algorithms

MDPI

Article

Control of a Mobile Line-Following Robot Using Neural Networks

Hugo M. Leal [1], Ramiro S. Barbosa [1,2,*] and Isabel S. Jesus [1,2]

[1] Department of Electrical Engineering, Institute of Engineering—Polytechnic of Porto (ISEP/IPP), 4249-015 Porto, Portugal; 1190660@isep.ipp.pt (H.M.L.); isj@isep.ipp.pt (I.S.J.)
[2] GECAD—Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development, ISEP/IPP, 4249-015 Porto, Portugal
* Correspondence: rsb@isep.ipp.pt

Abstract: This work aims to develop and compare the performance of a line-following robot using both neural networks and classical controllers such as Proportional–Integral–Derivative (PID). Initially, the robot's infrared sensors were employed to follow a line using a PID controller. The data from this method were then used to train a Long Short-Term Memory (LSTM) network, which successfully replicated the behavior of the PID controller. In a subsequent experiment, the robot's camera was used for line-following with neural networks. Images of the track were captured, categorized, and used to train a convolutional neural network (CNN), which then controlled the robot in real time. The results showed that neural networks are effective but require more processing and calibration. On the other hand, PID controllers proved to be simpler and more efficient for the tested tracks. Although neural networks are very promising for advanced applications, they are also capable of handling simpler tasks effectively.

Keywords: AGV; robot; Raspbot; PID; LSTM; CNN; AI; deep learning; line-follower

1. Introduction

In recent years, artificial intelligence techniques, such as machine learning and deep learning, have revolutionized various fields of science and engineering, including robotics [1–4]. These techniques have shown significant potential to enhance the autonomy and intelligence of robots, enabling them to perform complex tasks with greater efficiency and precision. One of the promising fields where these techniques are currently applied is the control of mobile robots, including Autonomous Guided Vehicles (AGVs).

Traditionally, techniques such as PID control and trajectory planning algorithms have been used in mobile robot control. However, these approaches face limitations in dynamic and unstructured environments. Deep learning (DL) techniques offer a data-driven approach that can enhance the adaptability and intelligence of these control systems. Inspired by recent advances in artificial intelligence, modern alternatives like LSTM and CNN networks have emerged, offering potential solutions to these challenges.

In this context, the development of a line-following robot control algorithm using DL techniques was chosen. This practical and relevant application enables one to demonstrate the effectiveness of neural networks in robot control, providing solid validation for new control approaches. By comparing traditional and modern techniques, this work aims to identify the advantages and limitations of DL, contributing to the development of smarter and more efficient robotic systems. On the other hand, the use of DL algorithms such as LSTM and CNN does not inherently require a mathematical model of the robot for its control. The need for a mathematical model depends on the specific application and the role

check for updates

Academic Editor: Frank Werner

Received: 28 November 2024
Revised: 30 December 2024
Accepted: 14 January 2025
Published: 17 January 2025

Citation: Leal, H.M.; Barbosa, R.S.; Jesus, I.S. Control of a Mobile Line-Following Robot Using Neural Networks. *Algorithms* **2025**, *18*, 51. https://doi.org/10.3390/a18010051

of neural networks in the control system. In the present case, the LSTM or CNN models are trained to predict control actions directly from sensor data, and the robot can be controlled using only the sensor inputs. This data-driven approach relies on neural networks to learn the relationships between sensor input and control output during training. Thus, LSTM is used to process sequential data, such as time series sensor readings, while CNN is used to handle spatial data such as camera images. The results obtained are very promising, showing that the quality and quantity of the training data used were sufficient to provide satisfactory control of the robot in a real scenario. Therefore, the main contributions of this work can be summarized as follows:

- Introducing the LSTM and CNN architectures for the control of a mobile robot.
- Providing detailed insights into the implementation of various control algorithms, with a particular focus on the LSTM and CNN networks.
- Comparison of artificial intelligence algorithms with the classical PID controller.
- Evaluating and comparing the performance of different control algorithms using various metrics.

The article is structured as follows. Section 2 presents the robot architecture and its main components. The control methods are described in Section 3. Section 4 discusses the details of the implementation of the three control strategies used, PID, LSTM, and CNN. The results are provided in Section 5, while Section 6 includes a discussion of the real experiments. Finally, Section 7 draws the main conclusions and outlines the future developments of the presented work.

Literature Review

In recent years, the use of artificial intelligence and deep learning for autonomous vehicle control has attracted a great deal of interest. In [5], the authors developed a Bluetooth- and vision-controlled AGV for COVID-19, capable of carrying 10 kg and following a labeled user at an optimal distance of 80 cm and a 55° detection angle. The AGV reduces contact risk, aids seniors by eliminating the need to push heavy loads, and operates via a mobile app or autonomous tracking within a 10 m range. The work by [6] introduced an AI-driven AGV for factory automation, using magnetic tape for navigation to handle goods. The limitations include misjudgment when tape breaks and an inability to change routes around obstacles. The study explored divergence-wheeled robots for smoother navigation in crowded environments. In [7], an AGV was presented for seamless indoor and outdoor navigation, utilizing a self-evolving free space detection (FSD) framework with online active machine learning and sensor data fusion. The framework outperforms DeepLabV3+ by self-learning from real-time multimodal data, reducing the need for large datasets while improving robustness in unstructured environments. In [8], an obstacle avoidance system was implemented for AGVs using deep action learning (DAL) to enhance visual navigation, object recognition, and decision-making. By integrating YOLOv4, SURF, and kNN, the system achieves real-time obstacle avoidance and navigation, meeting Industry 4.0 standards for speed, accuracy, and robustness in unstructured environments. In [9], the authors proposed an AGV obstacle avoidance system combining deep learning object detection and grid-based path planning for dynamic industrial environments. Tested in a simulated tobacco production workshop, the system achieved a 98.67% success rate in obstacle avoidance and improved task efficiency by 27.29%, offering a reliable solution for industrial AGVs. In [10], a deep-reinforcement-learning-based method for automatic PID adjustment to ensure smooth AGV movement was presented. Using the Deep Q-Learning Network (DQN), the approach transforms PID tuning into an action–value optimization problem. Simulations and tests demonstrate the method's effectiveness in achieving optimal PID settings without manual intervention. Another study [11] introduced a two-wheel

AGV platform utilizing sensor fusion and neural networks for intelligent path planning and navigation in dynamic unknown environments. The neural network processes sensor inputs to determine safe paths while avoiding arbitrarily shaped and moving obstacles, with simulations validating the approach for Industry 4.0 applications. The work in [12] reviewed recent applications of deep learning in UAVs for tasks like security, disaster rescue, and warehouse management, highlighting the key techniques, their performance, and limitations while discussing the challenges and future directions for UAV-based deep learning solutions. The work in [13] presented an AI algorithm for platform autonomy, enabling navigation without human intervention. Using cameras, LIDAR sensors, and a vision system, the platform employs convolutional neural networks for environment analysis and path optimization. Transfer learning trains the network, with the results demonstrating its effectiveness. In [14], a deep reinforcement learning approach for mapless AGV navigation using LiDAR and RGB cameras was presented. The NavACL-Q method combines curriculum learning with soft actor–critic to address challenges like sparse samples and partial observability. The results demonstrated a 40% performance gain over random starts and a 60% boost with a pre-trained feature extractor, outperforming map-based navigation.

2. System Architecture

This section describes the robot platform, its architecture, and the software tools used for the project.

2.1. Mobile Platform

The robot chosen for this study was the Raspbot AI Vision Robot Car from YahBoom [15]. It is equipped with the necessary sensors for implementing a line-follower, including infrared sensors and a camera. Table 1 lists a summary of the main characteristics of the Raspbot AI Vision Robot Car. It integrates vision functions based on open-source computer vision (CV) via Python programming. This enables the robot to be controlled via an app, infrared remote (IR), and the JupyterLab web interface.

Table 1. Main characteristics of Yahboom Raspbot AI Vision Robot Car.

Parameter	Yahboom
Sensors	Camera Lower and lateral infrared Ultrasound
Infrared module	4
No. of wheels	4
No. of motors	4
Microcontroller	Raspberry Pi 4B
Camera	5 Megapixels 1080P@30FPS/ 720P@60FPS/480P@90FPS interface CSI
Camera Gimbal	Yes
Ultrasound Gimbal	No

2.2. Robot Architecture

The Raspbot is equipped with interconnected components, including sensors, microcontrollers, drivers, and motors. This setup enables the robot to perform several tasks necessary for the realization of this project. Figure 1 presents a block diagram of the system's architecture, highlighting the main components of the system and their connections [15].

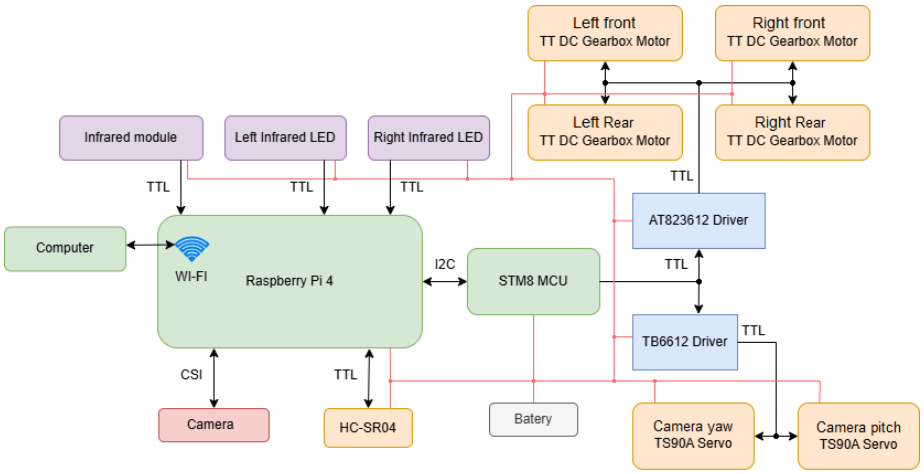
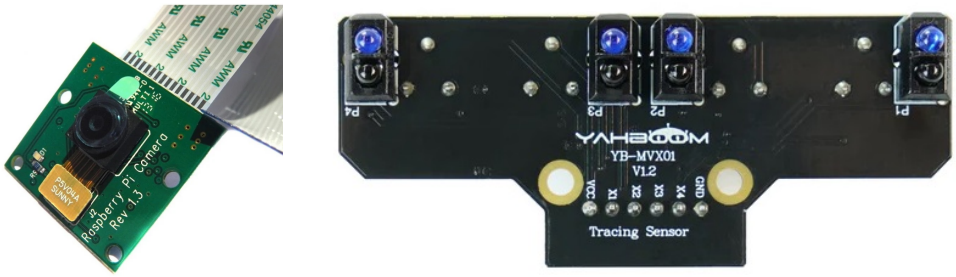


Figure 1. Block diagram of the robot architecture.

The green blocks represent the system's controllers, the main one being the Raspberry Pi 4 model B. The Raspberry will be responsible for controlling all the system's components. The robot is equipped with six motors: 4 generic TT DC gearbox motors from TTmotors in China [16] and 2 TS90A servo motors from TianKongRC, China [17]. In addition, it has a Raspberry Pi Camera Rev 1.3 [18] (Figure 2a), capable of recording still images at a resolution of 2592 × 1944 pixels. For line tracking, the robot employs a lower tracking module equipped with four tracking probes in an optimized spacing layout. The two inner probes accurately detect the black line, while the two outer ones assist the inner probes by providing additional detection and early warnings. This configuration enables the robot to navigate complex tracks, including 90° curves (Figure 2b) [15].

This setup provides remote access to the Raspberry Pi operating system through the JupyterLab environment [15].



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

Figure 2. (**a**) Raspberry Pi Camera Rev 1.3 [18] and (**b**) Yahboom's 4-channel infrared tracking sensor [15].

2.3. Software and Tools

For this work, an operating system (OS) distributed by the robot's manufacturer (based on Raspbian 5.10.63-v7l+) was used, which enables the user to connect the Raspberry Pi to a Wi-Fi network without having to connect it to a monitor or keyboard afterwards. Once connected, remote access to the system is also available through a JupyterLab 6.1.4 server (Figure 3).

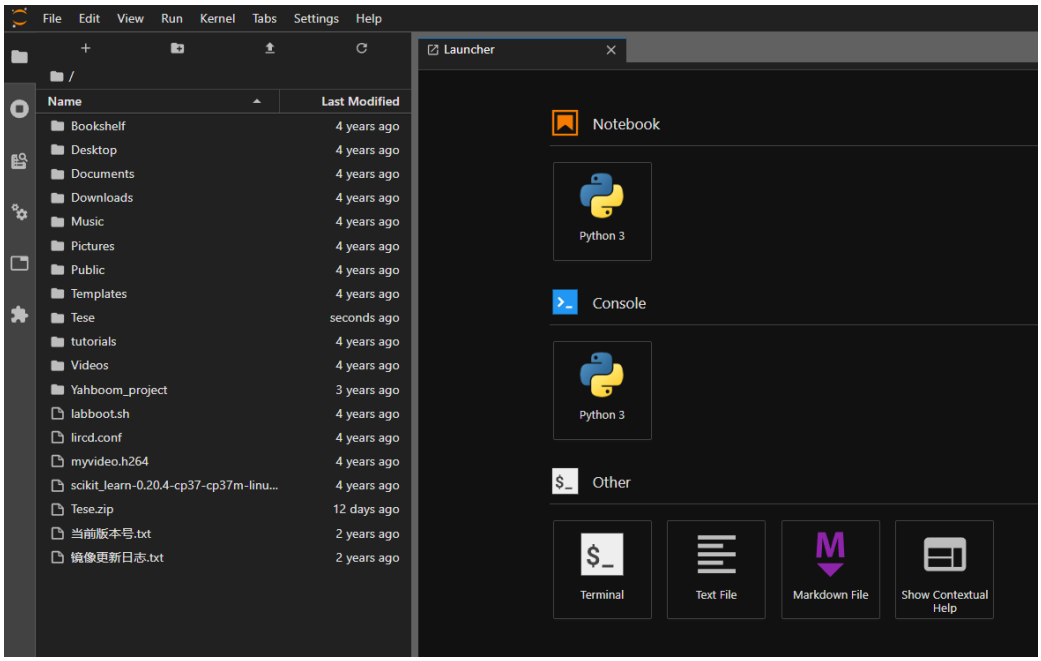


Figure 3. JupyterLab environment.

The data collected by the robot for training the neural networks were transferred to a more powerful computer equipped with an NVIDIA RTX 3060 GPU [19], a Ryzen 7 3700X AMD CPU (3.6 GHZ base clock speed) [20], and 32 GB of DDR4 RAM. This setup enabled faster neural network training and testing of different network configurations. The Python language (version 3.7.4) was selected because of the greater degree of familiarity already present and its powerful data visualization, which also helped the debugging process during the development of the work. The main tools included the Anaconda environment, the Jupyter Notebook, and libraries such as TensorFlow 1.14.0 and Keras 2.2.4 [21,22].

3. Control Strategies

The objective of this work was to develop, implement and test multiple control methods to assess their performance in the control of a line-following robot. For that purpose, classical and deep learning methods were studied. During execution, sensor data from the infrared sensors or camera are sent to the Raspberry Pi and then the controller implemented in the software will calculate the new motor speed values and send these new values to the motors, as illustrated in Figure 4. The control is created with two values, where a single value will be updated for both right motors (front and rear), and the other value will be updated for the left motors.

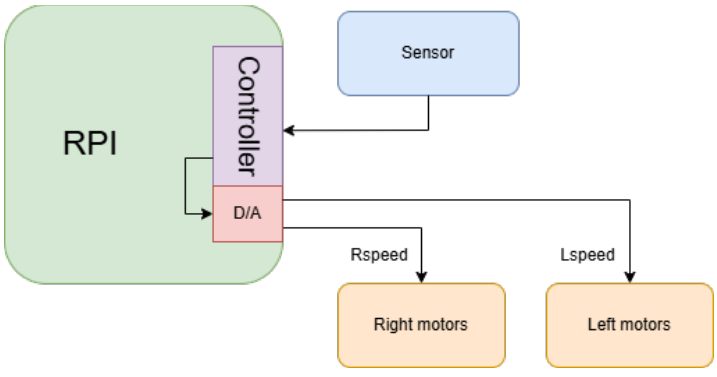


Figure 4. Schematic of the system with the software controller.

3.1. PID

For the classical methods, the PID (Proportional–Integral–Derivative) controller was selected. The PID is the most common controller in industry and is widely used in all control applications. It also represents an essential element of more sophisticated controllers [23–26]. Figure 5 presents a block diagram of a PID-controlled system [27,28].

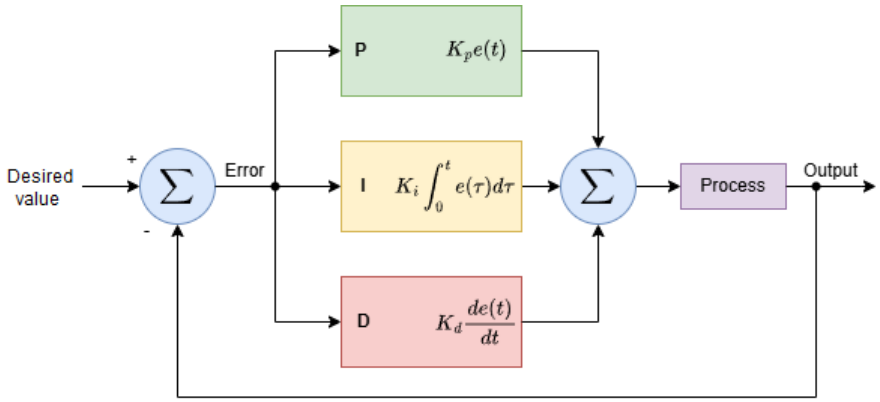


Figure 5. PID-controlled system.

The PID controller can be described as

$$u(t) = K_p\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)\,d\tau + T_d\frac{de(t)}{dt}\right) \tag{1}$$

where $u(t)$ represents the controller action output and $e(t)$ is a control error input, which is the difference between the desired reference value and the current value of the variable to be controlled. The controller output is thus the sum of three terms: the P-term (which is proportional to the error), the I-term (which is proportional to the integral of the error), and the D-term (which is proportional to the derivative of the error). The parameters K_p, T_i, and T_d are, respectively, the proportional gain, the integral time constant, and the derivative time constant [29].

The discrete PID controller can be obtained as

$$u[n] = K_p\left(e[n] + \frac{T}{T_i}\sum_{k=0}^{n} e[k] + \frac{T_d}{T}(e[n] - e[n-1])\right) \tag{2}$$

where T is the sampling period and n is the sample instant. Since T is constant (T = 1/80 s), the following simplified equation was used:

$$u[n] = K_p e[n] + K_i\sum_{k=0}^{n} e(k) + K_d(e[n] - e[n-1]) \tag{3}$$

where $K_i = \frac{K_p T}{T_i}$ is the integral gain and $K_d = \frac{K_p T_d}{T}$ the derivative gain.

3.2. LSTM

LSTM (Long-Short Term Memory) is an advanced type of RNN (recurrent neural network) developed by Hochreiter and Schmidhuber [30]. Traditional RNNs are characterized by using the previous output as one of the inputs of the network, similarly to PID controllers. However, RNNs have difficulty learning long-term dependencies due to the vanishing gradient problem. LSTM solves this problem by introducing a memory cell that can retain information over extended periods of time. Figure 6 shows the structure of an LSTM cell [31].

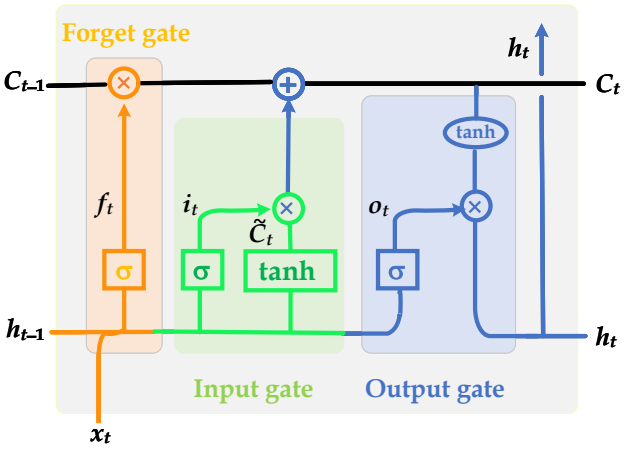


Figure 6. LSTM cell.

The equations for the forward pass of an LSTM cell are as follows:

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{4}$$

where W terms denote the weight matrices, $[h_{t-1}, x_t]$ the concatenation of the current input and the previous hidden state, the b terms denote bias vectors, σ is the sigmoid activation function, and i, f, o, and C are, respectively, the input gate, forget gate, output gate, and cell activation vectors; $*$ is the elementwise multiplication, and tanh function is the activation function.

The LSTM, by having the ability to capture long-term dependencies and process sequential data efficiently, has revolutionized many fields, including natural language processing, speech recognition, and time-series forecasting. Their advanced architecture, which incorporates memory cells and gates, enables them to learn and retain complex patterns over time, making them an essential tool in modern machine learning and artificial intelligence applications, including image classification [32].

3.3. CNN

A CNN (convolutional neural network) is a widely used discriminative deep learning architecture that learns directly from input data without the need for manual feature extraction. As a result, CNN improves on the design of traditional ANNs (artificial neural networks), such as regularized MLP (Multi-Layered Perceptron) networks. Each layer in the CNN takes into account optimal parameters to produce a meaningful output, as well as to reduce the complexity of the model [33].

They are specifically designed to handle a variety of 2D shapes, making them widely used in visual recognition, medical image analysis, image segmentation, natural language processing, and many other domains. The ability to automatically discover essential input features without human intervention makes it more powerful than a traditional network [34].

Table 2 shows the CNN LeNet (LeNet-5) architecture to classify images [22]. LeNet-5 is characterized by having a total of 7 layers (3 convolutional layers, 2 pooling layers, and 2 dense fully connected (FC) layers) and is a simple and efficient network generally used for the categorization of handwritten letters and numbers. In the present work, it was used for the classification of shapes of the black line in front of the robot. This architecture was chosen primarily for its lightweight and computationally efficient design, making it well suited for low-end hardware implementations, which is the case with respect to the Raspberry Pi utilized. It also offers several advantages, including ease of implementation, with fewer parameters than other CNNs, making it easier to train and less prone to overfitting in smaller datasets.

Table 2. LeNet-5 architecture.

Layer	Layer Type	Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32×32	-	-	-
1	Convolution	6	28×28	5×5	1	tanh
2	Average Pooling	6	14×14	2×2	2	tanh
3	Convolution	16	10×10	5×5	1	tanh
4	Average Pooling	16	5×5	2×2	2	tanh
5	Convolution	120	1×1	5×5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

4. Implementation

This section details the various control methods used for the control of the line-follower robot. The project will be divided into two experiments. The first one uses the track on the wooden floor and the infrared sensors with an LSTM network, and the second one, on a modified track, uses the camera and a CNN for robot control.

4.1. PID

The objective is to control the robot to follow a line drawn on the ground utilizing neural networks through infrared sensors. To achieve this, in order to collect and classify data to be used to train the LSTM network, a PID controller was first implemented. Therefore, whilst the robot is being controlled by the PID, the infrared sensor data and right and left speeds are saved for use in the next experiment. Figure 7 shows the main loop of the PID-controlled system. This loop is executed 80 times per second.

While the user does not interrupt the program, the algorithm will start on each loop by obtaining the new error value to be fed to the PID controller. This error is defined as the approximate distance between the central position of the infrared sensor module and the central position of the line. Consequently, the current position value is first calculated by the following equation:

$$Position = \frac{(0 \times S1) + (2750 \times S2) + (3550 \times S3) + (6300 \times S4)}{S1 + S2 + S3 + S4} \quad (5)$$

where S1 to S4 represent the binary logic values of each sensor. This estimate is completed using a weighted average of the sensor indexes multiplied by a specified value. The values

used can be chosen arbitrarily, so the controller gains would need to be adjusted accordingly. However, since the four sensors are not spaced equally apart, the solution found was to match the gains to the distance between the different sensors. By measuring this distance, the values were set to 0, 2750, 3550 and 6300, with the first weight being 0, kept in the equation for clarity of the weight used for each infrared sensor of the module, as shown in Figure 8.

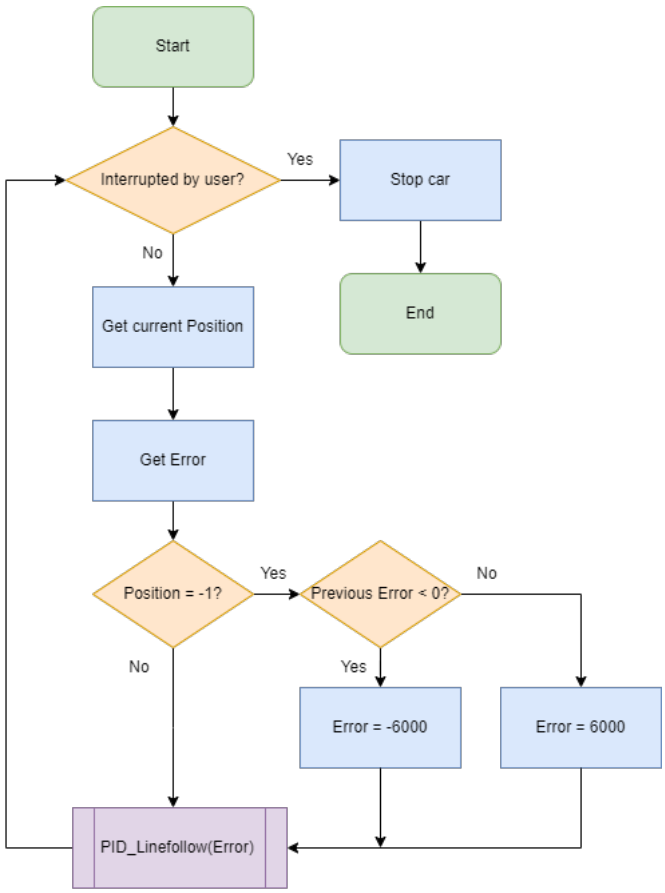


Figure 7. Main loop of the PID controller.

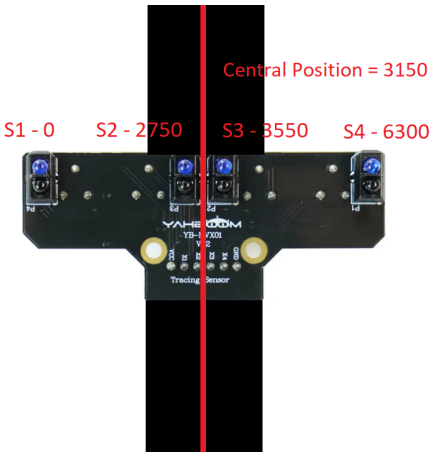


Figure 8. Diagram of the weighted average used by the system and the error calculation.

Finally, with the distance calculated, the error can be determined by subtracting the desired value (3150) from the current position, $Error = 3150 - Position$. In this way, if the error is 0, the sensor is centered with the line; if it is negative, the line is to its right; and, if it is positive, to its left. Furthermore, if the previous function returns -1, that is, the

sensor 'loses' the line, the error will be adjusted taking into account the previous error. If the previous error is negative, it will be changed to -6000, and, if positive, it will be set to 6000, as described in the implementation of Figure 7.

With the error determined, this value can be used as the parameter for the function `PID_Linefollow`. Figure 9 illustrates the steps of this function:

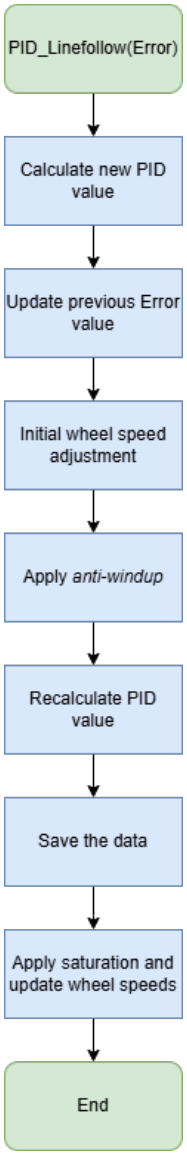


Figure 9. Steps of the PID controller.

- **Calculation of terms**: proportional ($P = error$), integrative ($I = I + error$), and derivative ($D = Error - previousError$), multiply by their respective gains, and add them to calculate the PID value.
  The gains were obtained experimentally, trying several combinations so that oscillations were minimized and responsiveness optimized. In this case, the gains are $K_p = 0.04$, $K_i = 0.00002$, and $K_d = 0.035$.
- **Initial wheel speed adjustment**: a default value is set for the motors to turn, in this case 70, called the base speed. The adjustment will add the result of the PID to the right-hand motors and subtract it from the left-hand motors so that the average value of 70 is maintained in the forward direction of the robot.
- **Apply the anti-windup** mechanism to ensure that the integrative action does not reach very high values, which could lead to the robot, when it loses the line, spinning

indefinitely, unable to get back on track. In this way, if the wheel speeds exceed the acceptable limit (less than −255 or more than +255), the amount by which these speeds exceed the limits will be added to the anti-windup value.

- **Recalculate the PID value**: due to the fact that the anti-windup changes the value of the integrative action, it is necessary to recalculate the output of the PID and the speed of the wheels.
- **Save data** for analysis, debugging, and then training the neural network.
- **Apply saturation** to the motor speeds so that they do not exceed the limits (−255 to +255), and finally apply these speeds to the motors.

The resulting scheme for the PID control is shown in Figure 10.

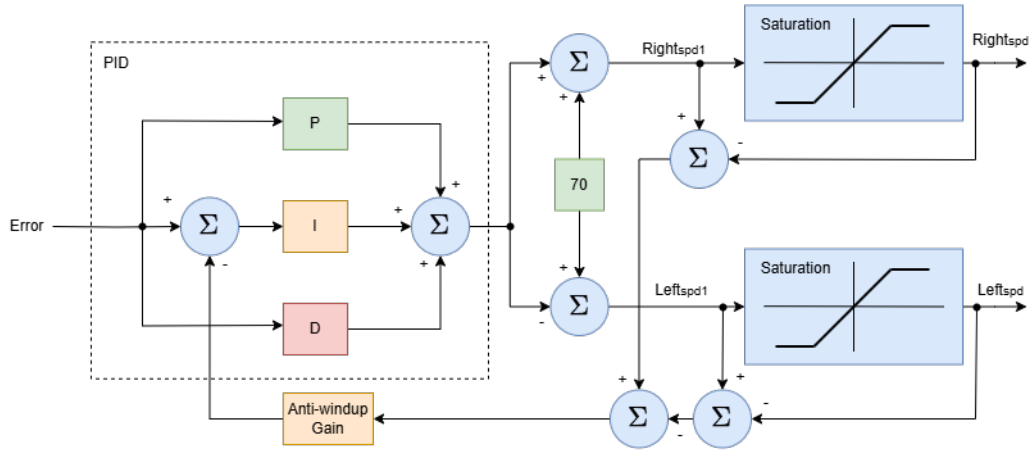


Figure 10. Block diagram of the implemented PID controller.

4.2. LSTM

With the data collected, it is now possible to train an LSTM neural network so that it can simulate the behavior of the PID controller. To accomplish this, it is necessary to choose a suitable model for this implementation.

Since the PID controller takes previous values to calculate the new output value, it is important to consider a model that has the ability to use past values to calculate new motor speeds. Therefore, an LSTM network was selected.

The implementation of an LSTM neural network involves several steps. First, the training data are prepared by organizing the CSV data, identifying the inputs and outputs, creating sequences, and splitting the data into training and test sets. The network is then defined by determining the depth, type, and size of each layer. The following step is the network training, which includes testing different configurations and depths while comparing the errors obtained in each case. Finally, the trained model is saved and can be transferred to the Raspberry Pi of the robot, where it will be used for control.

The LSTM model is designed to accurately deliver new left and right speeds to the motors, striking a balance between complexity and regularization to optimize performance while mitigating overfitting. Figure 11 illustrates the network model used in this implementation. Table 3 lists the network parameters.

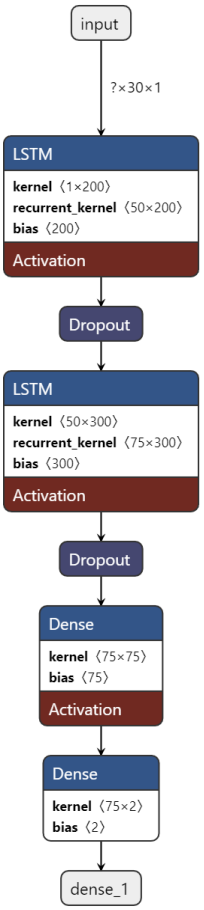


Figure 11. Diagram of the LSTM network used.

Table 3. Hyperparameters of the LSTM model.

Hyperparameter	Value
LSTM Units (Layer 1)	50
Input Shape	$(n\_steps, 1)$
Return Sequences (Layer 1)	True
Dropout (Layer 1)	0.1 (10%)
LSTM Units (Layer 2)	50
Return Sequences (Layer 2)	False
Dropout (Layer 2)	0.1 (10%)
Dense Units (Hidden)	50
Activation Function (Hidden)	ReLU
Dense Units (Output)	2
Optimizer	Adam
Loss Function	MSE (Mean Squared Error)
Batch size	32
Epochs	60
Validation Data	(X_test, y_test)

The network was trained along 60 epochs, that is, the number of times the entire training dataset is passed through the network; for that and to ensure that the model

is trained correctly, the dataset was divided into training and test data, the split was configured so that 80% of the data are used for training and 20% for testing. It was observed that training with more epochs enables the model to learn better, but too many can lead to overfitting. Figure 12 shows the evolution of the model error as it is trained.

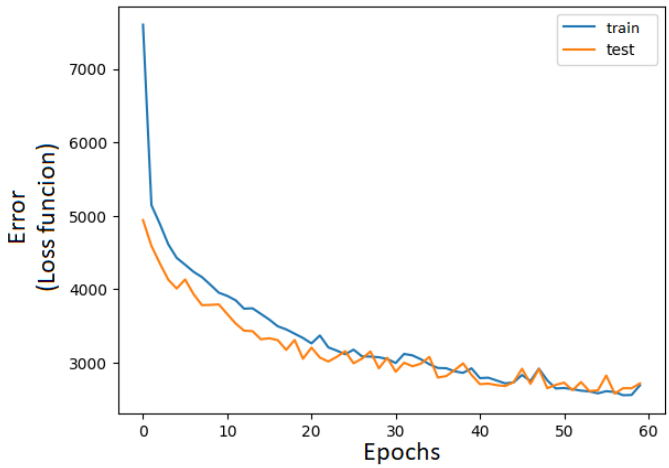


Figure 12. Evolution of the LSTM network's error during training.

The resulting controller is shown in Figure 13. Evidently, the PID controller and the anti-windup system were removed and replaced by the LSTM network.

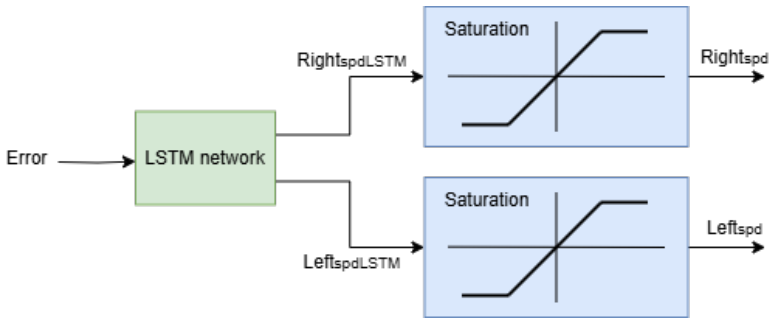


Figure 13. Block diagram of the implemented LSTM controller.

This approach enabled the robot to be controlled with a neural network using only infrared sensors as input. Its control is very similar to the PID control, which is expected, since data from that controller were used to train the neural network. However, existing similarities validate that the LSTM network was able to learn the associated patterns and temporal dependencies of the PID controller.

4.3. CNN

In the second experiment, the aim is to develop a platform for the robot to follow a line drawn on the ground using neural networks, as in the previous experiment, but using a camera. In this case, the Raspberry Pi camera mounted on the robot will point downward so that it can take successive images of the ground in front of it. These images will then be fed to a neural network that will control the robot movement along the path.

With this strategy, the use of an LSTM neural network would offer limited benefits compared to a CNN since the LSTM network was chosen for the previous experiment to simulate the behavior of the PID controller, which requires past information to calculate the new output. However, the use of real-time images to control the robot does not involve the temporal dependence found in the PID controller. However, the use of an LSTM network could easily, if configured with an incorrect number of steps for each sequence, lead to

the network memorizing the path it was trained to follow, capable of causing the robot to behave abnormally or incorrectly if placed on a different track. For these reasons, a CNN was chosen.

As in the first experiment, a large amount of data is required to train the neural network. In this case, the data will be a large number of images taken by the robot itself, so it is necessary to develop a method for obtaining these images.

As mentioned, the first step is to take several images of the path and save them for use in neural network training. This will be completed following the steps outlined in the flowchart of Figure 14:

- **Camera initialization and positioning**: During initialization, the camera is configured to capture images and is positioned correctly for the task. Additionally, the positions of the servos connected to the camera are adjusted to ensure that the camera is aligned pointing straight down for accurate image capture;
- **Taking a new image**: In each iteration of the loop, a new image is captured from the camera (Figure 15). This continuous image acquisition is crucial for real-time image processing;
- **Image processing for line detection**: The process begins by converting the color image to grayscale (Figure 16), which simplifies the complexity of the image and improves the effectiveness of binary thresholding. Binary thresholding is then applied (Figure 17), where a threshold value transforms the image into a binary representation, highlighting the relevant pixels in black and white. To optimize processing, the image is then cropped to keep only the top part (Figure 18), thus reducing unnecessary areas and improving the detection of the desired line near the robot, which is essential for the control operation;
- **Save image**: After processing, the final image is saved to a file. The file path is generated dynamically and the image is saved in .jpg format.

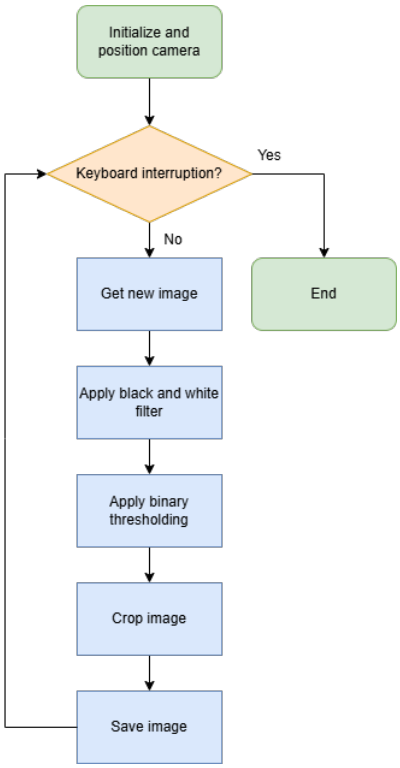


Figure 14. Process of capturing, processing, and saving images.

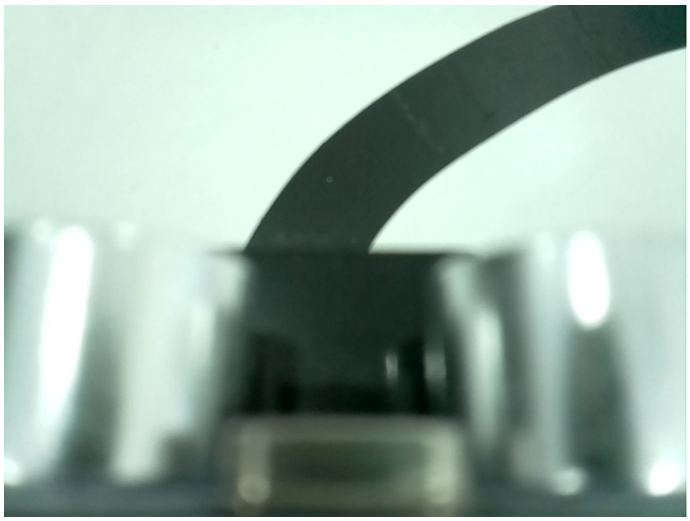


Figure 15. Example of a raw image.

Figure 16. Example of a grayscale image.

With this procedure, several images of the track are taken at different locations, positions, and angles to ensure that the neural network is trained to handle any situation the robot may encounter. The dataset has a size of 1800 images.

Figure 17. Example of an image with binary thresholding.

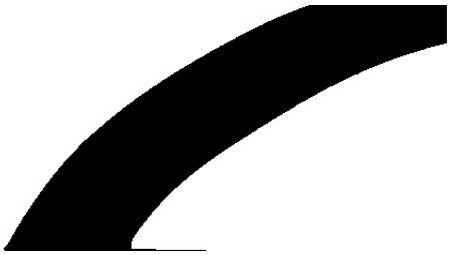


Figure 18. Example of a cropped final image.

With the dataset obtained, the next step is to classify the images so that they can be used to train the neural network. For this, each image will be assigned the action that the robot will have to complete. These actions will be divided into the categories listed in Table 4.

Table 4. Speed of the right and left motors for each category.

Category	Right Speed	Left Speed
forward	60	60
soft_left	50	70
soft_right	70	50
hard_left	30	90
hard_right	90	30
spin_left	−50	70
spin_right	70	−50
lost_line	-	-

The wheel speed values are between −255 and +255, where negative values cause the wheel to spin backward and positive values spin forward. Additionally, in the last category ('lost_line') corresponding to when the robot loses the line, the robot will move accordingly to the previous reading. So, if the vehicle was turning to the left on the previous reading, the robot, when losing the line, will spin to the left and vice versa to the right.

For the classification task, eight different folders were created, one with the name of each category, where all images taken in the previous step were manually classified. Although this classification method is simpler, it requires considerably more time. With this method, the robot will have eight different modes of operation. However, using more categories would lead to greater difficulty in classifying the *dataset* due to the greater ambiguity in distinguishing these categories.

With the dataset obtained and classified, it can finally be used to train the CNN network, as depicted in Figure 19:

- **Configure model**: In the first step, the CNN network is configured using the LeNet architecture, which is suitable for image classification tasks. The model is configured to receive 36 × 36-pixel grayscale images and classify them into one of eight categories, each representing a different driving command;
- **Load categorized images and assign labels**: The dataset containing images is loaded from the specified directories. Each image is read and resized to 36 × 36 pixels to match the input dimensions of the CNN. The images are stored in a list for further processing. The labels are derived from the directory structure: each folder corresponds to a driving command (e.g., 'forward' or 'soft_left'), which is then converted into numerical label;

- **Divide the dataset into training and test data**: To ensure a good generalization of the model, the dataset is split into training and test sets. The split is configured so that 75% of the data is used to train the model and the remaining 25% are reserved for testing. This split enables the model's performance to be assessed on data not observed after training;
- **Train the model**: The CNN is trained using the training data. To avoid overfitting, early stopping is implemented. This technique monitors validation loss and stops training if the model stops improving after a certain number of epochs, ensuring that the model does not overfit the training data and generalizes better;
- **Model evaluation**: After training, the model performance is evaluated by plotting the training and validation loss over iterations. These plots provide information on how well the model has learned to generalize. If the validation loss continues to decrease, this indicates that the model is improving;
- **Save the model**: Finally, after the model has been successfully trained, it is saved to disk. This enables the model to be used in the future.

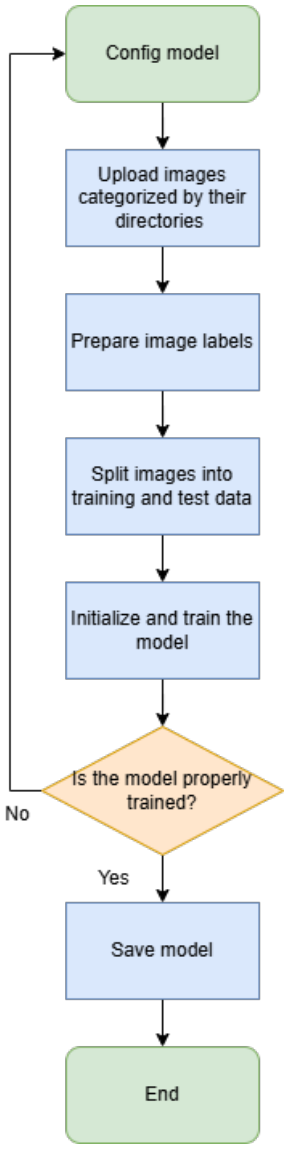


Figure 19. Process of training the CNN.

The CNN model used is shown in Figure 20, and its hyperparameters are provided in Table 5.

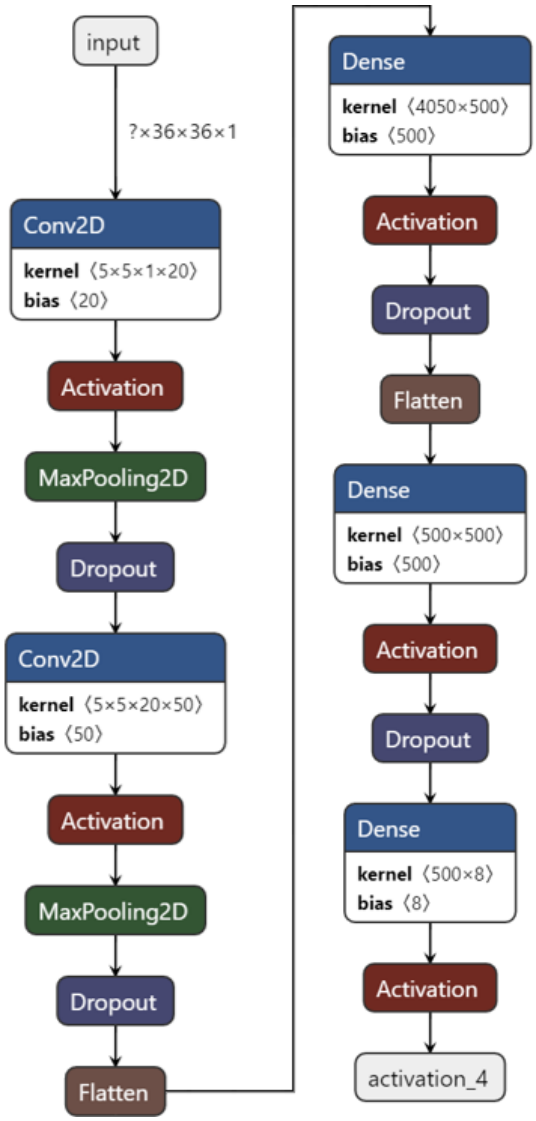


Figure 20. Structure of the CNN used.

Table 5. Hyperparameters of the LeNet model.

Hyperparameter	Value
Input Format	(height, width, depth)
Number of Convolutional Layers	2
Filters in First Convolutional Layer	20
Filters in Second Convolutional Layer	50
Filter Size (Both Convolutional Layers)	(5, 5)
Activation Function	ReLU
Pooling Size	(2, 2)
Pooling Stride	(2, 2)
Dropout (After Pooling Layers)	0.25
Dropout (Before Classification Layers)	0.5
Number of Dense Layers	2
Units in First Dense Layer	500

Table 5. *Cont.*

Hyperparameter	Value
Units in Second Dense Layer	500
Units in Output Layer	Number of classes
Output Activation Function	Softmax

As mentioned, an early stopping method has been set up during the training of the model, which will automatically evaluate the model's performance. If the validation error rises during three successive epochs, the model's training will be interrupted. The evolution of the model performance can be observed in Figure 21.

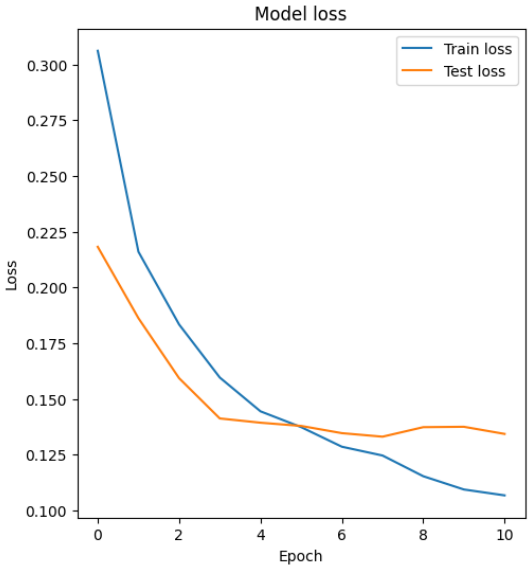


Figure 21. Evolution of the CNN error over the training.

Figure 22 shows the confusion matrix of the trained model, where each label value corresponds to the class indicated in Table 6. This shows how many times the model was able to correctly categorize the test data. Thus, the main diagonal shows the number of times the model correctly predicted the class, while the off-diagonal shows the number of times the model was incorrect, classifying one class as another. The performance of the system is satisfactory as it is able, in most cases, to correctly categorize the data with an accuracy (the ratio of correctly predicted instances (both positive and negative) to the total number of predictions) of 79.17%, precision (the ratio of correctly predicted positive observations to the total number of predicted positive observations) of 74.92%, recall (the ratio of correctly predicted positive observations to all actual positive observations) of 0.7375, and F1-score (the harmonic mean of accuracy and recall, providing a balance between the two) of 0.7360.

It should be noted that an accuracy target was not defined, as the objective was to strive for the highest accuracy possible within the scope and limitations of the project. Ultimately, the objective is that the robot is capable of navigating the track effectively and efficiently, which was met with an accuracy of 79.17%. Some ways to further improve this value would require more careful classification and a larger dataset.

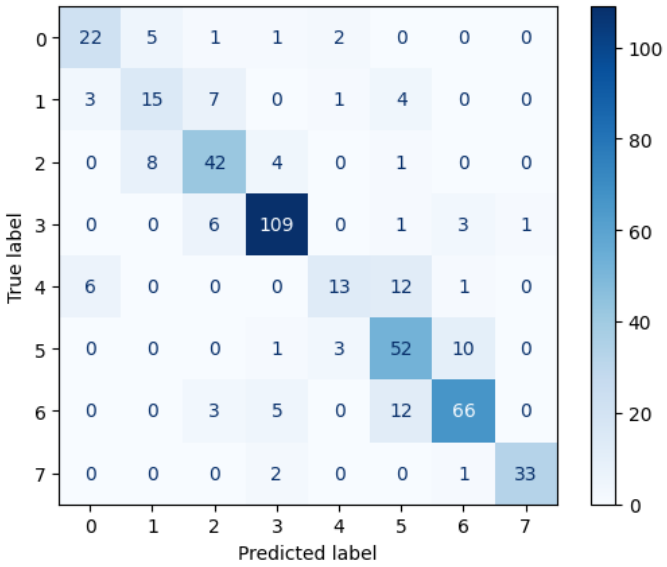


Figure 22. Confusion matrix of the CNN model.

Table 6. Number corresponding to each class of the CNN.

Class	Corresponding Label
forward	0
soft_left	1
hard_left	2
spin_left	3
soft_right	4
hard_right	5
spin_right	6
lost_line	7

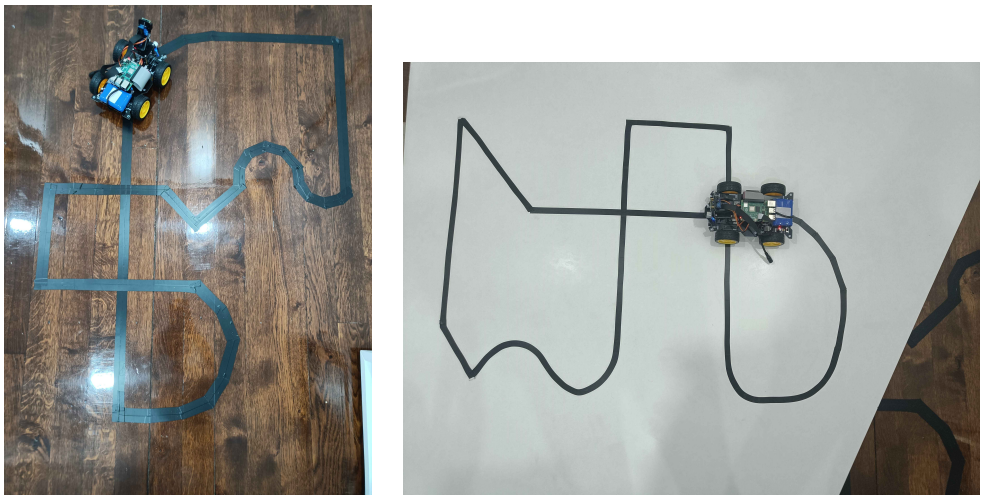
5. Results

This section presents the results of the implemented control methods using pre-built tracks to assess their performance.

5.1. Tracks

For testing, experimentation, and validation of the control methods implemented, two tracks were built, as illustrated in Figure 23. Each track was built in such a way as to fit the control methods developed. Both tracks have the same length of 475 cm.

The second track (Figure 23b) was built on a white platform so that noise from the camera feed could be reduced since the use of the camera for data collection during the first experiment would be greatly affected by the dark color, texture, and shininess of the wooden floor. The attempt to use the first track (Figure 23a) resulted in noisy post-processing images, making it difficult to distinguish the black line from the image. Other more advanced filters could have been implemented in the image processing step to avoid this problem. However, adding these filters would increase the processing overhead of the control loop, increasing the control loop execution time, and thus further decreasing the sampling rate.



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

Figure 23. (**a**) Track used for the first experiment and (**b**) track used for the second experiment.

5.2. PID

The performance of the PID control method is shown in Figure 24. For this experiment, the first track was used (Figure 23a). The figure illustrates the plots of the error, the output value of the PID with its terms P, I, and D, and the antiwindup effect over one lap of the track.

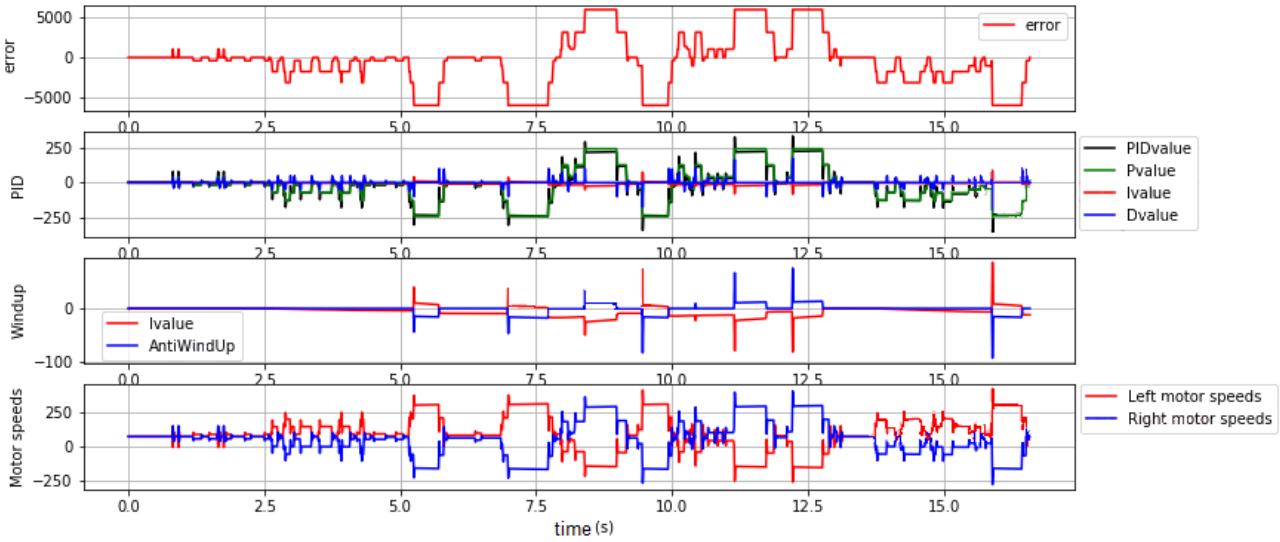


Figure 24. Plots of the error, PID terms, and corresponding output with the anti-windup scheme over one lap of the track.

5.3. LSTM

Figure 25 shows the results of the control with the LSTM network and the motor speeds on one lap of the track. The first plot displays the error in the infrared sensor readings, while the second plot shows the predictions generated by the LSTM network. Unlike the PID controller, where the new calculated value is added to or subtracted from the wheel speeds, the LSTM network directly outputs the wheel speeds. To compare the performance of both control methods, a reverse operation was performed: using the new wheel speeds, an approximate equivalent of the PID value was calculated. However, it is important to note that this approximation does not perfectly correspond to the actual PID value as the LSTM network is trained to emulate the behavior of the PID controller rather than replicate its calculations exactly. In fact, comparing the two control methods, we can

see that both are quite similar, as shown in Figure 26. Although some similarities can be observed, the differences in the initial conditions and variations throughout the experiment, even when using the same path, lead to notable differences between the two experiments.

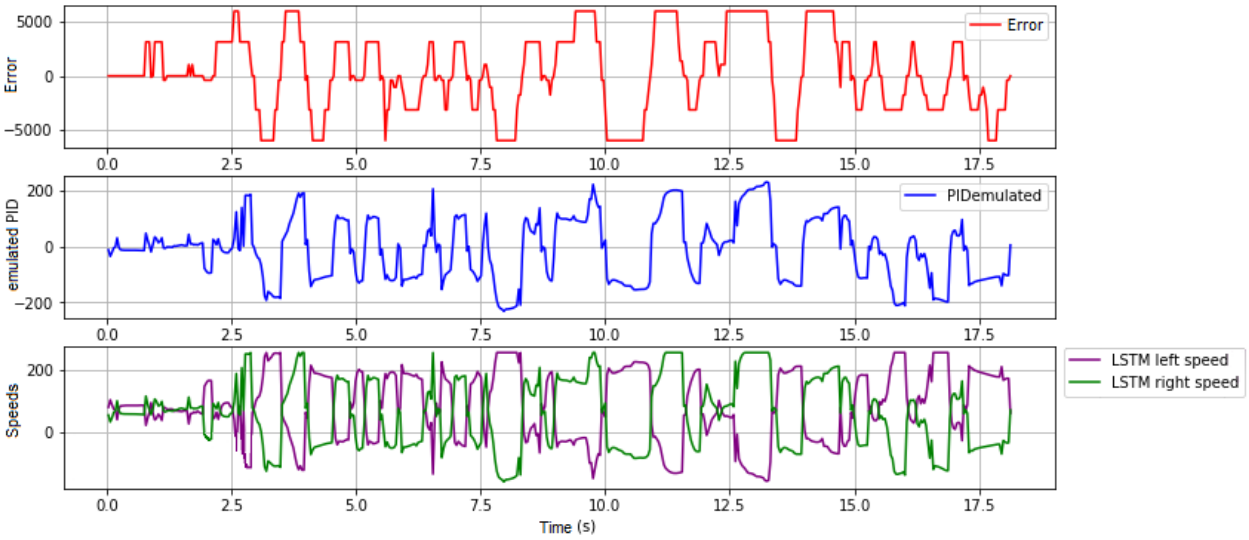


Figure 25. Plots of the control with the LSTM network and the motor speeds over one lap of the track.

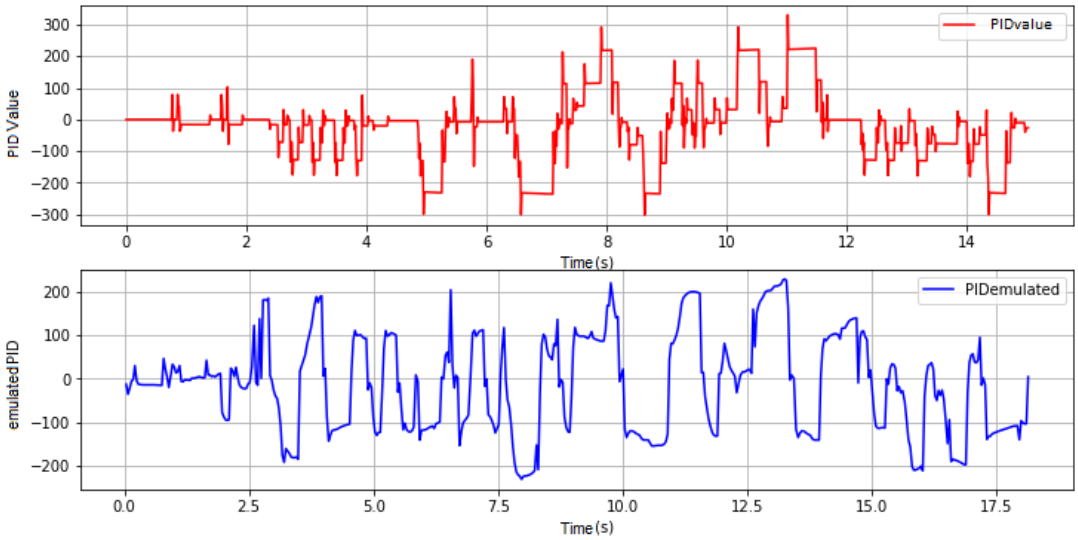


Figure 26. Comparison of the output of the PID and LSTM controllers.

5.4. CNN

The performance of the previous model was influenced by environmental and operational factors, which required adjustments to optimize behavior. One of the first changes was to replace the original track, placed on a shiny wooden floor that generated noise in the images, with a high-contrast black and white track (Figure 23b), which increased accuracy.

The lighting was also crucial because good lighting helps with track detection, but too much could cause excessive reflections. Adjusting the light intensity proved to be essential for consistent performance. Other sources of uncertainty can be attributed to the classification; as it was conducted manually, the ambiguity between classes led to some inaccuracy while classifying. The dataset could also be larger, which could result in a more accurate model with fewer errors.

Due to the high computational load of real-time image processing, the sampling rate was reduced to 24 frames per second, which required slowing down the robot to ensure that it had time to react to the track.

Figure 27 shows the graphs of the motor speeds and the CNN output over one lap of the track.

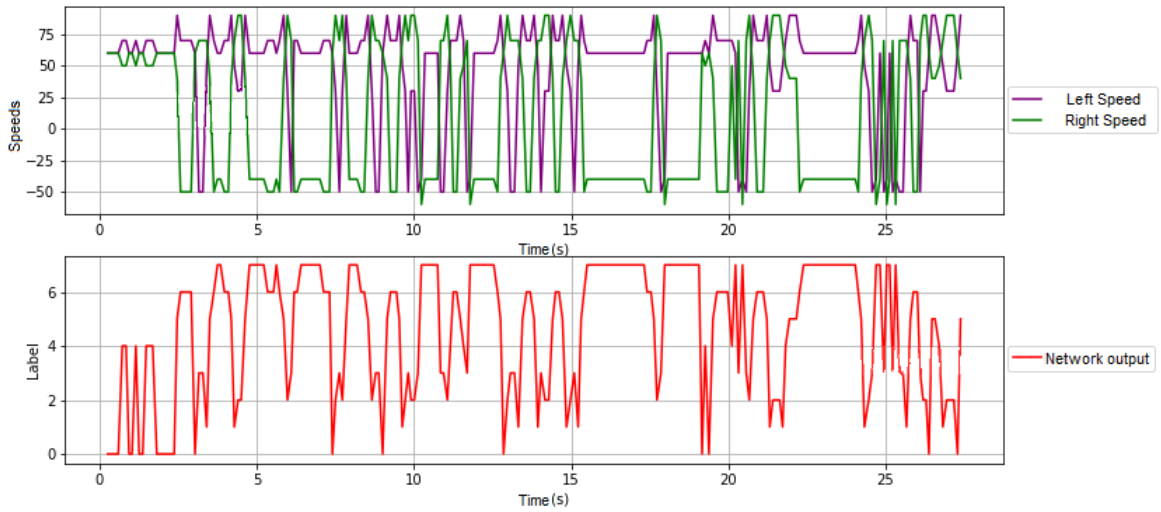


Figure 27. Plots of the motor speeds and the output of the CNN over one lap of the track.

6. Discussion

In general, all the methods used were able to efficiently control the robot along a line drawn on the ground. Consequently, both the PID controller and the LSTM and CNN networks proved to be feasible approaches for this type of control. However, due to the simple nature of the proposed line-following robot, PID control was revealed to be the most effective.

It was observed that, while the PID controller is the most efficient method, it is not perfect as it cannot overcome some obstacles on the track, such as sharp angles, which the CNN network can handle more easily. This shows that, although PID has more advantages over the other methods in the line-following experiment, neural networks and deep learning are powerful tools that can be used in more complex scenarios.

Since the control methods are quite different from each other, it is difficult to obtain common metrics to compare the performance between them. Table 7 shows the error performance indices of Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), the time taken for each method to cover one lap of the track and the robot oscillations along the path. The sign "+" indicates the level of oscillations; the higher the number of "+", the greater the oscillation. These data were obtained by running each method five times and obtaining the average values and standard deviations for each of the control methods. As the values obtained in each test run were quite consistent between each other, the somewhat arbitrary number of five test runs for each method was chosen. This value represents a fair equilibrium between the execution time required to execute each run and the variation in the standard deviation.

Table 7. Performance results.

Method	MSE	MAE	RMSE	Lap Time (s)	Oscillations
PID	10,684,111 ± 515,855	2257 ± 71	3267 ± 79	16.40 ± 0.22	+
LSTM	17,158,701 ± 1,598,305	3435 ± 242	4137 ± 192	20.53 ± 1.04	+++
CNN	21,040,166 ± 1,186,665	3857 ± 188	4585 ± 128	25.96 ± 0.29	++

As mentioned, in a central position, the error of the sensor reading is 0, while when it loses the line, the error becomes ±6000, with any value in this range defining an intermediate position value. With this approach, the error values, especially the MAE, of the PID controller are quite good compared to the other methods. Although the LSTM method exhibits higher errors, it is still satisfactory when evaluated against the other controllers.

It should be noted that the indices for the CNN method do not accurately reflect the network's performance because it does not use the infrared sensors for control that were used to calculate the error, but rather the camera, so the infrared sensors may not align with the line. In addition, as the line in the second path is less wide, in a central position, no sensor would detect the line, which will cause an increase in error.

Furthermore, the sample rate of each method was also measured, that is, how many samples each method can evaluate from the sensors and return new motor speeds. The sample rate was set to the highest value that the processor could handle. The PID method had a sample rate of 80 Hz, LSTM 25 Hz, and CNN 24 Hz.

7. Conclusions

This paper investigated the use of neural networks in the control of a mobile line-follower robot using the LSTM and CNN approaches.

Various control methods based on deep learning and PID were developed and tested on the robot, resulting in a comparative analysis between the techniques. This comparison demonstrated the advantages of neural networks for complex paths despite their greater sensitivity to factors such as lighting and computational load, which required a reduced sampling rate and limited speed.

The first approach, based on infrared sensors and using an LSTM model, proved to be efficient for simple paths, with a higher sampling rate and reduced dependence on environmental conditions. The second approach, which used a camera and a CNN, was more effective for complex paths but revealed to be sensitive to noise and reflections from the environment, requiring careful calibration. Another source of uncertainty was the classification process, which, being performed manually, is not free from biases.

Compared to the PID controller, neural networks showed better adaptation to difficult paths but with high computational complexity. PID remains practical for simple controlled paths, while neural networks offer great potential for AGVs in dynamic and unpredictable scenarios.

For future work, we can consider creating a simulation environment that allows us to test different scenarios, tracks, and configurations more quickly and under controlled conditions, thus reducing time and resources. The use of non-linear controllers over the traditional linear PID controllers could be an interesting research area. For better performance and reliability, a hybrid approach could be used that combines the neural networks of LSTM and CNN with the mathematical model of the robot. Another improvement would be the use of pre-trained networks, such as those trained on large datasets for autonomous navigation or path detection. To achieve this, the use of transfer learning (adapting an already trained network for new tasks) could significantly reduce training time and improve model accuracy under certain conditions. Testing the robot in more com-

plex environments, simulating industrial or urban scenarios, would also make it possible to validate the robustness of the models under more realistic and demanding conditions.

Author Contributions: Conceptualization, H.M.L. and R.S.B.; methodology, H.M.L. and R.S.B.; software, H.M.L.; validation, H.M.L. and R.S.B.; formal analysis, H.M.L.; investigation, H.M.L.; writing—original draft preparation, H.M.L.; writing—review and editing, H.M.L., R.S.B. and I.S.J.; visualization, H.M.L.; supervision, R.S.B. and I.S.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original data presented in the study are openly available from GitHub repository at https://github.com/HugoMLeal/TEDI1190660 (accessed on 13 January 2025).

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

References

1. Jain, N.; Kumar, R. A Review on Machine Learning and It's Algorithms. *Int. J. Soft Comput. Eng.* **2022**, *12*, 399–406. [CrossRef]
2. Sarker, I. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN Comput. Sci.* **2021**, *2*, 420. [CrossRef] [PubMed]
3. LeCun, Y.B. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
4. Moshayedi, A.; Li, J.; Liao, L. AGV (automated guided vehicle) robot: Mission and obstacles in design and performance. *J. Simul. Anal. Nov. Technol. Mech. Eng.* **2019**, *12*, 5–18.
5. Wong, C.; Soon, C. A Bluetooth and Vision Controlled Automatic Guided Vehicle. *Emerg. Adv. Integr. Technol.* **2022**, *3*, 53–63. Available online: https://publisher.uthm.edu.my/ojs/index.php/emait/article/view/11856 (accessed on 12 October 2024).
6. Lin, K.; Chen, H.; Li, G.; Tu, J.; Huang, S. Automatic Guided Vehicle with Artificial Intelligence Navigation. In Proceedings of the 2019 IEEE International Conference On Consumer Electronics—Taiwan (ICCE-TW), Taiwan, China, 20–22 May 2019; pp. 1–2.
7. Roche, J.; De-Silva, V.; Kondoz, A. A Multimodal Perception-Driven Self Evolving Autonomous Ground Vehicle. *IEEE Trans. Cybern.* **2022**, *52*, 9279–9289. [CrossRef] [PubMed]
8. Aryanti, A.; Wang, M.; Muslikhin, M. Navigating Unstructured Space: Deep Action Learning-Based Obstacle Avoidance System for Indoor Automated Guided Vehicles. *Electronics* **2024**, *13*, 420. Available online: https://api.semanticscholar.org/CorpusID:267145921 (accessed on 12 October 2024). [CrossRef]
9. Li, X.; Rao, W.; Lu, D.; Guo, J.; Guo, T.; Andriukaitis, D.; Li, Z. Obstacle Avoidance for Automated Guided Vehicles in Real-World Workshops Using the Grid Method and Deep Learning. *Electronics* **2023**, *12*, 4296. Available online: https://api.semanticscholar.org/CorpusID:264328025 (accessed on 12 October 2024). [CrossRef]
10. Chen, Y.; Li, D.; Zhong, H.; Zhao, R. The Method for Automatic Adjustment of AGV's PID Based on Deep Reinforcement Learning. *J. Phys. Conf. Ser.* **2022**, *2320*, 012008. Available online: https://api.semanticscholar.org/CorpusID:251515556 (accessed on 12 October 2024). [CrossRef]
11. Simon, J.; Trojanová, M.; Hovsovská, A.; Sárosi, J. Neural Network Driven Automated Guided Vehicle Platform Development for Industry 4.0 Environment. *Teh.-Vjesn.-Tech. Gaz.* **2021**, *28*, 1936–1942. Available online: https://api.semanticscholar.org/CorpusID:243855693 (accessed on 12 October 2024).
12. Carrio, A.; Sampedro, C.; Rodriguez-Ramos, A.; Campoy, P. A Review of Deep Learning Methods and Applications for Unmanned Aerial Vehicles. *J. Sens.* **2017**, *2017*, 3296874. Available online: https://onlinelibrary.wiley.com/doi/abs/10.1155/2017/3296874 (accessed on 12 October 2024). [CrossRef]
13. Bałazy, P.; Gut, P.; Knap, P. Positioning algorithm for AGV autonomous driving platform based on artificial neural networks. *Robot. Syst. Appl.* **2021**, *1*, 41–45. Available online: https://api.semanticscholar.org/CorpusID:237210213 (accessed on 12 October 2024). [CrossRef]
14. Xue, H.; Hein, B.; Bakr, M.; Schildbach, G.; Abel, B.; Rueckert, E. Using Deep Reinforcement Learning with Automatic Curriculum earning for Mapless Navigation in Intralogistics. *arXiv* **2022**, arXiv:abs/2202.11512. Available online: https://api.semanticscholar.org/CorpusID:247058694 (accessed on 12 October 2024).
15. Raspbot Repository. Available online: http://www.yahboom.net/study/Raspbot (accessed on 20 November 2024).
16. TTmotors. Available online: https://www.ttmotor.com/10-20mm/ (accessed on 13 January 2025).
17. Einstronic. Available online: https://einstronic.com/product/sg90-micro-servo-motor/ (accessed on 13 January 2025).
18. Raspberry Pi Hardware. Available online: https://www.raspberrypi.com/products/ (accessed on 20 November 2024).
19. NVIDIA. Available online: https://www.nvidia.com/en-eu/geforce/graphics-cards/30-series/rtx-3060-3060ti/ (accessed on 13 January 2025).

20. AMD. Available online: https://www.amd.com/en/support/downloads/drivers.html/processors/ryzen/ryzen-3000-series/amd-ryzen-7-3700x.html#amd_support_product_spec (accessed on 13 January 2025).
21. Anaconda Enterprise 4 Repository. Available online: https://docs.anaconda.com/anaconda-repository/ (accessed on 18 December 2024).
22. Zhang, A.; Li, M.; Smola, A.J.; Lipton, Z. *Dive into Deep Learning*; Cambridge University Press: Cambridge, UK, 2023. Available online: https://D2L.ai (accessed on 27 November 2024).
23. Abdullahi, Z.; Danzomo, B.; Abdullahi, Z. Design and Simulation of a PID Controller for Motion Control Systems. *Iop Conf. Ser. Mater. Sci. Eng.* **2018**, *344*, 012016. [CrossRef]
24. Tehrani, K.; Mpanda, A. PID Control Theory. 2012. Available online: https://api.semanticscholar.org/CorpusID:9908243 (accessed on 13 October 2024).
25. Inc, M. AVR221: Discrete PID Controller on tinyAVR and megaAVR Devices. Available online: https://www.ni.com/docs/en-US/bundle/labview/page/fuzzy-controllers.html (accessed on 12 March 2024).
26. Jesus, I.S.; Barbosa, R.S. Genetic optimization of fuzzy fractional PD+I controllers. In *ISA Transactions*; Elsevier: Amsterdam, The Netherlands, 2015; Volume 57, pp. 220–230.
27. Dorf, R.C.; Bishop, R.H. *Modern Control Systems*, 14th ed.; Pearson Education: London, UK, 2022.
28. Aboelhassan, A.; Abdel-Geliel, M.; Zakzouk, E.; Galea, M. Design and Implementation of Model Predictive Control Based PID Controller for Industrial Applications. *Energies* **2020**, *13*, 6594. Available online: https://api.semanticscholar.org/CorpusID:230600853 (accessed on 13 October 2024). [CrossRef]
29. Åström, K.J.; Hägglund, T. *PID Controllers: Theory Design and Tuning*; ISA Press: Research Triangle Park, NC, USA, 1995.
30. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. Available online: https://api.semanticscholar.org/CorpusID:1915014 (accessed on 13 October 2024). [CrossRef] [PubMed]
31. colah's Blog: Understanding LSTM Networks. Available online: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed on 26 December 2024).
32. Sak, H.; Senior, A.; Beaufays, F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. *arXiv* **2014**, arXiv:1402.1128.
33. Alzubaidi, L.; Zhang, J.; Humaidi, A.; Al-dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M.; Al-Amidie, M.; Farhan, L. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *J. Big Data* **2021**, *8*, 53. Available online: https://api.semanticscholar.org/CorpusID:232434552 (accessed on 14 October 2024). [CrossRef] [PubMed]
34. Upreti, A. Convolutional Neural Network (CNN): A comprehensive overview. *Int. J. Multidiscip. Res. Growth Eval.* **2022**, *3*, 488–493. Available online: https://api.semanticscholar.org/CorpusID:251888217 (accessed on 14 October 2024). [CrossRef]