Visual Servoing of A Line Following Robot

A Project Report

submitted in partial fulfillment of the requirements for the

Personal Professional Project

in

Industrial IT and Automation Engineering (IIA 3)

by

Malek Zitouni

Mohammed Karim Ben Boubaker

Under the supervision of

Mrs. Amani AYEB

Department of Physics and Instrumentation Engineering

National Institute of Applied Science and Technology

Tunis, Tunisia.

May 2024

Acknowledgement

We appreciate our supervisor, Mrs. Amani Ayeb, for her great patience and valuable insights, as well as Mr. Boulila, the jury president, for his support. We are also thankful to everyone who provided advice and encouragement throughout this project.

Additionally, we want to express our gratitude to our university for giving us the necessary resources and environment to be able to complete this project. We have greatly benefited from the guidance and knowledge shared by our instructors and peers, who have been essential in guiding us on our journey.

Contents

Acknowledgement 2

1 Introduction **4**

2 Hardware Design **5**

2.1 Overview 5

2.2 Mechanical Design 5

2.2.1 Design phase 6

2.2.2 Robot Structure 6

2.3 Electrical Design 7

2.3.1 Components 7

2.3.2 Circuit Diagram 8

2.4 Conclusion 8

3 Software Design **9**

3.1 Overview 9

3.2 Algorithms and Methods for Lane Detection 10

3.2.1 Thresholding 10

3.2.2 Warping 11

3.2.3 Pixel Summation 13

3.3 Algorithms and Methods for Lane Keeping 14

3.3.1 Steering Angle Calculation 14

3.3.2 Motors Control Algorithm 15

3.4 Deep Learning Integration 16

3.5 Conclusion 18

5 General Conclusion **19**

6 Bibliography **20**

Introduction

As a human driver, it's hard to keep the vehicle in the correct lane while maintaining a safe gap with the front vehicle for an extended period, especially with the presence of distracting technologies that can lead to accidents. According to a 2022 World Health Organization report, around 1.3 million people die yearly from road traffic accidents, calling the transportation industry to develop autonomous systems.

An essential component of autonomous vehicles is the Advanced Driver Assistance System (ADAS), designed to enhance safety. ADAS includes features like Adaptive Cruise Control, Automatic Braking/Steer Away, Lane-Keeping System, Blind Spot Assist, Lane Departure Warning System, and Lane Detection. These systems together monitor the vehicle's surroundings, reduce driver error, and improve overall safety. We provided an illustration that shows different sensors for the general ADAS [Figure 1].

This project will focus on the integration of Lane detection and Lane-Keeping systems using different computer vision and deep learning approaches for lane detection and different motor controlling techniques and algorithms for the lane-keeping system.

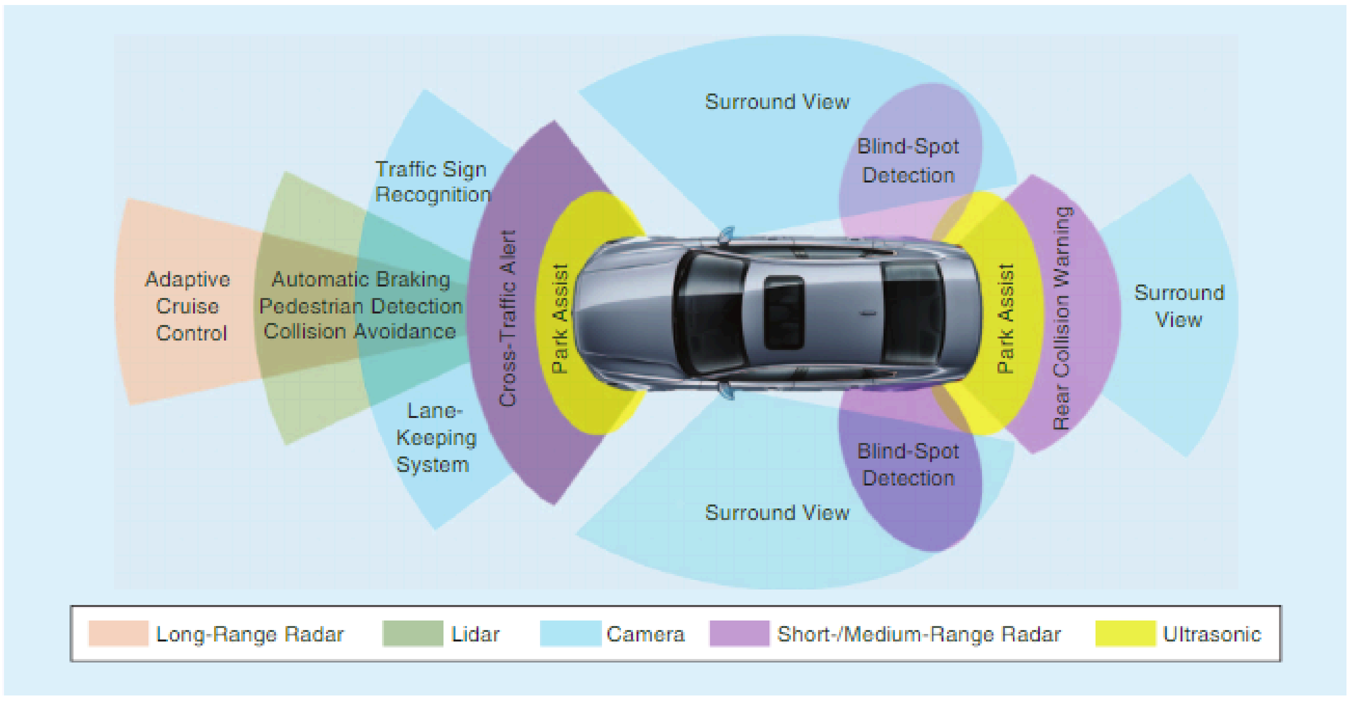


Figure 1 - Sensors used by the ADAS (Advanced Driver-Assistance System) [1]

We will start our project by designing the mechanical and electrical architecture that is necessary to test our software system, design choices will be studied and explained in the system design section, and then the software architecture, its implementation, and explanation of the used algorithms and methods such as Pixel Summation and thresholding will be presented in the software design section.

System Design

2.1 Overview

The integration of the lane detection and lane keeping systems requires a specific hardware and software setup. We can view these systems as separate units with a dedicated interface to ensure communication. Let's use block diagrams to represent our system's inputs and outputs. Each block represents a function that takes a specific input with a defined type and format. The output will also have a specific type and format, which will generally serve as the input for another block.

We have created a simplified illustration that explains the relationship between our system blocks. On the right are the software components, while on the left is the hardware component [Figure 2].

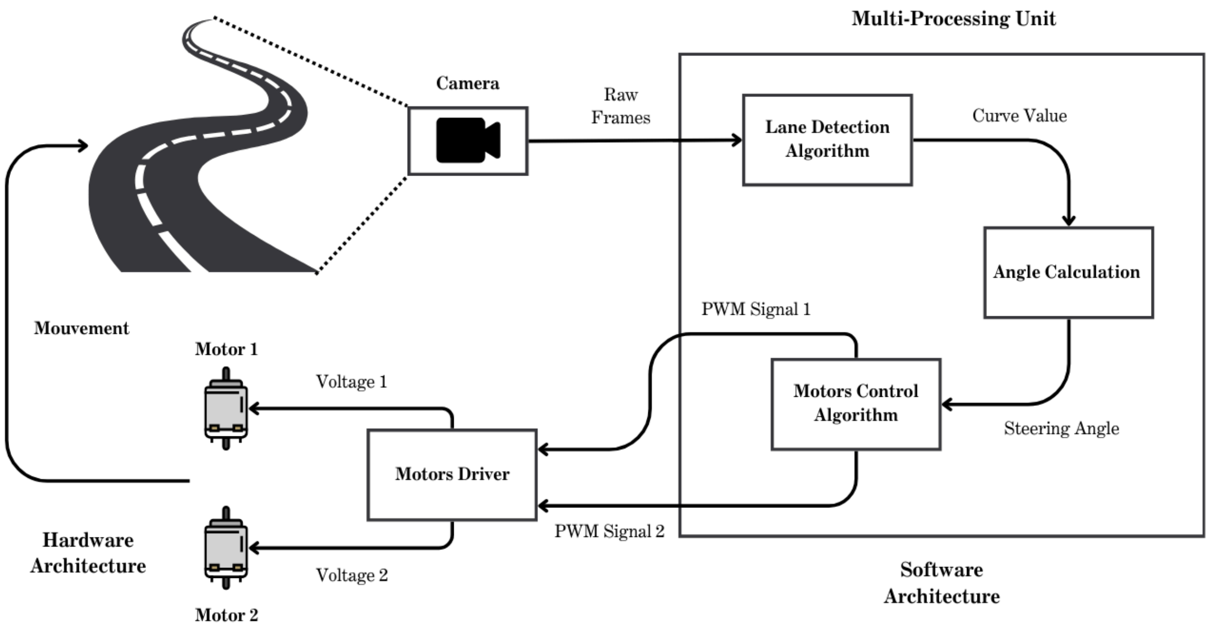


Figure 2 - Comprehensive System Diagram (Hardware and Software)

2.2 Mechanical Design

This section covers the physical structure and mechanical components of the prototype, our goal in this study is not to build a solid and durable mechanical structure, but rather a functional and reliable prototype that tests and validates our software approaches with minimal error and maximum stability.

2.2.1 Design phase

Our prototype will feature a 2-wheeled design with an additional point of contact that is represented by a ball caster wheel.
This specific design decision offers several advantages over other configurations. The 2-wheeled differential robot is simpler and more cost-effective to build and maintain. In Addition, the design provides excellent flexibility allowing the robot to rotate in curved and circular paths which is helpful for navigation in tight spaces.

Here's a comparative table that explains well our mechanical design decision [2]:

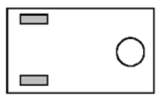
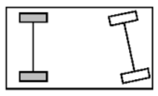
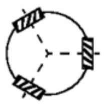
Configuration	Description	Advantages	Disadvantages	Application	Arrangement
2-Wheeled Differential Robot	Two driven wheels with a ball caster wheel for support	Simple construction, cost-effective, high maneuverability, can pivot in place	Limited stability on uneven terrain, requires precise balance	Indoor navigation, educational robotics	
4-Wheeled Robot	Two motorized wheels in the back, 2 steered and free wheels in the front	Stable, good traction, high load capacity	Complex control, less maneuverable	Outdoor exploration, industrial	
3-Wheeled Robot	Three motorized Swedish or spherical wheels arranged in a triangle	omnidirectional movement is possible	Complex control	Autonomous delivery, mobile robotics	

Table 1 - Comparative table between different robot wheel configurations, their descriptions, advantages, disadvantages and applications.

2.2.2 Robot Structure

The goal of the mechanical design process is to establish the correct dimensions and geometric features, ensuring the seamless integration of mechanical and electrical components with minimal manual machining.

The correct geometric measurements are crucial for the physical modeling that is used for software functionalities.

Our approach is to build the robot in two different stages. The first stage focuses on integrating most of the hardware, including the motors and their drivers. The second stage will cover the inside of the robot, providing a platform for camera support. The following figure [Figure 3] shows a real image of our robot, presenting what we described.

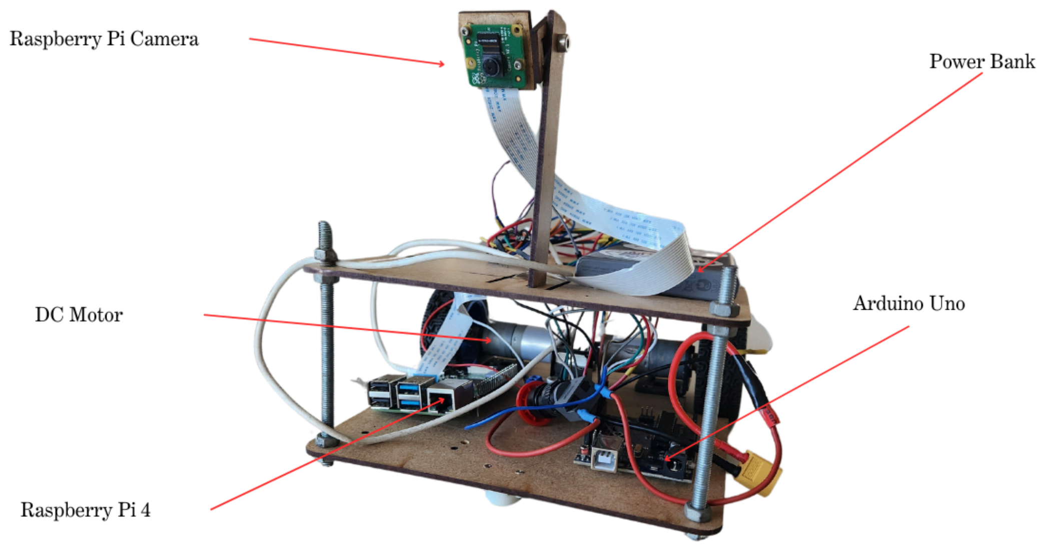


Figure 3 - Image of the Prototype

2.3 Electrical Design

2.3.1 Components

The current robot configuration uses two motors per wheel. Due to their simplicity and low cost, we have chosen DC motors with integrated Encoders. These motors are controlled through a motor driver circuit, which abstracts the directions and output signals by input pins and a PWM input pin. An external power supply is essential to power the DC motors. Since the maximum voltage of the DC motors is 12V, we will use 12V LiPo batteries for their ability to provide high current when needed.

The motor drivers will be controlled directly by a small Arduino Uno Connected to the Raspberry Pi 4, the need for a Raspberry pi is explained by the amount of processing power needed for image processing. Additionally, we will be using a Raspberry Pi camera to take pictures and stream video of the lane to our Computer.

Here's a summary table of the components:

Component	Purpose	Picture
DC Motors (12v)	Provide movement for the wheels	

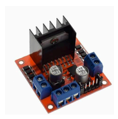
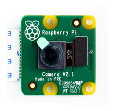
L298 Dual H-Bridge Motor Driver	Controls the DC motors	
12v LiPo Batteries	Power Supply for the DC motors	
Raspberry Pi 4	Controls motor drivers and processes complexe algorithms	
Raspberry Pi Camera	Captures images and streams video of the lane	

Table 2 - Summary of the Components used in our project and their purpose of use.

2.3.1 Circuit Diagram

The circuitry of our system involves direct connections between the PWM and GPIO pins of the Arduino and the motor driver, as well as connections between the power supply and the motors. Finally a USB connection between the Raspberry Pi 4 and the Arduino Uno to ensure the sending of the Steering Angle to the Arduino to Control the Motors.

2.4 Conclusion

Our mechanical structure was precisely designed to provide a solid platform for the seamless integration of the electrical components. This precision in CNC manufacturing ensured that all parts fit together without issues. After the success of the navigation test, we can conclude the readiness of our hardware for a software implementation.

The success of our hardware design provides a solid foundation for testing our software design. With the reliable performance of it, we can proceed with software development and debugging. This reliability helps to eliminate potential hardware-related issues during troubleshooting, allowing us to focus on optimizing our software.

Software Design

3.1 Overview

There are several techniques employed in lane detection systems, and we will focus on one approach comprising three stages of processing. These stages are based on digital image processing, also known as traditional computer vision, which utilizes various computer algorithms for image manipulation. The primary steps of this technique include Thresholding, Warping, and defining the Region of Interest (ROI). Further processing involves Pixel Summation (Histogram) followed by a custom Steering Angle Calculation Algorithm designed to our needs. Here is a diagram illustrating the relationship between the different stages of image processing techniques [Figure 4].

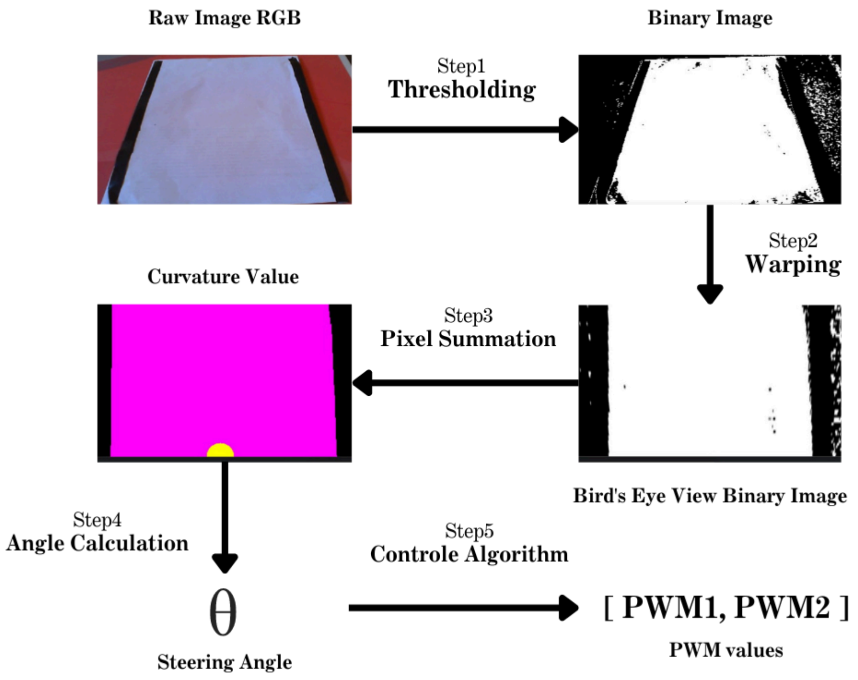


Figure 4 - Simplified illustration of the relationship between the different software stages

3.2 Algorithms and Methods for Lane Detection

3.2.1 Thresholding

3.2.1.1 Image Segmentation and Global Thresholding

Image segmentation is an important step in computer vision, especially for applications such as lane detection in autonomous driving. It involves subdividing an image into constituent sub-regions or distant objects that are easier to analyze.

Thresholding is an effective segmentation technique that converts an image into a binary format, simplifying the detection process.

Thresholding works by setting a specific threshold value that represents the pixel's intensity in foreground and background or 1 and 0 which are represented by white and black. Global thresholding applies a single threshold value T to the entire image I(x,y). The binary image B(x,y) is generated as

$$\begin{cases} 1 & \text{if } I(x,y) > T \\ 0 & \text{if } I(x,y) \leq T \end{cases}$$

3.2.1.2 Hue, Saturation, Value (HSV) Color Space

For lane detection, converting the image to the HSV (Hue, Saturation, Value) color space before thresholding is beneficial as it decouples color information (hue) from the intensity (value), making it robust against varying lighting conditions and shadows.

The conversion for an RGB image to the HSV color space is described by a transformation function:

$$I_{HSV}\left(x,y\right) = f_{RGB\rightarrow HSV}\left(I_{RGB}\left(x,y\right)\right)$$

3.2.1.3 HSV Thresholding

Once the image is in the HSV color space, thresholding can be applied to isolate the road from the rest of the objects. For white roads, the thresholds can be set as follows:

For lower whites, these define the minimum HSV values for a pixel to be considered part of the Lane:

$$White_{low} = (H_{low} = 0,\ S_{low} = 0,\ V_{low} = 180)$$

For upper whites, these define the maximum HSV values for a pixel to be considered part of the Lane:

$$White_{high} = (H_{high} = 180,\ S_{high} = 255, V_{high} = 255)$$

We can generate a Binary image where pixels within the specified HSV range are set to 1 (white), indicating the road, and all other pixels are set to 0 (black).

$$\begin{cases} 1 & if\, H\left(x,y\right) \in [H_{low}, H_{high}] \wedge S\left(x,y\right) \in [S_{low}, S_{high}] \wedge V\left(x,y\right) \in [V_{low}, V_{high}] \\ 0 & \text{otherwise} \end{cases}$$

Here is an example illustrating the thresholding applied on a raw image RGB [Figure 5]

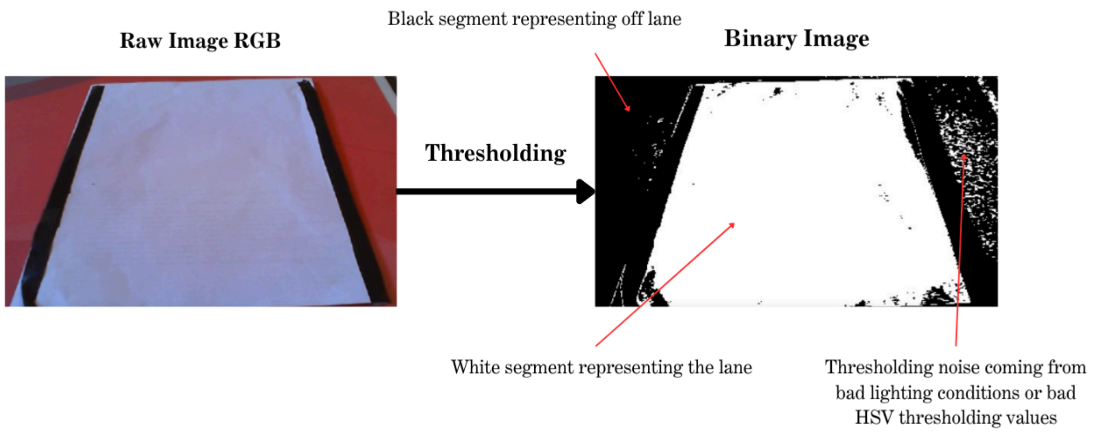

Raw Image RGB
Black segment representing off lane
Binary Image
Thresholding
White segment representing the lane
Thresholding noise coming from bad lighting conditions or bad HSV thresholding values

Figure 5 - example that illustrates the thresholding process, the left image is the original and the right image is the thresholded image.

3.2.2 Warping

3.2.2.1 Why Warping is important

The image obtained via the camera module has an angled perspective. The lane edges cannot be precisely located using this image as is. To fix this, the image warping concept is used. Image warping is a geometric transformation process that is used to get a birds-eye view of the path by changing the perspective of the image, [Figure 6] displays the effect of this technique.

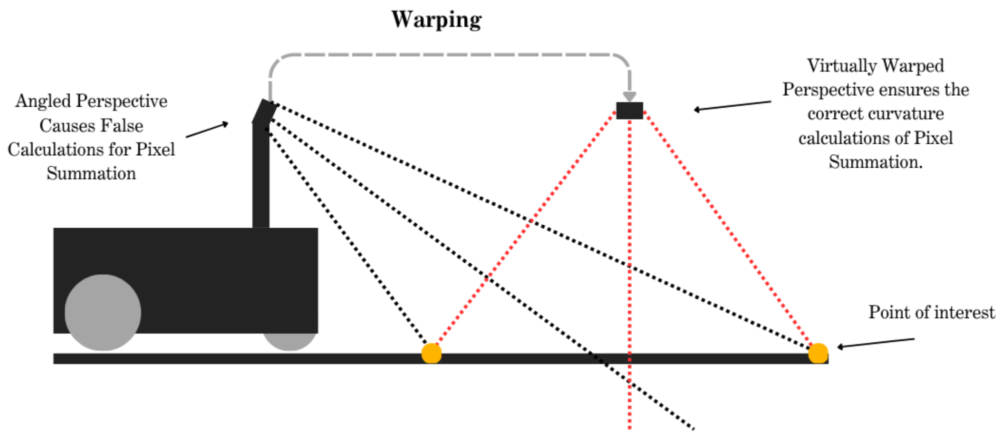

Warping
Angled Perspective Causes False Calculations for Pixel Summation
Virtually Warped Perspective ensures the correct curvature calculations of Pixel Summation.
Point of interest

Figure 6 - Illustration that explains the warping process, it is all about changing the perspective without changing the camera angle physically.

The warping process is also important as it's needed for the correct calculations of the converter done by the pixel summation algorithm, if we feed the algorithm with the wrong perspective it will give wrong values. So the transformation is mandatory in this case.

3.2.2.2 Perspective Transformation

The perspective transformation maps the coordinates of points in an image to new coordinates in another image, which finally changes the perspective.

This transformation can be represented mathematically by a homography matrix H. The new coordinates in the warped image are described by the following equation:

$$\begin{pmatrix} x' \\ y' \\ \alpha \end{pmatrix} = H \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Where:

- (x, y) are the coordinates of a point in the original image.
- (x', y') are coordinates of the corresponding point in the warped image.
- H is the 3x3 homography matrix
- α is a scaling factor

The matrix H is determined using a set of four points in the original image and their corresponding points in the desired bird's-eye view.

It can be calculated by setting a system of 8 linear equations. These equation are derived from the relationship between the coordinates of points in the two images and solve for the 9 unknown elements of the homography matrix

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4x'_4 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_1y'_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_1y'_3 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_1y'_4 \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = h_{33} \begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \\ x'_4 \\ y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \end{pmatrix}$$

Where:

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

Here is an example illustrating the warping in our project [Figure 7] :

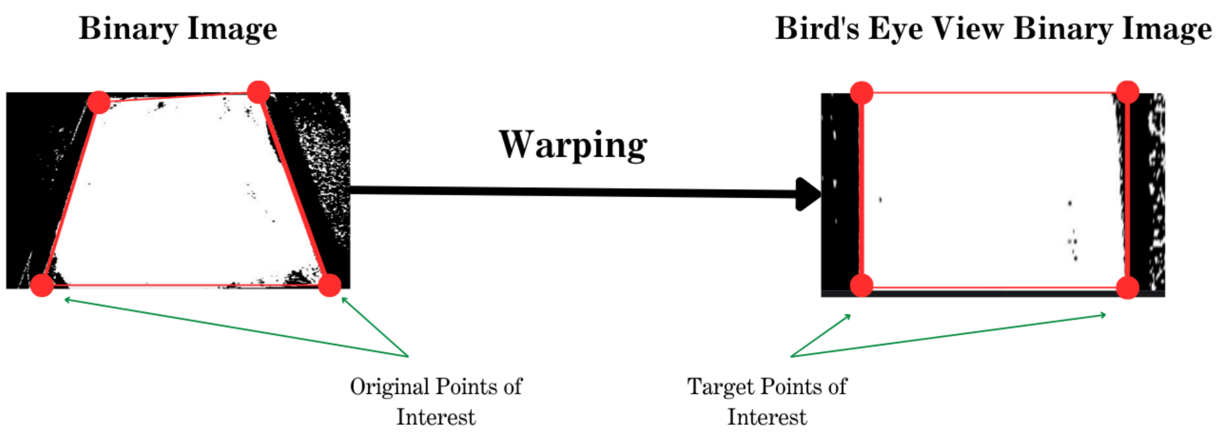

Binary Image
Bird's Eye View Binary Image
Warping
Original Points of Interest
Target Points of Interest

Figure 7 - Example of the effect of the Warping transformation for the original image which is on the left (Binary Image) and the output on the right (Bird's Eye View Binary Image).

3.2.3 Pixel Summation

The pixel summation algorithm works on binary thresholded and warped images to identify the curvature of a lane or path. This method involves summing up all the white pixels in each vertical column of the image. Mathematically, for a column j in the image with height H, the sum of white pixels Sj is calculated as :

where ***I(i,j)*** represents the intensity value of the pixel at position ***(i, j)*** . In our case, the intensity of Each pixel in the image is equal to 1 or 0 since the image is binary.

$$S_j = \sum_{i=1}^{H} I\left(i, j\right)$$

When lanes are present, there's a dense concentration of white pixels, resulting in a higher sum per column. By completing this calculation, we can spot lane geometry by detecting peaks corresponding to columns with a significant count of white pixels.

To calculate the curvature, we first determine the Lane Center. This is achieved by averaging the values of the histogram for the lower 1/4th part of the image, representing the base point or center of the lane. Mathematically, the Lane Center ***LC*** is calculated by averaging the sums ***S'j*** for rows in the lower ***25%*** part of the image.

$$LC = \frac{1}{W} \sum_{j=1}^{W} S'_j$$

Where ***W*** is the width of the image.

Next, we calculate the curve value by computing the average value of the histogram for the entire image. This average value represents the middle point. Mathematically, the Middle Point ***MP*** is calculated as:

$$MP = \frac{1}{W} \sum_{j=1}^{W} S_j$$

Subtracting the base point from the middle point gives us the curve value, which provides insights into the curvature of the lane. Mathematically, the Curve Value ***CV*** is given by:

$$CV = MP - LC$$

Here is an illustration that explains the algorithm [Figure 8] :

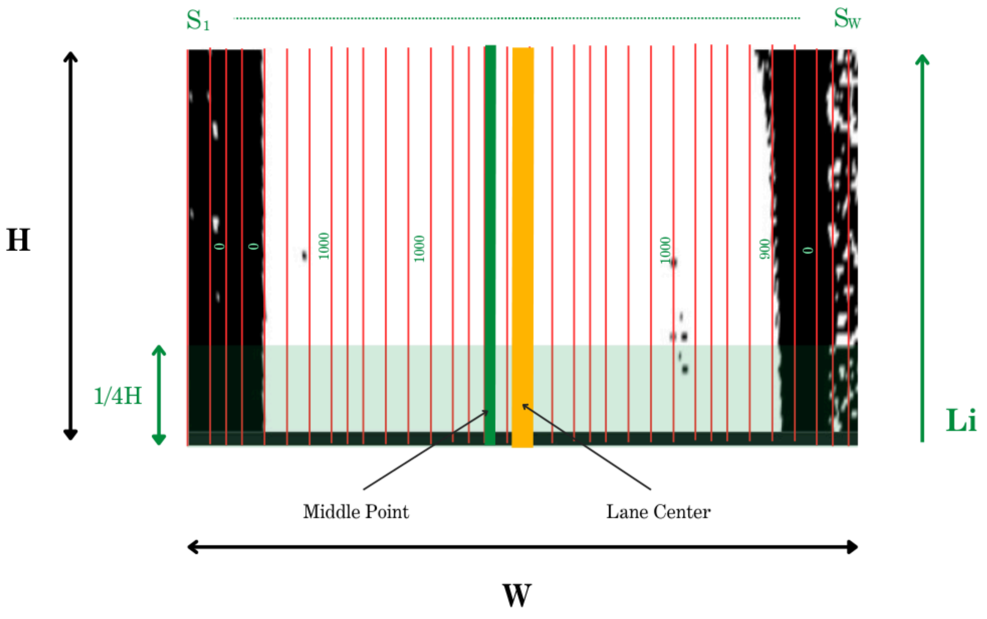


Figure 8 - Illustration of the Pixel Summation Algorithm for calculating the Curve Value

3.3 Algorithms and Methods for Lane Keeping

3.3.1 Steering Angle Calculation

The steering angle is determined based on the curve value. The relationship between the curve value and the steering angle can be modeled linearly or non-linearly depending on the robot's kinematics. We adopt a linear model for simplicity.

$$\theta = k \cdot C$$

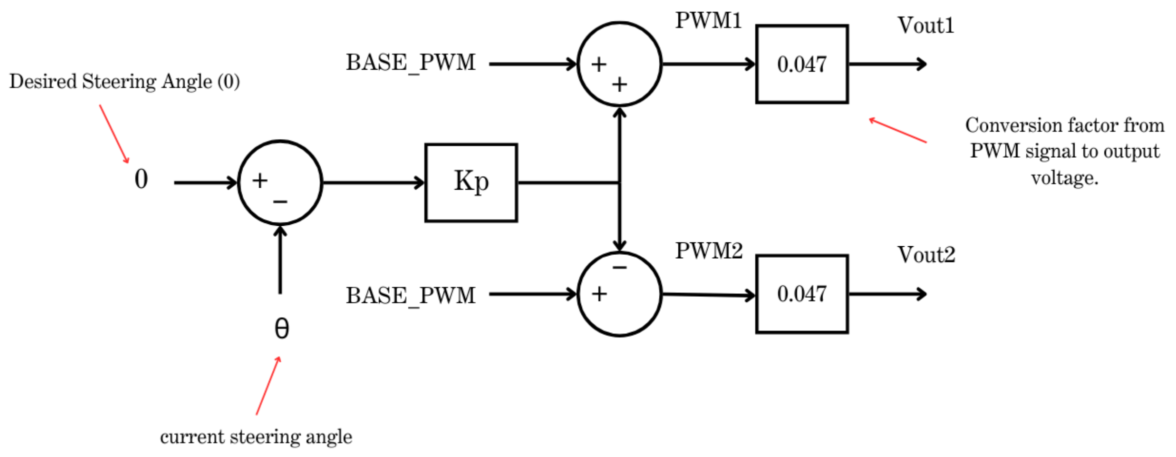
Where θ***(deg)*** is the Steering Angle, ***k*** is the proportional gain and ***C*** is the Curve Value.

Our objective is to ensure that the steering angle remains within the range of [-90, 90]. Since the values of the curve value are not predetermined we must determine the proportional gain k experimentally. This involves calculating curve values for the extreme or maximal curve values encountered during operating and mapping these values to the corresponding steering angles.

$$k = \frac{\frac{90}{C_{max}} + \frac{-90}{C_{min}}}{2}$$

3.3.2 Motors Control Algorithm

For motor control, we will use another controller that converts the steering angle into PWM1 and PWM2 signals that adjust the steering mechanism of the robot, here is the block diagram that explains the controller:



Desired Steering Angle (0)
0
θ
current steering angle
Kp
BASE_PWM
BASE_PWM
PWM1
PWM2
0.047
0.047
Vout1
Vout2
Conversion factor from PWM signal to output voltage.

Figure 9 - Block diagram of the Motors Controller implemented in Arduino Uno.

One method of calculating the proportional gain ***Kp*** is to set the maximum value of the ***PWM*** knowing that the maximum steering angle *θ* is 90. Since we know that the maximum value of ***PWM*** is 255, we will choose 200 as the maximum value for ensuring stability and we will choose the base **PWM** as 80.

We can determine the Proportional Gain ***Kp*** by using the following formula:

$$PWM_{max} = -k_p.\theta_{max} + PWM_{Base} \implies k_p = \frac{PWM_{Base} - PWM_{max}}{\theta_{max}} = 1.3$$

3.4 Deep Learning Integration

We talked about traditional computer vision techniques for lane detection such as thresholding, warping, and pixel summation. These methods are effective for specific controlled environments but they fail in a complex and dynamic one. To enhance the robot's capability we will use the power of neural networks for lane detection and following.

Our goal was to create a neural network model that could reliably predict the steering angle and the curve value after it was fitted using a dataset that included images, paths, and steering angles. The training phase of this approach can be presented by the following image [Figure 10].

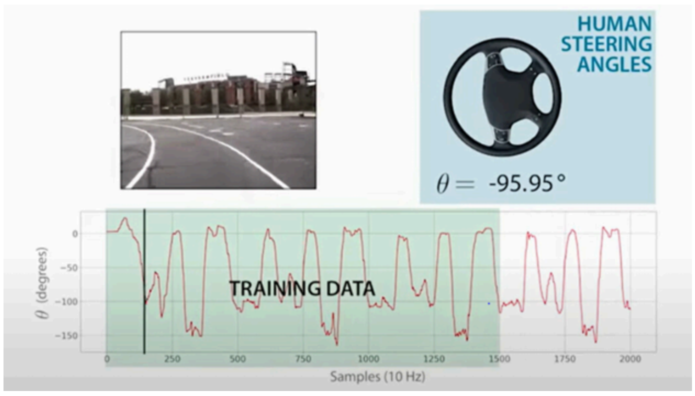


Figure 10 - Illustration of the training phase of the Deep Learning Approach

Artificial neural networks have proven to be very effective and flexible in dynamic contexts with plenty of noise and frequent changes. The three-layer neural network ALVINN, which was created especially for road tracking, served as the model's inspiration. The ALVINN network's design is illustrated in [Figure 11]. This system calculates the best steering direction to keep the car on course while utilizing information from a laser rangefinder and input from a camera. The network is trained to predict the steering direction properly using the backpropagation learning algorithm.

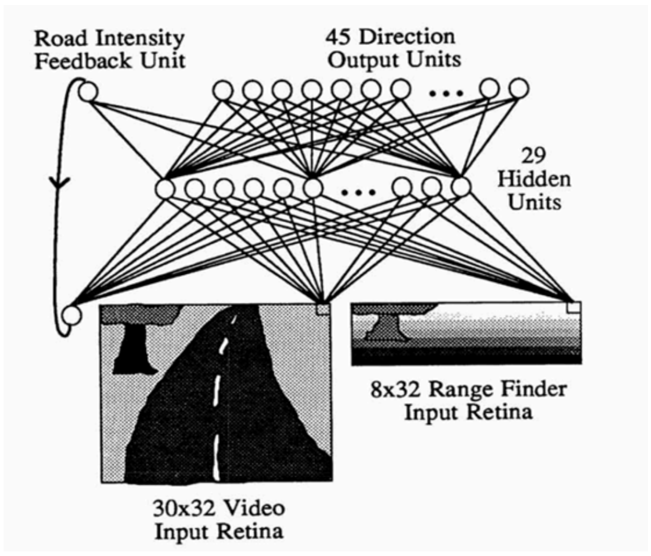

Road Intensity Feedback Unit
45 Direction Output Units
29 Hidden Units
8x32 Range Finder Input Retina
30x32 Video Input Retina

Figure 11 - The General Architecture of the ALVINN [3]

Such a model needs special well defined steps, we have presented a general illustration that explains it.

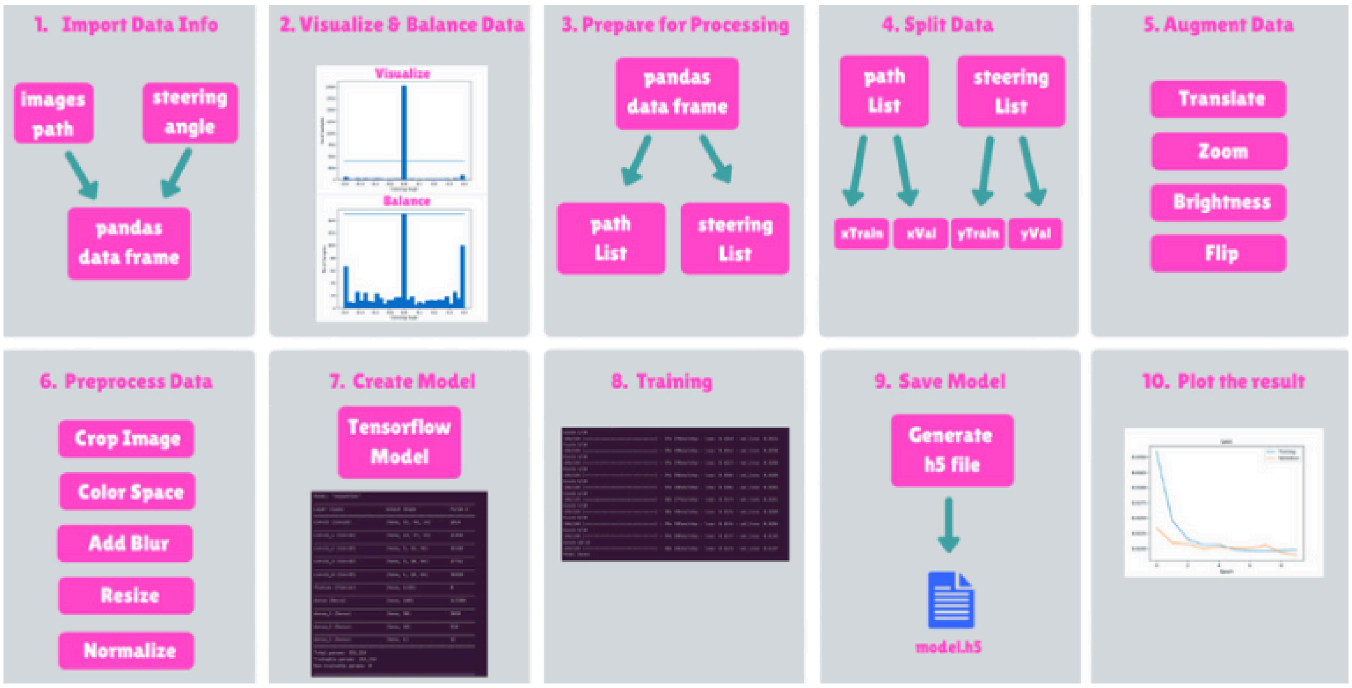


Figure 12 - Overview of the differents steps of our model

Step 1: Data Collection:

capture images from a Raspberry Pi camera, store these images along with their corresponding steering angles, and save this data into a log file.

Step 2: Initialize Data:

Initialize and import image data along with their corresponding steering angles from multiple CSV log files into a pandas DataFrame

Step 3: Visualize and Balance Data:

In order to prevent biased models, the dataset's steering angle distribution should be balanced. Before using balancing procedures, data visualization is shown in [Figure 13].

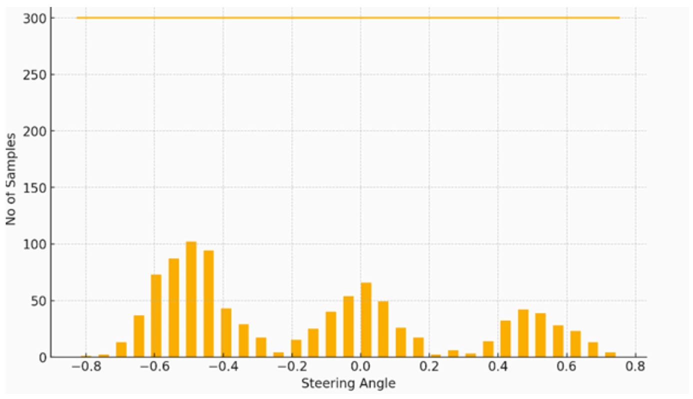


Figure 13 : Example of Data Visualization of Number of Samples and Steering Angle.

Step 4: Prepare Data:

Extract from the DataSet the associated steering angles and images captured and transform them into arrays for effective numerical computations.

Step 5: Data augmentation:

Create a more varied dataset for training the model without actually collecting new data.

Step 6: Preprocess Data:

Carries out a number of image preprocessing operations that are commonly employed in computer vision tasks:

- Gaussian Blur: smooth out edges by reducing noise.
- Normalization: Enhance gradient descent convergence during neural network training.

Step 7: Create the Model:

The model is designed by a group of convolution layers with some flatter and dense techniques, here is the architecture of the model [Figure 14]

The layer of input accepts 66 x 200 pixel photos with three RGB color channels.

For the convolution Layers, each layer uses a varied number of filters and kernel sizes to extract features from the input image. This facilitates the capture of different detail levels, ranging from edges to intricate patterns.

For Layer flattening, it prepares the multi-dimensional convolutional layer output for the fully connected layers by converting it into a one-dimensional vector.

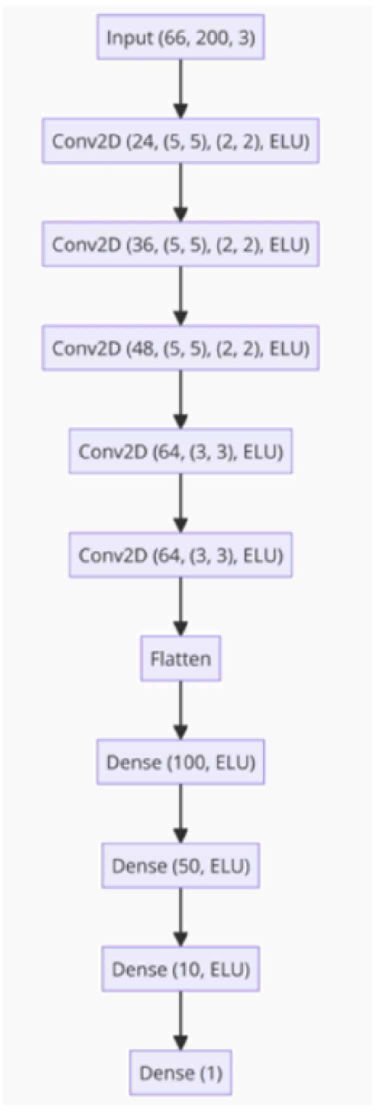


Figure 14 : Software Architecture of the Model

3.5 Conclusion

In conclusion, the implementation of various computer vision algorithms and techniques for lane detection with an integration with basic control methods has demonstrated its success for lane detection and lane keeping systems, yet these methods has its own limitations when it comes to dynamic environments, that's why in modern methods to design such systems combine both traditional approaches and modern neural network architectures to get the benefits of both.
Future work will focus on refining these models, improving real-time processing capabilities, and expanding the system's functionality to include additional ADAS features.

General Conclusions

In this project, we successfully developed a line-following robot utilizing visual servoing techniques, employing both traditional computer vision methods and modern deep learning approaches. This comprehensive system involved careful hardware and software design to achieve reliable lane detection and lane-keeping capabilities.

Traditional computer vision techniques require adjusting parameters for every environment, making them inefficient for real-world scenarios. Even slight changes in lighting conditions can cause significant masking errors, leading to incorrect calculations and system failures.

To address this problem, we suggested a deep learning approach, which has proven effective in modern self-driving cars. However, implementing deep learning was challenging for our project, as collecting hundreds of pictures with different steering angles in various conditions is a tedious task.

Another approach that we didn't explore in our study is to automatically find the right masking and warping parameters; this method can be explored in future studies.

We also opted for a different robot wheel configuration for simplicity. Using a 4-wheeled robot with two motorized wheels in the back and two free-steering wheels in the front is mechanically challenging and requires more equipment. However, it is the best choice for simulating our target: self-driving cars.

Our modular architecture of this system has proven its effectiveness, as it makes it easier to debug and troubleshoot and make the system explainable and presentable without ambiguity.

We opted for experimental methods for steering angle calculation but it can be enhanced by studying the kinematics of the system and fitting Curve Values to real measurements, with this approach we can improve the precision and the correctness of this angle, thus better information to be fed to the control algorithm to generate the desired outcome.

Bibliography

[1] Advanced Driver-Assistance Systems: A Path Toward Autonomous Vehicles - https://www.researchgate.net/publication/327253343_Advanced_Driver-Assistance_Systems_A_Path_Toward_Autonomous_Vehicles

[2] Mobile Robot Wheel Configurations and Kinematics - PPT - Mobile Robot Wheel Configurations and Kinematics PowerPoint Presentation - ID:1089388 (slideserve.com)

[3] End-to-end Autonomous Driving using Deep Learning: A Systematic Review - [PDF] End-to-end Autonomous Driving using Deep Learning: A Systematic Review | Semantic Scholar

[4] Lane Detection Techniques using Image Processing (researchgate.net) - https://www.researchgate.net/publication/353773538_Lane_Detection_Techniques_using_Image_Processing

[5] Image Segmentation - Image Segmentation (Part 1). Thresholding, Otsu's and HSV… | by Ralph Caubalejo | Towards Data Science

[6] Thresholding - Thresholding-Based Image Segmentation - GeeksforGeeks