CHALMERS
UNIVERSITY OF TECHNOLOGY

Car Vision: Lane detection and Following

Bachelor's Thesis in Electrical Engineering

Aditya Subramanian

Department of Electrical Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2017

Acknowledgements

This is a bachelor's thesis as part of Chalmers University of Technology's electrical engineering programme. The thesis was carried out on behalf of the embedded systems department of Infotiv in Gothenburg, Sweden. The aim of the thesis was to add functionality to the existing autonomous car education platform.

I would like to thank Doctor Jonathan Gustafsson and Olle Norelius at Infotiv for their tremendous help as technical advisors. This thesis would not be what it is without their guidance and encouragement. I would also like to thank Pierre Ekvall and Fredrik Lundgren also at Infotiv for believing in me and giving me the chance to conduct my thesis at Infotiv.

Abstract

During the past 17 years, technology has progressed at astronomical speeds. We have experienced not only the birth of many new technologies but also the miniaturization of both technologies new and old. It would have been audacious in the beginning of the new millennium to expect the amount of technological development. Perhaps the next biggest change to our lives is going to be self-driving or autonomous vehicles. Personally, my motivation was grounded in the fact that I received an opportunity to learn a tremendous amount of knowledge in both a niche, modern, highly-technological field and gained experience in the different facets of software development such as software architecture, design and testing. Additionally, having full autonomy over the decision process and design choices added a great sense of responsibility.

This thesis deals with lane detection and driving of such autonomous vehicles. The goal of this thesis is to program an existing autonomous car, built using a Raspberry Pi 2, an Arduino Mega and 4 DC motors, to successfully navigate a black and white mat which is supposed to function as a model of a racetrack. A small camera is mounted on top of the car and this is used in conjunction with different computer vision techniques to analyse data and correctly predict the course the car should navigate.

This thesis shows how line and lane detection is possible with the provided hardware. The system can find edges from a camera picture, classify these edges and calculate a virtual line that the car attempted to drive according to. A camera stream was implemented as a proof of concept which displayed the above in addition to an instrument panel which showed the errors the car tried to regulate. However, due to time constrains, this thesis was not able to correctly regulate the errors calculated.

To summarize, I inherited hardware from a previous project and built the software required for the car to be able to drive autonomously. The software created for this thesis had no influence from the software used for the previous project.

Table of Contents

1.Introduction 1
1.1 Goal 1
1.2 Project Specification 1
2. Technical Background 3
2.1 Hardware 3
2.2 Software 4
2.2.1 Python 4
2.2.2 Interface to the car 4
2.2.2 Architecture 4
2.3 Libraries 5
3. Method 8
3.1 Modules 8
3.1.1 Image Processing 8
3.1.2 Edge Detection 10
3.1.3 Classification 11
3.1.4 Virtual Line 11
3.1.5 Deviation 11
3.1.6 PID – Controller 12
5. Results and Discussion 13
6. Conclusion 15
7. References 16

1.Introduction

Over the past years, several companies such as Volvo and Delfi have invested in the research of self-driving or autonomous cars. The hope for future, I believe, is that all cars will be driven by computers and humans will have more of a monitoring role in the context of driving. Perhaps sometime in the future humans will be purely passengers in a car.

This thesis was done conducted on behalf of Infotiv under their Embedded Systems department. As a consultant company Infotiv wishes to strengthen the base technical competences of all its employees. As such, over the recent years Infotiv has developed different education platforms. One of these education platforms has been their autonomous vehicles platform. This thesis is intended to be part of the continuous development of this education platform at Infotiv.

Prior to this thesis, a previous project called "*Car Vision*" laid the groundwork for this thesis. The purpose of this project was to create an autonomous car, referred to in the rest of this report as "the car", which was capable of following road lines, markings and signs and drive accordingly. The car was tested on a cloth mat with white tape meant to function as a model of a race track. After the conclusion of the "*Car Vision*" project the car was able to drive on top of the white lines the car recognised as road lines and recognise different road signs such as stop signs and speed signs. Additionally, a remote interface was created which could show the view from the camera, show the last recognised sign and show the confidence of the software that it had recognized the sign correctly.

1.1 Goal

The goal of this thesis is to improve line detection and create functionality for the car to be able to recognise and drive in lanes. The hardware will be the same hardware used in the "*Car Vision"* which comprised of a Raspberry Pi 2, an Arduino Uno and 4 DC Motors to control the individual wheels.

These 3 components are mounted on top of a small, plastic frame. The testing for the car will be done on a black mat with white tape, modelling a racetrack. Some communication will be done over Wi-Fi using SSH protocol, but the goal is to transfer all functionality over to the Raspberry Pi to make the car a standalone product.

The thesis consists of three main parts in the following order of priority:

- On top of existing functionality from the "*Car Vision"* project, the car should be able to recognise and drive in lanes
- Software enhancements and optimization to previous project code
- Adding functionality to the user interface for the car

1.2 Project Specification

This thesis will only be dealing with an autonomous vehicle, referred to as "the car" for the rest of this report and a model of a racetrack, referred to as "the mat" for the rest of this report. It is not intended to run on a real road with a real car. As mentioned, the hardware will be a small autonomous car. In addition to the existing functionality, the following will be added to the car over the course of this thesis:

- Improvement in speed of line detection
- Distinguishing between striped and single white lines
- Drive in a lane. A lane is defined as the area confined within a striped and single white line.
- Simultaneously to the above, previous functionality should keep on operating.

The autonomous car should successfully navigate a small track with two lanes and models of various traffic signs such as stop and speed signs. All calculations will be preferably be performed on the on-board Raspberry Pi 2. If this not possible a computer will be used instead. The communication between the car and the computer in this case will be over Wi-Fi using the SSH protocol which will be outlined in more detail below. Furthermore, Olle Norelius will provide an encryption free, interface to simplify communication between the computer and the car. This interface will make it possible to display and monitor the status of the autonomous car in real-time and simplify the signal processing from the computer to the car.

2. Technical Background

This chapter covers the hardware, software and software architecture used.

2.1 Hardware

The hardware for this thesis was inherited from its predecessor, the "*Car Vision*" project (Figure 1). It is important to note that this thesis did not deal with any hardware modifications. As such, the hardware description is taken from the project documentation of the "*Car Vision*" project.

The hardware comprised of the following parts:

- Raspberry Pi 2
- PiCam
- Arduino Mega 2560
- Motor Shield
- Car chassis with motors and wheels
- 2 x 200 mAh Li-Po accumulators
- Powerbank (10 000 mAh)
- Wi-Fi Dongle (TP-LINK TL-WIN722N 150 Mbps)
- Red LED with resistor
- Breadboard with 5-3.3V level shift

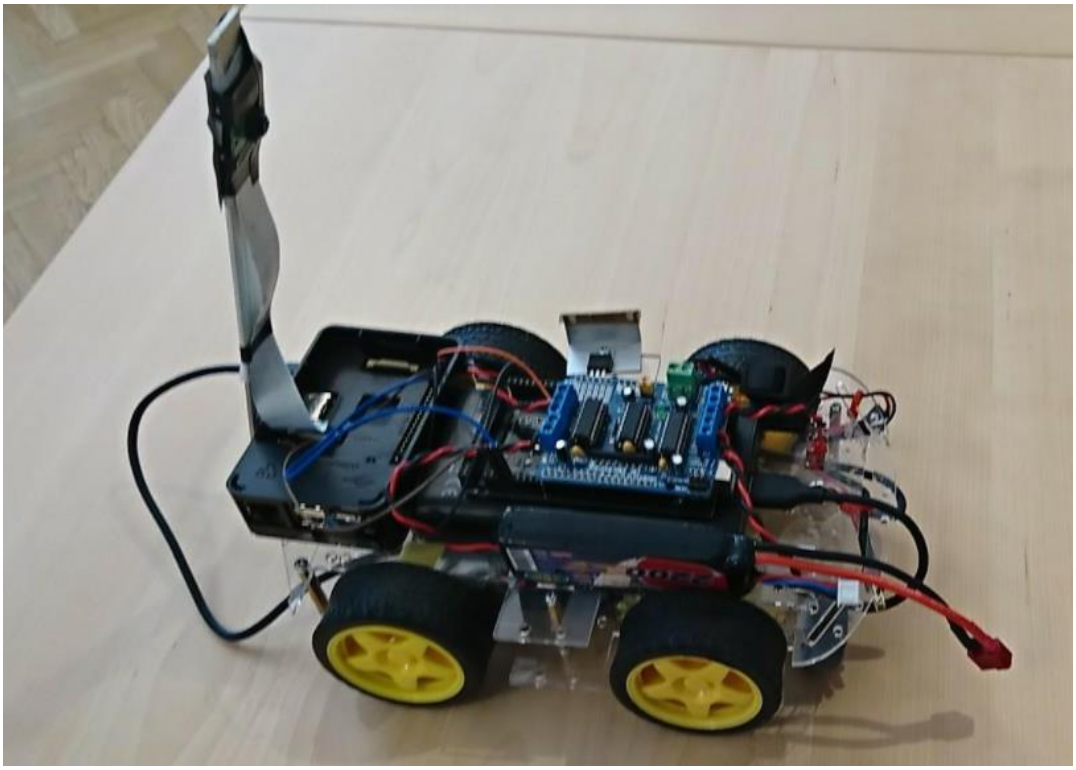

Figure 1: Hardwrae inherited from "Car Vision" Project

2.2 Software

This chapter will provide an overview of the software developed during the course of this thesis.

2.2.1 Python

Python was chosen as the programming language. This choice was based upon the recommendation and advice of Dr. Gustafsson. This recommendation was based upon the fact that Python is a programming language, despite its simplicity, has immense power and a swath of community driven, open-source libraries. Since Python is so widely used, an advantage it provided in hindsight compared to other programming languages is the tremendous amount of information available online which made debugging and trouble-shooting much easier.

2.2.2 Interface to the car

Olle Norelius provided an interface to ease communications with the car. This interface included a server where the software from this project could send commands to the car. This meant that instead of having to signals to each individual motor and wheel, the software could send commands which controlled the turn rate and speed of the car.

2.2.2 Architecture

The two backbones of the successful completion of this thesis were Test-Driven Development and SCRUM.

Test-Driven Development

Test-Driven Development or TDD "is an evolutionary approach to development which combines test-first development where you write a test before you write just enough production code to fulfil that code and refactoring" [1]. This is the standard practice at the Embedded Systems department at Infotiv. Test-Driven Development grounds itself in the philosophy that if one writes code first and then the tests are written for the code to pass and thus one may forget the original specification. Over the course of the thesis, the author has learnt to appreciate the rigidity Test-Driven Development provides to software development.

Software testing is generally comprised of different levels of tests. Ranging from unit test, the lowest level, to integration tests, the highest level. The purpose of unit tests is to test individual functions or modules (units) while integration tests are meant to test the whole process chain.

Before coding each function, a unit test was designed and implemented. A function was then considered completed if and only when it passed the test. The thought process then is that a test will only stop working if either the function is not fulfilling the requirements, in which case the function needs to be refactored or the test is expecting a worse result than the function provides in which case the test needs to be re-designed. An integration test for the whole thesis was built by chaining these unit tests together.

SCRUM

Scrum which 'is an Agile framework for completing complex projects." [2] , was the project management framework chosen for this thesis. The process starts with choosing an amount of time, referred to as a sprint, 2 weeks for this thesis. Every 2 weeks the author and Dr. Gustafsson had a sprint meeting where we discussed the progress of the thesis over the past 2 weeks and decided tasks which were to be completed over the next sprint. This gave a clear idea what was necessary and a status quo of the thesis. Twice a week during each sprint, the author and Dr. Gustafsson had a short meeting where we discussed the current issues and discussed solutions to these issues. This was to ensure that there were always tasks that needed to be completed. Scrum allowed for

constant evaluation over the progress and a structured plan for the successful completion of the thesis.

2.3 Libraries

As previously mentioned, Python has a tremendous number of open-source libraries. This thesis mainly used two such libraries.

2.3.1 OpenCV and Computer Vision

OpenCV or "Open Source Computer Vision Library is an open source computer vision and machine learning software library" [3]. OpenCV includes a wide-variety of functions dealing with computer vision.

A small note on computer vision. Human vision is a very complicated process and the conscious mind never sees the raw data from the eye. Here however we will treat the images as a matrix. Each element in the matrix represents a pixel. The number of elements in the matrix is related to the resolution of the matrix. There are three common models to represent pixels, RGB (Figure 2) or BGR (Red/Green/Blue or Blue/Green/Red), grey scale (Figure 3) or HSV (Figure 4) (Hue, Saturation, Value). Only RGB/BGR and grey scale representations were used in this thesis.

In the case of RGB representation each pixel has three values where each value corresponds to the colour intensity of the respective colour (red, green and blue). In the case of grey-scale, each pixel has one value which denotes the light intensity white/black for each respective pixel.

For this thesis OpenCV was used mainly for image processing and to store/load images. The specific usages are described in more detail under the method chapter of this report.

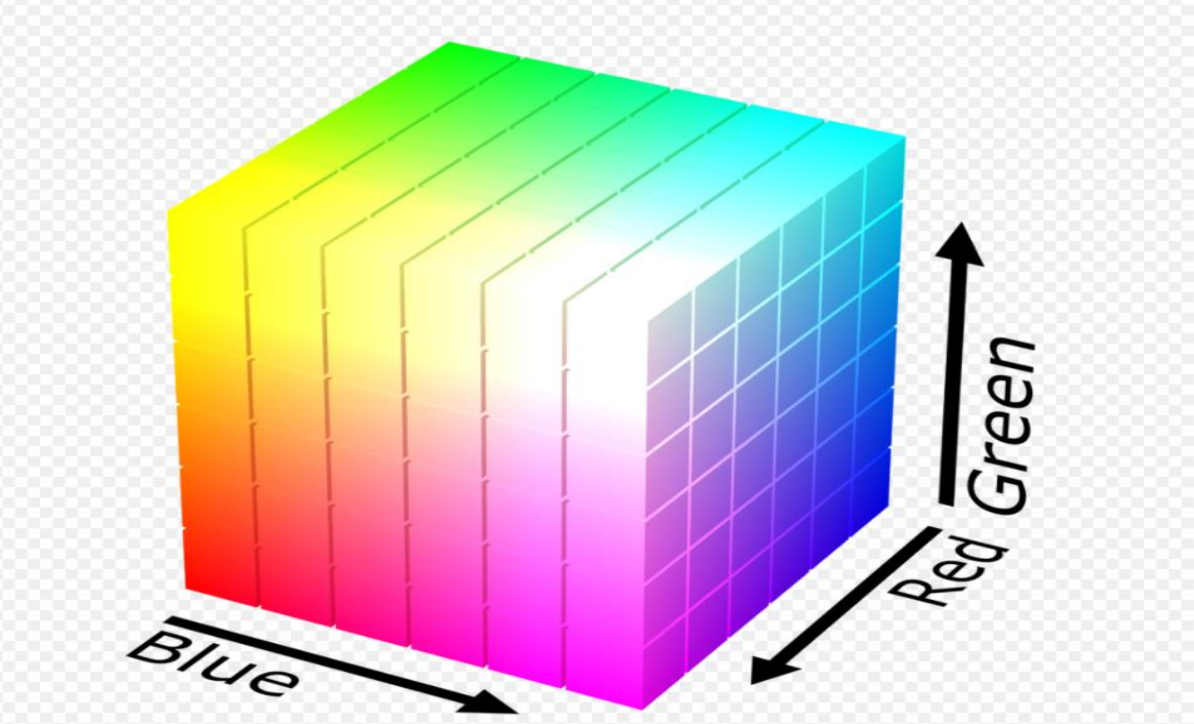

Figure 2: RGB color space

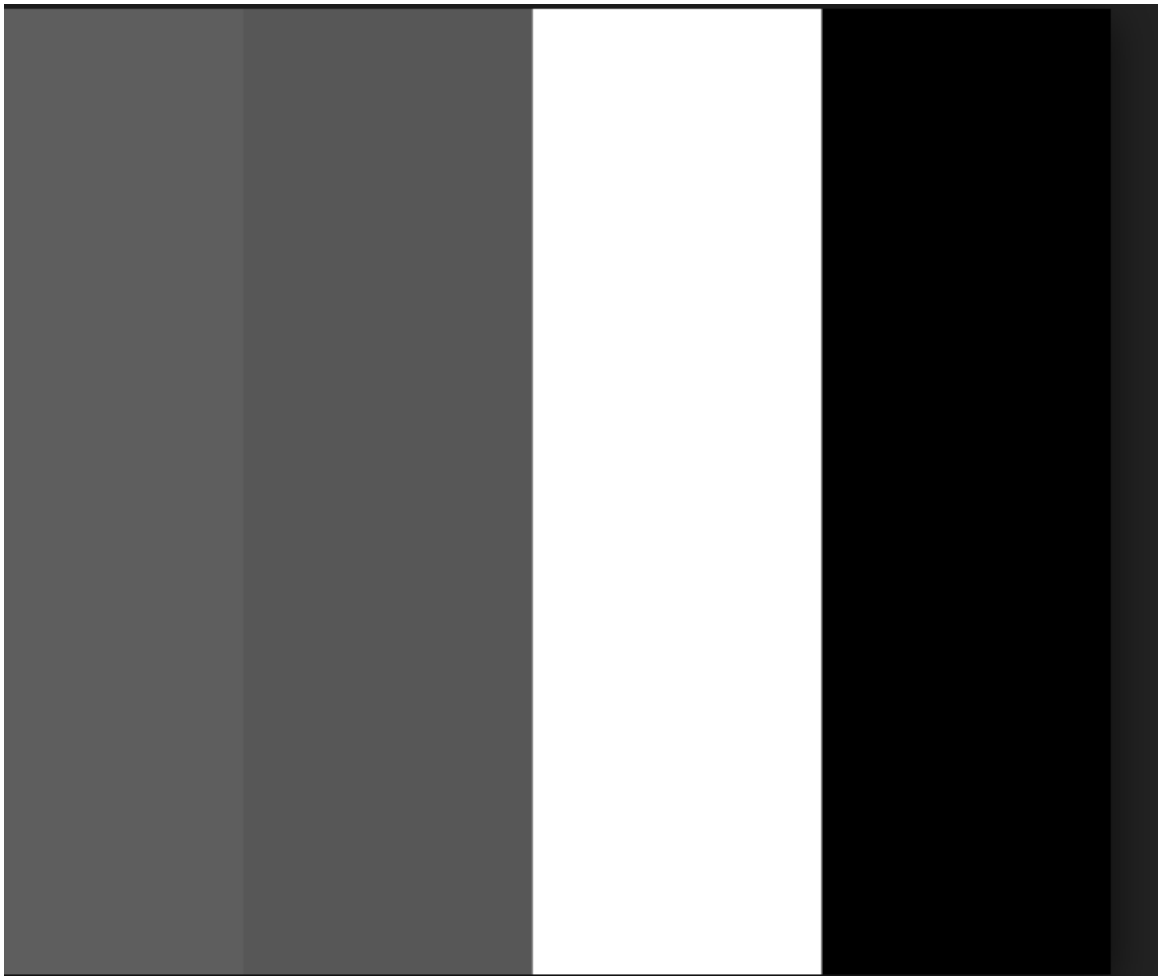

Figure 3: Gray scale colorspace

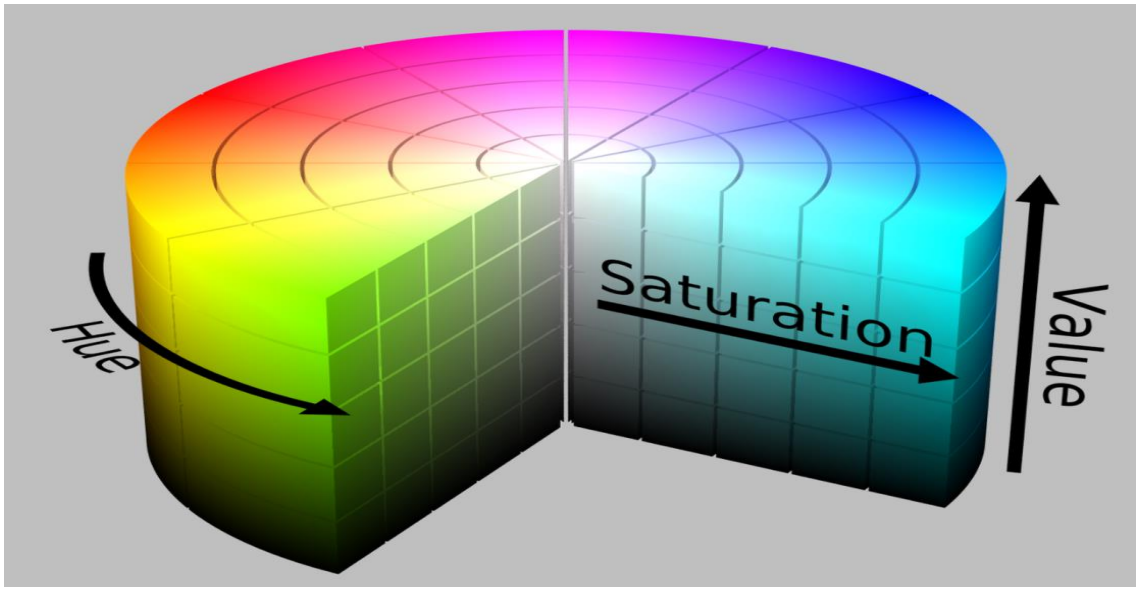

Figure 4: HSV Color space

2.3.2 NumPy and SciPy

NumPy is “the fundamental package for scientific computing with Python”[4], while SciPy is a similar library containing more full-fledged linear algebra functions. NumPy has provided the backbone for the numerical calculations and analysis done as part of this thesis. SciPy has been used exclusively for its clustering functions, specifically k-means clustering.

3. Method

As outlined in the Software Architecture section of this report, the Scrum framework was used for project management. During the first sprint meeting a detailed plan of the thesis was decided upon. Additionally, several key decisions including product design choices were made during the first meeting.

It was decided that the code would be modularized. This provides a great advantage in the form of each module being plug-and-play. It eases future development because input and output to each function is already in place.

Furthermore, a plan was outlined on what functionality needs to be implemented so a picture taken by the PiCam can be converted to steering the car. This plan included the function of each individual module, the input the module will receive and the output the module will send to the next module. Possible future problems and possible solutions were also discussed during the planning phase.

3.1 Modules

The modules below are listed in the order they are executed. Below (Figure 5) is a flowchart of the system architecture.

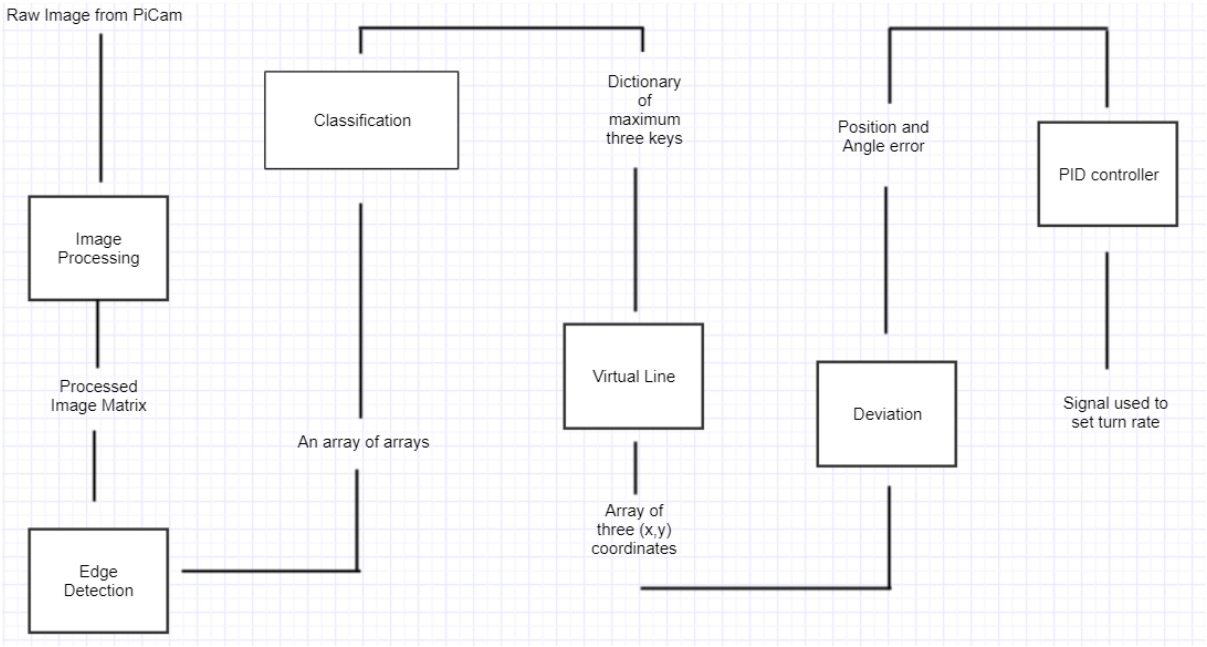

Figure 5: System Architecture

3.1.1 Image Processing

Input to module	Output from module
Raw image from PiCam	Processed Image Matrix

The image processing module is the first module. It receives its input from the PiCam in form of a raw image. Below is an example of an image this module can receive (Figure 6). The resolution of the picture received from the PiCam is 2592 x 1944. Due to this being too large to show on a standard laptop screen it is re-sized to 320 x 240. It is then converted to a gray-scale representation of the image from RGB representation using OpenCV's cvtColor() function. Grey scale is the preferred representation in computer vision as it makes it easier to analyze the image as each pixel is

represented by a singular value. After, the image is converted to grey-scale it undergoes common image processing techniques.

Figure 6: Image received from PiCam

Equalized Histogram:

First it undergoes histogram equalization. This "is a method that improves the contrast in an image to stretch out the intensity range" [5].

Thresholding

Afterwards a threshold is applied to the image using OpenCV's threshold() function with the binary style. This transforms the image from gray-scale to black and white. For example, if a threshold of 240 is chosen all pixels with a value larger than 240 are set to 255, else the value of the pixel is set to 0. At this point all the pixels in the image are either 255 (white) or 0 (black) (Figure 7).

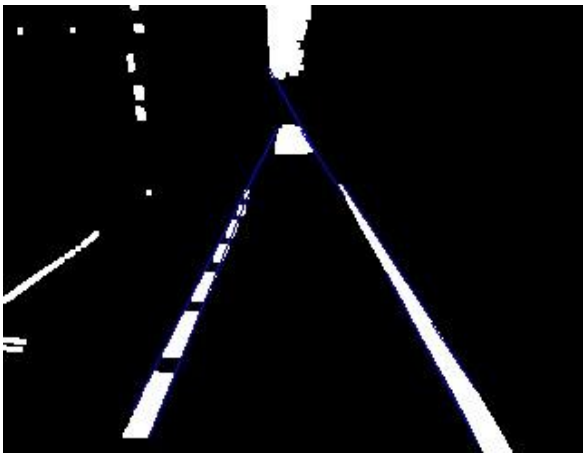

Figure 7: Image post thresholding

Erosion and Dilation

Erosion and dilation are two basic mathematical morphological transformations, usually applied on binary images, which mirror each other. The main usage of these operations is to increase the contrast in the image by either removing disturbance (commonly referred to in computer vision as noise) or enhancing by enhancing edges.

These operations require two inputs, an image to transform and a structuring element (or a kernel). For this thesis a 3 x 3 matrix was used. The corners of the matrix have values of 0.2 while the rest of the matrix have values of one. The reasoning behind this was that a kernel of only '1' values were eroding curves harshly and by having values close to zero,

The idea is to erode noise and make small white spots darker. "The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all pixels under the kernel is 1, otherwise it is eroded (made to zero)." Dilation is the opposite where "a pixel element is '1' if at least one pixel under the kernel is '1'. So [,] it increases the white region in the image or the size of foreground object increase" [6]

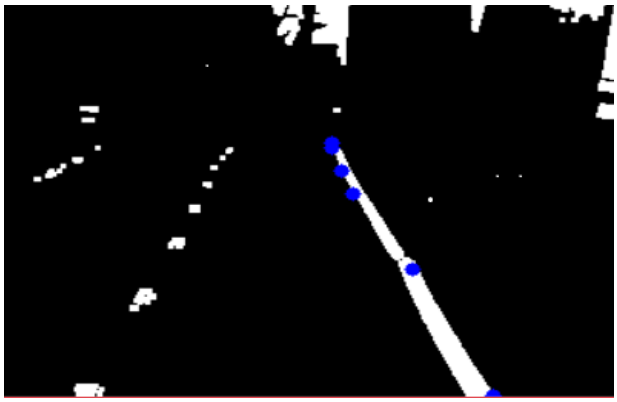

Figure 8: Pre-Region of Interest. N.B The blue dots have no relevance at this stage

Region of Interest

To avoid the image including surrounding furniture and other objects which would make the image noisier, a region of interest was implemented to create a horizon effect.

The region of interest was decided by placing the car furthest along the straight lane of the mat and taking a picture at this point. Figure 8 shows the picture that was taken. As seen on the picture the top third of the picture is simply noise and irrelevant data and therefore our region of interest is only the bottom 2/3rd of the picture.

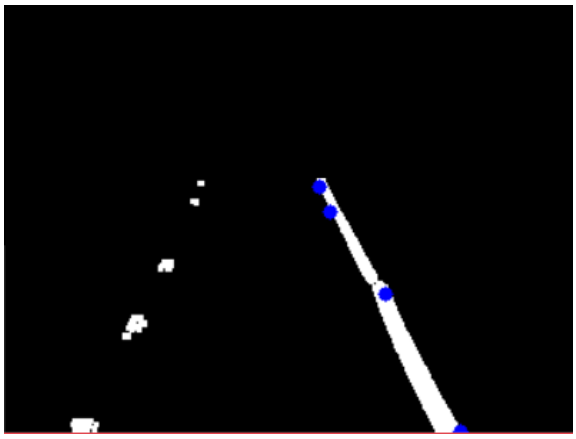

Figure 9: Post- Region of interest. N.B the blue dots have no relevance at this stage

The region of interest is implemented by drawing a black, filled rectangle. Also, two black, filled, triangles on each side are drawn underneath the rectangle. This was because right after approaching a curve, noise on the peripheral vision of the car would cause problems for the algorithms which follow. After applying the region of interest, the image will resemble the image in figure 9. This is done before erosion/dilation in case the horizon effect created is on the edge of a white line which will be removed and enhanced by erosion and dilation respectively.

3.1.2 Edge Detection

Input to Module	Output from Module
Processed Image Matrix	An array of arrays. Each array contains 4 values which are the start (x,y) and end (x,y) coordinates of line segments detected.

This module is responsible for finding all the lines in the processed image matrix. The original intention was to do this manually. The idea was that since the input is a binary (black and white) image, all pixel values are either 255 (white) and 0 (black). Thus, by differentiating the matrix and finding where the pixel value changed from either 255 to 0 or 0 to 255 would provide the edges.

However, while implementing this idea the author found a technique commonly used [7,8] for edge detection known as Hough Transform.

Hough transform "is a technique which can be used to isolate features of a particular shape within an image. Because it requires that the desired features be specified in some parametric form, the *classical* Hough transform is most commonly used for the detection of regular curves such as lines circles, ellipses, etc." [9] For the purpose of this thesis OpenCV's HoughLinesP() function was used.

HoughLinesP() is a more efficient, probabilistic version of Hough transform. The algorithm uses a line segment described by given parameters (length, r and angle, theta) and rotates it randomly throughout the image and saves how many points were on this line segment. Depending on the parameters used the algorithm rejects line segments which either have too few points, points too far away from each other or line segments which are too short.

3.1.3 Classification

Input to Module	Output from Module
An array of arrays. Each array contains 4 values which are the start (x,y) and end (x,y) coordinates of line segments detected.	Dictionary with maximum of three keys (left, middle and right) where each entry is a array with three (x,y) coordinates representing the start, middle and end of each line segment.

This module is responsible for classifying all the line segments found by the previous module and choosing the correct line segments. To do this the "K-means clustering" algorithm is used. [10] The number of clusters, k, is used as an input to the algorithm. Then the algorithm starts by guessing k number of centroids which it expects to be in the center of each cluster. After this, each point is assigned to the nearest centroid, creating clusters. Once the clusters are formed, centroids are updated by calculating the mean of each centroid. Finally, a "within cluster sum of squares" (WCSS) is used to measure the error. WCSS, measures the distance between each point and it's assigned cluster and summates this error for each cluster respectively. This process is iterated until the WCSS error is optimized to minima.

3.1.4 Virtual Line

Input to Module	Output from Module
Dictionary with maximum of three keys (left, middle and right) where each entry is a array with three (x,y) coordinates representing the start, middle and end of each line segment.	Array of three (x,y) coordinates

This module is responsible for finding the line according to which the car drives. The line is the mean line between either the middle and left lines (if the car is to drive according to left hand drive) or the mean line between the middle and right lines (if the car is to driving according to right hand drive). This is done by taking the mean of each (x,y) coordinates for the start, middle and end point of the respective lines. This virtual line is also the line the car is supposed to drive according to.

3.1.5 Deviation

This module is responsible for finding how far the car is from the virtual line. It calculates two errors, position and angle. Position error is how far away the car is from the virtual line in pixels. The middle point of the virtual line is used to calculate this difference. The function takes the difference in (x,y) coordinates of this point and the middle of the image as we know it's dimensions. The angle error is the angle between the virtual line and the normal of the car. It is shown in figure 5.

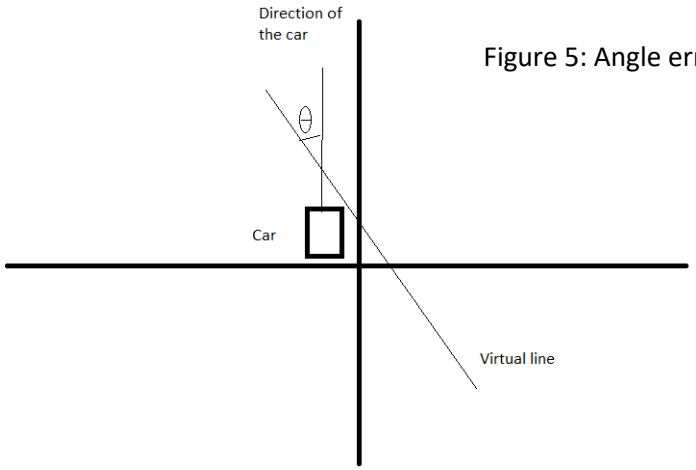

Figure 5: Angle error

3.1.6 PID – Controller

Input to Module	Output from Module
Position error: The pixel difference between the virtual line and the direction the car is facing Angle error: The angle with respect to the y-axis between the virtual line and the direction the car is facing	Output signal from PID used as the turn rate for the car

This module simulates and implements a PID (Proportional – Integral – Derivative) controller. Due to a very noisy signal used as an input only the Proportional part of the controller is implemented. Two different controllers regulate the angle and position errors. The individual controllers receive the error as an input and multiply the error by some constant before outputting this value. The output from the individual controllers are summated to be used as the signal to the car as the turn rate.

5. Results and Discussion

Along each step of the thesis, critical design choices were made. These choices were made both on a micro level, to benefit the individual modules performance, and on a macro level, to benefit the system. The philosophy in context of design choice was to make a decision early and to consider alternative solutions if and only if the original implementation was lacking. However, this meant that alternative solutions were not explored if the original implementation worked sufficiently well. Future research in this area however would benefit greatly from considering these alternatives and as such an overview of these alternatives is worth discussing.

OpenCV as a library provides a vast number of image processing and edge detection algorithms. Hough transform requires edges to be very clear visually. There needs to be a high contrast between the pixel where the edge is and the surrounding pixels. To aid the Hough Transform, thresholding was used which converts a grey image to a binary image, increasing the efficiency of Hough Transform. However, there are other functions provided by OpenCV such as canny edge detection and contour finding which can be used as alternatives for finding edges.

It was difficult to appreciate what changing the parameters of Hough Transform did. A lot of time was invested into researching and trying to improve the transform with no clear success. Therefore, the Hough transform, once functioning sufficiently, was treated as a black box. Any optimization was done in the modules around the edge detection module. One issue which caused issues was the fact that the Hough transform chosen was a probabilistic. This means that the same image used as an input several times will not return the same output. To counteract this effect, the output from the Hough transform was passed through a median filter. The transform was run several times and the median value was passed to the classification algorithm. Due to reasons unknown, despite the Hough transform being probabilistic, the Hough transform returned the exact same result for each frame up to 8 decimal places meaning that the median filter had absolutely no impact on the result as the median filter was simply outputting it's input.

The k-means clustering algorithm presented its own problems. In the original implementation of the clustering algorithm the raw output of the Hough transform, the coordinates of the extremes of each line segment was used. This created an issue where the two edges of the same road line would be classified as two different lines. This was because the algorithm was only receiving information about pixel coordinates of the line segment and no information about the angles relative to each other. To counteract this the output of the Hough transform was altered before being passed into the clustering algorithm.

Instead of using (x, y) coordinates of the extremes of the line segments, the following four values were used as input:

- The (x, y) coordinates of the mean point on the line segment
- The length of the line segment
- The angle of the line segment relative to the x-axis

This lead to better results because now the algorithm received the relative angles of the line segments in relation to one another. Also, the length of the line segments meant that the output would not be very short line segments.

The virtual line was trivial to implement by taking the mean of each respective start, middle and end point of the two lines which defined the region of the lane. Something to note in the virtual line is

that as the mean was taken the car always drove according to a straight line which meant that the car drove akin to a rally car driver i.e. hitting the apex of each curve.

Calculating the position and angle error in the deviation module presented the biggest issues to the successful completion of outlined goals. The original implementation simply calculated the difference in pixels between where the car was and where the virtual line was calculated to be. This did not work sufficiently well as both the angle and position error are dependent on the direction the car was facing and therefore the error calculated was not a physical quantity in the real-world context. Approaching the conclusion of the thesis, a solution where transforming the picture to a birds-eye, real-world representation using trigonometric functions was tried and stabilized the output.

The PID controller was easy to implement but difficult to finetune. Firstly, as the input to the PID was very noisy with large variance, it was not possible to implement the integral and derivative parts of the controller. Secondly, as two individual controllers were used and later summated, it was problematic to finetune the proportionality constants for both controllers as it was difficult to appreciate the effects of altering each constant. The solution that was tried to solve this issue was to graph and analyse both errors. However, this did not lead to any conclusions that could be used to improve the performance of the PID controller.

The result of this was that the system did a very good job of detecting edges, finding and correctly classifying the lines but it was not possible to implement a PID controller which could regulate the path of the car. This meant that the car was not able to drive around the mat autonomously. Something became apparent, is that one small error in one of the individual modules compounds to cause an even bigger error for all the following modules. A camera stream was designed which showed the live camera feed, the edges the Hough transform detected, how the k-means clustering algorithm classified these edges and the virtual line that was calculated. Additionally, an instrument panel was implemented as an addition to the camera stream which showed the angle error and position error calculated.

There are improvements that can be made which would potentially improve the performance of the system. In general, the improvements should be to either stabilize the input to the PID controller or choose better values for the individual components of each controller. Finding and fixing the reason why the Hough transform as an input to a median filter was returning the exact same output for a single frame, regardless of the number of iterations despite it being probabilistic, should in theory stabilize and produce a better result from the Hough transform and compound. A potential solution could be to add a birds-eye view transformation as part of the image processing module, so all the numerical analysis can be relatable to real-world physical quantities instead of pixels in a picture. If this leads to a better performance of the autonomous driving capabilities of the car, it will make it exponentially easier to finetune the PID controller by graphing and analysing the input and outputs of the controller.

Lastly, something that is worth considering is the calculation of virtual line. Instead of taking the mean of two lines, it could perhaps instead find the mean and search the y-values of the edges detected by the Hough transform for an appropriate edge to use.

6. Conclusion

In context of comparing and reflecting on the goals set out for the thesis in the beginning, the system was able to do a great job of detecting lines. The implementation of distinguishing between striped and single white lines was never attempted as part of this thesis as it was deemed that it had no bearing on the car, in context of its capability to drive autonomously. Neither was any testing done on verifying if previously functionality from the "*Car Vision"* functioned parallelly to functionality developed as part of this thesis.

The goals outlined in the introduction were changed over the course of the thesis. All focus was invested in the autonomous driving capabilities of the car which meant that there was no effort made in verification of previous functionality. However, the findings of this thesis show that the lane detection and following is possible with the hardware used. The system was able to find and detect lines from camera pictures and correctly plan the correct path the car should follow. This was verified by the live camera stream implemented. Further work in stabilizing the input and increasing the performance of the PID controller should result in the creation of an autonomous vehicle which can successfully navigate the mat.

7. References

Barba, R. and Carrillo, J. (2017). *VAUTPL/Detecting-lines*. [online] GitHub. Available at: https://github.com/VAUTPL/Detecting-lines [Accessed 19 Oct. 2017].

Docs.opencv.org. (2017). *Histogram Equalization — OpenCV 2.4.13.5 documentation*. [online] Available at: https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html [Accessed 29 Nov. 2017].

Docs.opencv.org. (2017). *Morphological Transformations — OpenCV 3.0.0-dev documentation*. [online] Available at: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html [Accessed 30 Nov. 2017].

Docs.scipy.org. (2017). *K-means clustering and vector quantization (scipy.cluster.vq) — SciPy v1.0.0 Reference Guide*. [online] Available at: https://docs.scipy.org/doc/scipy/reference/cluster.vq.html [Accessed 30 Nov. 2017].

Homepages.inf.ed.ac.uk. (2017). *Image Transforms - Hough Transform*. [online] Available at: https://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm [Accessed 30 Nov. 2017].

NumPy. (2017). *NumPy — NumPy*. [online] Available at: http://www.numpy.org [Accessed 29 Nov. 2017].

Opencv.org. (2017). *About - OpenCV library*. [online] Available at: http://opencv.org/about.html [Accessed 29 Nov. 2017].

Scrum Alliance. (2017). *Learn about Scrum?*. [online] Available at: https://www.scrumalliance.org/why-scrum [Accessed 29 Nov. 2017].

Sung, G. (2017). *road_lane_line_detection*. [online] GitHub. Available at: https://github.com/georgesung/road_lane_line_detection [Accessed 19 Oct. 2017].

Wambler, S. (2017). *Introduction to Test Driven Development (TDD)*. [online] Agiledata.org. Available at: http://agiledata.org/essays/tdd.html [Accessed 29 Nov. 2017].