



THE UNIVERSITY OF
WESTERN
AUSTRALIA

Autonomous Driving on a Model Vehicle

Lane Detection and Control

Final Report for GENG5511 and GENG5512

Student: Nicholas Burleigh (21107266)
Master of Professional Engineering (Electrical & Electronic Engineering)


Academic Supervisor: Professor Thomas Braunl
School of Electrical, Electronic & Computer Engineering

Word Count: 7912

Date of Submission: 20 May 2019

Declaration of Authorship

I, **Nicholas Burleigh**, declare that this project is my own work and suitable acknowledgement has been made for any sources of information used in preparing it.

Signature: 

Signed 20 May 2019

Abstract

Autonomous driving systems have increasingly become a topic of anticipation in recent years. In development of these systems, it is useful to test potential strategies in a controlled environment, and a common means of accomplishing this is using miniature model vehicles on a track, both in a physical environment and in simulations. These model vehicles typically include the expensive sensory and computational hardware that the full-size road vehicles use.

The aim of this research was to construct a functional autonomous driving system on the comparatively inexpensive Eyebot hardware and software developed at the University of Western Australia, with its limited computational power and sensory equipment. The primary focus was on creating a comprehensive and reliable model of road markings, aiming to develop a system wherein a vehicle could reliably navigate a track in both simulated and physical environments with no errors. Subsequent to this aim was to implement more complex driving behaviours.

Methods would first be developed and tested in a simulated environment, using the Eyesim software to run test drives in a virtual environment. After a method was identified as conceptually sound in this software, it would then be implemented in the practical environment. From there, the issues in transferring from simulation to reality would be identified and software was adjusted accordingly.

A functional system was created which was capable of navigating the simulated and practical tracks while remaining within the road boundaries. Due to spatial constraints the practical track contained extreme circumstances relative to the expected environment. The robot was able to successfully navigate this track, albeit with a lower rate of successfully detecting lanes.

Contents

Abstract.....	2
Contents.....	4
1. Introduction.....	6
1.1. Introduction to Autonomous Vehicles.....	6
1.2. Model Vehicle Driving Competitions.....	6
1.3. Scope.....	7
2. Literature Review.....	7
2.1. Autonomous Vehicle Models.....	7
2.2. Lane Perception, Interpretation and Control.....	8
3. Design Process (Design & Build).....	12
3.1. Requirements.....	12
3.2. Resources.....	12
3.3. Design Philosophy.....	13
3.4. Metrics of Success.....	13
4. Final Design.....	14
4.1. Eyebot Autonomous Driving Model.....	14
4.2. Low-Cost Method.....	14
4.3. Hough Lines Method.....	14
4.4. End-to-End Neural Network.....	22
5. Assessing Driving Methods.....	22
5.1. Performance Assessment.....	22
5.2. Further Criticism of Methods.....	24
5.3. Limitations.....	24
6. Topics for Further Investigation.....	25
7. Conclusions.....	25
8. References.....	26
9. Appendix.....	30
Appendix A – Initial Lane Perception & Control Algorithm.....	30
Appendix B – Informative Lane Model Algorithm.....	35
Appendix C – Simulation Test Track.....	37
Appendix D – Real Test Track.....	37

1. Introduction

1.1. Introduction to Autonomous Vehicles

Autonomous vehicles have become an object of desire in recent years, with many companies and universities racing to implement their own autonomous driving system. Autonomous vehicles are typically recognised as having six levels of autonomy [1]:

- L0 – No Automation
- L1 – Drive Assistance
- L2 – Partial Automation
- L3 – Conditional Automation
- L4 – High Automation
- L5 – Full Automation

The end goal in developing autonomous driving systems is to reach the L5 level of automation, wherein the vehicle is capable of directing itself with no human input required. To date, the highest level of automation commercially available is L3 in a recent Audi model [2], allowing for driving assistance in highly structured environments and parking without user input. However, as the level of vehicle automation it has increased, some have begun to worry about the dangers of the adolescent stages of autonomous vehicles [3]: they are automated enough that the driver can feel comfortable losing focus on the road, but require the human to take over if a problem arises. If the human is not prepared, safety issues arise. To this end, some vehicle manufacturers have declared that they will not put a partially automated vehicle on the market, instead striving to release a fully automated product [4]. Some have since reconsidered their position, but the dangers posed by partially autonomous vehicles remains [5]. Consequently, it has become valuable to develop autonomous driving methods in environments where the risk to human life is minimal. Part of this is running autonomous vehicles and verifying their functionality in simulated environments prior to driving on a road [6].

1.2. Model Vehicle Driving Competitions

One method of developing and testing autonomous driving strategies is on small model vehicles in simulated road environments. In universities, it has become commonplace to hold competitions wherein model vehicles drive around a track while completing various activities.

Once a strategy has been validated in these settings it becomes safer to implement it in a full-scale model in a controlled environment, with the eventual goal of deploying it on a road with traffic. The types of situations that this project wishes to emulate are primarily the Carolo Cup [7], and the Audi Autonomous Driving Cup [8]. The kinds of tasks that these competitions require include [9]:

- Driving within lanes as quickly as possible.
- Capacity to detect and avoid obstacles.
- Adherence to all traffic signs.
- Intelligent navigation of intersections and crossings.
- Ability to park in both standard and parallel manners.

1.3. Scope

The University of Western Australia (UWA) currently has two autonomous driving projects in development: a project on a real, full-scale car; and a project using model robots on a small-scale track. This project focuses on assisting in the development of the model autonomous driving system. The hardware that this project was chosen to be developed on was the UWA Eyebot hardware and software.

Specifically, this project focuses on developing a system for the Eyebot to perceive lanes and drive within them. The aims of this system were to use low-cost image processing methods to allow the robot to react within a reasonable time frame, and to design a system which is capable of managing as wide an array of situations as possible.

2. Literature Review

2.1. Autonomous Vehicle Models

Autonomous driving system can broadly be divided into the structures described in the following figure [10]: sensing, perception, planning and control. In model autonomous vehicles, it is desired to mimic the sensory [11] and computational [12] capabilities of a full-scale car as much as possible. These are the aspects in which this project is most limited. There has yet to be a standard for all of the sensors that should be present in an autonomous vehicle, and the only sensors common across every implementation are cameras. Other commonly used technologies are LIDAR, radar and ultrasonic sensors [13]. The inclusion of multiple different types of sensors allows for sensor fusion [14]. This is a means of combining the data from multiple sensors to enhance the reliability of the resulting environment model [15].

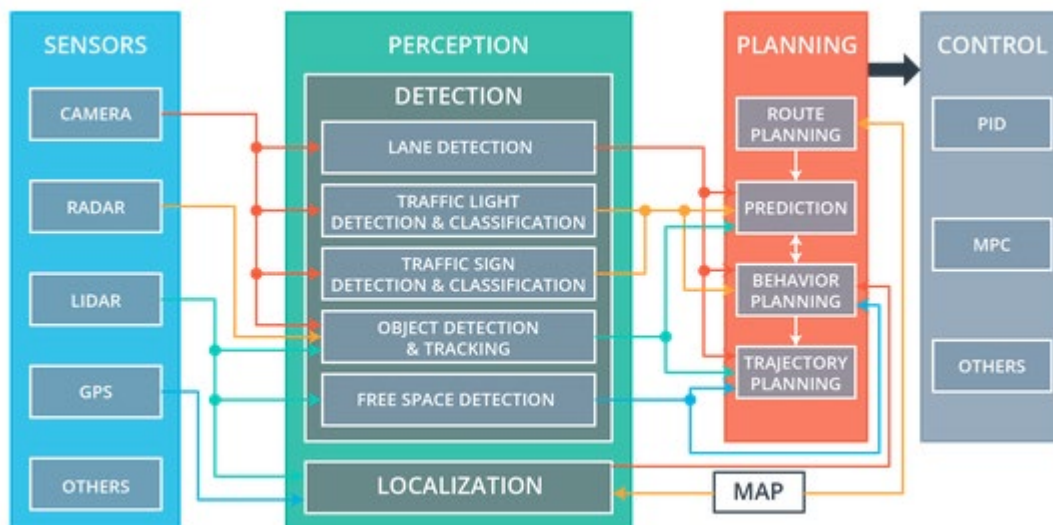


Figure 1 Autonomous Vehicle Model [10]

2.2. Lane Perception, Interpretation and Control

2.2.1. Separation of Stages

2.2.1.1. Low-cost Image Processing

A previous UWA student devised a method for lane detection that was designed to be suitable for low-power mobile phones. The essence of this method was to apply a specialised steerable filter to a grayscale image, and was formerly regarded as a potential solution for quick lane detection. [16].



Figure 2 Steerable Filter Applied to Road Image [17]

A version of this algorithm was available in C++ for testing and potential implementation. [18]

2.2.1.2. Hough Lines

2.2.1.2.1. Processing

A standard image processing method of extracting shapes, and by extension lanes, from an image is Hough lines [19]. This is a means of filtering an edge-detected image into groups of lines, and has been shown to be effective on a Raspberry Pi [20]. Broadly, the idea is that for each edge in an image, a sinusoidal graph can be created for all of the lines defined by polar coordinates that intersect it [21].

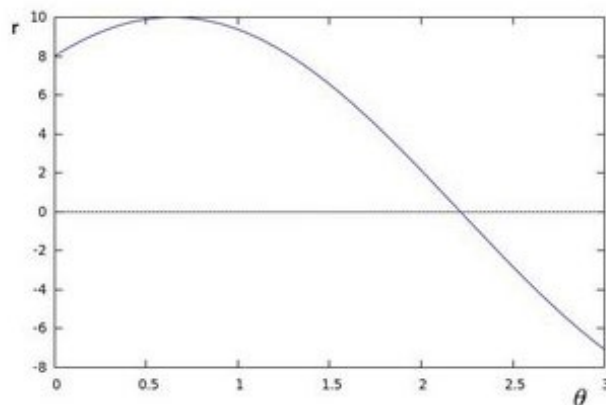


Figure 3 Hough Line Polar Graph [21]

This graph defines all the possible linear lines that can intersect that point on the image, and a graph of this nature can be produced for every edge pixel in the image. Each individual graph can be classified as belonging to the same line if they have a point of intersection [22]. For instance, in figure 4 below, the polar line $(r, \theta) = (9, 1)$ intersect all three data samples. Therefore, it is likely that the three samples all belong to the same line characterised by $(r, \theta) = (9, 1)$.

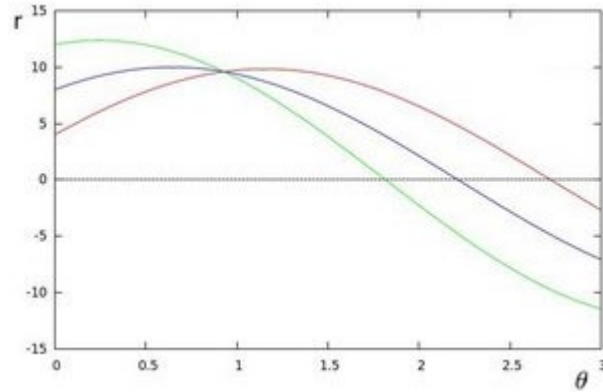


Figure 4 Hough Line Intersecting Polar Graphs [21]

The benefit of this method is that it has a great deal of tuning parameters which allows for controlling data according to the user's purposes. In this project that is of use because it allows quick elimination of lines that are too small to be considered a lane. The tuning parameters include [21]:

- Threshold: how many points need to intersect at the one point for it to be defined as a line.
- Minimum line length: the minimum distance between the two furthest points in the line for it to be classified as a line.
- Maximum line gap: The maximum distance between two points for them to be considered in the same line.

The end goal of this processing method is to translate all lines in the image into a useable set of vectors.

The steps involved in this method are to first apply a Gaussian blur filter to the image. This has the effect of blurring out any small details that may be mistaken for a lane. Following this, an edge detection filter is applied to eliminate all irrelevant data and leave only edges. Finally, the Hough transform is applied to create a set of vectors based on lines in the image [19].

2.2.1.2.2. Clustering

The issue with this method is that the output from the Hough lines process isn't a guaranteed set of lanes, but a set of lines. The lines could belong to the same lane or they could belong to separate lanes; the lines might not even belong to a lane at all. Therefore, it is necessary to apply data clustering strategies in order to successfully group the lanes.

Rudimentary means of narrowing the data down involve eliminating lines that are not white, and lanes that are too small, but this alone does not suffice. A standard means of grouping the lines is agglomerative hierarchical clustering [23]. In this method, each data point (in this instance, line) is given its own independent cluster. If the shortest distance between a set of clusters is lower than the set threshold value, then the two clusters are merged. This process continues until no more groups satisfy the requirements for merging [24]. With respect to this implementation, the lines are clustered according to their angle and y-intercept [23]. However, it is clear from that description alone that the lanes must be relatively straight for this to be successful.

2.2.1.3. Neural Network

2.2.1.3.1. Processing

Alternatively, there are a variety of methods to extract lanes from an image involving deep learning. One of these involves using a convolutional neural network to output a feature map of the lanes [25]. Notably, the output of this network is an image. Consequently, it is more akin to filtering out all non-lane data before a method of processing the data is used. This has the effect of improving reliability of the detected lanes, but on its own is not useful for this project. Other methods have built upon this to include extraction and clustering of lanes. This is achieved by training a neural network to output beginning and ending line values, including depth data so that curves in 3D spaces may be created [26].

2.2.1.3.2. Planning & Control

Planning a path for the vehicle to drive typically involves some form of trajectory generation: projecting a point in space based on the driving data and attempting to drive there. Path planning must adhere to the rules of the road and avoid any obstacles in the vehicle's path [27]. For driving in more complex environments, spline-based trajectory planning is an effective approach that can be generated to remain comfortably within lane boundaries [28]. For control of the vehicle according the planned trajectory, a PID controller for the steering angle is sufficient [29].

2.2.2. End to End Processing

All of the above methods assume a model wherein the goal is to reliably identify the lanes so that planning and control may be handled by separate processed and algorithms. However, this is not always the case; a novel means of autonomous driving was implemented by Nvidia wherein an image was fed into a neural network, and the networks outputted steering directions [30]. In effect, the perception and planning stages are combined into one neural network. This was accomplished by passively taking images of the road while a human was driving, and the angle of the driver's steering wheel was recorded at that moment. Subsequently, the network was trained with those steering angles as the output and then the vehicle control strategy emulates the desired motion.

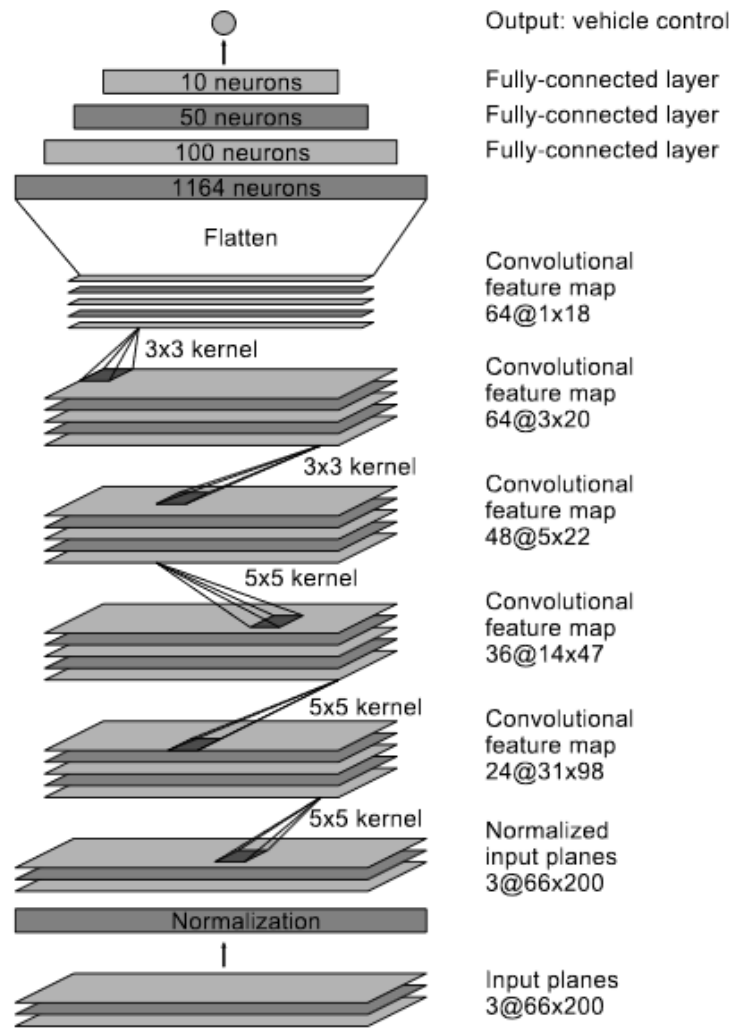


Figure 5 End to End Autonomous Driving Neural Network [30]

This method has proven to be extremely effective, with many implementations and sets of training data available. The disadvantage of this method is that the problems in neural networks are difficult to diagnose, and if any problem is encountered, exactly what happened cannot be identified [31]. The only recourse is to replicate the situation in which it happened and train the network to react correctly. Entrusting all tasks to a relatively opaque neural network is a problem because they are not endless in their possibilities. Significant criticism has been levelled against the way they have become a catch-all solution in computer science. [31] [32] This criticism becomes especially important to consider when implemented on sensitive software such as autonomous vehicles, which are responsible for human lives.

3. Design Process (Design & Build)

3.1. Requirements

The requirements of the final design are as follows:

- The system shall be capable of identifying lanes in the environment.
- The system shall be capable of driving comfortably within lanes.
- The system shall be capable of managing minor damage to the lane markings.
- The system shall be capable of running on the Eyebots.

3.2. Resources

3.2.1. Hardware

As previously specified, the aim of this project was to implement an autonomous driving system on UWA's Eyebots [33]. Consequently, the physical implementations of the project are developed and tested exclusively on the Eyebots. These machines contain a Raspberry Pi Model 3B for processing tasks. For sensing, they contain one front-facing camera, and 3 Position Sensitive Devices (PSDs), one each on the front and sides of the robot. The Eyebot has pre-existing software installed for interfacing with these sensing devices. It also has software for sending drive commands to the motors in terms of desired linear and angular speed, and an internal coordinates system based on the wheel motion.

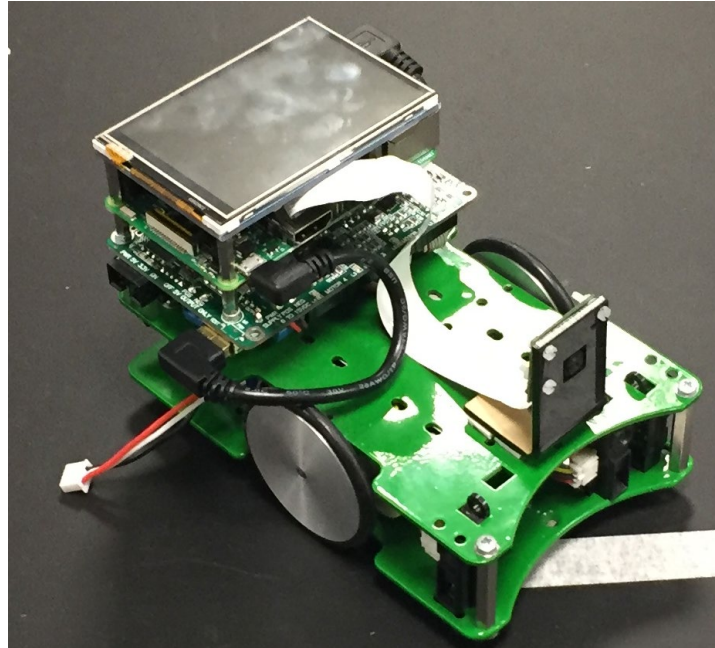


Figure 6 The Eyebot

3.2.2. Software

The Eyesim simulation system is available for driving virtual versions of the robots [34]. The simulation system has functionality for inserting custom objects and floor textures. Using this, driving tracks may be created and tested within the simulation system.

OpenCV was used for image processing in the robots. This software library contains a wide array of solutions for image processing. This library was used for non-neural network-based image processing in the project. [21] [35] [36]

Tensorflow was used for neural network functionality in the project. This is a platform for machine learning [37] with all of the features necessary to implement the Nvidia end-to-end neural network.

3.3. Design Philosophy

There are many valid methods of detecting lanes, so it was important to take into consideration the reasons for creating this system. Among this project's goals were to create a baseline for future projects to develop upon it, therefore it was desired to create a system that could be understood by a new user entering into the project and provide a means of assisting with the development of future systems.

Therefore, the general approach of this project was to begin at the method with the least complexity and build upwards to include as much as possible.

3.4. Metrics of Success

There are several metrics that may be used to evaluate the robustness of these various implementations. The primary and most important metric is the car's capacity to drive comfortably within the lanes within the test tracks available. A means of quantifying this would be the total time without a failure.

This is a generic approach that assumes all sections of the track have an equally likely probability of failure. In actuality, the track was created to expose the robot to a variety of different circumstances, and these will have varying failure rates. Thus it is also necessary to document the reliability for each individual driving case.

Beyond this, an important factor in an autonomous vehicle is its reactivity. If a situation changes, it must be able to identify as such and react as quickly as possible. This is naturally dependent upon the rate at which input data can be processed. Therefore this quality will be measured in terms of Frames per Second (FPS) for each implementation.

Furthermore, the robot successfully following the track does not mean that it is functioning perfectly. It indicates that it is successfully detecting lanes, but it does not indicate that it is successfully detecting lanes reliably. Therefore it is necessary to indicate a general rate at which lanes are successfully and correctly detected.

Finally, another concern is the amount of data available for interpretation. This is relevant as this data would eventually be used for purposes in addition to maintaining lanes, such as parking. It is difficult to

define a statistical metric for this, so success here will be characterised by how much data has been successfully categorised compared to how much data could have been categorised.

4. Final Design

4.1. Eyebot Autonomous Driving Model

Taking into account the limitations of the Eyebot, the revised autonomous driving model is:

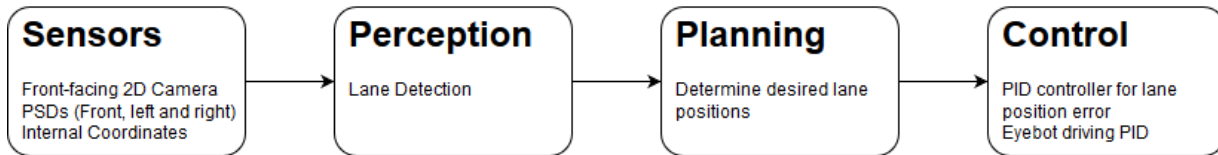


Figure 7 Eyebot Autonomous Driving Model

Flow diagrams of the implemented lane detection methods may be found in the appendix.

4.2. Low-Cost Method

The lane processing method based on standard image processing was first attempted. C++ code for the method was available within the robotics department of UWA and could be altered to accept images from the Eyebot. The code for this had several flaws though: the first being that it ostensibly would only function on images of resolution 160x120, which is a smaller than desired image size. The second issue was that lanes as the Eyebot sees them are not identical in nature to the lanes that car, which this method was tested on, would see. The Eyebot is significantly closer to the ground and consequently the lanes can look much larger in the image. Consequently this image processing algorithm would detect far away lanes but not the desired lanes. Based on this information, it was decided not to move forward with a planning and control method based on this implementation.

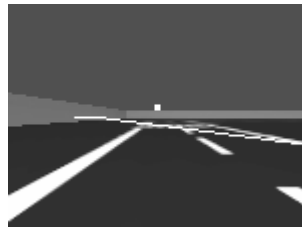


Figure 8 Low-Cost Method Running in Simulator

4.3. Hough Lines Method

4.3.1. General Processing

The standard method for preparing the Hough Transform was used. Images were captured from the Eyebot camera, converted to grayscale, a Gaussian blur was applied, a canny edge detection filter was applied, and then the Hough transform was applied. OpenCV was used for all filtering functionality. As previously stated, the Hough transform has three tuning parameters: the line threshold, the minimum line length, and the maximum line gap. In order to gain a complete picture of the environment, those

two parameters were set to the minimum practicable value. The threshold value was manually tuned until the camera was detecting all edges with minimal lines of identical nature. The rudimentary post-processing step of deleting any lines that were not touching white on the raw image was taken to further refine the data.

4.3.2. A Functional Implementation

4.3.2.1. Selecting Lanes

As is detailed in the succeeding sections, clustering lanes effectively is a difficult process. Therefore, for the initial implementation to validate the concept, a more simplistic approach was used using observable rules within the environment. The system was based on the assumption that each section of a lane will have two lines: one for the left side of the lane, and another on the right side of the lane. Ergo the extracted lines are filtered into two groups based on whether their white side is on the left or right of the lane. Smaller lines are omitted as this method does not allow for fine details. Once the groups have been split, the two lists of lines are compared to each other. If the space between two lanes has an average white colour, they are not too close to each other, and the extrapolated lines do not intersect within the image, then the lanes are assumed to belong to the same group and grouped together.

Lanes were filtered into left lanes and right lanes based on the following set of rules: if a lane is on the far left or right of the image then it is classified as the corresponding lane. If the lane does not meet either criterion then it is determined to be left or right based on its angle within the image. Lanes are selected by choosing the closest one to the vehicle unless many are present; this is assumed to be a pedestrian crossing and the closest match to the previous is selected. If the lane's starting point is not within a reasonable range of the expected starting point, that lane is ignored and control is accomplished based on the other lane.



Figure 9 The Initial Implementation Detecting a Left Curved Lane

4.3.2.2. Planning & Control

A simplistic implementation is again used for planning and control of the vehicle. The robot may be calibrated by placing it in the middle of a straight road, parallel to the lanes. The program then uses these as the expected values of the robot. The error needs to be taken from a static reference point, and there is no guarantee that the existing lanes will intersect any one point. Thus the lanes are extrapolated to the bottom of the image and that is used as the reference point for the calibrations.

The error is passed to a PID controller for which the P, I and D values are manually tuned based on what appears to work.



Figure 10 Robot Driving on Track

4.3.3. An Informative Implementation

The previous method is sufficient for successfully following the lanes. However, its functionality would be limited to the specific task it was created for. It keeps four lines necessary for identifying the nearest lanes and discards all other data while naively following the lanes using a simplistic control method. With reference to the autonomous vehicle model, the means of perception is extremely limited. Therefore, a method that operates using a more complete environment model is desirable for future use.

With this in mind, a driving strategy was built in addition to the previous method with the intention of profiling world data to the maximum possible extent, still using the Hough transform as the base image processing method. The intention in developing this method was twofold: to group all lines belonging to the same lane together, and to accurately map images on the camera to real-world positions. The former was accomplished by testing various clustering methods and implementing the most successful one, and the latter was accomplished by mapping the relationship between camera positions to distance from the vehicle.

4.3.3.1. Grouping Lanes

4.3.3.1.1. Agglomerative Clustering

The difficulty with developing a comprehensive robust lane model is that very few assumptions may be made. The number of lanes present in the image is unknown, though the maximum may be defined. Furthermore, it is difficult to define which lanes to follow.

The standard convention when using the Hough transform is agglomerative clustering, as stated previously. The lines are grouped by their angle and y-intercept to form clusters. The principle is sound on a generally straight road but runs into significant issues on a highly curved track like those used in this project. Another issue with this method which was noticed during testing it was that the camera was so close to the ground on the Eyebots that the white lines took up a more substantial portion of the

screen than they did in the images taken on a standard car. This ultimately led to a difference in results clustering robustness when compared to their original intended usage.

4.3.3.1.2. Spectral Clustering

This is a clustering strategy wherein a two-dimensional dataset is dimensionally reduced to a one-dimensional dataset, and clustering is based on the new dataset [38]. In this instance, a distance matrix was created for the lines

This method was tested with the hope that it would be able to identify the special cases for which rules were difficult to define. As can be seen in the figures below, it was successful in this task in many circumstances and had potential to be successful in all of the available cases with additional tuning. However, this method is very computationally expensive to the extent that on the simulator, the frame rate would reach as low as 8 frames per second. Given this, this method was discarded and not tested on the physical Eyebots.

4.3.3.1.3. Final Method

Given the failure of implementing standard clustering methods, a specialised implementation of agglomerative clustering was used that best suited the circumstances that the robot would face. This idea operated on the assumption observed during development that large lines were considerably more reliable for being an approximation of the lane than the small lines. Based on this assumption, the clustering algorithm would select the largest line from the unused dataset and start a new lane group. A linear line would be calculated from this line, and for each unsorted line, the Sum of Squared Error (SSE) would be calculated for the start and end points from the projected linear line. The line with the smallest error would be added to the lane group and least squares regression would be used to create a new linear line for the lane group. The SSE would be recalculated based on the new lane group. This loop continues until either no lines remain unsorted or the lowest SSE value exceeds a specified threshold value.

If the threshold value is exceeded before all lines are sorted, a new lane group is created starting with the largest line remaining in the unsorted group. This process continues until all lines are sorted. The intent behind implementing this method was to use the larger lines as a reliable baseline for build lanes and slowly skewing it in the direction of any curves that exist.

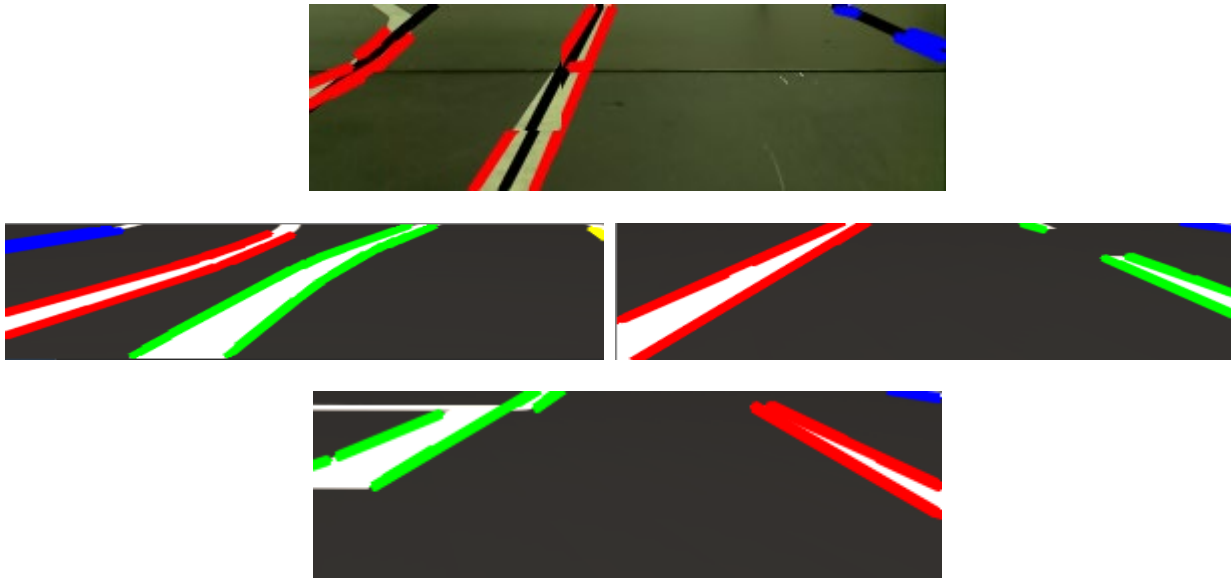


Figure 11 Clustering Successfully in Complex Situations

4.3.3.2. Mapping Camera Positions to Coordinates

Many autonomous vehicles contain stereo cameras so that they may extract position data from the captured images [39]. The Eyebot contains only a single camera so doing this is not possible. The intention of the project was to accomplish autonomous driving on comparatively cheap hardware, simply buying and installing stereo cameras would have been inconsistent with the project's goals. However, both the projects that this was based on and this project make use of a uniformly flat surface for the road. When being able to make the assumption of a flat surface, it is easy to emulate the effect of position sensing by simply placing a reference point within the camera's field of view and mapping the position of that object with respect to the robot.

4.3.3.2.1. Mapping in the Simulator

Creating a coordinates transformation system is trivial in the simulator using the already built tools, as the PSDs can be set to give exact distances with no error. All that need be done is to place a white line on the surface and black a wall directly parallel and perpendicular to that line. The virtual robot is also placed perpendicular to the line so that the PSDs may be used to determine distances to the front and side walls. The Hough transform may then be used to attain the exact camera coordinates of this line and the PSDs may be used to determine the exact real-world coordinates of the end of the line. The robot is then simply manually pulled around as much as possible while keeping the end of the line in frame and the real-world positions of the walls relative to the robot, and the camera position of the line are recorded for as long as desired.

From there, it is a simple matter of determining the relationships between camera coordinates and the simulator coordinates. This method yielded the relationships shown below.

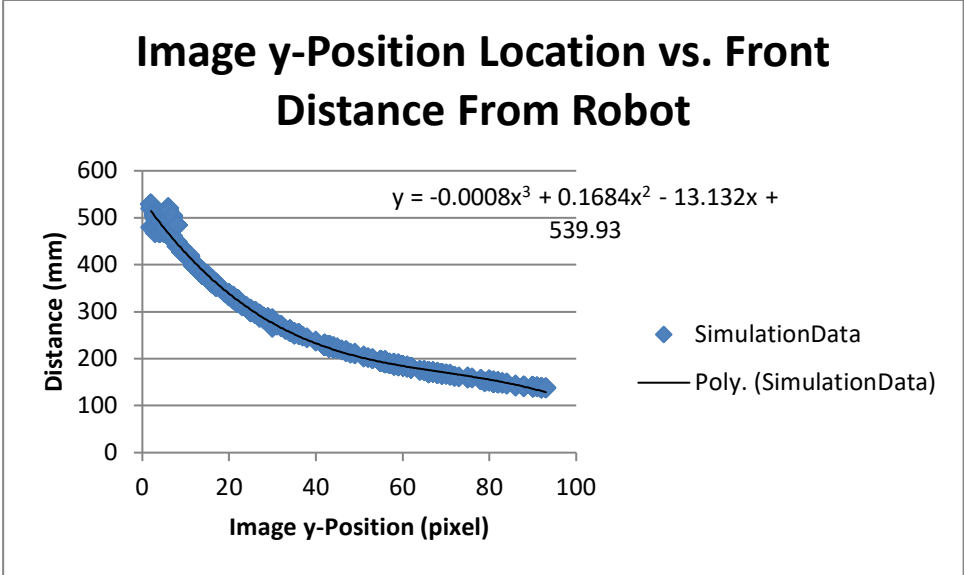


Figure 12 Mapping Image Ground Distance Positions to y-Positions in Images

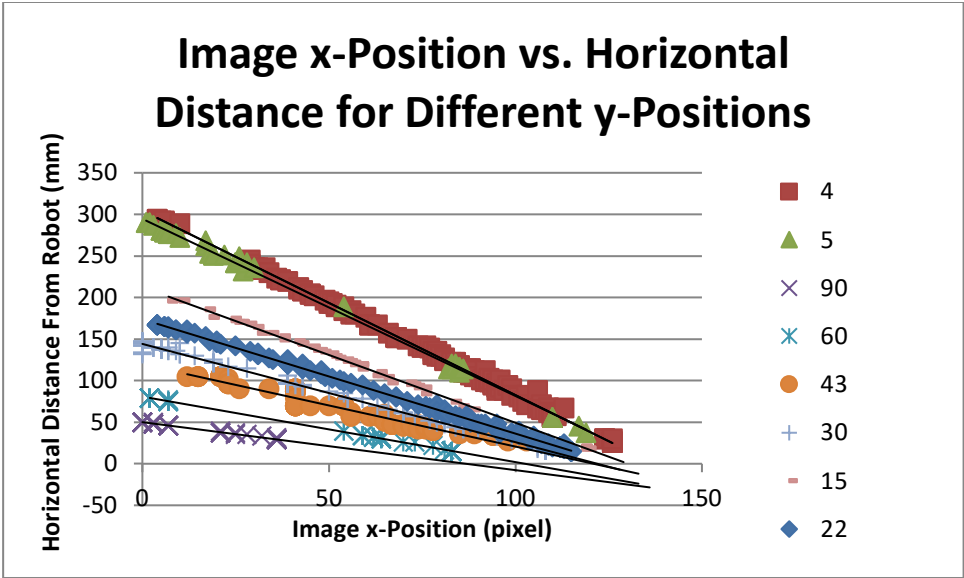


Figure 13 Mapping Image Sideways Distance to x-Positions for varying y-Positions

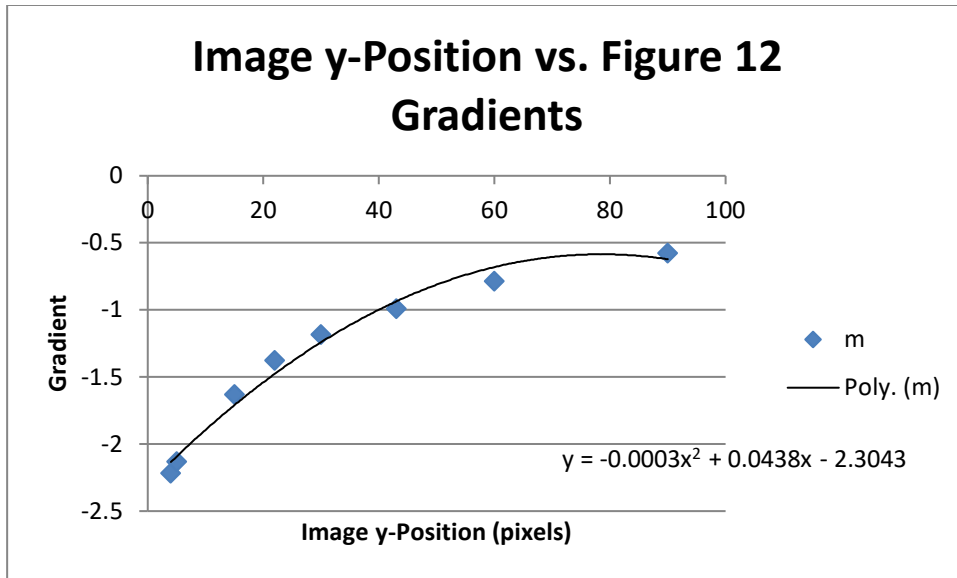


Figure 14 Building a Function to Get the Gradient for the x-Position vs. Horizontal Distance equation for a given y-Position

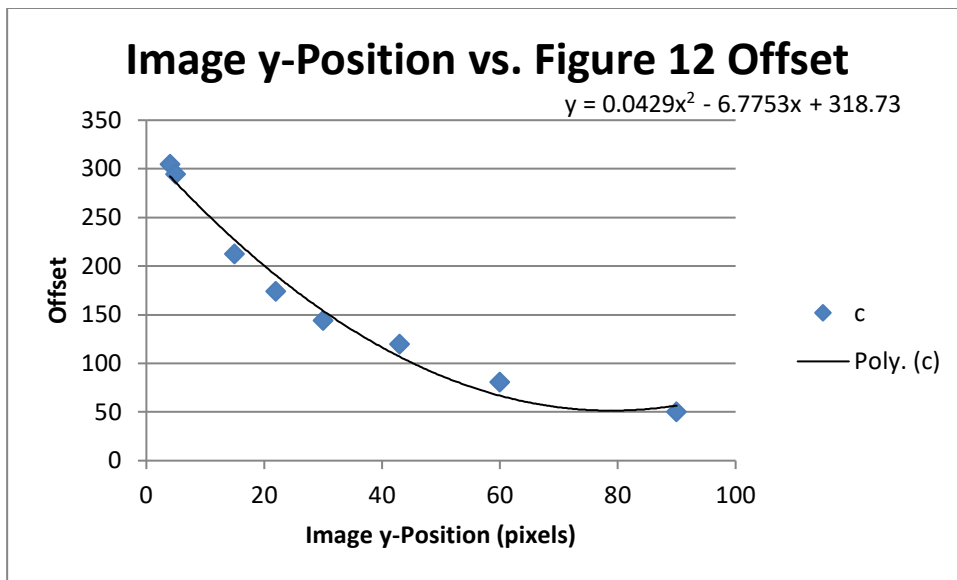


Figure 15 Building a Function to Get the Offset for the x-Position vs. Horizontal Distance equation for a given y-Position

4.3.3.2.2. Mapping in Reality

When transferring from the simulator to reality, the transformation no longer holds true. This is likely because of small imperfections in camera placement and angle when compared to a perfectly reproducible simulated object. This method is not practicable in the real world due to the large variance in PSD values at any given time. This causes a very imprecise calibration that cannot produce coordinates capable of subtle control.

Instead, accurate generally accurate mapping was accomplished by assuming that the same general relationships that the simulator's camera-position transformations adhered to would also be true in reality, and that the goal in transferring to reality was to calibrate that equation with constants which

would work. With this in mind, the real-world calibration was accomplished by placing a marking on the driving surface and moving the robot so that the marking is at the top and bottom edges of the image, as well as the middle of the image. At these points, the front-facing distance was measured directly with a tape measure. The relationship between the forward distance of the camera and the y-position of the image may then be accurately mapped.

Following this, the horizontal position of the point with respect to the camera was then measured using the same method. The camera was again placed so that the reference point was at the top, middle and bottom of the image. At each of these positions, the robot was moved sideways so that the reference point was at the middle of the x-position and the far left of the image for each of the three y-positions.

4.3.3.3. Path Planning

The intent of creating this more complex environment model is so that the robot may be directed towards a specific point in space, rather than using a reactionary control strategy based only on camera coordinates. Using this more refined system of lane detection, the position of the nearest lane is determined with respect to the robot. Once the lane is selected, the real-world position of the lane's beginning is estimated and the coordinates are transformed so that the x-axis is parallel to the lane and the y-axis is perpendicular to the lane [40]. The goal at this point is to select a point to drive towards. If the selected lane is estimated to be a left lane then on the local coordinates system the following coordinates are set:

$$(x, y) = (x_{lane}, y_{lane}) + \left(0, -\frac{Width_{robot}}{2} - Offset\right)$$

Where $Width_{robot}$ is the width of the robot and $Offset$ is the desired distance the robot should be from the lane. If a right lane is selected then the coordinates are:

$$(x, y) = (x_{lane}, y_{lane}) + \left(0, \frac{Width_{robot}}{2} + Offset\right)$$

These coordinates are then transformed to their position relative to the robot.

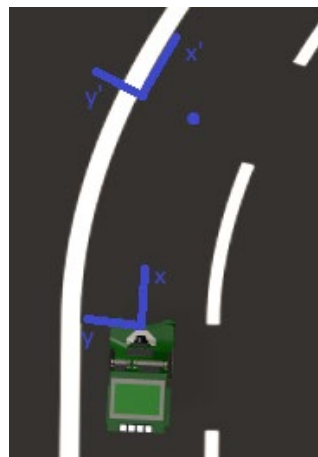


Figure 16 Transforming Coordinates to be Parallel to the Lane and Selecting a Point to Drive to

4.3.3.4. Control

The PID error parameter uses the angular speed of the vehicle. To get this, a uniform arc is assumed from the robot's current position to its end position. The length of the arc is estimated in millimetres and the desired finish change in angle is assumed to be the angle of the lane relative to the robot. From here, the angular speed is determined using the equation:

$$\omega = \frac{\theta v}{d}$$

Where ω is the angular speed (degrees/second), θ is the finishing angle (degrees), v is the robot's linear speed (mm/s) and d is the length of the arc (mm).

4.4. End-to-End Neural Network

A neural network was trained based on the model used by Nvidia. Nvidia did not provide code for implementing their network. However, implementations of it have been created by others and made available for public use. Rather than spend time translating Nvidia's model into usable code, one of these pre-existing implementations was used [41]. The original model was designed to output a steering wheel angle, which the Eyebot does not have, so it was changed to output a parameter for the Eyebot's angular speed. It was made to output in the form degrees per metre to be scalable with speed.

5. Assessing Driving Methods

5.1. Performance Assessment

5.1.1. Lane Keeping Performance

Running on the real-world track, the two Hough transform methods were run until an error occurred. The results are:

Method	Time Running Before Error (s)
Standard	382
Informative	130

Table 1 Run Times without an Error

Notably, the standard method's failures are rarely for failure to identify lines is often for reasons external to the detection algorithms. The real-world driving area is made from two tables connected in the centre. The heights of the tables differ by several millimetres, resulting in the back of the robot getting caught when driving from the lower table to the higher table. When the robot gets over it, its direction is shifted to the left or right.

The informative method also has a specific, regular point of failure on the track, where two separate sections of the track are within view of each other. The lane processing algorithm can misinterpret the wrong lanes as the desired lanes to follow, and attempts to follow them instead, resulting in leaving the track. The reason that this happens in this method and not in the standard method is that the robot control for the informative method wasn't able to be calibrated to the smoothness of the standard method. This resulted in the robot rotating left and right while driving. Occasionally, if it happens in the

wrong way, this can result in the conditions for identifying a crossing being satisfied, resulting in the robot picking the wrong lane to follow. By tuning PID parameters to smooth the robot's travel, this problem could potentially be eliminated.

5.1.2. Reactivity

As stated, the measure of success for this is the frame rate of the detection method. The results are as follows:

Method	Average Frames Per Second
Standard	16
Informative	19
End-to-End Neural Network	7.4

Table 2 Average Frame Rates

The decided minimum permissible frame rate was 10 frames per second. The two Hough transform methods were well above this threshold and performed satisfactorily. Unfortunately, the neural network method was not capable of achieving this standard on a Raspberry Pi. Notably, the informative method performed the better than the initial method. This is likely due to the greater degree of planning that went into its development resulting in a more efficient algorithm.

5.1.3. Situation Handling

The strategies were also assessed based upon their ability to handle various situations. For an autonomous vehicle, failing even once is unacceptable, so they will be measured against their ability to constantly perform:

Situation	Driving Method		
	Standard	Informative	Neural Network
Straight	Yes	Yes	Yes
Corner	Yes	Yes	Sometimes
Crossing	Yes	Yes	Yes
Dual Lines	Often	Often	Sometimes

Table 3 Driving Capability for Given Circumstances

The lanes are also measured against the rate at which they are successfully identifying a situation on a frame-by-frame basis.

Situation	Driving Method		
	Standard	Informative	Neural Network
Straight	Yes	Yes	Not Applicable
Corner	Often	Yes	Not Applicable
Crossing	Yes	Yes	Not Applicable
Dual Lines	Often	Often	Not Applicable

Table 4 Detection Rate for Given Circumstances

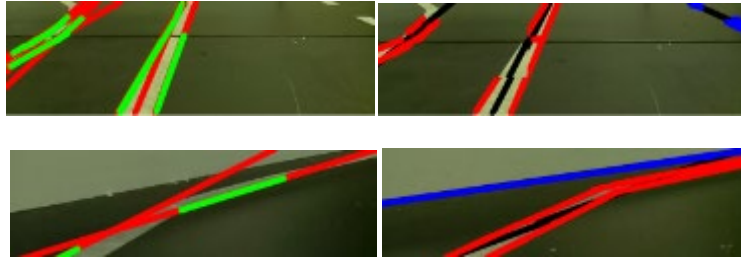


Figure 17 Initial Hough Method (left) vs. Informative Method (right) in identifying complex circumstances.

5.2. Further Criticism of Methods

The issue with developing lane detection and control methods based on standard image processing techniques is that it requires the designer to define every circumstance. For instance, if the lanes to follow are defined as the closest lanes to the robot on the left and right, then those conditions will follow the wrong lanes when a pedestrian crossing is present. If this is resolved by programming a degree of memory into the lane detection, this can result in situations where the wall is closer to the previous values than the new positions of the lanes, and the robot will follow the wall.

This is a problem because the method is trying to define and categorise a chaotic environment which will not play by the designer's rules. If a new situation arises, the designer must program a new means of detecting and handling that situation. Then there is the possibility that the new conditions will conflict with existing conditions, causing problems for existing situations which previously worked.

Neural networks solve this problem by training from existing situations. If something goes wrong, the network is trained on similar situations to the one that caused a problem until it can deal with them. This causes the issue that if something does go wrong it is very difficult to determine what the exact problem that caused it was. However, the limited training of the network severely impacted its ability to function. The angular speed may also be a much more sensitive parameter than steering angle, causing errors to have a greater impact.

5.3. Limitations

The main concern with regard to the validity of these implementations in a broad sense is their robustness in a variety of tracks. In simulations there are no boundaries to the tracks available as Eyesim allows for easily implementing new tracks that can be created using standard image design tools. However, the situation is much more limited in the real-world testing setting. Little space was available within the laboratory for creating test tracks, with any tracks created needing to be confined to a small area.

This is an issue because the primary measure of success for a project such as this is successfully functioning in a diverse range of circumstances. This is difficult to accomplish in an environment where the capacity to test different scenarios is limited to a small area.

Although the simulator implementations do not share this problem, they are also flawed in that they offer a pristine environment where imperfections must be explicitly put in. This runs contrary to the real-world scenario where effort must be exerted to eliminate imperfections.

Another significant flaw in these implementations is the complete dependence on one sensor.

6. Topics for Further Investigation

The ultimate goal is still to have a robot performing fully autonomous driving within its environment. With relation to lane detection, it would be extremely beneficial to further refine the neural network method and refine it to a state where it can drive reliably. The lower frame rate may ultimately be a somewhat acceptable sacrifice if it can be made to make the robot drive well enough.

In terms of creating a fully autonomous system, sign recognition and obstacle detection are natural requirements for creating a complete system. Beyond that, other requirements are for the vehicle to be capable of performing advanced manoeuvres such as overtaking and parking. Lastly, guided driving would be highly desirable. In the current driving method, the robot drives in a straight line. It would be desirable for the robot to be given a target point in a map and drive to it in the shortest method possible. Given that the current real-world track can be fully traversed by driving straight, this could be developed by first having the car drive straight until it reaches the desired point. Once a localisation algorithm is working on the straight line, the algorithm could then be refined to take the shortest path there using more advanced driving commands.

7. Conclusions

Multiple methods were successfully designed and implemented to work within the Eyesim simulation system and the test track. The implementations, however, are far from fully robust. The driving environments were extremely controlled and the robots nonetheless had limited driving success. When inserted into an entirely foreign driving environment that the robot wasn't designed in, it is possible that the Eyebot would not be capable of fully navigating the track in its current state.

Within the confines of this project, the most success in adhering to lanes was with the original Hough lines implementation, with the more informative Hough lines implementation being the second best and the neural network having the worst performance. The standard image processing methods are an effective way of driving the robot within the limited environments but they are ultimately limited solutions in how successful they can be when every possible situation must be accounted for by the designer.

Ultimately, the future of autonomous driving is likely to be heavily reliant on neural networks to function. With more training, a neural network is capable of surpassing the Hough lines method in driving performance. The methods developed in this project have potential to be effective in training a neural network to drive smoothly, but focus in the future should be placed on neural networks.

8. References

- [1] Intel, "Autonomous Driving – Hands on the Wheel or No Wheel at All," Intel, 11 April 2018. [Online]. Available: <https://newsroom.intel.com/news/autonomous-driving-hands-wheel-no-wheel-all/>. [Accessed 22 April 2019].
- [2] P. Maric, "2018 Audi A8: We've driven the world's first Level 3 autonomous vehicle," Car Advice, 8 September 2017. [Online]. Available: <https://www.caradvice.com.au/581052/2018-audi-a8-weve-driven-the-worlds-first-level-3-autonomous-vehicle/>. [Accessed 22 April 2019].
- [3] A. Davies, "The Very Human Problem Blocking the Path to Self-Driving Cars," Wired, 1 January 2017. [Online]. Available: <https://www.wired.com/2017/01/human-problem-blocking-path-self-driving-cars/>. [Accessed 22 April 2019].
- [4] K. Naughton, "Ford's Dozing Engineers Side With Google in Full Autonomy Push," Bloomberg, 17 February 2017. [Online]. Available: <https://www.bloomberg.com/news/articles/2017-02-17/ford-s-dozing-engineers-side-with-google-in-full-autonomy-push>. [Accessed 30 April 2019].
- [5] M. Martinez, "Ford rethinks Level 3 autonomy," Automotive News Europe, 20 January 2019. [Online]. Available: <https://europe.autonews.com/automakers/ford-rethinks-level-3-autonomy>. [Accessed 22 April 2019].
- [6] H. Schoner, "The Role of Simulation in Development and Testing of Autonomous Vehicles," Daimler, 7 September 2017. [Online]. Available: http://dsc2017.org/Docs/Keynotes/SimulationForAutonomousDriving_DSC17_Schoener.pdf. [Accessed 30 April 2019].
- [7] Volkswagen, "Carolo Cup 2018: Ready for the Autonomous Future," Volkswagen, 2019. [Online]. Available: <https://www.volkswagenag.com/en/news/stories/2018/03/carolo-cup-2018.html>. [Accessed 22 April 2019].
- [8] A. Wittke, "Audi Autonomous Driving Cup 2018: a focus on artificial intelligence," Audi, 21 November 2018. [Online]. Available: <https://blog.audi.de/audi-autonomous-driving-cup-2018-a-focus-on-artificial-intelligence/?lang=en>. [Accessed 22 April 2019].
- [9] Audi, "Audi Autonomous Driving Cup 2018 Rulebook," 16 April 2018. [Online]. Available: https://www.audi-autonomous-driving-cup.com/wp-content/uploads/2018/04/2018-04-16_en_Rulebook_2018_V1.0_EU.pdf. [Accessed 22 April 2019].

- [10 D. Silver, "How the Udacity Self-Driving Car Works," 7 September 2017. [Online]. Available:
] <https://medium.com/udacity/how-the-udacity-self-driving-car-works-575365270a40>. [Accessed 22
April 2019].
- [11 Audi, "Audi Autonomous Driving Cup 2018 - Sensors," 2018. [Online]. Available: [https://www.audi-
\] autonomous-driving-cup.com/car/sensors/](https://www.audi-autonomous-driving-cup.com/car/sensors/). [Accessed 22 April 2019].
- [12 Audi, "Audi Autonomous Driving Cup 2018 - Computers," Audi, 2018. [Online]. Available:
] <https://www.audi-autonomous-driving-cup.com/car/computers/>. [Accessed 22 April 2019].
- [13 Barnard, "Tesla & Google Disagree About LIDAR — Which Is Right?," Clean Technica, 29 July 2016.
] [Online]. Available: <https://cleantechnica.com/2016/07/29/tesla-google-disagree-lidar-right/>.
[Accessed 30 April 2019].
- [14 S. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. Eng and M. Ang, "Perception, Planning,
] Control, and Coordination for," MDPI, 2017.
- [15 D. Galar and U. Kumar, "Sensor Fusion," Science Direct, 2017. [Online]. Available:
] <https://www.sciencedirect.com/topics/engineering/sensor-fusion>. [Accessed 17 May 2019].
- [16 B. Zeisl, "Robot Control and Lane Detectionwith Mobile Phones," Technical University of Munich,
] Munich, 2007.
- [17 B. Zeisl, "modFrame.pgm," 23 May 2018. [Online]. Available:
] http://robotics.ee.uwa.edu.au/vision/lines/lane_detection_algorithm/lanewrapper/modFrame.pgm. [Accessed 18 May 2019].
- [18 Z. Bernhard, "Index of /vision/lines/lane_detection_algorithm/lanewrapper," University of
] Western Australia, 23 May 2018. [Online]. Available:
http://robotics.ee.uwa.edu.au/vision/lines/lane_detection_algorithm/lanewrapper/. [Accessed 17
May 2019].
- [19 H. Z. Z. X. & H. M. Li, "Two-stage Hough Transform Algorithm for Lane Detection System Based
] on TMS320DM6437," IEEE International Conference on IST, Beijing, 2017.
- [20 R. Jahan, P. Suman and D. Singh, "Lane Detection Using Canny Edge Detection and Hough
] Transform on Raspberry Pi," International Journal of Advanced Research in Computer Science,
Lucknow, 2018.
- [21 OpenCV, "Hough Line Transform," OpenCV, 2017. [Online]. Available:
] https://docs.opencv.org/3.4.3/d9/db0/tutorial_hough_lines.html. [Accessed 30 April 2019].

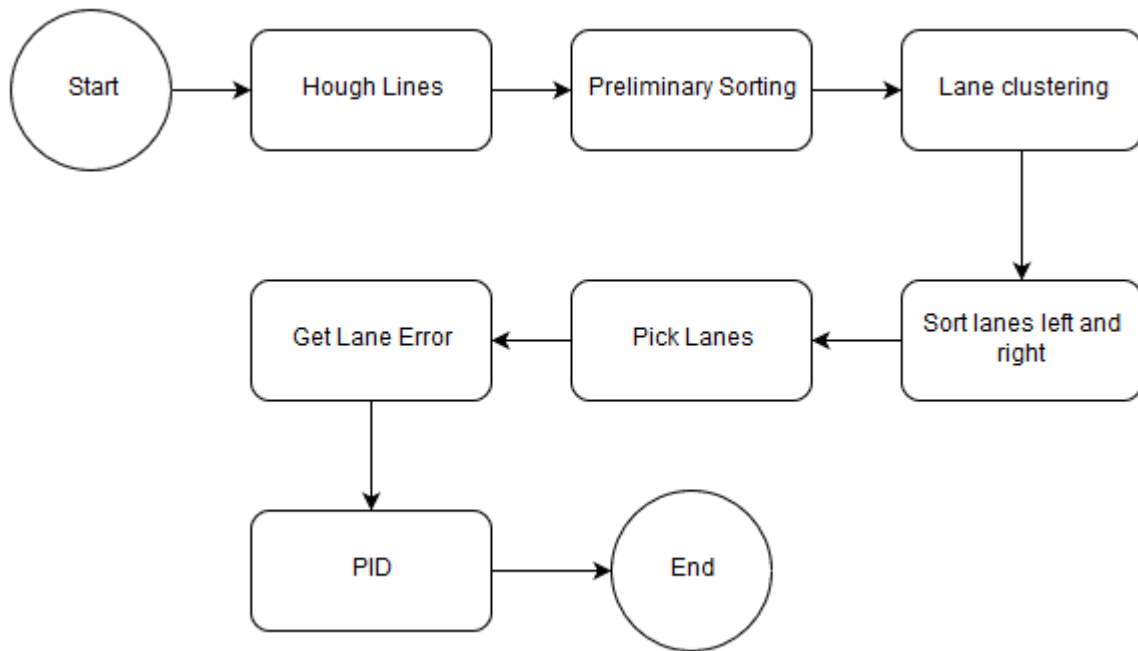
- [22 A. Solberg, "Hough Transform," 21 October 2009. [Online]. Available:
] <https://www.uio.no/studier/emner/matnat/ifi/INF4300/h09/undervisningsmateriale/hough09.pdf>
. [Accessed 30 April 2019].
- [23 R. Hota, S. Syed, S. Bandyopadhyay and P. Radhakrishna, "Simple and Efficient Lane Detection
] using Clustering and Weighted Regression," in *15th International Conference on Management of Data*, Mysore, 2009.
- [24 Stanford, "Hierarchical agglomerative clustering," Stanford, 7 April 2009. [Online]. Available:
] <https://nlp.stanford.edu/IR-book/html/htmledition/hierarchical-agglomerative-clustering-1.html>.
[Accessed 2019 April 30].
- [25 Q. Zou, H. Jiang, Q. Dai, Y. Yue, L. Chen and Q. Wang, "Robust Lane Detection from Continuous
] Driving Scenes Using Deep Neural Networks," Wuhan University, 2018.
- [26 B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T.
] Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates and A. Ng, "An Empirical Evaluation of Deep
Learning on Highway Driving," Stanford, 2015.
- [27 C. Shackleton, R. Kala and K. Warwick, "Sensor-Based Trajectory Generation for Advanced Drive
] Assistance System," University of Reading, Reading, 2013.
- [28 M. Gloderer and A. Hertle, "Spline-based Trajectory Optimization for Autonomous Vehicles with
] Ackerman drive," University of Freiburg, Freiburg im Breisgau.
- [29 R. Marino, S. Scalzi, G. Orlando and M. Netto, "A Nested PID Steering Control for Lane Keeping in
] Vision Based Autonomous Vehicles," in *American Control Conference*, St. Louis, 2009.
- [30 M. Bojarski, D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U.
] Muller, J. Zhang, X. Zhang, J. Zhao and K. Zieba, "End to End Learning for Self-Driving Cars," Nvidia,
Holmdel, 2016.
- [31 G. Marcus, "Deep Learning: A Critical Appraisal," New York University, New York, 2018.
]
- [32 J. e. a. Battaglia, "Relational inductive biases, deep learning, and graph networks," DeepMind,
] Google Brain, MIT, University of Edinburgh, 2018.
- [33 T. Braun, M. Pham, F. Hidalgo, R. Keat and H. Wahyu, "EyeBot 7 User Guide," 29 August 2018.
] [Online]. Available: <http://robotics.ee.uwa.edu.au/eyebot/EyeBot7-UserGuide.pdf>. [Accessed 30
April 2019].

- [34] T. Braunl, "EyeSim VR - Unity Based EyeBot Simulator," University of Western Australia, 2019. [Online]. Available: <http://robotics.ee.uwa.edu.au/eyesim/>. [Accessed 10 May 2019].
- [35] OpenCV, "Smoothing Images," OpenCV, 7 April 2019. [Online]. Available: https://docs.opencv.org/3.4.6/dc/dd3/tutorial_gaussian_median_blur_bilateral_filter.html. [Accessed 10 May 2019].
- [36] OpenCV, "Canny Edge Detection," OpenCV, 7 April 2019. [Online]. Available: https://docs.opencv.org/3.4.6/d7/de1/tutorial_js_canny.html. [Accessed 10 May 2019].
- [37] TensorFlow, "Why TensorFlow," Google Brain, 2019. [Online]. Available: <https://www.tensorflow.org/>. [Accessed 10 May 2019].
- [38] N. Doshi, "Spectral Clustering," Towards Data Science, 4 February 2019. [Online]. Available: <https://towardsdatascience.com/spectral-clustering-82d3cff3d3b7>. [Accessed 10 May 2019].
- [39] G. Rudolph and U. Voelzke, "Three Sensor Types Drive Autonomous Vehicles," Sensors Online, 10 November 2017. [Online]. Available: <https://www.sensormag.com/components/three-sensor-types-drive-autonomous-vehicles>. [Accessed 10 May 2019].
- [40] University of Texas, "Rotational coordinate transformations," 31 March 2016. [Online]. Available: <https://farside.ph.utexas.edu/teaching/celestial/Celestial/node122.html>. [Accessed 1 May 2019].
- [41] S. Chen, "Autopilot-TensorFlow," 23 April 2019. [Online]. Available: <https://github.com/SullyChen/Autopilot-TensorFlow>. [Accessed 13 May 2019].

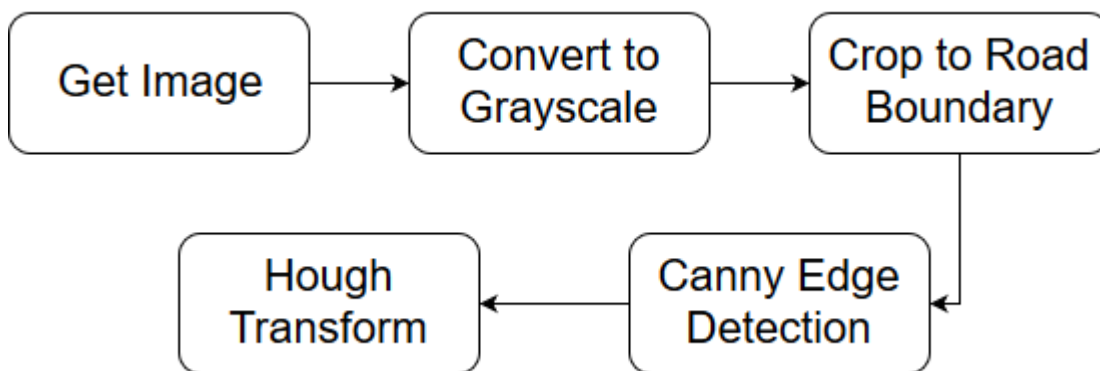
9. Appendix

Appendix A - Initial Lane Perception & Control Algorithm

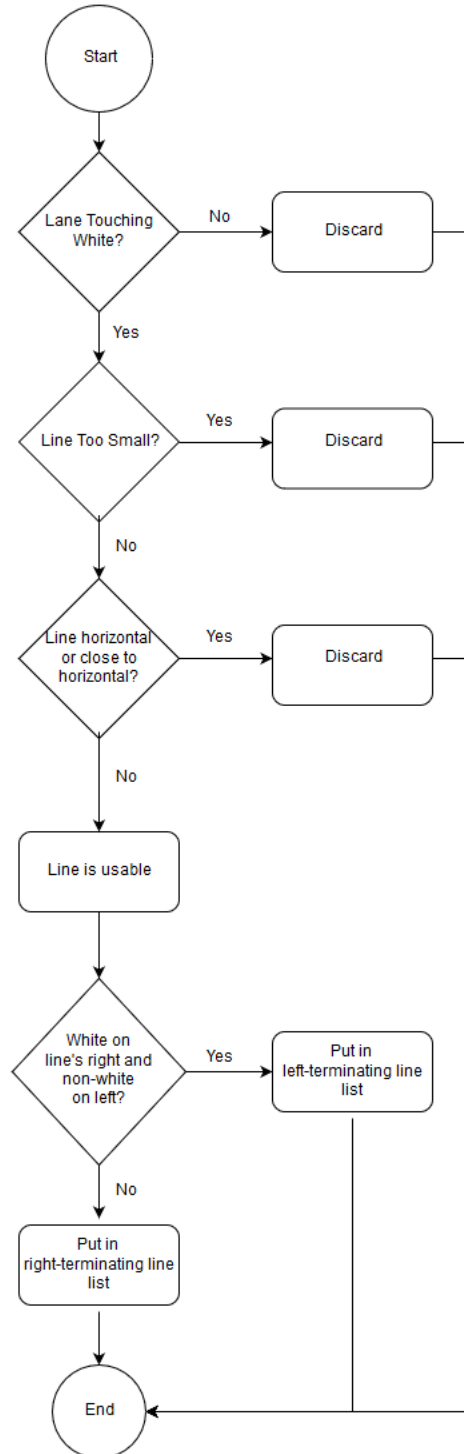
Appendix A.1 - Overview



Appendix A.2 - Hough Lines

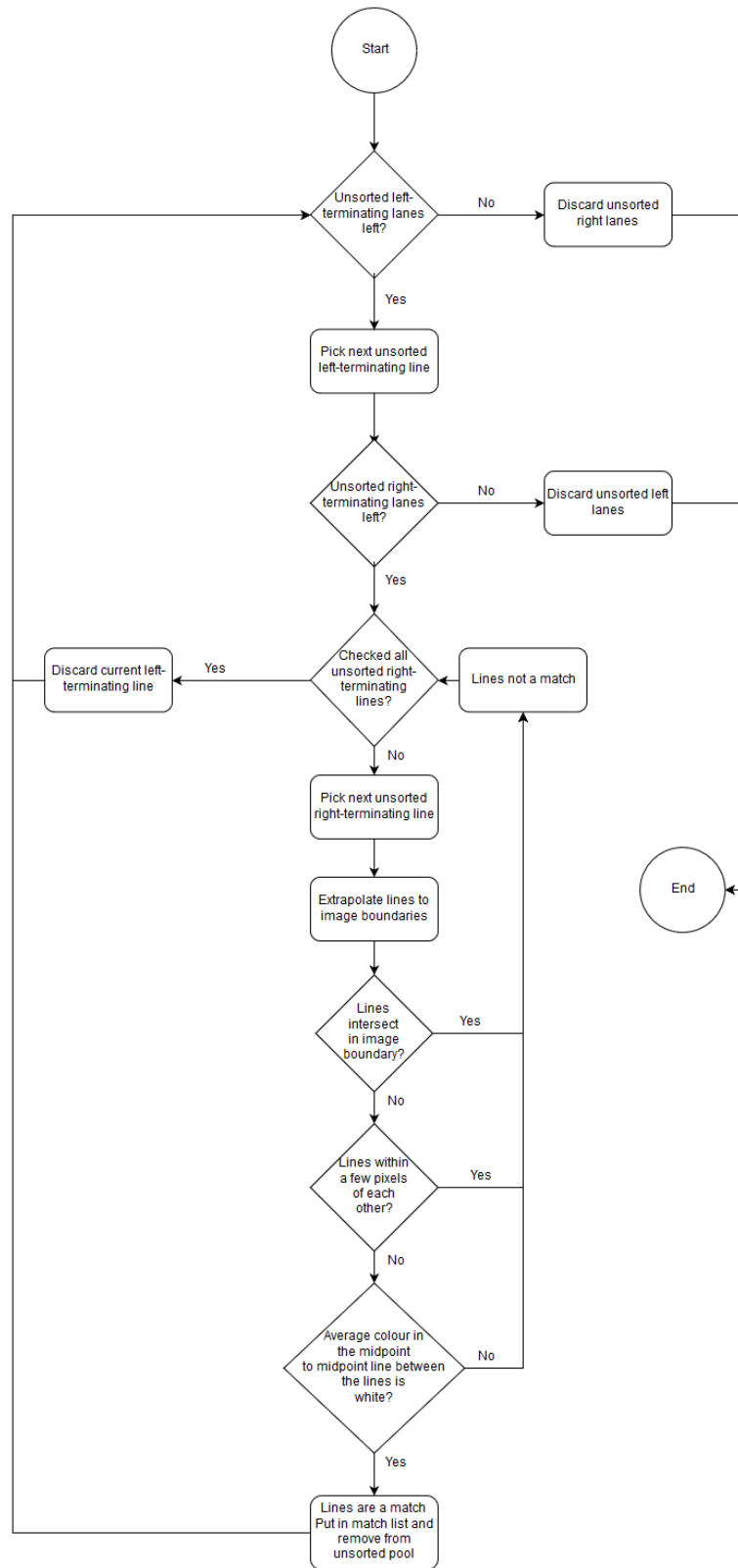


Appendix A.3 – Preliminary Sorting

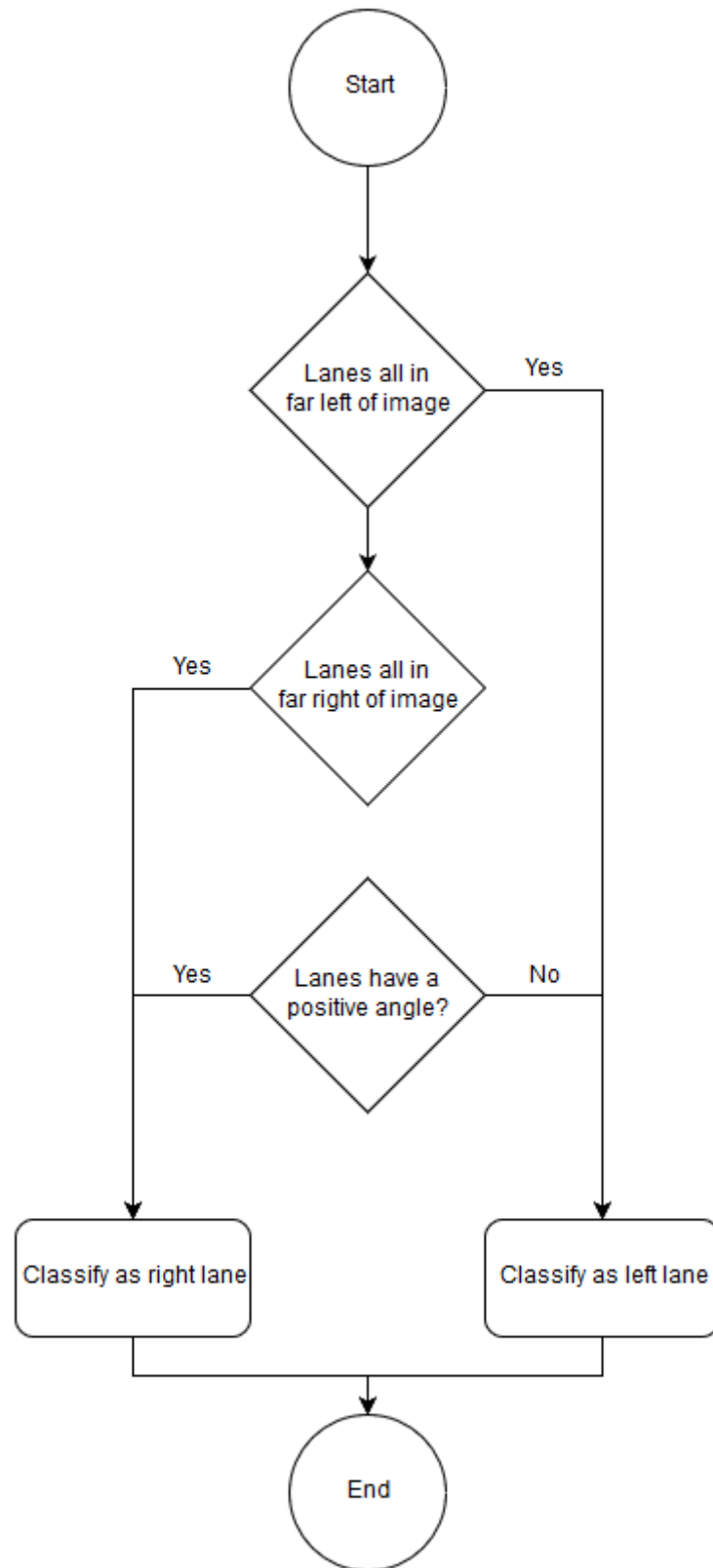


This process is repeated for each line extracted from the image.

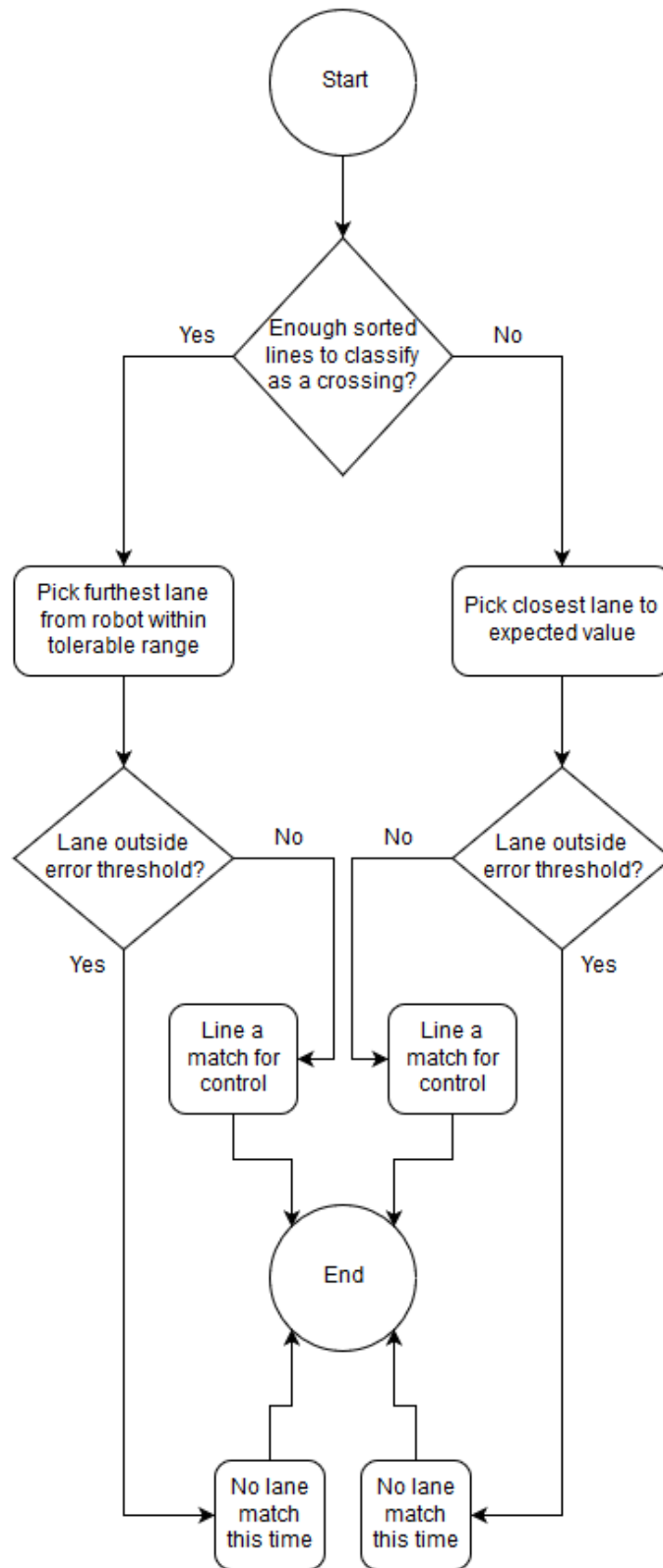
Appendix A.4 – Lane Clustering



Appendix A.5 – Sort Lanes Left and Right

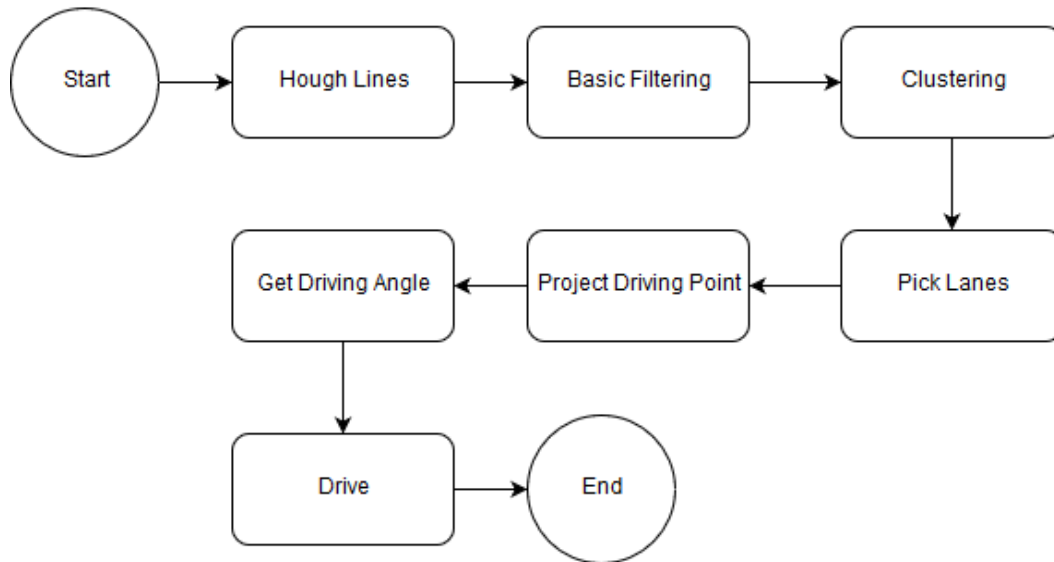


Appendix A.6 - Pick Lanes



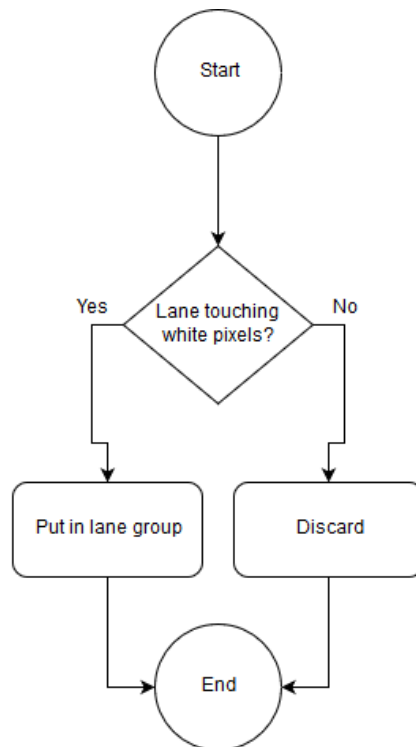
Appendix B – Informative Lane Model Algorithm

Appendix B.1 – Overview



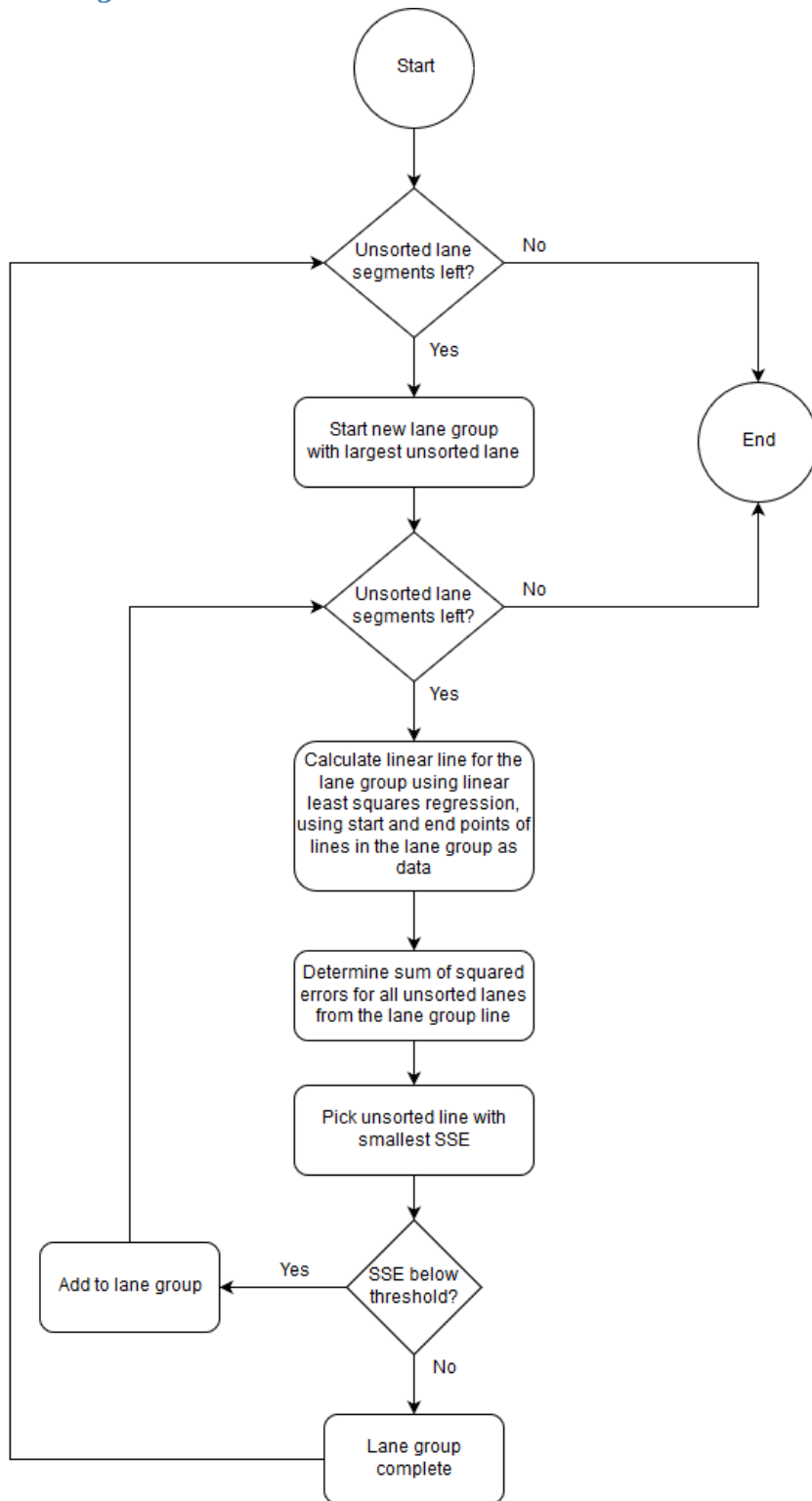
Pick Lanes operates as in the previous section and driving operates as described in the report body.

Appendix B.2 – Basic Filtering

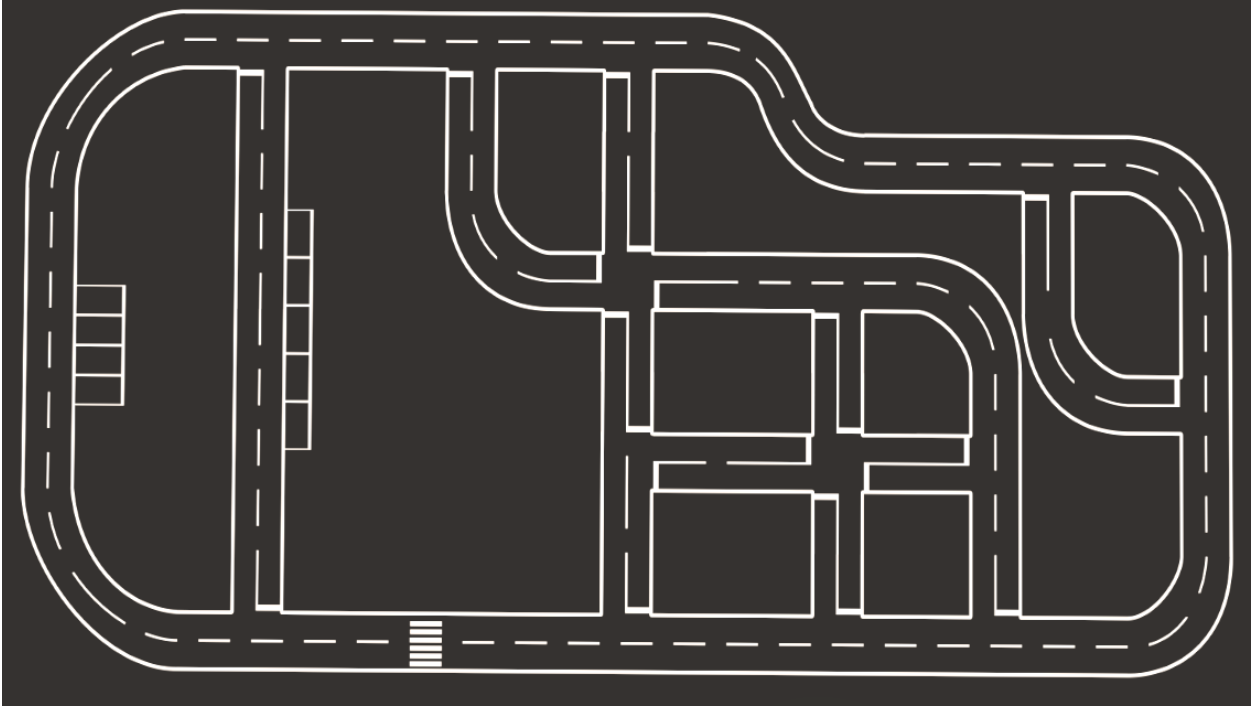


This is applied for each line.

Appendix B.3 – Clustering



Appendix C - Simulation Test Track



Appendix D - Real Test Track

