado CIATEQ A.C., Centro de Tecnología Avanzada, Querétaro 76150, Mexico; rjaramillom@ipn.mx

² Instituto Politécnico Nacional, Unidad Profesional Interdisciplinaria de Ingeniería Campus Zacatecas (UPIIZ), Zacatecas 98160, Mexico

³ CIATEQ A.C., Centro de Tecnología Avanzada, Querétaro 76150, Mexico

* Correspondence: echn_7@hotmail.com (E.C.-N.); tibarrap@ipn.mx (T.I.-P.)

Abstract

Autonomous navigation in mobile robots operating in dynamic and partially known environments demands the coordinated integration of perception, decision-making, and control while ensuring stability, safety, and energy efficiency. This paper presents an integrated navigation framework for differential-drive mobile robots that combines deep learning-based visual perception, reinforcement learning (RL) for high-level decision-making, and a Lyapunov-based trajectory reference generator for low-level motion execution. A convolutional neural network processes RGB-D images to classify obstacle configurations in real time, enabling navigation without prior map information. Based on this perception layer, an RL policy generates adaptive navigation subgoals in response to environmental changes. To ensure stable motion execution, a Lyapunov-based control strategy is formulated at the kinematic level to generate smooth velocity references, which are subsequently tracked by embedded PID controllers, explicitly decoupling learning-based decision-making from stability-critical control tasks. The local stability of the trajectory-tracking error is analyzed using a quadratic Lyapunov candidate function, ensuring asymptotic convergence under ideal kinematic assumptions. Experimental results demonstrate that while higher control gains provide faster convergence in simulation, an intermediate gain value ($K = 0.5I$) achieves a favorable trade-off between responsiveness and robustness in real-world conditions, mitigating oscillations caused by actuator dynamics, delays, and sensor noise. Validation across multiple navigation scenarios shows average tracking errors below 1.2 cm, obstacle detection accuracies above 95% for human obstacles, and a significant reduction in energy consumption compared to classical A* planners, highlighting the effectiveness of integrating learning-based navigation with analytically grounded control.



Academic Editor: Manoj Gupta

Received: 29 December 2025

Revised: 27 January 2026

Accepted: 13 February 2026

Published: 17 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons](#)

[Attribution \(CC BY\)](#) license.

Keywords: reinforcement learning; convolutional neural networks; Lyapunov candidate function

1. Introduction

Traditionally, trajectory planning has relied on geometrical algorithms and classic control techniques. However, the increasing complexity of the operational environment has driven the integration of computer vision and artificial intelligence, marking a significant evolution in autonomous navigation.

Recent literature shows a clear trend toward the hybridization of techniques to address specific challenges. For instance, robotics manipulation and logistics, predictive control, and reinforcement learning (RL) have facilitated the execution of complex motions [1]. Likewise, the integration of ant colony algorithms with RL has optimized navigation in container terminals [2]. In parallel, in the agriculture domain, advances in perception have been achieved through robust segmentation models capable of handling illumination variability [3], while sensor fusion combined with convolutional neural networks (CNNs) has enabled trajectory tracking at low computational cost [4]. Moreover, models that combine RL with fuzzy logic have demonstrated improvements in travel time in real environments [5]. Finally, recent multi-objective approaches have significantly increased the success rate without compromising velocity [6].

Regarding trajectory planning, optimized classic algorithms such as FHQ-RRT* have reduced computational time by approximately 40% to 77%. Similarly, the combination of global planning algorithms with the dynamic window approach (DWA) has enabled a balance between global planning and dynamic obstacle avoidance, improving efficiency and safety in dynamic environments [7]. Nevertheless, these classic methods present important limitations in unstructured environments, where the absence of semantic information restricts their ability to adapt smoothly to external disturbances.

To mitigate these limitations, deep learning (DL)-based algorithms such as OctoPath [8] and RRT-GPMP2 [9] have improved generalization capabilities and smoothness of generated trajectories. More recently, deep reinforcement learning (DRL) algorithms, including DQN, DDQN [10] and PL-TD3 [11], have endowed agents with greater autonomy to learn complex navigation policies. However, a recurring weakness of systems based exclusively on reinforcement learning (RL) is their “black box” nature, as they lack formal mathematical guarantees of stability, which may compromise system safety in critical situations.

The RL–Lyapunov paradigm [12] has emerged as an alternative to integrate stability theory into learning-based environments. However, most approaches reported in the literature address perception, decision-making, and control as decoupled modules or embed them implicitly within learned policies, without a formal integration that simultaneously guarantees semantic perception, adaptability, and mathematical stability of the system. In particular, a methodological gap persists in autonomous navigation for mobile robots, where deep learning-based methods incorporate advanced visual perception but lack explicit stability guarantees, while control schemes with proven stability do not integrate semantic information from visual sensors in unstructured environments.

To address this gap, this work proposes an integrated methodology for the autonomous navigation of a differential-drive mobile robot that combines deep learning-based visual perception, reinforcement learning-based decision-making, and an analytical control scheme with guaranteed local asymptotic stability through an explicitly designed Lyapunov candidate function. Unlike purely learning-based approaches, system stability is not left implicit within the learned policy but is formally guaranteed through an analytical control law, while semantic perception is employed for efficient obstacle detection and avoidance.

The project is structured into five interconnected stages: data acquisition, deep learning architecture design, system modeling and simulation, efficient trajectory generation, and experimental validation. For navigation, an RL-based algorithm driven by an occupancy map generated from an RGB-D camera is implemented. Unlike previous methods, this system incorporates an obstacle avoidance module based on CNNs for semantic environment classification, coupled with a trajectory control strategy grounded in a Lyapunov candidate function. This combination enables not only efficient and robust navigation in

the presence of obstacles but also a formal guarantee of system stability, a feature that is typically absent in purely learning-based approaches.

The remainder of this paper is organized as follows. Section 2 describes the materials and methods employed, including data acquisition, neural network architecture design, the robot's mathematical model, and the experimental environment. Section 3 presents the results obtained from both simulation and real-world experimentation, analyzing obstacle detection performance, travel-time minimization, trajectory control, and energy consumption. Section 4 discusses the results and compares the proposed approach with classical algorithms. Finally, the last section presents the conclusions and future work.

2. Materials and Methods

The development of this project is structured into five methodologically interconnected stages, as shown in Figure 1. The workflow progresses from the acquisition of sensorial information to physical validation, integrating artificial vision, deep learning, and control theory under a clearly defined hierarchical scheme.

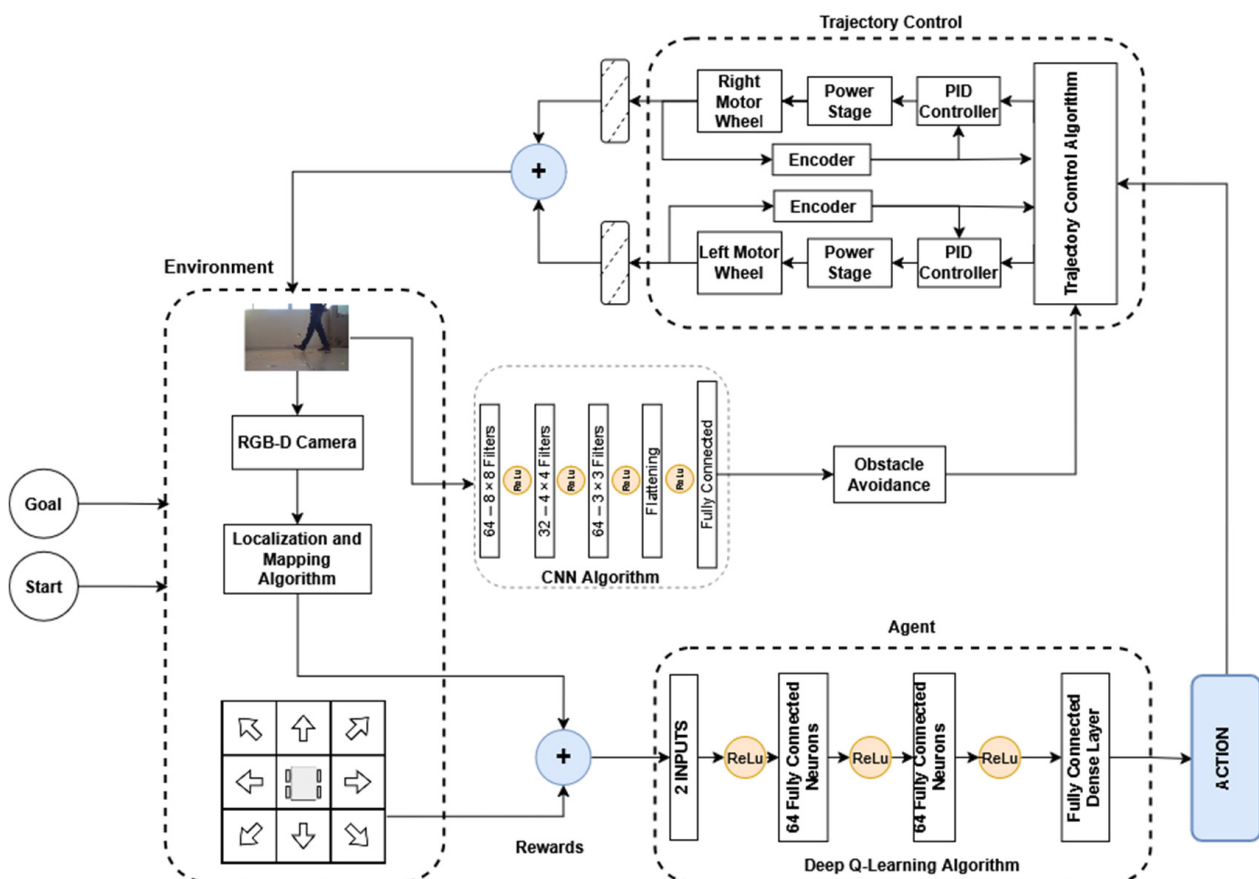


Figure 1. Methodological architecture for autonomous navigation based on deep learning and reinforcement learning.

The process begins with the collection and processing of training data from the perception models. Subsequently, DL architectures are designed and optimized, followed by the modeling and simulation stages, which allow the navigational logic to be validated before proceeding to the physical implementation stage. Trajectory generation focuses on energy efficiency, and finally, experimental validation is carried out in a controlled environment. In this context, energy efficiency is associated with smooth trajectories and with the reduction in the control effort required to track the robot's motion.

The methodological architecture operates under a closed-loop scheme: the robot perceives its environment using an RGB-D camera, processes the information using a CNN for object detection, and employs an RL algorithm to decide on the optimal navigation actions. Within this scheme, the CNN is exclusively responsible for the semantic perception of the environment, while the RL algorithm operates at the decision-making level, selecting navigation actions or references based on the perceived state. The control law governing movement is based on the Lyapunov candidate function, which ensures stability and energy efficiency. This function is not learned but analytically designed, allowing local asymptotic stability of the system to be guaranteed.

The trajectory control block, composed of PID algorithms and sensory feedback via encoders, transforms the planned trajectory into angular velocity signals for the wheels of the differential robot. PID controllers are employed at the actuation level due to their robustness, simplicity of implementation, and suitability for real-time applications, acting as a low-level loop that stably executes the references generated by the higher-level modules.

2.1. Collection and Training Data Preprocessing

To train the artificial neural network, a dataset consisting of 4200 images was built using an RGB-D Astra Pro-Plus (ORBEC) camera from the manufacturer Orbbec Inc. in Shenzhen, China. The RGB images were acquired in a real environment under controlled illumination conditions using a native resolution of 640×480 pixels at 30 fps and were subsequently resized for processing, ensuring that the model learned representative characteristics of the physical operational environment. The camera employs structured light technology (850 nm) and provides an RGB field of view of $66.1^\circ \times 40.2^\circ$, which allows consistent visual information to be captured at different scales and distances.

The dataset was structured into three balanced classes (1400 images per category): left obstacle, right obstacle, and no obstacle. These categories were selected to directly map visual perception to the robot's discrete navigation decisions. During data acquisition, the considered obstacles mainly included humans and rigid signaling objects (cones), which are representative of real operational scenarios. The distance between the robot and the obstacles ranged from 0.2 m to 7 m, where the lower limit corresponds to the minimum safety distance and the upper limit corresponds to the maximum observable distance within the experimental area.

The sample size was determined experimentally to achieve adequate generalization while avoiding overfitting. This process was carried out through successive training sessions of the neural network using subsets of increasing size, evaluating accuracy, loss, and F1-score metrics for both training and validation sets. The final dataset size was established when further increases in the number of samples did not produce significant improvements in model performance.

All images were resized to 224×224 pixels, and the intensity values were adjusted to facilitate gradient convergence. Specifically, the intensity values were normalized to the $[0, 1]$ interval. Additionally, data augmentation techniques were applied, including shearing and perspective transformations (10%), color jittering (10%), and the addition of Gaussian noise (10%). Rotation, mirroring, and zoom transformations were not considered, as they alter the spatial location of obstacles within the image and directly affect position and distance estimation.

For dataset construction, a standard split was adopted: 80% for training, 10% for validation, and 10% for testing.

Model performance was evaluated using the following metrics: accuracy, true positive rate (recall or sensitivity), false positive rate (FPR), and precision. Furthermore, classifier performance was analyzed using class-wise confusion matrices in order to identify specific

misclassification patterns among the defined categories. These metrics allow the evaluation not only of classification accuracy but also of operational safety by minimizing false negatives in obstacle detection.

The performance metrics used to evaluate the classification results are formally defined as follows.

- Accuracy represents the proportion of correctly classified instances, including both positive and negative cases. It is mathematically defined as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

- Recall, also referred to as sensitivity, represents the proportion of actual positive instances that are correctly identified by the classifier:

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

- The false positive rate, also known as the false alarm probability, indicates the proportion of actual negative instances that are incorrectly classified as positive:

$$FPR = \frac{FP}{FP + TN} \quad (3)$$

- Precision measures the proportion of instances predicted as positive that truly correspond to positive cases:

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

These metrics allow the evaluation not only of the classifier's overall performance but also of its operational safety, particularly by minimizing false negatives in obstacle detection.

2.2. Deep Learning Architecture for Obstacle Detection

Due to the limitations of the embedded hardware platform, a custom convolutional neural network (CNN) architecture was designed instead of relying on large pretrained models. This decision was motivated by the need to ensure real-time inference with low computational and energy consumption, which is critical for autonomous mobile robotic platforms. While widely used architectures such as VGG19 contain approximately 138 million parameters, the proposed architecture comprises around 13 million parameters, achieving a suitable balance between representational capacity and computational efficiency. The proposed model is illustrated in Figure 2.

The network processes input images of size $224 \times 224 \times 3$ and is composed of four sequential convolutional blocks with 32, 64, 128, and 256 filters. This progressive configuration allows the gradual extraction of increasingly complex features while maintaining stability during training. Each block employs ReLU activation functions followed by batch normalization and MaxPooling2D layers for spatial downsampling. The use of batch normalization contributes to stabilizing the activation distributions and accelerating convergence during training.

After feature extraction, the resulting feature maps are flattened and passed to a multilayer perceptron with L2 regularization to mitigate overfitting. The progressive reduction in the number of neurons in the dense layers (1024–512–256) enables a gradual decrease in feature dimensionality, improving model stability and reducing the risk of

overfitting. The output layer uses a softmax activation function to classify the three possible environmental conditions.

The architecture is detailed as follows:

1. Input layer (224, 224, 3): The model receives RGB images of 224×224 pixels.
2. Convolutional and normalization layers:
 - Conv2D(32, 3×3) with ReLU activation, followed by normalization and Max-Pooling2D(2,2).
 - Conv2D(64, 3×3) with ReLU activation, followed by normalization and Max-Pooling2D(2,2).
 - Conv2D(128, 3×3) with ReLU activation, followed by normalization and Max-Pooling2D(2,2).
 - Conv2D(256, 3×3) with ReLU activation, followed by normalization and Max-Pooling2D(2,2).
3. Flatten layer: Converts multidimensional feature maps into a one-dimensional vector.
4. Fully connected layers:
 - Dense layer with 1024 neurons and L2 regularization (0.001).
 - Dense layer with 512 neurons and L2 regularization (0.001).
 - Dense layer with 256 neurons and L2 regularization (0.001).
5. Output layer: Dense layer with 3 neurons and softmax activation corresponding to each class.

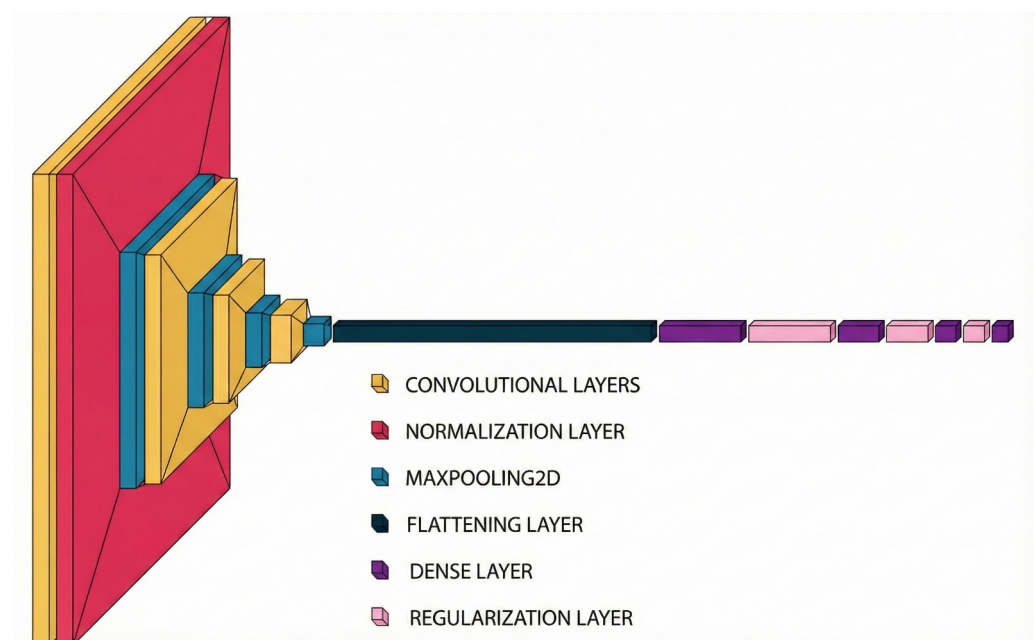


Figure 2. Architecture of a convolutional neural network proposed for obstacle detection.

Categorical cross-entropy was used as the loss function during training. An early stopping criterion with a patience of 10 epochs was applied as a regularization mechanism to prevent overfitting. Given the compact nature of the CNN architecture, convergence typically occurred at early stages of training; therefore, the maximum number of training epochs was limited to 15 as an upper bound. In practice, stable convergence was consistently observed within the range of 5 to 15 epochs.

The training process was implemented in Python 3.11.7 using an Intel Core i5-10300 processor (4 cores, up to 2.50 GHz), an NVIDIA GeForce GTX 1650 GPU, 16 GB of RAM, and Windows 11. The optimizer selected for training was Adam [13], with a learning rate of

1×10^{-3} and a batch size of 32. The hyperparameters used during training are summarized in Table 1.

Table 1. Hyperparameters of the proposed architecture.

Hyperparameters	Value
Maximum epochs	15
Early stopping (patience)	10
Learning rate	1×10^{-3}
Batch size	32
Optimizer	Adam

The proposed architecture was compared against representative CNN models commonly reported in the literature. Table 2 summarizes the main characteristics of each architecture in terms of depth, number of parameters, and model size.

Table 2. Comparison of classic CNN architectures and the proposed architecture, considering depth, size, and number of parameters.

Architecture	Developed by	Depth (Layers)	Size (MB)	Number of Parameters
AlexNet	[13]	8	240	60 million
VGG19	[14]	19	550	138 million
SqueezeNet	[15]	18	5	1.2 million
GoogleNet	[16]	22	50	4 million
Proposed CNN	-	8	53	13 million

This comparison is intended to provide a qualitative assessment of the static computational complexity and memory requirements of each architecture, which are critical factors for deployment on embedded robotic platforms. While larger pretrained models offer high representational capacity, their parameter count and memory footprint often limit real-time implementation. In contrast, the proposed CNN aims to balance architectural simplicity and expressive capability, targeting efficient inference under constrained hardware resources. Model size is reported as a comparative indicator of memory demand and corresponds to the stored weight footprint under a standard floating-point representation.

2.3. Minimization of Travel Time

It is possible to optimize the robot's trajectory to minimize travel time by considering direction changes induced by the presence of obstacles. This problem is analyzed based on the scenario illustrated in Figure 3, where the robot must modify its original path to avoid a collision.

The diagram represents a scenario in which a four-wheeled differential mobile robot moves from point D to point A while avoiding an obstacle located along its forward path. The trajectory is divided into two main segments: the segment $\overline{DC}$, traveled at velocity v_1 , and the segment $\overline{CA}$, traveled at velocity v_2 . The velocities v_1 and v_2 may differ due to the robot's dynamics during the avoidance maneuver since the change in direction involves deceleration and control adjustments that reduce the effective velocity in the second segment. For this analysis, these velocities are assumed to be constant within each segment.

The parameter x corresponds to the horizontal distance between the robot's central axis and the turning point C, while b represents the vertical distance between the obstacle and point C. The objective of this analysis is to determine the optimal value of x , independently of the specific values of v_1 and v_2 , that minimizes the total travel time and defines the point at which the robot should initiate the turning maneuver to avoid the obstacle.

The total travel time is expressed as

$$f(x) = \frac{a-x}{v_1} + \frac{\sqrt{x^2+b^2}}{v_2} \quad (5)$$

where a corresponds to the total distance between point D and point C when the turn is performed at the farthest admissible position relative to the obstacle.

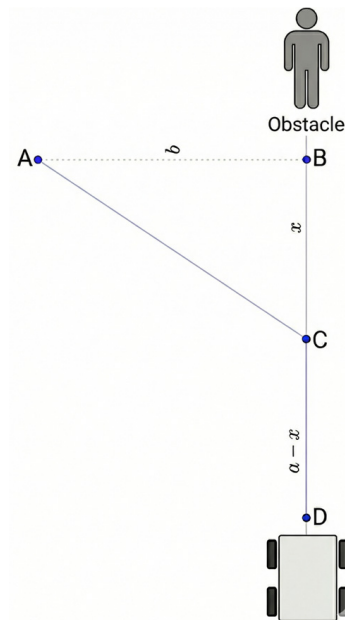


Figure 3. Geometrical scenario for minimization of travel time.

The search domain for the optimal turning point is defined as $x \in [0, a]$, where $x = 0$ represents a limiting case with zero clearance between the robot and the obstacle, and $x = a$ corresponds to the farthest possible point at which the turn can be initiated. Due to the structure of the function $f(x)$, it exhibits a global minimum within this interval. An analysis of the first and second derivatives shows that, for the parameters considered, the minimum travel time is attained at the boundary $x = a$.

Consequently, this result indicates that, regardless of the relative magnitudes of v_1 and v_2 , the total travel time is minimized when the robot initiates the turn at the farthest permissible point from the obstacle, thereby maximizing the safety distance prior to the maneuver. This optimal value can be computed directly once the geometric parameters of the environment are known.

In terms of integration with the navigation system, the RL algorithm is responsible for determining the next target point within the occupancy map, taking into account obstacle locations and the final goal. From this target point, the Lyapunov-based trajectory controller guides the robot from its current position to the new point generated by the RL algorithm, ensuring stability and convergence during motion.

Finally, the following decision cases are considered:

- Centered obstacle: when left and right avoidance trajectories present equivalent distances and energy consumption, the decision is made pseudo-randomly. Since both alternatives involve the same energetic and temporal cost, either option is equally valid from an efficiency standpoint.
- Obstacle detected at a distance smaller than x : the robot stops immediately, prioritizing safety over travel time optimization.

2.4. Trajectory Control Based on the Lyapunov Candidate Function

To analyze the stability of the trajectory-tracking process, a Lyapunov-based approach is adopted at the kinematic level of the control architecture. The objective is not to directly control the actuator dynamics but to ensure the convergence of the position tracking error toward zero by generating stable velocity references for the low-level controllers.

Let $r_e \in R^2$ denote the position tracking error between the robot's current pose and the desired reference point generated by the navigation module. A quadratic function of the tracking error is adopted as a Lyapunov candidate function:

$$V(r_e) = \frac{r_e^T r_e}{2} \quad (6)$$

The function $V(r_e)$ is locally positive in domain D , such that $V(0) = 0$ and $V(r_e) > 0$ for all $r_e \in D$. The domain D is defined as a region of the state space in which the Lyapunov candidate function is continuously differentiable, positive definite, and whose derivative along the system trajectories is negative definite. Mathematically, this domain is expressed as $D \subset R^n$. Moreover, if $\dot{V}(r_e) \leq 0$, then the system is asymptotically stable.

Under standard control-oriented assumptions, the error dynamics are shaped as

$$\dot{r}_e = -K \cdot r_e \quad (7)$$

where $K \in R^{2 \times 2}$ is a symmetric positive definite gain matrix. Substituting (7) into the time derivative of the Lyapunov candidate function yields

$$\dot{V}(r_e) = -r_e^T \cdot K \cdot r_e \leq 0 \quad (8)$$

which guarantees local asymptotic stability of the tracking-error dynamics in a neighborhood of the reference trajectory. This result ensures that the error converges exponentially to zero under ideal kinematic conditions.

Based on this formulation, the following reference velocity law is obtained:

$$\dot{q}[k] = J(q[k])^{-1} \cdot K \cdot r_e[k] \quad (9)$$

where $J(q)$ denotes the Jacobian matrix of the differential-drive robot, and $\dot{q}$ represents the desired wheel angular velocities. It is important to emphasize that this control law operates at the trajectory-generation level, producing velocity references rather than directly commanding the actuators.

The physical execution of these references is handled by classical PID controllers at the low level, which compensate for actuator dynamics, delays, and unmodeled effects. As a result, the Lyapunov-based formulation guarantees stability of the reference tracking process, while robustness against real-world disturbances is provided by the embedded PID loops.

At the simulation level, where actuator dynamics and measurement noise are neglected, gain values close to $K = I$ yield the fastest convergence and lowest tracking error. However, experimental results revealed that such aggressive gains may induce oscillations due to inertia, delays, and sensor noise. Consequently, an intermediate gain value of $K = 0.5I$ (where I is the identity matrix) was selected for experimental implementation, representing a practical trade-off between convergence speed and robustness.

The pseudocode of Algorithm 1 of the controller is summarized as follows:

Algorithm 1 Pseudocode of the Lyapunov-based controller

```

1: Initialize
2:   Samples, robot parameters, desired trajectory.
3: Controller Calculation
4:   Loop:
5:     Error calculation.
6:     Jacobian matrix J(q(t)).
7:     Control parameters.
8:     Control law  $\dot{q}[k] = J(q[k])^{-1} \cdot K \cdot r_e[k]$ 
9:     Control actions.
10:   End Loop
11: Visualization

```

2.5. Kinematic Modeling of a Software-Constrained Differential-Drive Platform

The robotic system used in this study is configured with four omnidirectional wheels. Although the hardware inherently supports holonomic motion, a software-defined constraint is implemented to emulate a differential-drive locomotion model. This approach allows leveraging the stability provided by the four contact points.

As shown in Figure 4, the robot's state in the global reference frame is defined by its coordinates (x_1, y_1) and its orientation angle θ . To ensure the mathematical consistency of the model for both simulation and physical implementation, the following idealized conditions are assumed: rigid body dynamics, software-imposed non-holonomic constraint, and virtual axis synchronization.

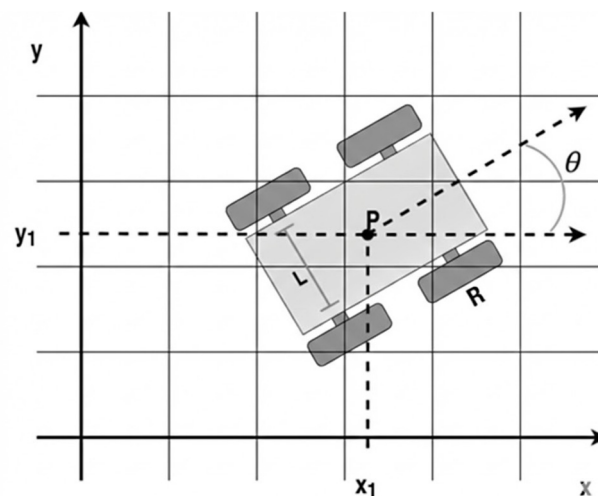


Figure 4. Kinematic model and coordinate system of the software-constrained four-wheeled robot.

Under these constraints, the mathematical representation is defined by the standard kinematic equations of a differential-drive robot [17], where v is the resulting linear velocity and w is the angular velocity:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \quad (10)$$

In this restricted model, the software assigns differential speeds to the four motors individually. The front and rear wheels on the same side receive identical speed commands. Let v_l and v_r be the target velocities for the left and right sides, respectively.

$$v_l = \frac{1}{R}(v - wL/2) \quad (11)$$

$$v_r = \frac{1}{R}(v + wL/2) \quad (12)$$

where R is the wheel radius and L is the lateral distance between wheels.

2.6. Testing Environment

The experimental tests were conducted at the Unidad Profesional Interdisciplinaria de Ingeniería Campus Zacatecas, within a 4.5×7 m area delimited by perimeter obstacles. Multiple navigation scenarios were evaluated, in which the spatial distribution of obstacles varied between trials, in order to analyze system performance under different environmental configurations. The testing environment is shown in Figure 5.

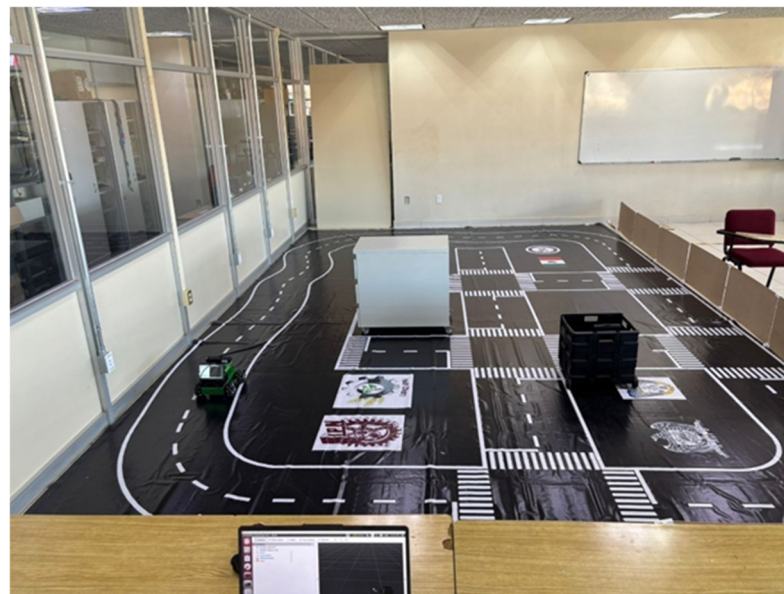


Figure 5. Testing environment used for experimental validation.

The experimental validation was conducted using an omnidirectional four-wheel Mecanum robot. However, to maintain non-holonomic requirements, the robot's movements were restricted. This obligates the robot to work with a differential traction configuration. The kinematic restriction defined by the software does not allow lateral movement.

The obstacles considered during the experimental tests included humans and commercial traffic cones. The cones used have approximate dimensions of 26 cm in width, 26 cm in length, and 45 cm in height. Both people and cones were placed at different positions within the working area, at distances ranging from 0.5 m to 7 m with respect to the robot, where 7 m corresponds to the maximum distance within the experimental area. In the case of human obstacles, they were considered as obstacles whenever they were within the effective detection range of the RGB-D camera.

The evaluation was carried out using the Robot Operating System (ROS) framework together with the Gazebo simulator [18] in order to compare the results obtained in the simulation with the physical prototype. The navigation environment was modeled using a cellular discretization approach, whose characteristics are described below:

- Each cell has a size of 50 cm $\times$ 50 cm, selected considering the physical dimensions of the robot, as this value corresponds approximately to twice its width, thereby ensuring safe motion between adjacent cells.

- This cell size also accounts for the minimum reliable operating distance of the RGB-D camera (0.4 m) and its approximate accuracy of ± 3 mm per meter, ensuring consistent obstacle detection.
- The map coordinate system was defined with the origin (0,0) located at the upper-left corner, allowing a straightforward matrix-based representation of the environment.
- The selected cell width enables the workspace to be represented by an integer number of cells, simplifying trajectory planning.

The robot and the goal are located at the center of the corresponding cell. This choice is a direct consequence of the cellular discretization approach: when an obstacle is detected within a cell, the entire cell is considered occupied. Similarly, placing the robot and the goal at the center of a cell ensures a consistent representation of free and occupied space, regardless of whether an obstacle partially or fully occupies a given cell.

During navigation, the robot captures an RGB-D image in each cell. The acquired image is processed through a thresholding procedure to determine whether the area is free or occupied. The resulting cell occupancy information is fed back to the reinforcement learning (RL) algorithm, which estimates the optimal trajectory based on the current configuration of the environment.

The characteristics of the robot used in the experiments are as follows:

- Dimensions: 260 mm width $\times$ 324 mm length.
- Wheels: 44 mm width $\times$ 97 mm diameter.
- Weight: 3 kg.
- Operating system: Ubuntu 18.04 LTS with ROS Melodic.
- Camera: Orbbec Astra Pro-Plus.
- Field of view: Horizontal $58.4^\circ \times$ Vertical 45.7° .
- Depth sensor accuracy: ± 3 mm at 1 m.
- Nominal speed: 85 RPM.

The control and navigation system was developed in Python 2.7 and C++, with support from the ROS Melodic environment. Simulation and visualization of the results were performed using Gazebo and RViz. The software architecture is organized into four main modules: perception, control, reinforcement learning, and mapping.

- Perception:
 - a. Node of image acquisition of RGB-D images.
 - b. Node of image processing.
 - c. Node of obstacle classification.
 - d. Node of distance and location estimation.
- Control:
 - a. Node of odometry.
 - b. Node of ROS control, in charge of the communication with the microcontroller.
 - c. Node of controlled trajectory.
- Reinforcement learning algorithm:
 - a. Node of decision-making.
 - b. Node of rewards estimation.
 - c. Node of optimal policies calculation.
- Mapping:
 - a. Occupancy map generation node.
 - b. Obstacle avoidance node.

Energy consumption was measured directly from the main battery supplying all system components using an ACS711 integrated circuit from the manufacturer Allegro MicroSystems in New Hampshire, United States together with an ESP32-C3 module from the manufacturer Seeed Studio in Shenzhen, China. The measurement system operates at 3.3 V with a sampling period of 10 ms. Data transmission is performed every 10 ms, resulting in an approximate current consumption of 120 mA (0.40 W), while the ACS711 integrated circuit consumes approximately 4 mA at 3.3 V (13.2 mW). The total power consumption of the measurement system is therefore approximately 0.413 W, and the current sensor accuracy is $\pm 5\%$.

The diagram of the circuit is shown in Figure 6.

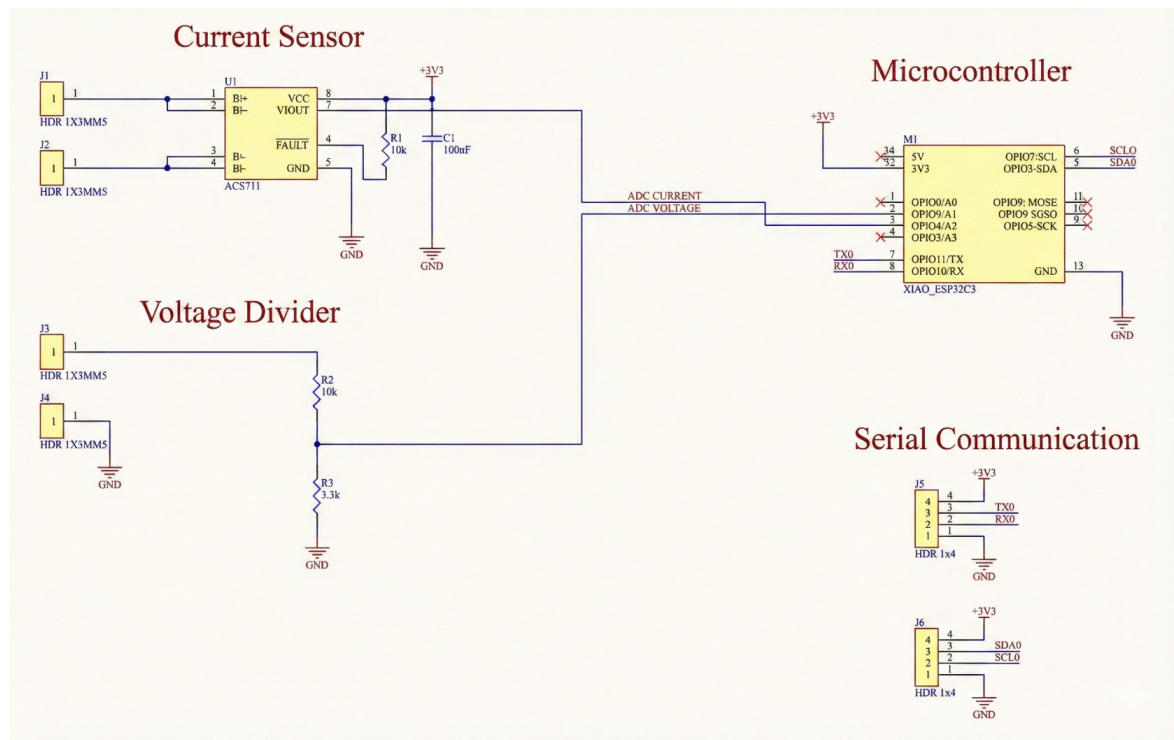


Figure 6. Current measurement circuit with ACS711 circuit.

Figure 7a,b, show the design of the electronic PCB and the physical module.

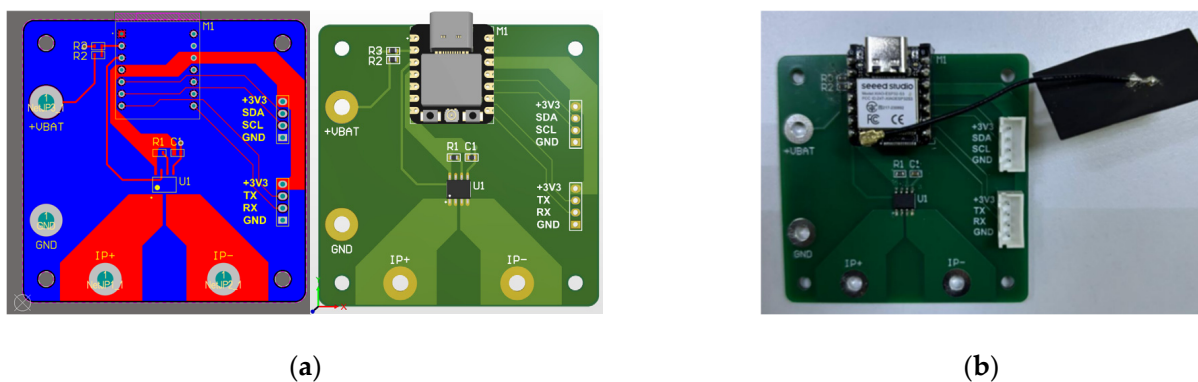


Figure 7. Module for current and voltage measurement (a) PCB design of the module; (b) Physical PCB module.

3. Results

In this section, a quantitative and qualitative analysis of the proposed system is presented. The results are organized into perception subsystems, validation of mathematical models, control, and energy efficiency.

3.1. Obstacle Detection

The performance of the proposed CNN architecture was evaluated and compared against several classical models using the same dataset described in the methodology section, with the aim of analyzing both classification capability and the trade-off between performance and computational cost.

Training was conducted in Python 3.11.7 using an Intel Core i5-10300 processor (4 cores, up to 2.50 GHz), an NVIDIA GeForce GTX 1650 GPU, 16 GB of RAM, and Windows 11. A batch size of 32 was used to ensure comparable training conditions across architectures. An early stopping criterion with a patience of 10 epochs was applied to prevent overfitting.

Table 3 presents a comparison of the performance results obtained for the evaluated CNN architectures.

Table 3. Comparison of performance results of the evaluated CNN architectures.

Metric	AlexNet	VGG19	SqueezeNet	GoogleNet	Proposal CNN
Accuracy	0.9620	0.9705	0.9620	0.9708	0.9261
Precision	0.9460	0.9550	0.9390	0.9720	0.9515
Sensitivity (recall)	0.9755	0.9775	0.9630	0.9610	0.9700
Specificity	0.9885	0.9905	0.9880	0.9895	0.9890
Training time (min)	37.3	168.76	40.19	65.48	55.2

Figure 8 graphically illustrates the comparative performance of the different architectures. GoogleNet achieves the highest precision and F1-score values, while VGG19 exhibits strong accuracy and sensitivity at the expense of significantly longer training times. SqueezeNet, on the other hand, shows acceptable performance with reduced training time and lower computational cost.

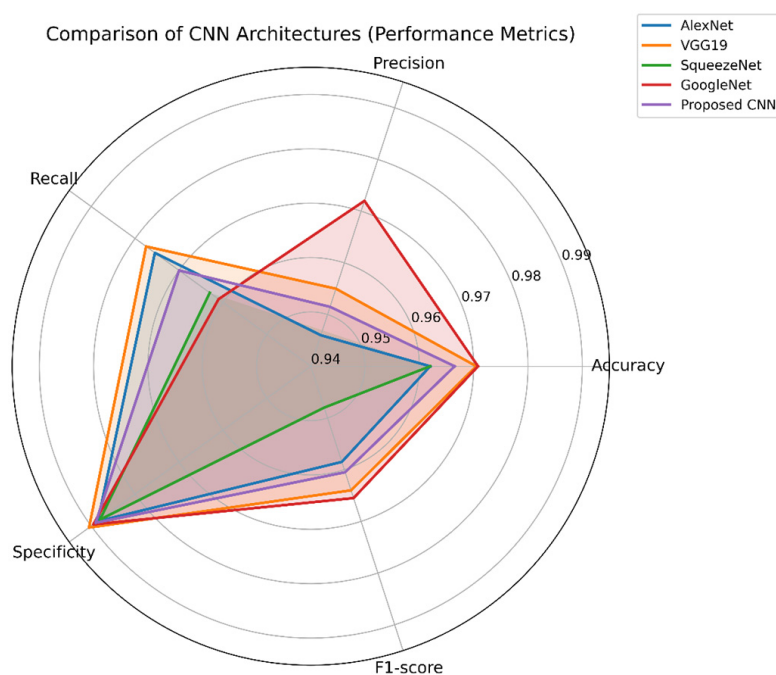


Figure 8. Comparison of the performance metrics of the evaluated CNN architectures.

The proposed CNN is positioned as an intermediate solution: although it does not reach the peak accuracy of deeper architectures, it achieves competitive sensitivity and specificity values while maintaining a moderate training time. This relationship supports the claim that the proposed architecture offers a reasonable balance between classification performance and computational cost.

It is worth noting that image acquisition was performed at a rate of 30 fps using the RGB-D camera. However, the images used for training were manually selected and labeled; therefore, inference FPS are not reported at this stage. Nevertheless, training time serves as an indirect indicator of the computational complexity associated with each architecture.

For error analysis, class-wise confusion matrices were evaluated. The proposed model exhibited greater difficulty in distinguishing between the “left obstacle” and “right obstacle” classes, particularly under partial occlusion conditions. In such cases, the overall accuracy was reduced to 92.61%. However, the high sensitivity achieved (0.9700) indicates that the system is highly reliable for obstacle detection, prioritizing robotic safety over classification ambiguity. The confusion matrix for the three proposed classes is shown in Figure 9.

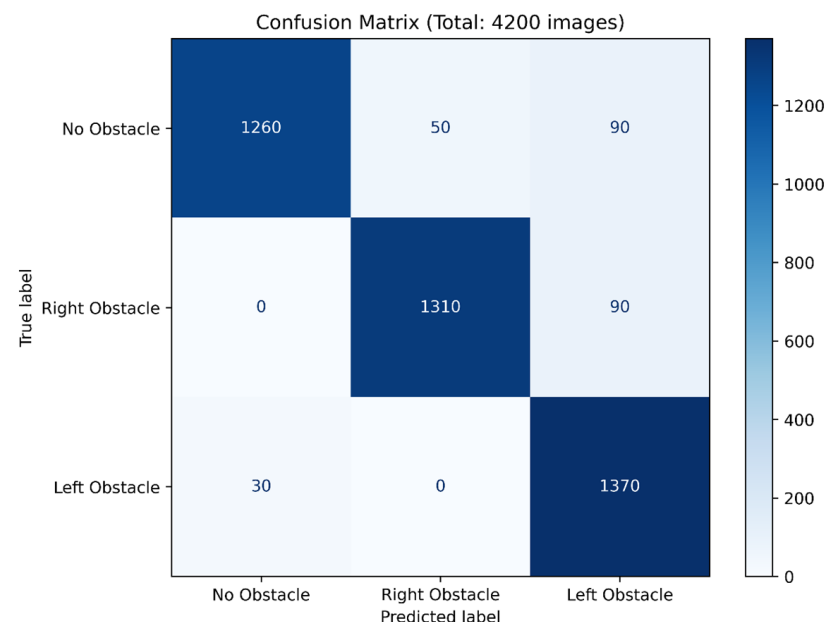


Figure 9. Confusion matrix.

3.2. Validation of Travel Time

This section analyzes the validity and consistency of the proposed travel-time minimization model. It is important to emphasize that the objective is not to validate the minimum travel time against an external ideal trajectory or a different planning algorithm, but rather to evaluate the internal behavior of the proposed formulation and its dependence on the turning point x under different velocity and geometric configurations.

The total travel time is expressed as a function $f(x)$, where x represents the lateral distance from the robot’s central axis to the turning point. The parameters a and b define the geometric relationship between the robot, the obstacle, and the target, while v_1 and v_2 correspond to the robot velocities before and after the turning maneuver, respectively. Although different velocity values are considered, the analysis focuses on identifying the optimal turning point that minimizes the total traversal time.

A parametric analysis was conducted by varying the distances a and b with a granularity of 0.2 m, equivalent to the robot width, and by considering different combinations of v_1 and v_2 . The key variable in this analysis is the turning point x . When the condition $x = a$ is satisfied, the total travel time reaches its minimum value, and the optimal travel time

coincides with the minimum travel time, resulting in a zero relative error. For values of x different from a , the total travel time increases, indicating a loss of optimality.

Figure 10 illustrates the behavior of the total travel time $f(x)$ as a function of the turning point x for a representative experimental configuration with $a = 1$ m, $b = 1$ m, $v_1 = 1.7$ m/s, and $v_2 = 0.34$ m/s. The curve clearly exhibits a global minimum at $x = a$, highlighted by a dashed vertical line. This result confirms that, regardless of the specific velocity values, the optimal turning point is always located at the boundary of the domain when $x = a$. Any deviation from this condition leads to a monotonic increase in travel time.

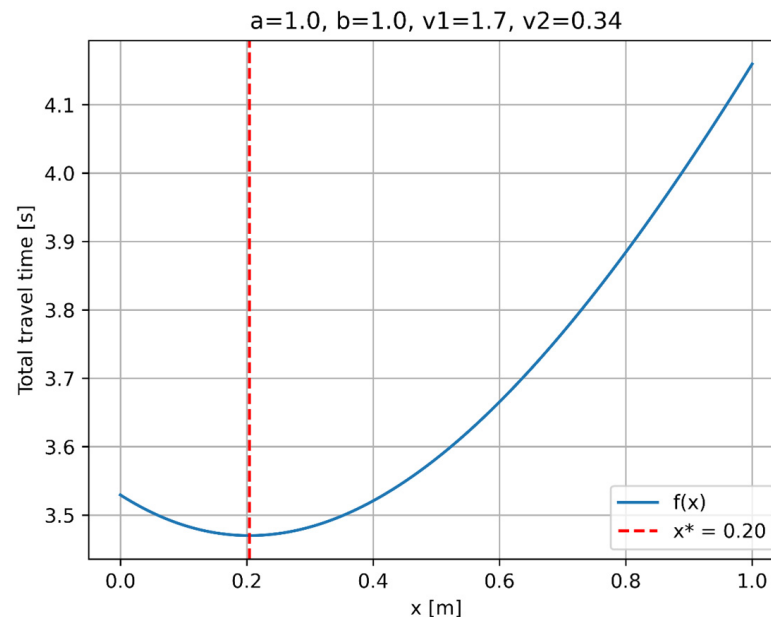


Figure 10. Total travel time according to the distance of the obstacle for the parameters used in experimental implementation.

The analysis also shows that when $v_1 \leq v_2$, no interior extremum exists within the domain, and the global minimum of the travel time function is again found at the boundary $x = a$. This behavior reinforces the conclusion that the optimal solution x^* is primarily governed by the geometric configuration rather than by fine-tuning of the velocity values.

The proposed model is employed directly online as part of the navigation strategy. The optimal turning point x is computed in real time and used to define the next reference point delivered by the reinforcement learning algorithm. The robot then moves from its current position to this new reference using the trajectory controller based on the Lyapunov candidate function. In this context, the travel-time minimization model acts as a local optimization mechanism embedded within the overall navigation and control architecture, rather than as a purely offline heuristic.

3.3. Obstacle Evasion

Obstacle detection and evasion tests were conducted under controlled illumination conditions using two types of obstacles: humans and traffic cones. These scenarios were selected to represent obstacles with significantly different geometric and visual characteristics, allowing the robustness of the perception system to be evaluated.

Table 4 presents the detection rate obtained for each type of obstacle. In this context, it is important to distinguish between detection capability (recall) and collision occurrence since a delayed detection may result in insufficient evasive action even if the obstacle is eventually identified.

Table 4. Obstacle detection and collision rates for the evaluated object types.

Object	Sample	% Detection	Collision
Person	100	95	5
Cone	100	88	12

For human obstacles, the system achieved a detection rate of 95%. The collision events (5%) were mainly associated with situations in which the obstacle entered the effective field of view (FOV) of the RGB-D camera at its limits, at distances shorter than 0.5 m, thereby reducing the temporal margin available to execute an evasive maneuver.

For traffic cones, the detection rate was 88%, with a collision rate of 12%. This performance difference can be attributed to the physical characteristics of the obstacle, as cones exhibit smaller height, reduced apparent volume, and a more compact geometry, which decreases their projected area in the image and makes early detection more challenging, particularly at longer distances.

The system reaction time, defined as the interval between obstacle detection and the transmission of the control signal to the actuators, exhibited variations depending on the computational load of the system. Under nominal operating conditions, latencies in the range of 90 ms to 200 ms were observed. When image resolutions higher than 640×480 pixels were used, delays of up to 500 ms were recorded, which negatively affected the evasion capability in dynamic scenarios. For this reason, a resolution of 640×480 was selected as a compromise between perceptual accuracy and response time.

Overall, the results indicate that the CNN-based perception system is capable of detecting and evading obstacles in a reliable manner, prioritizing robot safety even under variations in reaction time, which makes it suitable for integration within the proposed autonomous navigation framework.

3.4. Trajectory Control Based on Lyapunov

The simulation stage was carried out under ideal kinematic assumptions to serve as a preliminary functional validation of the hierarchical architecture. The controller implementation was carried out in discrete time with a sampling period of 10 ms. This value was selected from a practical perspective, aiming to adequately capture variations in energy consumption and to enable continuous data transmission via WiFi. Considering that the maximum rotational speed of the motors under load is approximately 85 rpm, this sampling period provides about 70 samples per revolution, which is sufficient to estimate the average energy consumption per traveled meter and to capture the power peaks associated with acceleration and braking phases.

The controller design is based on a Lyapunov candidate function, employing a quadratic tracking-error function, equivalent to the MSE, as a Lyapunov candidate at the kinematic level. This formulation enables the analysis of the local stability of the trajectory-tracking error at the kinematic level for both linear and nonlinear motion regimes. From a theoretical standpoint, a gain value of $K = 1I$ guarantees stability and yields the fastest system response, since lower values reduce the convergence speed.

One of the main advantages of this formulation is that, being a polynomial function, it can be applied to the tracking of nonlinear trajectories, such as a lemniscate. Figure 11 illustrates the tracking of a lemniscate trajectory in the XY plane, comparing the reference path with the trajectory followed by the robot. The same Figure 11 also presents the temporal evolution of the MSE, showing a rapid reduction in the tracking error during the initial phase of motion.

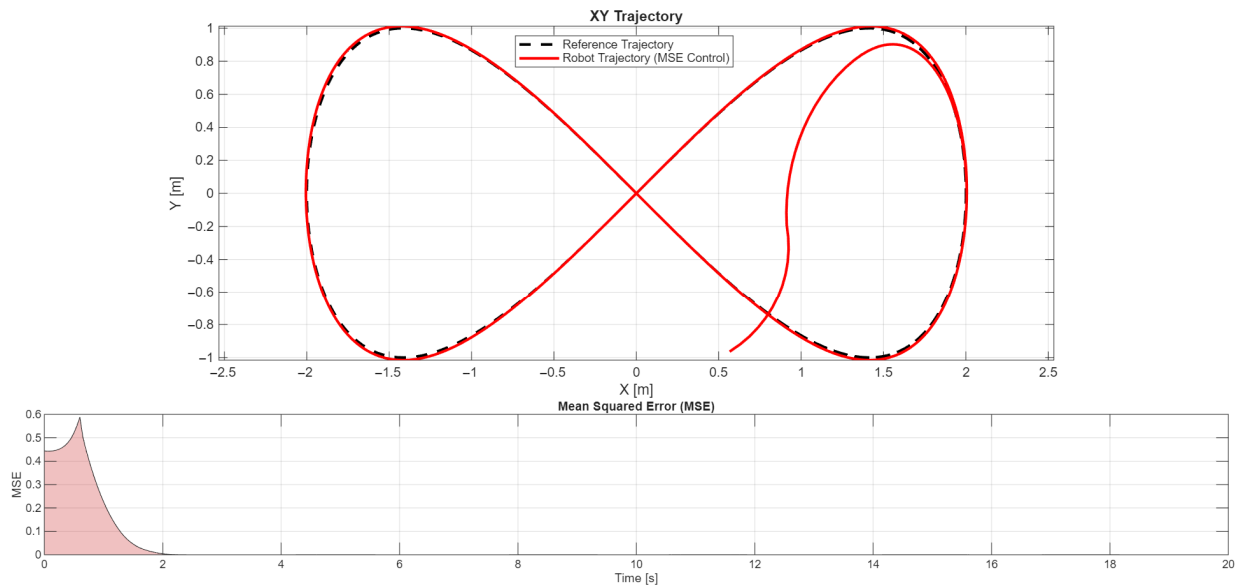


Figure 11. Lemniscate trajectory tracking using the Lyapunov-based controller: reference and robot trajectories in the XY plane and corresponding mean squared error (MSE) evolution over time.

At the simulation level, gain values close to unity produce the lowest MSE and the highest convergence rate, as illustrated in Figures 12 and 13 for linear and nonlinear trajectories, respectively. However, in physical implementation, ideal assumptions such as the absence of inertia, actuator delays, and measurement noise no longer hold. Under these conditions, excessively fast responses may induce oscillations and compromise system stability. As shown in the Control Inputs plot, the control signal converges rapidly to the reference value of 1 m/s (dashed line), demonstrating the stability of the proposed controller. In the case of angular velocity, it converges to 0 as it is a straight trajectory (purple line).

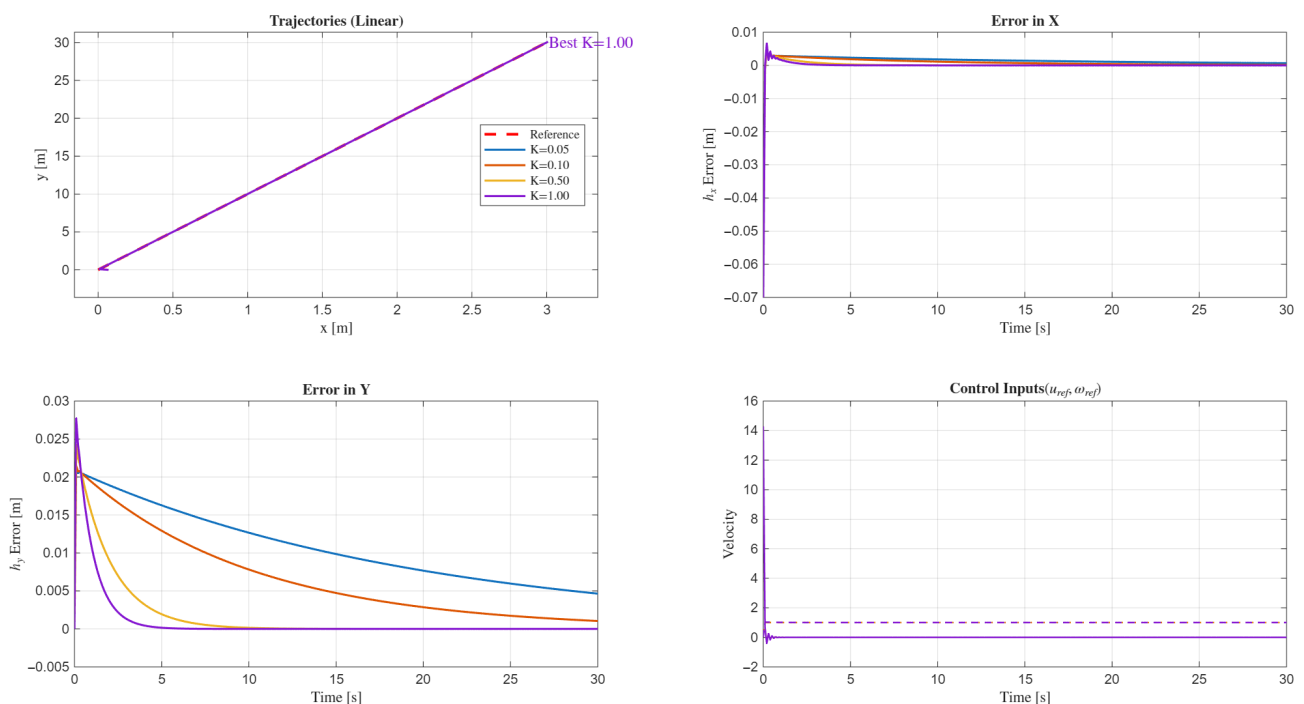


Figure 12. Parametric evaluation of the Lyapunov gain K for linear trajectory tracking, including reference and robot trajectories, tracking errors in the X and Y directions, and corresponding control inputs.

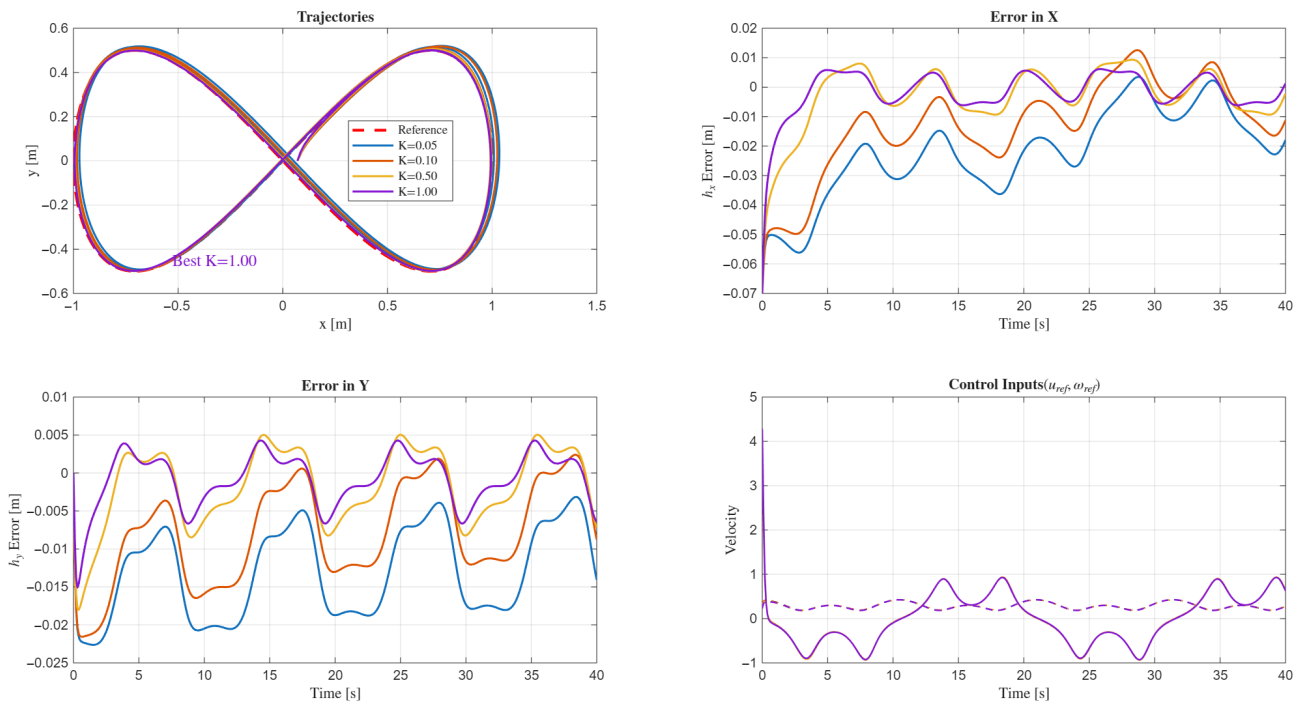


Figure 13. Parametric evaluation of the Lyapunov gain K for nonlinear (lemniscate) trajectory tracking, showing XY-plane trajectories, tracking errors in the X and Y directions, and control inputs.

For this reason, although $K = 1I$ is theoretically optimal, an intermediate value of $K = 0.5I$ was selected for experimental implementation. This value represents an adequate compromise between response speed and robustness, allowing stable trajectory tracking without amplifying the effects of unmodeled dynamics or sensor noise.

It is worth noting that, although Figure 12 does not explicitly report numerical percentage reductions in the MSE, its purpose is to qualitatively illustrate how the tracking error decreases significantly for higher values of K . The interpretation of Figure 13 confirms that the controller converges rapidly and maintains stability throughout the trajectory, validating its integration within the proposed navigation framework.

3.5. Energy Consumption

The energy consumption evaluation compared the proposed system with classical planners such as A*, with the objective of analyzing the energy efficiency of the generated trajectories. To isolate the energy costs associated exclusively with navigation, the net energy was computed by subtracting the robot's baseline power consumption at rest (11.71 W per unit time). Additionally, energy consumption is expressed in joules per meter (J/m), allowing performance normalization independently of the traveled distance.

The maps used to evaluate performance are shown in Figure 14; these originate from prior research [19] that establishes the validation of optimal trajectory generation. Map 1 (Figure 14a) considers the initial position at coordinates (0,0), located at the upper-left corner, and the goal at coordinates (9,9). In this case, a vertical trajectory free of obstacles can only be achieved by displacing a dynamic obstacle that varies in each iteration. Map 2 (Figure 14b) is a variant of Map 1 but includes random dynamic obstacles to increase environmental complexity. Maps 3 and 4 (Figure 14c,d) consider only static obstacles but differ in the initial agent position, which does not correspond to coordinates (0,0). For each map, between three and four experimental runs were performed, yielding very similar energy results across executions, particularly in the shape and magnitude of the consumption peaks.

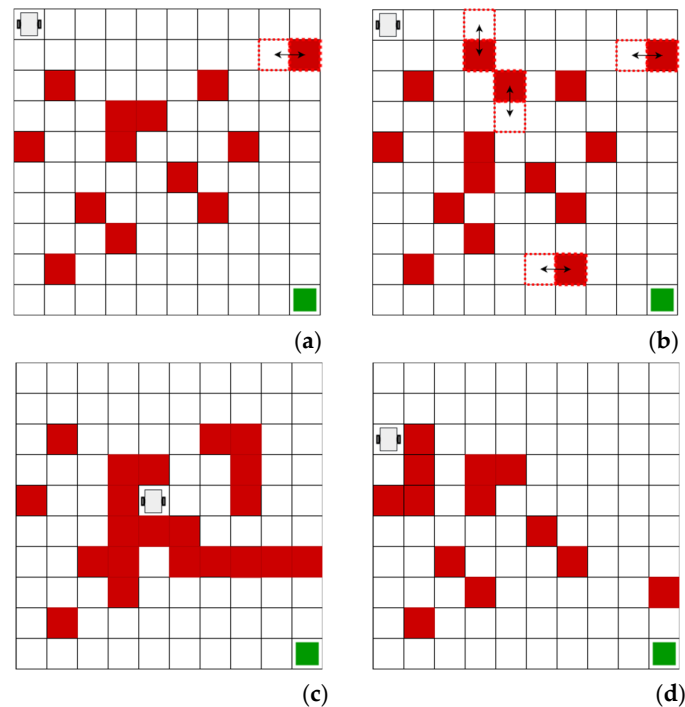


Figure 14. Maps used for reinforcement learning tests: (a) Map 1 with static and dynamic obstacles; (b) Map 2 with static and dynamic obstacles; (c) Map 3 with static obstacles; (d) Map 4 with static obstacles.

Energy consumption is a critical factor in the performance of mobile robots, as it directly impacts autonomy, efficiency, and operational viability in real-world scenarios. This section analyzes experimental measurements of energy consumption both at rest and during motion, as well as during the execution of trajectories obtained with the different proposed algorithms. The objective is to compare the efficiency of each approach in terms of traveled distance, number of turns, trajectory time, and energy consumption in order to determine the relationship between computational complexity and energy usage.

Figure 15 presents power consumption at rest. This was measured using the PCB for data collection, developed in a sample period of 10 ms, to isolate the motor consumption from the basic system consumption. An average of 11.71 W was observed.

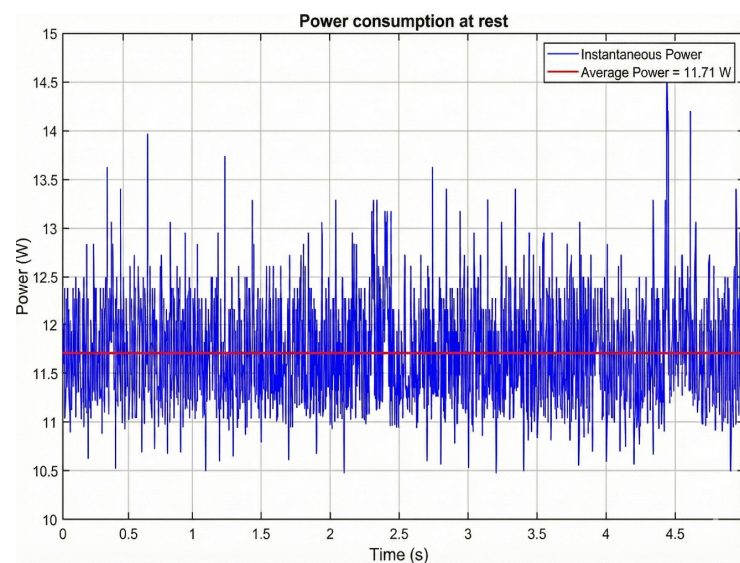


Figure 15. Robot's power consumption at rest.

The prototype uses four JGB37-520R90-12 motor reducers from the manufacturer Hiwonder Technology Co. in Shenzhen, China. The total consumption during a linear trajectory, which corresponds to higher demand due to having four simultaneously active motors, is shown in Figure 16. The graph includes a start signal at 1.45 s and a stop signal at 4.2 s. The average consumption in this case is 19.72 W, equivalent to 8.01 W of average consumption per motor.

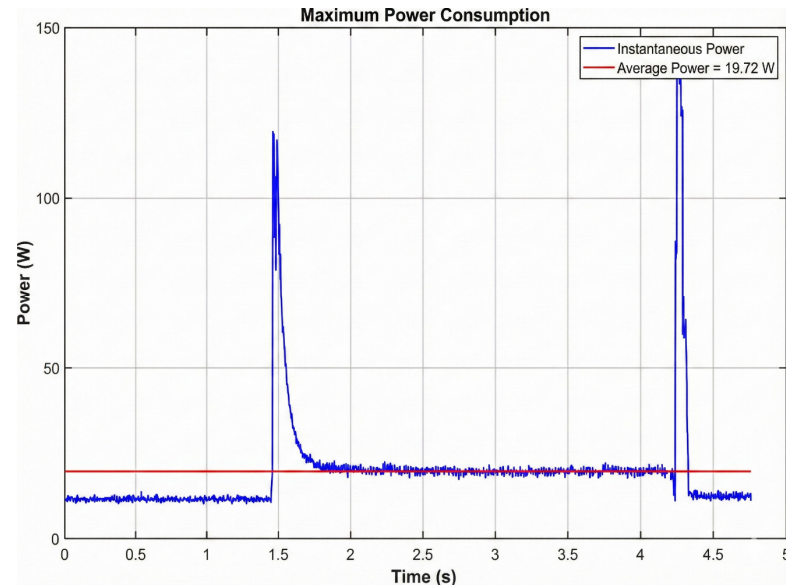


Figure 16. Robot's power consumption during a linear trajectory.

Subsequently, the energy consumption during the trajectories in all four maps considered in the simulation was evaluated. Figures 17–20 show profiles of energy consumption in each case. For these measurements, the average resting value (11.71 W) was used as a reference, and an exponential moving filter with a 0.3 factor was applied to smooth the data. It is noteworthy that the results only correspond to the execution phase of the trajectory, excluding any previous training algorithms. In all experiments, the trajectory-tracking control was implemented using the Lyapunov candidate function with a value of $K = 0.5I$.

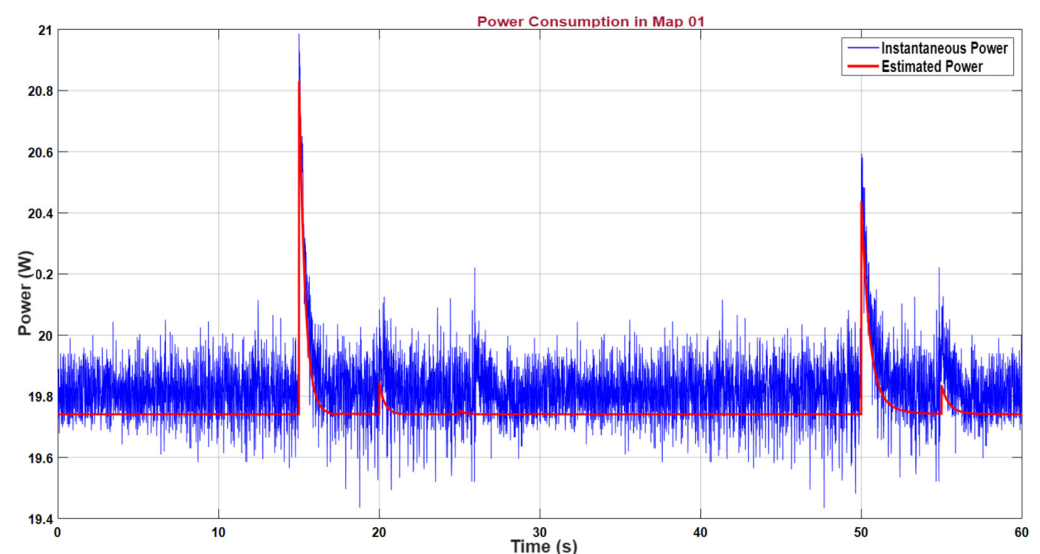


Figure 17. Robot's energy consumption during the trajectory in Map 1.

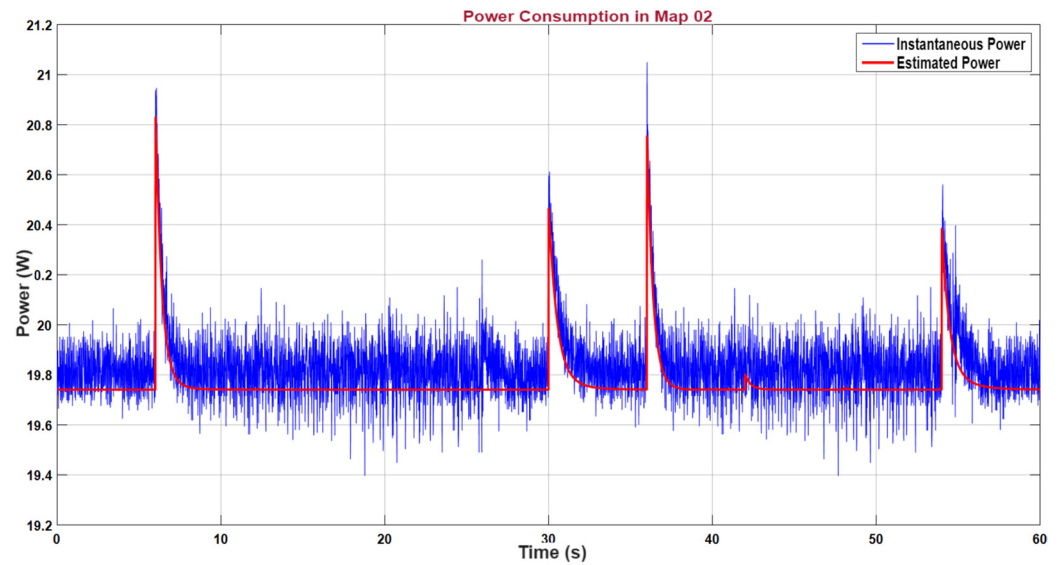


Figure 18. Robot's energy consumption during the trajectory in Map 2.

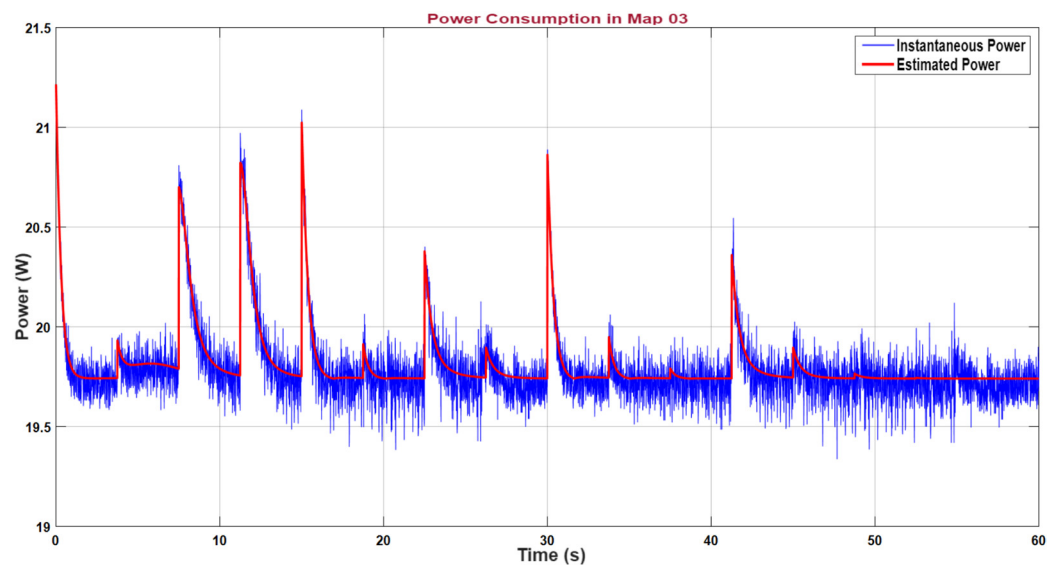


Figure 19. Robot's energy consumption during the trajectory in Map 3.

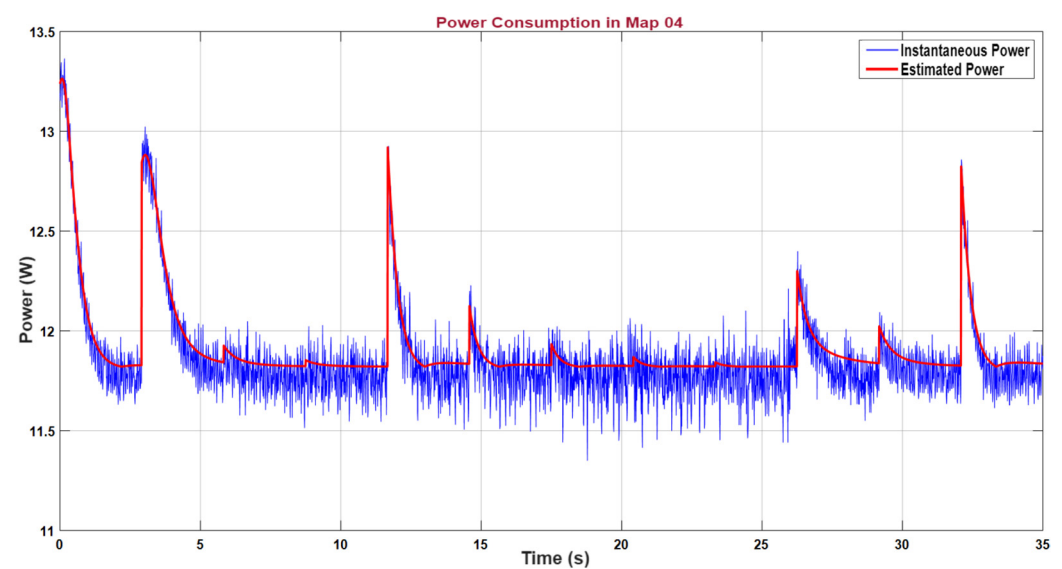


Figure 20. Robot's energy consumption during the trajectory in Map 4.

Table 5 provides a comparative summary of the performance of the four algorithms considered: classical reinforcement learning, DQL, A*-4, and A*-8. The column “Environment knowledge” indicates whether the algorithm requires prior information about the map to operate. Classical algorithms such as A depend on complete knowledge of the environment; without this information, these methods cannot generate valid trajectories. In contrast, reinforcement learning-based approaches operate without prior knowledge of the environment.

Table 5. Comparison of energy performance and trajectory characteristics for the different algorithms.

Map	Algorithm	Distance (m)	No. Turns	Environment Knowledge	Energy (J)	Trajectory Time (s)	Energy Saving vs. A*-4 (%)	Energy Saving vs. A*-8 (%)	Total Energy (J/m)
1	RL	13.89	3	NO	79.5	64	−34.0	−12.8	1104.2
	Proposed DQL	13.89	3	NO	78.7	66	−34.6	−13.7	1093.1
	A* (4 actions)	22.00	10	YES	120.5	100			2651
	A* (8 actions)	13.89	4	YES	91.2	76			1266.7
2	RL	13.31	4	NO	78.0	64	−32.1	−11.3	1038.1
	Proposed DQL	13.31	4	NO	77.5	62	−32.6	−11.9	1031.5
	A* (4 actions)	18.00	5	YES	115.0	95			2070
	A* (8 actions)	13.89	7	YES	88.0	72			1222.3
3	RL	17.89	8	NO	82.0	68	−32.7	−12.7	1466.9
	Proposed DQL	17.89	7	NO	80.5	66	−34.0	−14.3	1440.1
	A* (4 actions)	22.00	10	YES	122.0	102			2684
	A* (8 actions)	18.48	9	YES	94	78			1737.1
4	RL	13.89	6	NO	78.9	65	−33.5	−11.8	1095.9
	Proposed DQL	13.89	4	NO	77.1	63	−35.0	−13.8547	1070.9
	A* (4 actions)	18.00	4	YES	118.7	98			2136.6
	A* (8 actions)	13.89	8	YES	89.5	74			1243.1

In order to provide a quantitative measure of the variability in energy consumption, the standard deviation (σ) and the coefficient of variation (CV) were computed for each map, as summarized in Table 6.

Table 6. Statistical summary of energy consumption variability across the evaluated maps.

Map	Average (W)	σ	CV (%)
Map 1	19.75	0.45	2.3
Map 2	19.80	0.35	1.8
Map 3	19.74	0.35	1.8
Map 4	11.80	0.40	3.4

To quantify the variability of energy consumption across executions, the standard deviation and coefficient of variation (CV) were computed for each map. In all cases, the CV remained below 5%, which can be interpreted as indicative of smooth and energetically stable trajectories. Nevertheless, a potential bias may exist due to the relatively low sampling frequency used in the measurements, since fast power peaks might not have been fully captured. As future work, a sampling frequency close to 1 kHz is recommended to improve the temporal resolution of energy measurements.

The reported energy consumption includes the entire system electronics, as all sensors, processing units, communication modules, and actuators are powered by the same battery. Consequently, the reported values represent the total system energy consumption during trajectory execution.

Overall, reinforcement learning-based algorithms (RL and DQL) exhibit significantly lower energy consumption compared to A*-4 and A*-8, demonstrating the advantage of optimizing trajectories not only in terms of distance or time, but also from an energy-efficiency perspective.

4. Discussion

This work demonstrates that a structured integration of classical reinforcement learning with stability-based control represents a robust alternative to purely deep learning-driven approaches for autonomous navigation. In particular, the explicit decoupling between high-level decision-making and low-level motion execution, where learning-based modules generate navigation references, while stability is ensured at the kinematic level through an analytically designed control law.

While a significant portion of recent literature on autonomous navigation focuses on the direct use of deep reinforcement learning (DRL) to map sensory inputs to velocity commands, this study proposes a hierarchical architecture in which learning is restricted to generating discrete decisions or subgoals, while continuous execution is handled by a Lyapunov-based controller. This design not only improves system interpretability but also facilitates validation and tuning on real robotic platforms, where delays, inertia, and noise cannot be neglected.

One of the most relevant findings of this study is the strong correlation observed between trajectory shape and energy consumption. Classical planning algorithms such as A* tend to generate trajectories with abrupt direction changes, typically constrained to 45° or 90° increments. Although such behavior may be efficient from a geometric standpoint, it introduces a large number of maneuvers that translate into current peaks during acceleration and deceleration phases of the motors.

In contrast, agents trained using RL and DQL learned navigation policies that favor smoother and more continuous trajectories, significantly reducing the number of required turns. The observed reduction, ranging approximately from 36% to 50% depending on the evaluated map, confirms that trajectory optimization should be addressed as a coupled problem involving geometry, dynamics, and energy, rather than as a purely distance-minimization task.

Although energy consumption peaks were detected during start and stop events—mainly associated with motor polarity changes—these remained within expected ranges and did not significantly affect average power consumption. The consistency between simulated and experimental energy profiles further supports the validity of the proposed approach and its applicability to real robotic systems with limited energy resources.

Another key contribution of this work is the experimental validation of a Lyapunov-based reference generation strategy embedded within the learned navigation framework. Unlike approaches in which learning algorithms directly control actuator velocities, the proposed scheme guarantees local asymptotic convergence of the tracking error at the kinematic level, provided that the stability conditions of the reference generator are satisfied.

In this hierarchical scheme, the trajectory planning algorithm operates as a generator of references for the Lyapunov-based controller. The Lyapunov candidate function ensures that the discrete sub-objectives provided by the navigation layer are reached with asymptotic convergence.

The decision to implement PID controllers in the low-level actuation stage was driven by their minimal computational load. Despite its architectural simplicity compared to advanced control laws, the results demonstrate that the controller executes successfully. Future work may integrate advanced control techniques.

From a theoretical perspective, under ideal kinematic assumptions, a gain value of $K = 1I$ ensures stability and yields the fastest response. However, experimental results show that this value may induce oscillations in the presence of delays, inertia, and unmodeled dynamics. Selecting an intermediate gain ($K = 0.5I$) mitigated these effects by generating smoother reference velocities that can be reliably tracked by the low-level PID controllers.

under real experimental conditions, while maintaining a tracking error below 1.2 cm, which is suitable for autonomous navigation in confined environments.

One aspect worth highlighting in the experimental validation is the use of a chassis equipped with omnidirectional wheels restricted to a differential-drive configuration. This setup offers advantages, including a substantial reduction in lateral friction during turns. This reduction in frictional resistance contributes to the efficiency observed in the generated trajectories. However, this configuration also introduces challenges, such as mechanical vibrations inherent in the contact between the roller and the ground, which can introduce noise into the sensory feedback.

Regarding the control signals, their variability was evaluated during real-world experiments by analyzing the commanded angular velocities sent to the motor drivers. The measured signals remained bounded and exhibited low-amplitude fluctuations around their nominal values, mainly during acceleration and deceleration phases and when abrupt heading corrections were required. The root-mean-square (RMS) variation in the normalized control inputs remained below 6% of the nominal operating range, indicating that no high-frequency oscillatory behavior was induced by the Lyapunov-based reference generator or the low-level PID controllers. These results confirm that the proposed hierarchical scheme does not amplify measurement noise or mechanical vibrations into unstable control actions, which is essential for safe long-term operation in physical robotic platforms.

This trade-off between performance and robustness highlights the importance of experimental validation when combining learning algorithms with control strategies and emphasizes that optimal configurations in simulation are not always directly transferable to physical systems.

Despite the encouraging results, several limitations should be acknowledged. First, experimental validation was conducted in a controlled environment with uniform illumination. Abrupt lighting variations, strong shadows, or reflective surfaces could degrade the performance of the proposed CNN-based perception module. Regarding perception, a global formulation was used to ensure low latency in embedded systems. This represents a geometric simplification compared to spatial segmentation. However, methodological architecture is modular, allowing for the replacement of the CNN classification with a segmented semantic, which increases the capacity of generalization in complex environments without affecting the controller.

Additionally, the RGB-D sensor used in this study has a limited effective range, which constrains the robot's maximum safe velocity, particularly in dynamic environments where reaction time is critical. Furthermore, although basic statistical metrics were used to evaluate energy consumption variability, extended experimental campaigns were not conducted to rigorously characterize repeatability under strictly controlled conditions. Additionally, the Lyapunov-based analysis is formulated at the kinematic reference level and does not constitute a global stability proof of the full electromechanical system.

Finally, the sampling frequency employed for energy measurements, while sufficient to capture overall trends, may not be adequate to record very short electrical transients, potentially introducing a bias in the estimation of peak consumption.

As a natural extension of this work, the integration of complementary sensors such as LiDAR is proposed to enhance environmental perception under low-light conditions or increased geometric complexity. The application of sensor fusion techniques could further improve system robustness against partial perception failures.

Another promising research direction involves the implementation of online or adaptive learning mechanisms, enabling the robot to adjust its navigation policies in response to unmodeled changes such as mechanical wear, variations in ground friction, or accumulated odometry errors. Moreover, employing higher sampling frequencies for energy

measurement would allow a more detailed analysis of transient phenomena associated with aggressive maneuvers.

Together, these developments could consolidate the proposed approach as a scalable, safe, and energy-aware solution for autonomous navigation in real and dynamic environments.

5. Conclusions

This work has presented an integrated methodology for autonomous navigation in differential mobile robots that combines deep learning-based perception, reinforcement learning for decision-making, and Lyapunov-based trajectory control. The proposed framework advances the state of the art by explicitly decoupling high-level navigation decisions from low-level control execution, enabling adaptability while preserving formal local stability guarantees at the reference generation level.

Experimental results confirm that the Lyapunov-based trajectory controller ensures local asymptotic convergence of the tracking error toward navigation subgoals, achieving average tracking errors below 1.2 cm in real-world tests. For the experimental platform considered, a control gain of $K = 0.5I$ was identified as an appropriate operating point for the kinematic reference generator, providing stable and smooth trajectory tracking under physical constraints and unmodeled dynamics.

From a perception standpoint, the convolutional neural network developed for obstacle detection achieved classification accuracies above 95% while maintaining low computational complexity, making it suitable for real-time deployment on embedded robotic platforms. In addition, the analytical formulation for travel-time minimization enabled consistent identification of optimal turning points, which was directly integrated into the online navigation process.

A key contribution of this study lies in the experimental evaluation of energy consumption. The results demonstrate that reinforcement learning-based navigation policies yield significantly lower energy consumption compared to classical planners, primarily due to smoother trajectories and a reduced number of turning maneuvers. This confirms that trajectory planning should be addressed not only as a geometric or temporal optimization problem but also from an energetic perspective.

The experimental validation was conducted in a controlled indoor environment with predefined lighting conditions and obstacle configurations, which represents a current limitation of the study. Future work will focus on extending the proposed framework to more complex and unstructured environments, incorporating multimodal perception and adaptive learning mechanisms to enhance robustness, autonomy, and energy efficiency in real-world applications. Moreover, the stability analysis focuses on the reference tracking dynamics and does not attempt to establish global stability of the complete actuator–sensor loop.

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

Funding: This research was supported by the Instituto Politécnico Nacional (IPN) under grant number SIP/20254992.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Acknowledgments: The authors have reviewed and edited the output and take full responsibility for the content of this publication.

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

References

1. Li, J.; Wang, A.; Lyu, Y.; Ding, Z. Global Path Guided Model Predictive Path Integral Control: Applications to GPU-Parallelizable Robot Simulation Systems. *Comput. Electr. Eng.* **2024**, *120*, 109645. [\[CrossRef\]](#)
2. Hau, B.M.; You, S.-S.; Bao Long, L.N.; Kim, H.-S. Efficient Routing for Multiple AGVs in Container Terminals Using Hybrid Deep Learning and Metaheuristic Algorithm. *Ain Shams Eng. J.* **2025**, *16*, 103468. [\[CrossRef\]](#)
3. Reyes-Reyes, E.; Santiago-Nogales, B.N.; Rodríguez-Cerón, J.R.; Silva-Ortigoza, R.; Roldán-Caballero, A.; García-Sánchez, J.R.; Marciano-Melchor, M.; Taud, H.; Tavera-Mosqueda, S. Robust Multilevel Average Controller for Trajectory Tracking in Mobile Robots Powered by Solar Panels: Integrating Actuators and Power Electronics Stage. *Results Eng.* **2025**, *26*, 105311. [\[CrossRef\]](#)
4. Xaud, M.F.S.; From, P.J.; Leite, A.C. Robust Visual Servoing and CNN-Based Thermal Imaging for Sugarcane Row Following with a Skid-Steering Mobile Robot. *IEEE Access* **2025**, *13*, 143166–143195. [\[CrossRef\]](#)
5. Azimirad, V.; Khodkam, S.Y.; Bolouri, A. A New Hybrid Learning Control System for Robots Based on Spiking Neural Networks. *Neural Netw.* **2024**, *180*, 106656. [\[CrossRef\]](#) [\[PubMed\]](#)
6. Liu, H.; Dong, W.; Zhang, Z.; Wang, C.; Li, R.; Gao, Y. Optimization-Based Local Planner for a Nonholonomic Autonomous Mobile Robot in Semi-Structured Environments. *Robot. Auton. Syst.* **2024**, *171*, 104565. [\[CrossRef\]](#)
7. Zhai, X.; Tian, J.; Li, J. A Real-Time Path Planning Algorithm for Mobile Robots Based on Safety Distance Matrix and Adaptive Weight Adjustment Strategy. *Int. J. Control Autom. Syst.* **2024**, *22*, 1385–1399. [\[CrossRef\]](#)
8. Trasnea, B.; Ginerica, C.; Zaha, M.; Macesanu, G.; Pozna, C.; Grigorescu, S. OctoPath: An OcTree Based Self-Supervised Learning Approach to Local Trajectory Planning for Mobile Robots. *Sensors* **2021**, *21*, 3606. [\[CrossRef\]](#) [\[PubMed\]](#)
9. Meng, J.; Stoyanov, D. RRT-GPMP2: A Motion Planner for Mobile Robots in Complex Maze Environments. *arXiv* **2024**, arXiv:2412.07683. [\[CrossRef\]](#)
10. Lee, M.-F.R.; Yusuf, S.H. Mobile Robot Navigation Using Deep Reinforcement Learning. *Processes* **2022**, *10*, 2748. [\[CrossRef\]](#)
11. Lin, Y.; Zhang, Z.; Tan, Y.; Fu, H.; Min, H. Efficient TD3 Based Path Planning of Mobile Robot in Dynamic Environments Using Prioritized Experience Replay and LSTM. *Sci. Rep.* **2025**, *15*, 18331. [\[CrossRef\]](#) [\[PubMed\]](#)
12. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS 2012)*; Curran Associates, Inc.: Red Hook, NY, USA, 2012; Volume 25.
13. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980.
14. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2014**, arXiv:1409.1556.
15. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5 MB Model Size. *arXiv* **2016**, arXiv:1602.07360.
16. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going Deeper with Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
17. Rapalski, A.; Dudzik, S. Energy Consumption Analysis of the Selected Navigation Algorithms for Wheeled Mobile Robots. *Energies* **2023**, *16*, 1532. [\[CrossRef\]](#)
18. Wijaya, G.D.; Caesarendra, W.; Petra, M.I.; Królczyk, G.; Glowacz, A. Comparative Study of Gazebo and Unity 3D in Performing a Virtual Pick and Place of Universal Robot UR3 for Assembly Process in Manufacturing. *Simul. Model. Pract. Theory* **2024**, *132*, 102895. [\[CrossRef\]](#)
19. Jaramillo-Martínez, R.; Chavero-Navarrete, E.; Ibarra-Pérez, T. Reinforcement-Learning-Based Path Planning: A Reward Function Strategy. *Appl. Sci.* **2024**, *14*, 7654. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.