DUBLIN INSTITUTE of TECHNOLOGY
Institiúid Teicneolaíochta Bhaile Átha Cliath

Automated License Plate Recognition

Final Year Project Report

DT211
BSc in Computer Science (Infrastructure)

Soslan Tuaev
C12420642

Supervisor: Mark Foley
Second Reader: Basel Megableh

School of Computing,
Dublin Institute of Technology,
Kevin Street,
Dublin, Ireland

DECLARATION

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signature of Student: 

Date: 08/04/2016

Acknowledgments

I would like to thank all my colleagues and lecturers in DIT Kevin Street for giving me the support and knowledge needed to finish this project. A big thank you to my wrestling coaches in the Hercules Club and the DIT elite athlete support programme for allowing me to spend more time on my project and supporting me by keeping me on track during stressful times. Thank you to my parents and sisters for pushing me to study harder and a finally thanks to my friends Kamil Swituszak and Greg Wolasewicz for helping get test data for the project.

Abstract

An automatic license plate recognition system is a system that can successfully extract license plate number from vehicles using image processing techniques. These systems are used by for tolling on motors, car parks, law enforcement and in many other instances.

The automatic license plate recognition system created in this project is locate a license plate in an image by filtering an image and searching for areas of interest that would be a license plate. Every license plate is saved as a jpg file when located and then further processing is done to extract the numbers. Using segmentation to locate numbers and letters in a license plate and then classifying the numbers and letters using a neural network that has been trained to recognise number 0-9 and 12 letters. Finally, the extracted data is stored in a database and displayed in the user interface of the system where the user can use this data to create blacklisted vehicles and set alerts for when that blacklisted vehicle is located.

The ALPR uses Harris corner detection to locate corners in a filtered image and these corners are used to locate a license plate by scanning the image with a license plate sized window and counter the highest area with corner which should be the license plate if filtered correctly.

The located license plate is then segmented by using region props and filtering them by setting values that the region props needs to match to be considered a character.

The segmented characters are then sent to a neural network that is used to classify the segmented images.

The classified characters are formatted in the format of the license plate and stored in a database. This database will hold records of the time and date the vehicle was caught by the ALPR and the data can be used to create blacklists to that when ever a blacklisted license plate is caught by the ALPR, it will notify the user and add the alert time and date to a database.

Table of Contents

Chapter 1. Introduction 10
1.1 Introduction 10
1.2 Background 10
1.2.1 License plates in Ireland 10
1.2.2 Current Systems 11
1.3 Goals and Objectives 11
1.4 Objectives 12
1.5 Timeline 12
1.6 Structure of this Document 13
1.7 Conclusion 13

Chapter 2. Similar Systems and Algorithms 14
2.1 Introduction 14
2.2 Alternative Existing Solutions 14
2.2.1 GV-LPR 14
2.2.2 IPConfigure - LPR 15
2.2.3 An Garda Síochána 17
2.3 ALPR Algorithms 18
2.3.1 Locating by Colour 18
2.3.2 Locating by Corners 19
2.3.3 Segmentation 20
2.4 User Requirements 21
2.5 Conclusion 21

Chapter 3. Technology Research 22
3.1 Introduction 22
3.2 Image Processing Languages (Back End) 22
3.2.1 MATLAB 22
3.2.2 OpenCV 23
3.3 MATLAB and OpenCV comparison 23
3.3.1 Speed: 23
3.3.2 Memory Management: 23
3.3.3 Ease of use: 24
3.3.4 Development Environment: 24
3.3.5 GUI Development: 25
3.3.6 Debugging: 25
3.4 Storage 25
3.4.1 MySQL 26
3.4.2 Oracle 26
3.4.3 MongoDB 26
3.4.4 Local 26
3.5 Desktop GUI (Front End) 26
3.5.1 Java 27
3.5.2 Python 27
3.6 Conclusion 27

Chapter 4. Design 28
4.1 Introduction 28
4.2 System Architecture Diagram 28
4.3 Use Case Diagram 28
4.4 User Interface prototype 30
4.4.1 Login Interface for Database 30
4.4.2 Main Interface for Database 31
4.4.3 Changes to the Design 31
4.5 Image Processing Back-end 32
4.5.1 License Plate Extraction 33
4.5.2 Character Segmentation 34
4.5.3 Character Recognition 34
4.6 Design Methodologies 36
4.6.1 Waterfall 36
4.6.2 Agile 37
4.6.3 Spiral 37
4.6.4 Chosen Design Methodology Conclusion 38
4.7 Conclusion 39

Chapter 5. Implementation 40
5.1 Introduction 40
5.2 System Overview 40
5.2.1 User interface 40
5.2.2 Importing video and image 41
5.2.3 Locating the License Plate 42
5.3 Character segmentation 46
5.3.1 Locating Objects for Segmentation 47
5.4 Character Recognition 48
5.4.2 Storing Results 55
5.5 Conclusion 58

Chapter 6. Testing and Evaluation 59
6.1 Introduction 59
6.2 What is Software Testing? 59
6.2.1 White Box Testing 59
6.2.2 Black Box Testing 59
6.3 Testing the ALPR 60
6.3.1 Importing Video/Image to ALPR Test Case 60
6.3.2 Locating License Plate Test Case 60
6.3.3 Segmentation Test Case 61
6.3.4 Character Recognition Test 61
6.3.5 ALPR Interface Functionalities 62
6.4 Conclusion 62

Chapter 7. Conclusion 63
7.1 Introduction 63
7.2 Initial Objective 63
7.3 The System 64
7.4 Final Notes 65

Table of Figures

Figure 1: Irish registration plate after 2013 and pre 2013 11

Figure 2: GV-LPR feature chart 15

Figure 3: GV-LPR Solution diagram 15

Figure 4: IPConfigure embedded system demo of browser GUI 16

Figure 5: IPConfigure embedded system watch list configuration 17

Figure 6: An Garda Síochána badge 17

Figure 7 : Close-up of the ALPR system in traffic corps vehicles 17

Figure 8: RBG(left) to binary(right) whites only [9] 19

Figure 9: Harris corner detector(right) applied to vehicle (left) 19

Figure 10: section highest value of 24 plotted corner dots(left). Actual image of cropped section(right) 20

Figure 11: Binarization example 20

Figure 12: MATLAB Logo 22

Figure 13: OpenCV logo 23

Figure 14: MATLAB IDE environment 24

Figure 15: MATLAB GUI developer 25

Figure 16: ALPR system architecture diagram 28

Figure 17: Use Case Diagram 29

Figure 18: ALPR interface login prototype 30

Figure 19: ALPR main interface prototype 31

Figure 20: ALPR final interface design 32

Figure 21: Main processing stages in ALPR 33

Figure 22: Process of locating a license plate 34

Figure 23: 3 Layer Neural Network 35

Figure 24: Delta rule calculation [18] 35

Figure 25: Multilayer Neural Network for hand written digit recognition [19] 36

Figure 26: Waterfall Lifecycle Model 37
Figure 27: Agile Development Methodology 37
Figure 28: Spiral Development Methodology 38
Figure 29: AUP Lifecycle 38
Figure 30: ALPR interface 40
Figure 31: MATLAB GUIDE GUI builder 41
Figure 32: Reading frames from video in MATLAB 42
Figure 33: selecting smaller region from input image 42
Figure 34: selecting smaller region of input image 43
Figure 35: Moravec's corner detector 43
Figure 36: Harris corner detector formula 44
Figure 37: Applying Sobel mask and Gaussian filter to image 45
Figure 38: Harris corner detector and threshold 45
Figure 39: Increasing the size of image by adding weight and height of license plate 46
Figure 40: plotted coordinates on blank matrix 46
Figure 41: resized original image with plotted coordinates 46
Figure 42: Loop to sum coordinates found in box size 46
Figure 43: blank matrix highest area with corner coordinates 46
Figure 44: original image with highest area of corners 46
Figure 45: Binarization of license plate 47
Figure 46: MATLAB regionprops function 47
Figure 47: Locating bounding box and area with regionprops function 47
Figure 48: Filtering to select character bounding box coordinates 48
Figure 49: Training set images 49
Figure 50: Storing single number 3D images into one 3D matrix 49
Figure 51: Converting to matrix values of images to double precision values 50
Figure 52: Converting every slice of the 3D array of image values into a 2D vector array 50
Figure 53: Creating the target output for the neural network 50

Figure 54: Randomising weights for the neural network to train 51
Figure 55: Feed forward, hidden layer 51
Figure 56: Feed forward, output layer 51
Figure 57: Back propagation formula in English 52
Figure 58: Back propagation in MATLAB 53
Figure 59: Sending bounding box coordinates to recognition 53
Figure 60: Highlighting characters in license plate image 54
Figure 61: Resizing cropped image blob 54
Figure 62: Feed forward neural network character classification 55
Figure 63: Formating result and writing to text file 56
Figure 64: Storing license plate image and displaying extracted license plate values in the interface 56
Figure 65: User interface 57
Figure 66: Copying data to log and displaying log data to interface 57
Figure 67: Blacklisting functionality implementation 58
Figure 68: Root mean square error of Neural Network 61
Figure 69: ALPR system process diagram 64

Chapter 1. Introduction

1.1 Introduction

An Automatic License Plate Recognition (ALPR) is an Intelligent Transport System that has an important part in numerous real world applications such as motorway tolls, unattended parking lots and traffic law enforcement. Some of these ALPR systems have restricted working conditions due to different environments and the effect they have on the image. Some of these effects can be the different illumination, limited vehicle speeds or a designated range of distance between the camera and vehicle.

Most ALPR systems are privately owned by government companies and are expensive to acquire. The aim of this project is to create an inexpensive automatic license plate recognition system that uses a neural network to recognise and read the digits of a license plate, with a high rate of success in identifying and extracting license plate number from vehicles. The system will be usable for everyday situations such as tracking vehicle license plates that has recently entered private parking or estate.

The recognition of license number plates is based on digital image processing. This project will have a system will use a video source of moving vehicles and process the frames of the video to locate the area of a license plate. Once a license plate is located, it digits will be segmented for recognition using a neural network to classify the digits that were segmented and displayed on a user interface along with the time and date it was captured.
The system will have a front-end interface where the user will be able to search the using a specific license plate number to see where a vehicle was or to search a location and see all the vehicles that have passed through there.

The complexity of this project is in the localisation of the license plate, extraction of the license plate and the recognition of characters using a neural network. The captured images will have to go through many images processes before the license plate number can be extracted in ASCII characters.

1.2 Background

1.2.1 License plates in Ireland

Owners of automobiles driven on public roads are required by law to register their vehicles with the department of motor vehicles, and attach their license plates to their vehicle so that it is publically and legibly displayed [1]. These license plates generally consist of a unique combination of numbers and letters that reference the specific vehicle and the owner of the vehicle. Every country has their own combination of alphanumeric license plate number but Ireland is known for having the simplest to understand license plate in the world because of the information they hold about the car. For instance, the Irish license plate format since 2013 is YYY-CC-SSSSSS where as before it was 2013, it was YY-CC-SSSSSS. The YYY stands for the year the vehicle was made, CC stands for the county it was registered (e.g. D for Dublin; KY for Kerry) and SSSSSS is a 1- to 6-digit sequence number (starting with the first vehicle registered in the county that year/period) [2]. Usually vehicles that have previously been registered in the UK and imported to Ireland would have the full 6-digit sequence.

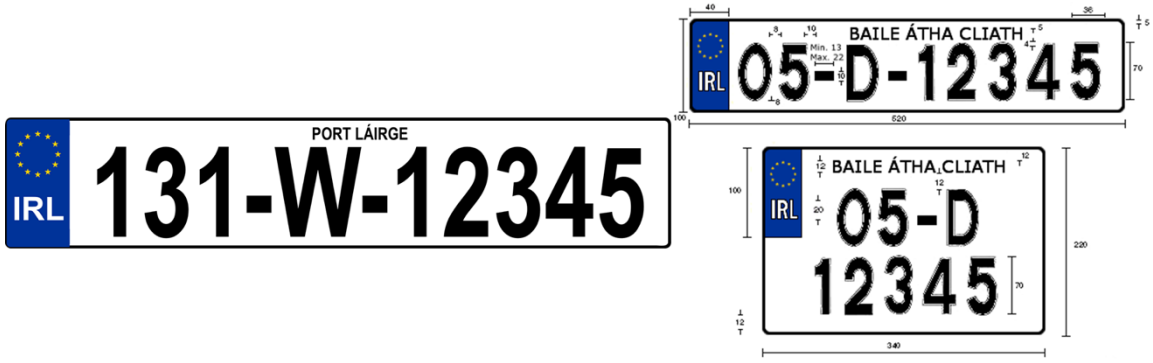

Figure 1: Irish registration plate after 2013 and pre 2013

1.2.2 Current Systems

Modern ALPR systems feature state of the art image processing technology to locate, and identify vehicle license plates. They use infrared illumination and digital camera to take images of incoming or outgoing vehicles using a frame grabber. These systems have very powerful on-board processers running the image processing software to locate and extract license plate numbers from vehicles which can then be used in many ways, such as:

- Stored in a database and checked against blacklisted vehicles
- Motorway tolling
- Information gathering

These modern systems don't need much assistance once they are set-up correctly and they are capable of processing thousand of license plates per day [3].

Automatic License Plate Recognition systems are an important part of modern day transportation and an interesting subject because of it complexity due to the diversity of different license plate formats on the road and the non-uniform outdoor illumination conditions when acquiring images. This can cause poor performance by the system because the images acquired could have unsuitable light conditions, noisy patterns, poor edge enhancement which will affect the accuracy of character recognition.

There are four main modules in this system: Image pre-processing, license plate detection, license plate character segmentation and optical character recognition [4] which will be looked at further in this paper. These modules become more difficult when working with images that have been taken from different angles e.g. inclined angles.

Most modern ALP recognition systems reduce the complexity of their systems by having a prefixed angle, distance and lighting at night to help increase accuracy and performance of the system. These systems can gain more accuracy by using specific features of local license plates, such as the number of characters, the number of rows, the size of the plate or even the colour of the background of the plate e.g. yellow UK registration plate or white EU registration plate.

1.3 Goals and Objectives

The goal of this project is to create an ALPR system that has a consistent rate of recognition of license plates and digits. Below is the list of features that are going to be implemented in this project.

- Capture frames from video source
- Image pre-processing
- License plate localisation
- Character Segmentation
- Optical character recognition (OCR) using a neural network
- Store extracted data with additional appropriate information

A system will be created that can perform image processing tasks. The system will process captured vehicle images and locate the license plate region of the image. Once the license plate is located the system will perform image processing algorithm that will perform character segmentation and optical character recognition (OCR) to extract the number and letters from the license plate.

A front-end will also be created for the user to be able to see extracted license plates and be able to perform other tasks such as blacklisting and searching cars. For this, a database must be developed to store the extracted information of the vehicles license plate. The database will need to hold information about where the image was captured which would include the time stamp and location.

1.4 Objectives

To achieve the requirements specified above, the following objectives will need to be researched in order to complete the system:

- Research and compare image processing technologies that can be used to create an ALPR and identify the most proficient one
- Implement a license plate recognition algorithm with the chosen technology
- Train a neural network with datasets
- Designing a user friendly GUI that will output the license plate information captured

1.5 Timeline

The research and information gathering of the project was done in semester one. Following the research documentation and design of the project, the implementation stage started in early February 2016. The implementation of the system side will be developed first, followed by the development of the database and connecting them together. Testing will be done as the implementation is done. The system will be reviewed and tested to see if anything needs to be added or changed in March during the last weeks of the project along with the documentation.

1.6 Structure of this Document

The remainder of this document contains the following chapters:

- **Chapter 2:** A review of similar systems and algorithms in order to understand how current ALPR systems operate and little view of algorithms in an ALPR system such as OCR.
- **Chapter 3:** A review and comparison of technologies that will be used to create the ALPR system. What programming languages, image processing software's and databases will be needed.
- **Chapter 4:** This chapter will look at what design methodology best suits this project and designs will be created for the system and how it will look.
- **Chapter 5:** Prototypes that have been developed so hard will be reviewed here to see how they were done. Examples of code will be described with the prototype interface.
- **Chapter 6:** The testing methodologies will be described here and the one chosen for testing the ALPR system.
- **Chapter 7:** Issues and risks will be stated in this chapter to better understand what need to be focused on and where main challenges are.

1.7 Conclusion

This chapter went over the general idea of the what needs to be done and what will be coming up in the next chapters. Goals and objectives have been set and the following chapters have been briefly introduced.

The next chapter is the Similar Systems and Algorithms where similar ALPR system will be reviewed on how they work and their main features.

Chapter 2. Similar Systems and Algorithms

2.1 Introduction

This chapter will look at Automatic license plate recognition systems and the method at are applied in an ALPR system such as license plate isolation, optical character recognition (OCR) and segmentation and detect license plates and characters. A background research will be conducted on similar automatic license plate recognition systems. A look into the approaches and techniques of these systems and how they perform under different circumstances e.g. lighting conditions, camera position, different cameras and see at how these systems perform in the real world and what they do.

2.2 Alternative Existing Solutions

The newest ALPR systems are programmed to be versatile to suit any application. They are programmed as single integrated packages that have several different modules. Most of these ALPR systems are connected to a central database where information is contained about registered vehicles and owners of the vehicles in that country. Recent ALPR systems in the UK are capable of detecting license plate numbers of vehicles traveling at speeds of almost 200 miles per hour.

A look into three of these systems will be a part of this chapter. There aren't many ALPR systems that are free and easy to use and most ALPR systems are run by government agencies (e.g. Law Enforcement and motor way tolls) or commercially such as private parking, which the public doesn't have access to. Because of that, an overview of these applications will be difficult to conduct because it is expensive and hard to get access to their interfaces or their system architecture and due to lack of detailed information about high end ALPR systems, it will be hard to apply heuristics. Below is a review of some of these systems and how they work.

2.2.1 GV-LPR

"The GV-LPR solution is an effective video security and parking monitoring solution that can enforce parking lot security" [5]. The GV-LPR is an intelligent PC based LPR that can read and recognize license plates for up to 8 lanes and has a 99% accuracy rate when running at its best performance. A good feature of this system is that it can store the license plate images when captured in a JPEG format and store it in a dedicated database.

GV-LPR has a database function, which is very useful for parking lots because of it can effectively utilize parking spaces and can be used to acquire information of vehicles in the a parking lot such as vehicle counting, monthly parking frequency, or average parking time during holiday/weekday or during the day or night time [5].

The GV-LPR system has a lot of functions other than just detecting the license plates. The system can also capture the drivers face and the car type which can be used when investigating a fraud activity or cash shrinkage in parking lots.

The GV-LPR is the ideal system for parking lots because of its many features and that it can be integration with so many external devices, making It a very flexible system for parking

management. *Figure* 2 shows the GV-LPR Features Chart and in figure 3 is the GV-LPR Solution.

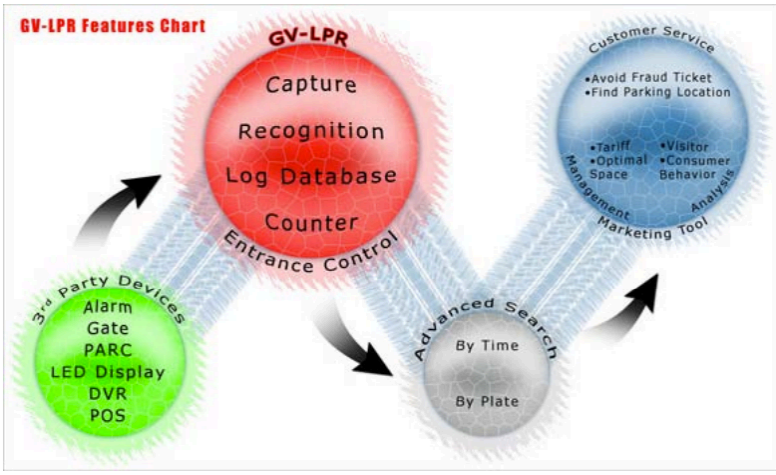

Figure 2: GV-LPR feature chart

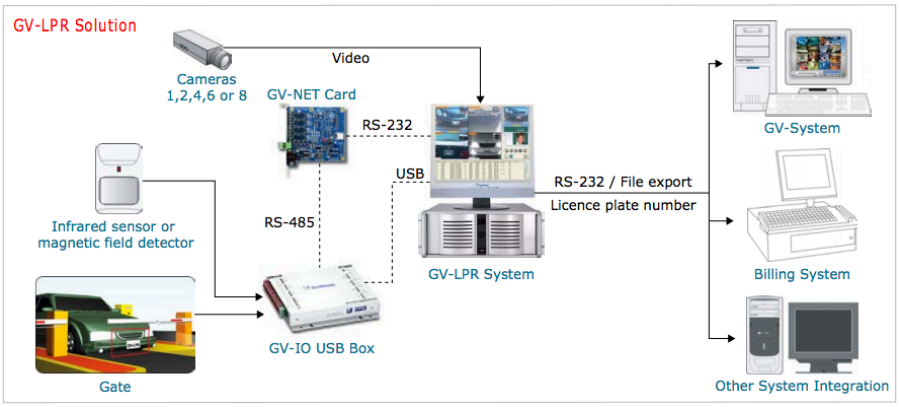

Figure 3: GV-LPR Solution diagram

2.2.2 IPConfigure - LPR

IPConfigure LPR is a server-based and camera embedded application for accurate license plate recognition. The LPR application is mostly used in the 50 states of the US but it also supports many other countries and subnational entities. The LPR camera captures stationary and moving vehicles up to 55 KPH. License plates can be read from up to 36 meters from the vehicle and at angles up to 35 degrees vertically and horizontally. IPConfigure sells IR (Infrared) camera along with the LPR application. The LPR can capture vehicle license plates during day and night conditions if it is used along side one of their IR cameras or white light LED illuminators. IPConfigure LPR has a two applications that is offers. These will be their server based LPR and embedded LPR system

The server based IPConfigure LPR was developed for the US government [6]. It provides highly accurate real time license plate recognitions along with full colour images during the day time that can provide forensic details such as the make, mode and colour of the vehicle and what's more is that at times it can capture images of the driver too.

Server based LPR key features:

- Displaying captured results in real time.

- Triggering events on license plate match's e.g. blacklisted vehicles.
- Full text plate search.
- Captures full colour license plate images for forensic detail.

The embedded LPR system is developed for commercial use. It was the first LPR to be used with IP cameras. The embedded LPR supports Axis ARTPEC-4 or better cameras that are capable of independently recognising, recording and reading license plates from within the camera itself [6]. The embedded LPR is a standalone system that doesn't need an external server. All of the processing is done by the Axis ARTPEC-4 or better camera and the LPR can be searched and queried from inside the camera through the camera IP link. The LPR also has push notifications that can be configured to send recognized license plates, such as blacklisted or wanted vehicles to external systems and third party video management systems.

Embedded LPR system key features:

- Commercial
- Camera UI text search
- Camera bases I/O triggered events
- On-board storage of up to 250,000 license plates
- API for 3rd party applications
- Push notifications.
- Watch list

IPConfigure has a demo version available online. The demo doesn't demonstrate how the system works but it allows the user to try some of their functionalities such as searching for captured license plate numbers and images from a certain period of time or by simply searching and exact date and time or the license plate number. Below is the online browser interface of the embedded system.

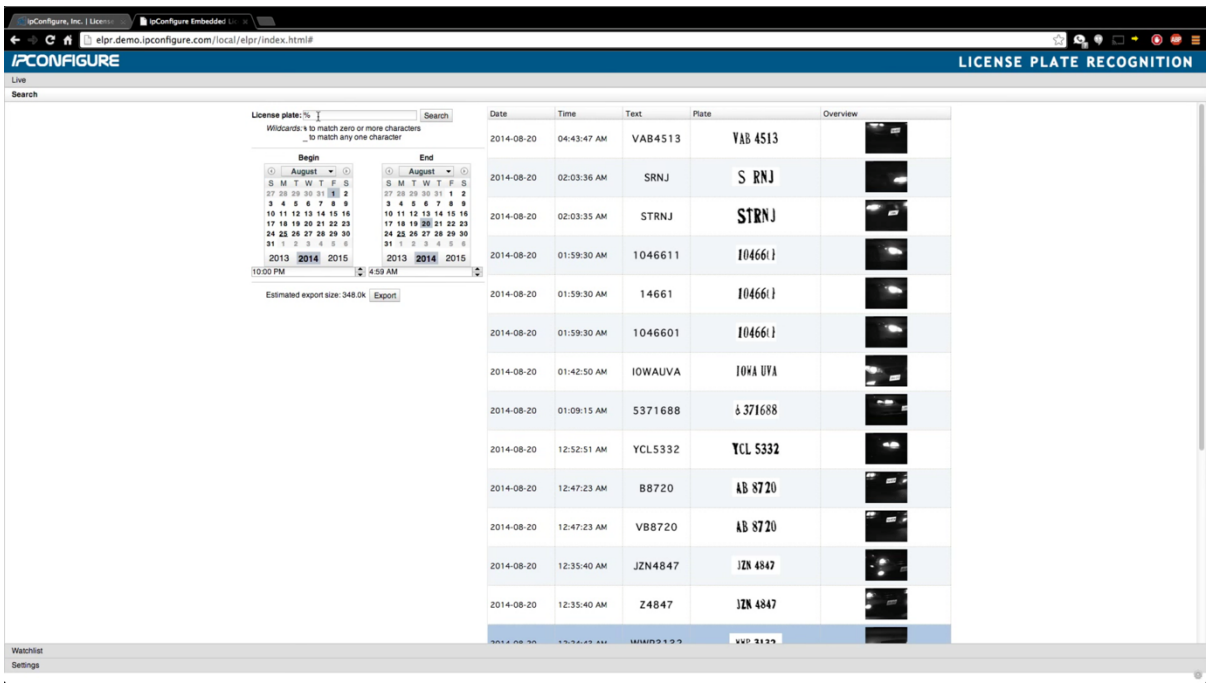

Figure 4: IPConfigure embedded system demo of browser GUI

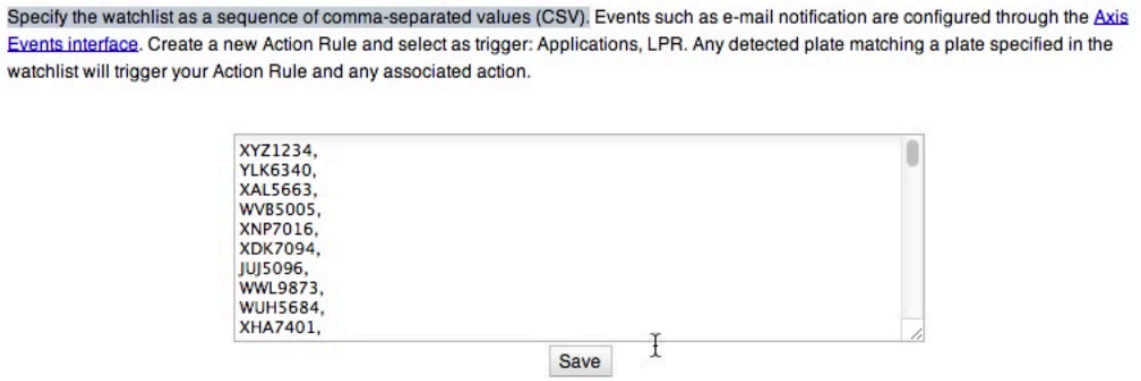

Figure 5: IPConfigure embedded system watch list configuration

2.2.3 An Garda Síochána

Figure 6: An Garda Síochána badge

An Garda Síochána is the police force of Ireland and they use automatic license plate recognition systems in their Garda Traffic Corps vehicles to assist them in their fight against crime.

The ALPR's they use are some of the best in the world, performing at incredibly accurate and fast rate with high quality equipment like the infrared cameras they have. The Garda Traffic Corps like many other police forces worldwide, use this technology to read vehicle registration plates at a rate of six number plates per second on vehicles travelling up to 180km/h [7].

Figure 7 : Close-up of the ALPR system in traffic corps vehicles

The traffic corps ALPR has many features which will be listed below:

- Speed Detection of travelling vehicles in front of patrol vehicle

- Video recording of on-the-scene evidence of speeding and offences
- Identification of stolen, untaxed, uninsured, criminal suspect vehicles etc
- Runs in the background while officers are attending other issues demands dictate
- In-depth details about vehicles

The ALPR systems have been rolled out to the Garda in 2010 and 95 (of which 91 are in Traffic Corps vehicles), are now fitted with the technology [7]. ALPR vehicles are now deployed in every Garda Division.

2.3 ALPR Algorithms

As indicated by platerecogintion.info [8], ALPR's have two essential technological issues. These two issues are *"the quality of the quality of the license plate recognition software with its applied recognition algorithms and the quality of the image acquisition technology, the camera and the illumination."* [8]. This implies that the better the image recognition algorithm is, the better the system will work and the following functions will be more efficient:

- Higher recognition accuracy
- Faster processing speed
- Better handling of a variety of license plates
- Handling a wider array of picture quality

Previously ALPR systems were bound to specific countries, for example an ALPR system in Germany would only recognise German license plates. This was on account of the geometrical proportions of license plates as well as it syntax were crucial parts of the ALPR. Without the assumption of character ratios, character distribution, font type and plate colour, the algorithm may not have found the plate in the image [8]. Todays most advanced ALPR systems are capable of reading all kinds of license plates from all over Europe.
This section will look at two methods of locating and isolating a license plate from an image. The first step in most license recognition systems is to convert the input image into binary from its original colour space.

2.3.1 Locating by Colour

The first method of locating a license plate is by searching for the colour of the license plate in the image. This is useful when the license plates always have specific colour format, such as a white background and black digits. This method uses colour values to remove anything that is not the specified value the system is looking for. Every RBG image has a value from 0-255 for every pixel in the image which determines the the colour of that pixel. In this instance a white background license plate is being located, so when a conversion to binary is being done, anything below the RBG value of the colour white (255) will be set to 0's and all the white pixels to 1's which will give the result in *Figure 8.*

Figure 8: RBG(left) to binary(right) whites only [9]

Figure 8 shows the final result of the conversion to binary and it is seen that all the colours that are not white have been removed during the binarization but it also has white pixels that are outside the license plate region, which are not desirable. This is a problem and it would be severally larger is the image that was used was of the whole vehicle with other objects in the image or if the vehicle colour was white.
This method is a good starting point for locating a license plate but it wont do the job without additional processing. Next will be another method that could be used with this one to increase the chances of detection.

2.3.2 Locating by Corners

This method uses corner detection such as the Harris corner detector algorithm [10]. The Harris corner detector algorithm can be used to plot points in an image where it has found a corner. This is useful for locating license plates in an image because a license plate has many corners, from the corners of the license plate holder to the corners of the digits in the license plate. This can be used to an advantage by scanning sections of an image to find a section with the highest amount of plotted corner points. The system will assume that the section with the highest plotted corners point is the license plate and crop it out for further processing. In *Figure 9* it shows an image of a license plate with the Harris corner detector algorithm applied and plotted with dots.

Figure 9: Harris corner detector(right) applied to vehicle (left)

Figure 9 shows the full image of a vehicle and all the corners found in that image. The centre of the image has the highest amount of dots which can indicate that a license plate it located there. The next step would be to scan every plotted dot with a specified section size and the value of every section with the amount of dots it found and only keep the one with the highest value. *Figure 10* shows a scanned section that was extracted because it had the highest amount of plotted corner dots and it came out to be the correct location of the license plate.

Figure 10: section highest value of 24 plotted corner dots(left). Actual image of cropped section(right)

Is a good method of locating license plates in an image but the downside is that if another region of the image has a higher amount of plotted dots, that region will be extracted.

2.3.3 Segmentation

OCR is a technology that converts images into editable formats. For the purpose of this assignment it will be in relation to images of license plates. The process of converting the image to an editable document is performed in several steps [6].

The first step in the OCR process is to load an image from a source, in this case the image will be a located license plate crop. Once the image is loaded, the most important features like the resolution and inversion are detected because the OCR algorithm expects a predefined range of font sizes and background colours so the image must be rescaled and inverted if necessary [6].

The next process is to remove any skew or noise that affects the image quality which is done by an algorithm to realign the image and remove such noise using Gaussian blur.

The next step is to turn the image into binary which is essentially converting the image to black and white because OCR technologies require bi-tonal images. This is an important step as incorrect binarization can cause inaccurate calculations [6]. Below in *figure 11* it shows an image being correctly and incorrectly binarized.

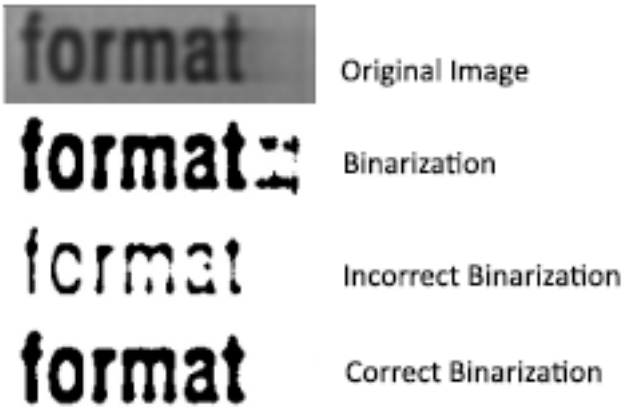

Figure 11: Binarization example

Once the image is correctly binarized, the next step is the main algorithm of any OCR and that is the recognition of characters. The first step in any character recognition system is segmentation. Segmentation is the process of locating and extracting every character found in an image. This can be done by converting a binary image to a logical format of 1's and 0's and measuring the area of the 1 values because 1 (logical value of black pixels) represents the characters. The last thing to do is to classify the segmented objects into characters which will be discussed further in the paper.

2.4 User Requirements

Following the review of similar systems above and the demo of the IPConfigure system, this ALPR project will have the following user requirements:

- The user will be able to choose a video source for the ALPR
- An aesthetic design for an easy view of the database and easy navigation for the user
- Access and view stored license plate numbers
- Search license plate numbers in the database
- The user should be able to blacklist license plate numbers and if the blacklisted number comes up in the ALPR, the system will push notifications to the user.

2.5 Conclusion

As mentioned before, ALPR systems are generally used by private companies for tolling or by law enforcement to help track criminal vehicles and these systems are not cheap. From the review of similar systems, there have been many shared characteristics that are in all of the systems. These characteristics are, capturing an image, processing the image to extract the license plate, store it in a database and check it against previously stored license plates to determine if the vehicle should go through (applied to parking lots) or to see if it is flagged for any alerts such as a stolen vehicle.

By taking the information gathered from reviewing the similar systems. The information gathered will help in the development of the ALPR system at hand. The demo of the IPConfigure system helped visualize how the project design will be implemented and how the database queries will be done. Further demoing of the system will help identify common mistakes which will then be avoided in the development of the ALPR system. The design of the ALPR system will look to address issues with error prevention and recovery as well as ease of access. On-screen instructions and error messages will be implemented to help users understand the system. During the development of the ALPR system, the review of similar systems will help determine what elements will need to be focused and developed.

Having absorbed the information on in this chapter, Chapter Three will help determine the technologies that will be used in this project by reviewing similar technologies and comparing them to choose the ones that will deliver an ALPR system which best meets the user's requirements.

Chapter 3. Technology Research

3.1 Introduction

This chapter will look at and review technologies that are going to be used in this project, as well as corresponding technologies. Several technologies have been researched to find a suitable platform where the final result will be delivered. Technologies such as Java, MATLAB and MySQL will be examined in detail as well as corresponding technologies to decide what is the most appropriate technology to use for building an ALPR system. This chapter will introduce and examine the technologies for its positives and negatives to determine why they were chosen and why the others were not.

3.2 Image Processing Languages (Back End)

For image processing technologies, a look into the two most popular image processing tools will be conducted and examined for the development on an ALPR system. The most common and popular image processing tools for developing ALPR systems are, MATLAB and OpenCV which will be compared in this chapter. The two tools will be reviewed to determine which one is most suitable for this project.

3.2.1 MATLAB

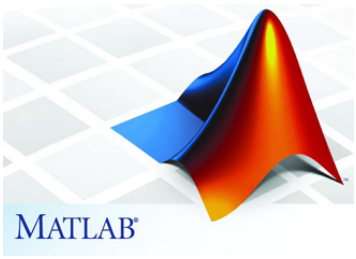

Figure 12: MATLAB Logo

MATLAB which stands for matrix laboratory, is a commercial high performance language for numerical computation and visualization. MATLAB incorporates computation, visualization and programming in a simple to utilize environment where issues and solutions are communicated in familiar mathematical notation. “MATLAB is an interactive system where basic data elements is an array that does not require dimensioning, allowing you to solve technical computing problems, especially those with a matrix or vector formulations in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran” [11].

Typical uses of MATLAB include:

- Algorithm development
- Application development, GUI building
- Data analysis and visualization
- Math and computation
- Simulation, modelling and visualization
- Scientific and engineering graphics

3.2.2 OpenCV

Figure 13: OpenCV logo

OpenCV is an open source C++ library for image processing and computer vision that was developed by Intel. The library has many built in functions that are mainly aimed at real time image processing. It has hundreds of image processing and computer vision algorithms which make developing advanced computer vision applications easy and efficient if the developer has previous knowledge of image processing and C++ [12].
Key features of OpenCV include:

- Optimized for real time image processing
- Primary interface of OpenCV is in C++
- There are also C, Python and Java interfaces
- OpenCV runs on all desktop operating systems
- Optimized for Intel processors

3.3 MATLAB and OpenCV comparison

3.3.1 Speed:

MATLAB is built on Java, which means that when MATLAB code is running, the computer will be busy interpreting all the MATLAB code that is then turned into Java and finally executed. OpenCV is a library with functions that are written in C and C++ which means that it is closer to directly provide machine language code to the computer to get executed [13]. Ultimately OpenCV will get more image processing done and not as much interpreting. Because of this, OpenCV programs are faster than programs written in MATLAB.

3.3.2 Memory Management:

Since OpenCV is based on C, every time the program allocates memory, it has to release it again. If there is a loop in the code that allocates chunks of memory and doesn't release it, the program will get a leak where it will keep growing the amount of memory until it crashes the program. On the other hand, because of MATLAB's high-level nature, it can automatically allocate memory to the program and release it in the background once execution of the code has ended [13]. MATLAB's memory management is considered better because of this and it is better for video analysing in the sense that the programmer doesn't need to worry about memory while the executed code is running for long periods of time.

3.3.3 Ease of use:

Compared to OpenCV which uses C++, MATLAB's language is a lot easy to learn and understand with additional help of the build in "help" function that explains functions inside the IDE and dedicated forums. Since it is a high-level scripting language, the developer doesn't need to worry about libraries, memory management or other low-level programming issues. MATLAB is great for prototyping and developing test algorithms which enables the user to get certain parts of a project done quicker and more efficiently compared to OpenCV. It could take 15 lines of code in OpenCV to perform a task that would otherwise take 2 lines in MATLAB to perform. Most of these lines would consist of "housekeeping code" [13] which doesn't need to be done in MATLAB as much as it does in OpenCV which has a very high learning curve.

3.3.4 Development Environment:

MATLAB comes with its own IDE as shown in the *Figure 14*.

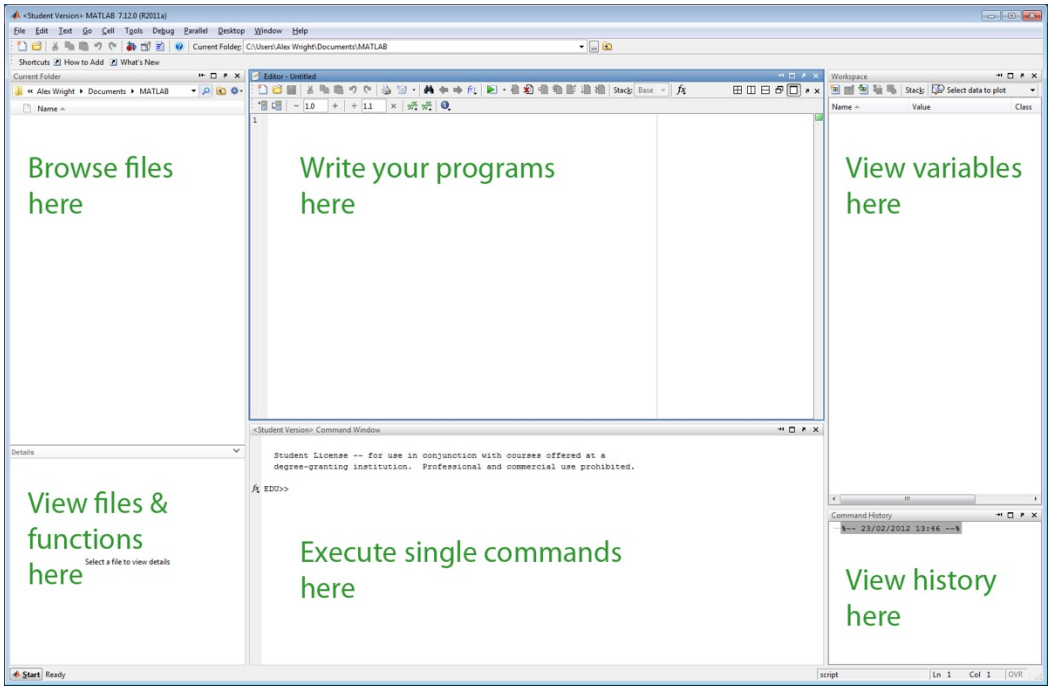

Figure 14: MATLAB IDE environment

Because MATLAB is based on Java, Mathworks have developed an IDE for Windows, Linux, and OSX users. In that capacity, when MATLAB is introduced, it is generally introducing the programming environment and the IDE too. The development environment is very nicely presented and easy to understand from the get go.

OpenCV doesn't have an IDE when it is installed. Instead its used without an IDE e.g. Notepad++, or a C programming IDE for instance it can use Microsoft Visual Studio or NetBeans for Windows or Xcode for OSX which isn't as efficient as the MATLAB IDE. Because it doesn't have its own IDE, it may take some time to set up a C IDE of the developer's choice since it isn't an easy thing to do and requires a guide of various complexity [12] and problems that can occur.

3.3.5 GUI Development:

MATLAB has a build in GUI developer built in to its IDE that is called GUIDE that similar to the Java "WindowBuilder" in the Eclipse IDE. MATLAB's GUI developer is very easy to use because of it "drag and drop" functionality and easy to understand functions which help save time doing it the tedious way of programming a GUI without actually seeing it before launching. There is also a downside to MATLAB's GUI developer due to its limitation of only being able to use the few objects e.g. buttons, edit boxes etc. that it provides.

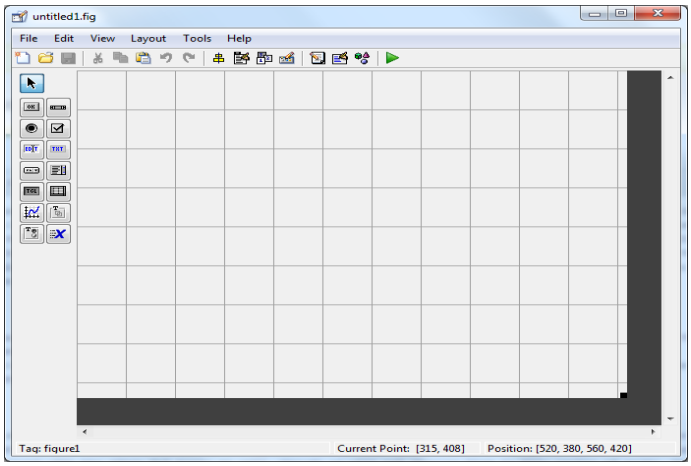

Figure 15: MATLAB GUI developer

Since OpenCV is mostly used with C++, it can be integrated with GUI builder in Microsoft Visual Studio if using it. Otherwise the programmer would need to manually create a GUI which times up a lot of development time.

3.3.6 Debugging:

MATLAB offers many additional debugging options that OpenCV if used without an IDE. Because of MATLAB's development environment, it is easy to utilize its debugging tools. One great element of MATLAB is that if the developer needs to rapidly see the output of a line of code, the semi-colon at the end can be overlooked to do so. Additionally, MATLAB is a scripting language, when execution is ceased at a specific line, the client can sort and execute their own particular lines of code on the fly and view the resulting output without having to recompile and link again [12]. Adding to this is the powerful functions for showing information and images.

3.4 Storage

An overview of different type of ways to store extracted information will be done in this section to find the most suitable one for this project. An overview of two types of databases will be conducted for relational and non-relational databases and also a look into storing the data locally. A database can be used to store and organise information gathered from the process images by ALPR system. For an ALPR system, there isn't that much to put into a database except for the license plate number, time, date and location of ALPR in operation, which are the main 4 variables to be stored in the database. A well designed database will allow the system to easily and efficiently query and retrieve information.

3.4.1 MySQL

MySQL is an open source relational database management system that is very popular for its robustness, openness and speed. It runs on all platforms including OSX, Windows and Linux. Although it is often associated with web-based applications, it works just as good with desktop applications and is very easy to integrate with MATLAB and Java programs. It is a server-based system that allows for cross-platform development and can be combined with server side languages such as PHP [14]. Developed independently in the mid 1990s, MySQL was purchased by ORACLE and is now a part of their suite of products, benefitting from their commercial success and experience. MySQL grew in popularity due to its incorporation in free to use web development platforms such as WAMP and its Linux equivalent, LAMP.

3.4.2 Oracle

Oracle is a very large and multi-user database management system. It is a relational database system developed by Oracle Corporation which was one of the first commercially developed and marketed relational databases and is a leader in this area. Oracle databases store data that is accessed and queried by using SQL statements such as ‘SELECT’, ‘INSERT’, ‘DELETE’ etc. The down side of Oracle databases is that large scale projects can generate considerable costs and may require advanced technical skills when it comes to configuration, unlike open source alternatives. Oracle demands a large amount of memory and may affect performance of the system.

3.4.3 MongoDB

MongoDB is an open source NoSQL document oriented data storage solution that is written in C++ and is widely accepted as the leading NoSQL database. It is exceedingly adaptable and supports a wide variety of programming languages furthermore it supports SQL like querying. As opposed to a relational database, a non-relational database does not use tables or relations and is instead built around the idea of a 'document' or non-unified collection of information. It is a schema free database which does not need the objects stored to follow a specific structure. Non-relational databases are not the best choice for every system because they have limited in the types of functions they can perform. For instance, MongoDB can’t join different data types across the database.

3.4.4 Local

Storing the data extracted by the ALPR system locally can be done by writing the data to a text file with a specific format which can then be easily to read and manipulate. It would be the same a querying a database using select statements and updates except updates would be done by writing to the text file and selects would be done by reading from the file. There wont be as many features as a database would have, such as primary keys and foreign keys that create connections but that wont be too important unless multiple ALPR systems are connected to try and create a network.

3.5 Desktop GUI (Front End)

This section of the chapter will look at languages that can be used to create the front-end of the system. The front-end of the system will be used by the user to access the data stored of license plates and perform additional features such as blacklisting specific license plates and creating push notifications. For this project a desktop GUI application will be created as

opposed to a web based application. An overview of Java and Python will be done and a final decision of which one will be used is going to be in the conclusion of this chapter.

3.5.1 Java

Java is an object-orientated language similar to C++, but simplified to dispose language elements that cause common programming mistakes. Java code is compiled into byte code which is then executed by a Java interpreter. Java is a good choice for creating GUI applications because it can run on most operating systems because of the Java interpreters and runtime environments knows as Java Virtual Machines or Java plug-in. These VM's exist on Unix, Mac OSX and Windows machines. Byte code can also be converted into machine language by just-in-time compiler [13]. Java's swing library is a very easy to use GUI development tool. Java GUI applications run considerably faster that other programming languages and have many features. There are a lot of Java IDE's that can be used to develop GUI's and provide excellent debugging features. The Eclipse IDE has a very easy and efficient GUI builder called WindowBuilder as mentioned before. It is packed with very high level features that are "drag and drop" and very useful designing a front end along with connecting to a database.

3.5.2 Python

Python is a high-level general purpose programming language that focuses on improving developer productivity and code readability. The syntax of core Python is minimalistic. At the same time, the standard library incorporates a lot of valuable functions. Python supports numerous programming paradigms, including structured, object orientated, functional, basic and aspect-orientated. The code in Python is organised into functions and classes that can be joined into modules. Reference implementation of the Python interpreter is CPython, supporting most heavily used platforms. It is appropriated under a free license Python Software Foundation License, that can be utilized without limitation in any application. Tkinter is the standard GUI library for Python which provides a fast and easy way to create GUI application using a powerful object-orientated interface.

3.6 Conclusion

After examining and considering the various technologies available for use and taking into consideration the requirements of the ALPR system. OpenCV could have been a better choice for creating a real time ALPR system because it performs video processing faster than MATLAB but given the time this project has to be completed in, MATLAB was chosen as the image processing technology because of a smaller learning curve than OpenCV and easier use with its IDE. MATLAB has all the features needed for this project including a GUI. Java for the front end and MySQL for the database were chosen before the actual implementation of the project began. But that changed once implementation of the ALPR system began. The choice changed to only using MATLAB for the development of the ALPR system and storage of the extracted data would be local in text files.

The choice was changed because MATLAB has all the feature and functions to create a GUI that is just as good as Java and Python. The built-in GUI builder is just as easy to use as any other builder except it doesn't have as many aesthetic features as a Java or Python GUI which aren't need that much for an ALPR system. Data storage locally was decided to save time on creating a full database that would do the same thing but be more time consuming which can be spent improving the accuracy of the ALPR system. MATLAB was the best choice because everything will be run from MATLAB for better performance and debugging.

Chapter 4. Design

4.1 Introduction

The aim of this chapter is to use the information gathered from the research in the previous chapter to design a functional ALPR system. This chapter will focus on the overall design of the ALPR system. Examination of the system architecture, use case diagrams and the structure of stored data. An overview of the client side interface design and it functionalities. An overview of available methodologies for software development to find their strengths and weaknesses to decide which one fits best for this project with a justification. The back-end section of this chapter will introduce the image processing techniques that will be explained in more detail in the next chapter.

4.2 System Architecture Diagram

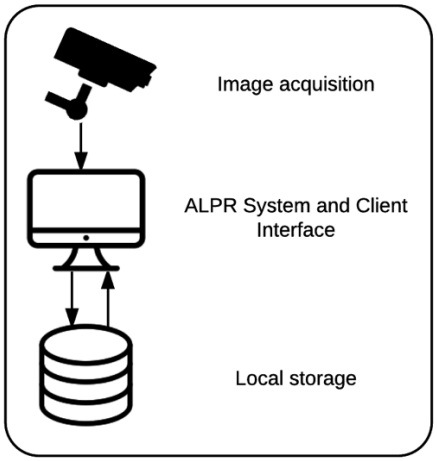

Figure 16: ALPR system architecture diagram

Figure 16 shows the overall system architecture of the ALPR system. The diagram shows the manner in which the system processes will flow, with the user having control of what is going to the image processing of ALPR system and also having access to stored data from previous logs. The users machine is the main component of the system because it is deciding the video/image source and accessing storage. In the interim report, it was stated that *"the camera collects video footage that is then processed by the ALPR system on the computer and sent to the database after the extraction of the license plate is complete."* But that has since changed to pre collected video footage and static images being imported into the ALPR for processing. The client interface is where the user can decide what source is being imported into the ALPR and what to do with the data extracted data such as searching previous logs of extracted data by date, by specific license plate numbers or blacklisting vehicles to create notifications if that vehicle is caught in one of the video sources.

4.3 Use Case Diagram

Use case diagram - a graph showing the relationship between actors and use cases and is an integral part of the use-case model, which allows to describe the system at a conceptual level. Precedent - the possibility of the modeled system (part of its functionality), through which the user can get specific, measurable and desired outcome him. Precedent corresponds to a separate service system determines one of the options of its use and describes the typical way

users interact with the system. Use Cases are usually used for external specification of system requirements.

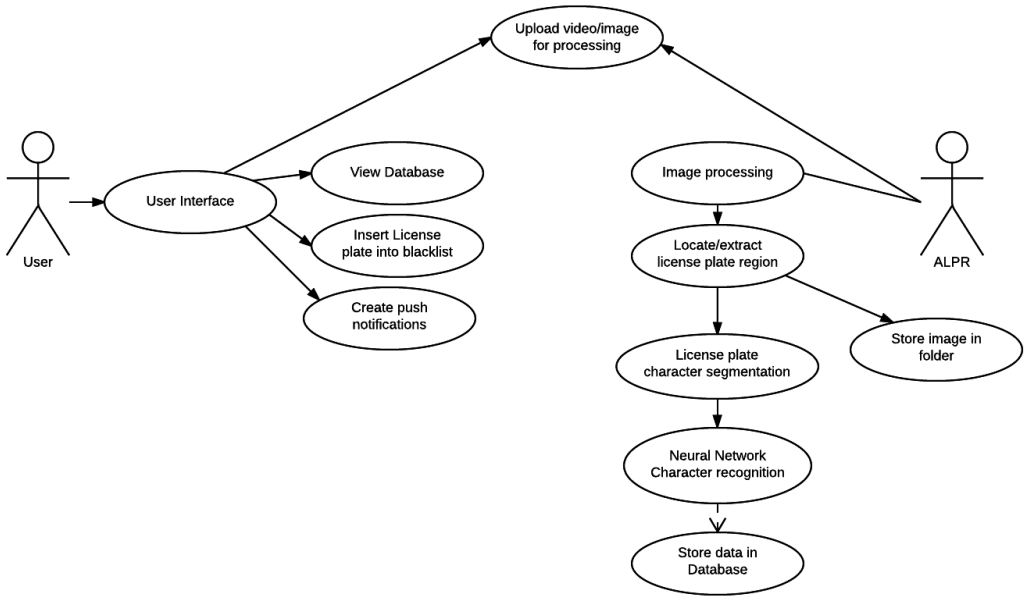

Figure 17: Use Case Diagram

Above in *Figure 17* is the use case diagram of the ALPR system. Each use case will be explained to better understand the system.

Use Case actors

- **ALPR:** The ALPR use case actor is the back-end of the ALPR system.
- **User:** This is the user that will use the front-end of the ALPR system to interact with the database and perform functions such as importing video/image or adding license plate numbers to blacklists.

User

- **Upload video/image:** The user will have the ability to insert the name or path of a video/image source that will be processed to extract the license plate from
- **View database:** Once the user interface is opened, the user will be able to see all previously stored license plate data on the screen and have the option to see blacklisted license plate number or previous blacklist alerts
- **Insert license plate into blacklist:** The user will be able to manually insert/delete data in the database
- **Create push notifications:** This is an automatic happens when the user adds a license plate number into the blacklist. Once the blacklisted number matches an license plate, a notification will pop up on the user interface and also be logged with the time/date it was notified

ALPR

- **Upload video/image:** The ALPR system will accept different video and images formats uploaded by the user

- **Image processing:** The process of extracting the license plate from the source will begin by sending frames of a video or the single image that was uploaded to further functions
- **Locate/extract license plate region:** This function will locate the license plate region using the Harris corner detector algorithm that was mentioned in the research chapter.
- **Store image in folder:** Once a license plate is extracted, it will be stored in a folder as a jpg file and then sent to the character segmentation
- **License plate character segmentation:** Once a license plate is located, it will be sent to this function. The first step in character segmentation is to turn the image to binary and find the area of the objects in the image. This will help decide which objects are characters by creating a area value threshold that will only select characters between a specific value
- **Neural Network character recognition:** The segmented characters will be sent to this function where they will be resized to 24x14 pixels and ran against a neural network which will check the weight value of the pixels to determine what neurons will activate to help classify the characters to a readable format.
- **Store data in database:** The final step is to store the extracted data in a readable format.

4.4 User Interface prototype

Paper prototyping is a low fidelity method of usability testing that is helpful for different types of applications such as conventional software programs and websites. The idea behind paper prototyping is that it permits the designer to test the interface before starting to code the software. One of the reasons for using paper prototyping is that the design and interface can be tested before coding. It is also easier to make quick changes to the design if something isn't clear, which helps prevent errors [15]. One of the main advantages to the designer is that paper prototyping allows the testing of the interface in a "perfect" scenario as there will be no code or bugs that could cause an issue.

4.4.1 Login Interface for Database

A prototype of the login screen that was created using Lucidchart.com who can provide a one-year free license for students when contacted directly. This online tool allows the user to quickly create all types of diagrams for developers and designer such as use case diagrams, paper prototypes, ERD's etc.

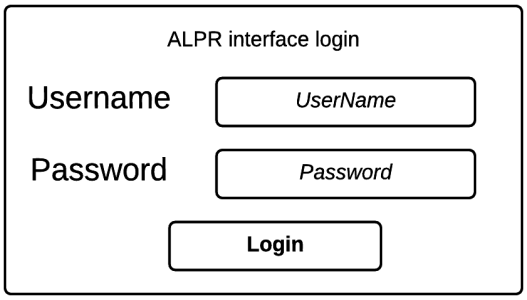

Figure 18: ALPR interface login prototype

The login screen will be the first thing a user will be prompted with when opening the interface. The user will need to enter a Username and Password to gain access to the actual interface. The login screen will have two texts fields and one button. The screen will be simple just like the prototype because there is no guarantee that the user will have prior experience with computers. This is why the login screen needs to look simple and be intuitive.

4.4.2 Main Interface for Database

The ALPR main interface is where all the collected data is going to be accessed from. This low fidelity prototype was also created using Lucidchart.com. Below in *Figure 19* is the prototype interface.

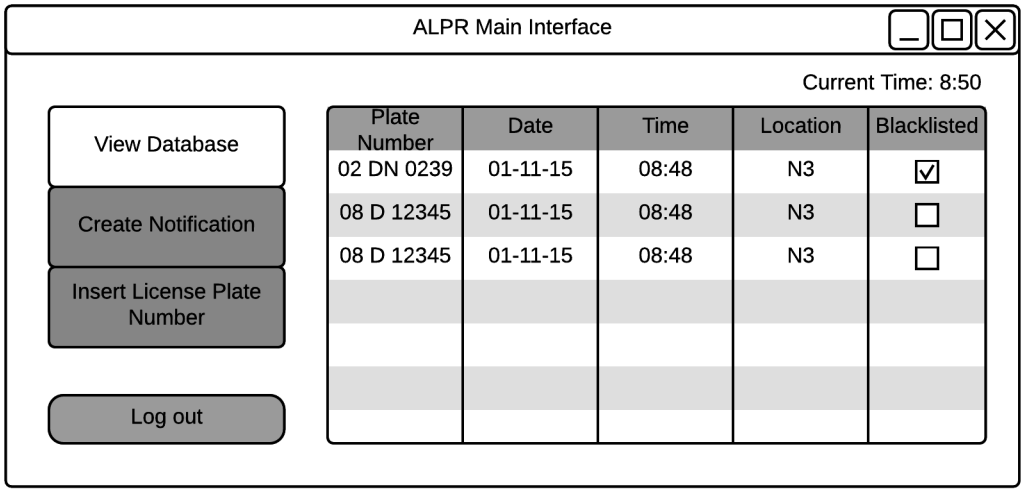

Figure 19: ALPR main interface prototype

The design of the prototype interface is made to be simple and intuitive for the user to use. Clear buttons on the left side of the interface will be used to perform the main functions of the system. The "View Database" button will display information stored in the MySQL database. "Create Notification" button will be used to create notifications for specific license plate numbers. The notifications will alert the system if the specified number has been caught by the ALPR system.

The database view has a blacklisted column which has a tick box option. A tick will be added to a previously notified license plate number so the user will know that a blacklisted vehicle has passed that location. Additional features might be added to the system so that it would send an email or SMS message to the user as additional notification.
The prototype will change as the system is developed and additional functions may arise. Changes will be documented in the next section

4.4.3 Changes to the Design

The design of the interface has changed during the development of the system. The login screen has been removed because it was unnecessary and difficult to implement without an SQL database. The client side interface also had small changes to it. The main things that change in the client interface is that the user can now see exactly what image is being processed, the cropped license plate the binarization of the license plate and which characters are being segmented. *Figure 20* is the final user interface design of the ALPR system

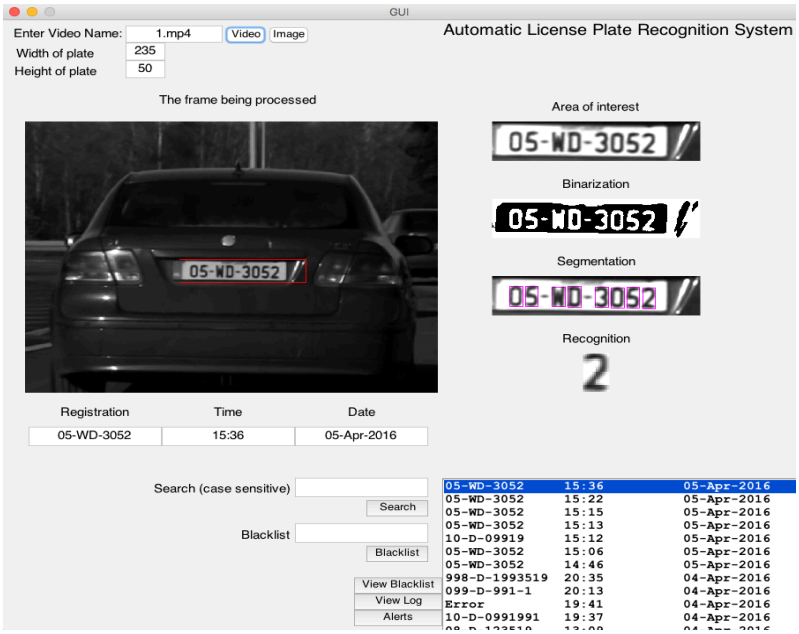

Figure 20: ALPR final interface design

The design is simple and easy for a new user to understand. The top left corner has an edit box for a source file of a video or image. The user can then leave the default width and height for the license plate being processed or change it depending on the distance of the vehicle. The large image in the middle is the frame that is being processed with a red outline of the located license plate. The area outlined with red is the area of interest that is being shown on the right in a larger size. The area of interest is cropped from the frame image and sent to the segmentation function where the license plate has been binarized and every character is being segmented as shown in the "Segmentation" image with bounding boxes around every segmented character.

The final result is displayed under the large image of the vehicle with the date and time it was captured. The bottom right corner displays the log of saved license plates and it is updated every time a new license plate number is extracted and added to it. There is a search box for the log of license plate and also a blacklist box where the user can type and add license plates to the log of blacklisted vehicles. All the logs can be used using the three bottom buttons.

4.5 Image Processing Back-end

Industrial ALPR's improve the recognition of license plate from image by using infrared cameras which significantly improve the accuracy of the an ALPR. For this project a Canon 600D DSLR camera will be used to take video footage and images of vehicles. The back-end of this ALPR techniques such as object detections, image processing and pattern recognition to locate and extract the license plate. The different types of license plates or environments can cause a challenge when detecting and trying to recognise a license plate from an image. Some of these problems can be caused as followed [16]:

- Location: License plates can exist in different locations of an image or be confused with something else
- Quantity: Being unable to locate a license plate if it is not in the image or locating too many license plates and not being able to process them correctly
- Size: Camera angle and distance can change the size of a license plate

- Font: International license plates have different fonts
- Occlusion: License plate could be obscured by dirt
- Inclinations: License plate could be tilted

Some of these problems can be prevented with a fixed camera location and distance from the vehicle. ALPR systems extract license plate number from images using the four main stages described in *Figure 21*. Image capturing is done using a camera. When selecting a camera, features such as the camera resolution, shutter speed, orientation and light have to be considered. The extraction of the license plate can be based on features such as the existence of characters, colour or the boundary from the image. "*The third stage is to segment the license plate and extract the characters by projecting their colour information, labelling them, or matching their positions with templates."* [16]. Character recognition can be done using methods such as template matching or using classifiers such as neural networks. These four stages are what makes a good performing ALPR system.

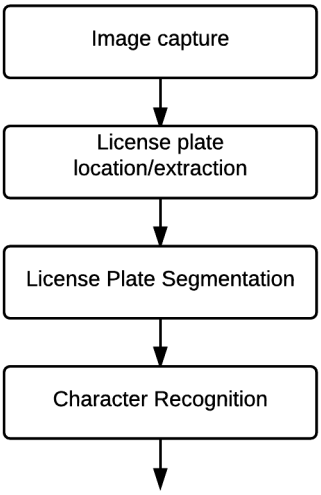

Figure 21: Main processing stages in ALPR

4.5.1 License Plate Extraction

In the license plate extraction stage, the license plate needs to be extracted from the source video/image to further process. A license plate can be distinguished by its features and doesn't require the processing of all the pixels in an image. The system will only need to process the pixels that have similar features of a license plate. The main feature that a license plate it can be distinguished by is the rectangular shape of the license plate with the corners being used to an advantage as mentioned in previous chapters.

Harris corner detection is used to find all the corners in an image, for this system the user can specify a smaller region of the image to be processed by pre allocating an area of the image that will definitely have a vehicle pass through. This area will be check for corners using the Harris algorithm followed by scanning the image 130x50 pixels at a time, depending on what the user has inputted the license plate dimensions, to locate the area with the highest amount of corners within a specific threshold. The flow of this process is shown in *Figure 22*.

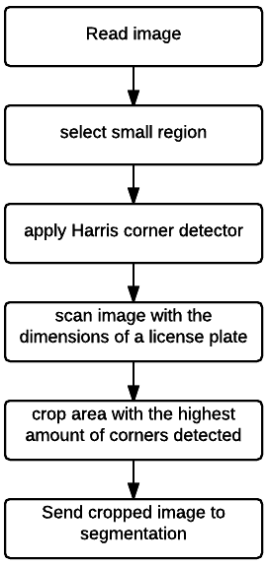

Figure 22: Process of locating a license plate

4.5.2 Character Segmentation

Character segmentation is a process that needs to be done by an ALPR system to find single characters in a license plate that will later be classified. This is a very important part of an ALPR system. The goal is to segment the characters, without losing any features of the characters. Segmentation is done by processing the to grey scale and binarization. An algorithm is then used to segment the characters from the license plate to obtain the characters individually. If the camera used for the source video/image is poor, then additional filtering can be applied to the segmented characters to enhance and removing any noise that might be in the image.

An easy way to segment objects in an image is to use the regionprops function of image processing technologies e.g. MATLAB, OpenCV. This function locates connected components (objects) in a binary image. For a character, the properties such as the area value which is the actual amount of pixels in the component is needed and the bounding box of the component. These are used to predict if the located component in the binary image is a character or not. An example of this would be to set the min and max value of an area that is considered to be a character. If the character 0 is a 24x14 image, then the area of that component would be between 100-170 pixels. This way anything that is not between the specified min and max value is discarded and not sent to recognition. To make better use of the regionprops function would be to take the bounding box of the component and give that a min and max value also.

4.5.3 Character Recognition

Character recognition is the most important part of any ALPR because if an ALPR system can't classify segmented character, it isn't much of an ALPR system. Character recognition can be done in many ways but the two most common are template matching and using a neural network. This system will use a neural network to perform character recognition. Neural network has been mentioned before in this report but now an explanation of how it classifies characters will be given. A feed-forward neural network will to classify the segmented characters to their corresponding ASCII.

4.5.3.1 Neural Network

Efficient neural networks have multiplayers of nodes that consist of an input layer, hidden layer(s) and the output layer. Neural networks are commonly used to train pattern recognition which is exactly what is needed for character recognition. The neural network used in this project uses a multilayer neural network that consists of 3 layers, the input layer, the hidden layer and the output layer. These layers have multiple node that are connected with "weight". The aim of training a neural network is to assign correct weights for these edges [17]. These weights are used to determine the correct output vector for a given input vector.

To train the neural network, a back propagation algorithm was applied. Back propagation is a supervised training scheme which simply means that the neural network is learning from labelled training data that is guided during training [17]. Back propagation is basically changing the neural network weights by the learning rate of the network to better match the expected output.

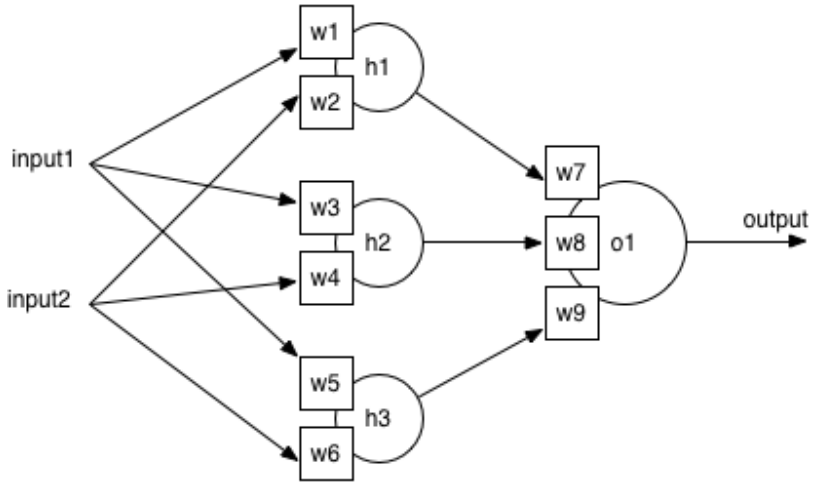

Figure 23: 3 Layer Neural Network

4.5.3.2 Back propagation

The first step in training a neural network is to assign random weights within a specified range. For every input in the training set, the neural network activates and its output is observed [17] and compared against the output that it should have been. The error is calculated using the Delta rule which is calculated using the formula in *Figure 24*.

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

Figure 24: Delta rule calculation [18]

Target being what the expected weight should be, out being the activation value of the neural network. This error value is subtracted by the value of the current weight and multiplied by the learning rate of the neural network [18]. This is repeated to all the weights until the neural networks output error reached a predetermined value or the number or training cycles (epochs) is reached. *Figure 24* shows a neural network for the recognition of hand written digits with the dimensions of 28x28 pixels [19]. It has 784 input layers (28x28 = 784) 15 hidden layers and 10 output layers that correspond to 0-9 digits.

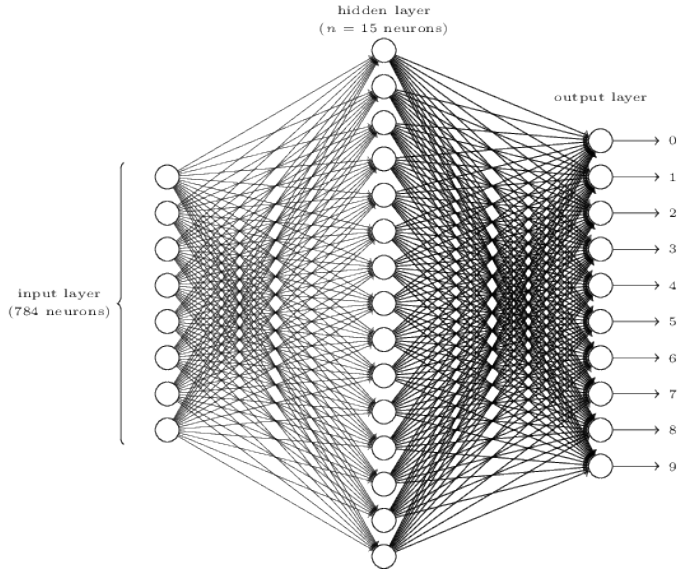

Figure 25: Multilayer Neural Network for hand written digit recognition [19]

The input neurons go through every hidden layer to check if the input weight is higher than the hidden layer threshold. If the weight is higher than the threshold, that neuron is activated and the same process is being done to the output layer. If the first output layer is activated, then that means the inputted image was the number 0. If the send output layer is activated, then that means the inputted image was the number 1 and so on.

4.6 Design Methodologies

The ALPR system will be designed using one of the methodologies that will be reviewed in this section of the Design chapter. Three principle design methodologies will be reviewed. These are the waterfall, agile and spiral design models. Keeping in mind that this project is being developed by single developer, so a suitable lifecycle will need to chosen from various models:

4.6.1 Waterfall

The waterfall model is a sequential design progression in which the process constantly flows forwards. In the traditional model, once a stage of development has been completed, it cannot be revisited. The water fall relies on the requirements being clearly stated and understood prior to beginning the development of software. Once the development begins on the system. The waterfall model assumes there will be no changes made to the requirements so there is no way to revise or add requirements later in the lifecycle. The main benefit of this methodology is a well-structured, laid out plan and it works best with a large of designers, developers and testers e.g. corporate environment.

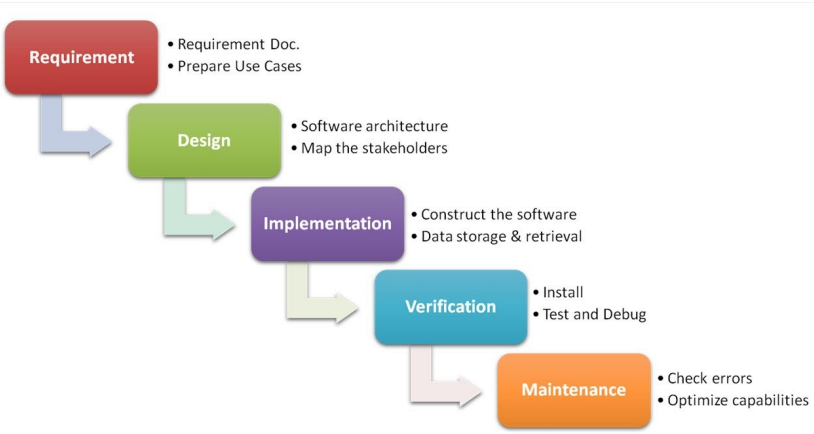

Figure 26: Waterfall Lifecycle Model

4.6.2 Agile

Agile methodology allows the system to be implemented in iterations. Developer can make new scope decisions a little at a time throughout the project as new requirements become clear. It supports the constant revision and re-visitation of previous stages of the planning and development process. Agile methodologies give flexibility in the development of the system and allows the developers to respond quickly to changes within the project requirements. Every increment that is completed in the project will incorporate the following [20]:

- Requirements Engineering
- Design
- Implementation and coding
- Verification/testing and Write up
- Delivery of Increment to System

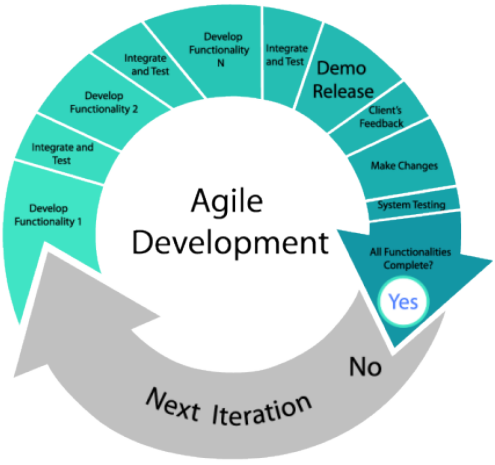

Figure 27: Agile Development Methodology

4.6.3 Spiral

The spiral model is similar to an incremental methodology but it has a heavy focus on risk analysis [21]. The spiral model is broken into 4 phases:

- Design – What needs to be accomplished.
- Evaluate – Determine the best way to accomplish goals. Focus heavily on risk-reward and what has worked for similar systems.

- Develop – Build the system that has been designed.
- Review – Compare the system to the design and determine how closely they match and how to continue.

The spiral model is repetitive. Each phase is repeated until the goal is accomplished. This methodology is very effective when applied to larger project due to the focus on risk analysis. This methodology is best suited when the project requirements are unclear and subject to change after the project has begun.

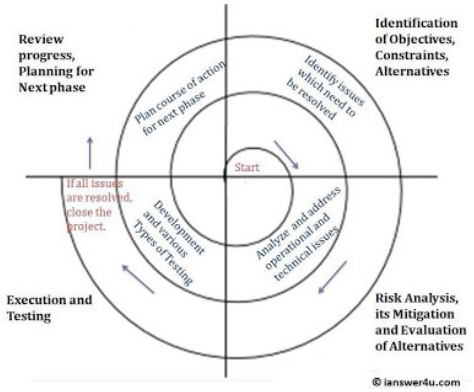

Figure 28: Spiral Development Methodology

4.6.4 Chosen Design Methodology Conclusion

The design methodology that best suits this project is the agile design methodology. Specifically, the Agile Unified Process (AUP). This is a simplified version of IBM Rational Unified Process (RUP) and is a concept built on an easy to understand implementation of agile practices. The agile methodology is chosen for the ALPR system because the requirements can change during the development of the system and time provided to finish the project is short.

AUP supports useful techniques and concepts, such as test-driven development (TDD) and agile change management. TDD is important in a code heavy project as it allows for individual "modules" to be developed independently and later fitted into a larger project [16]. *Figure 19* shows the 4 phases and 7 workflows of the AUP cycle.

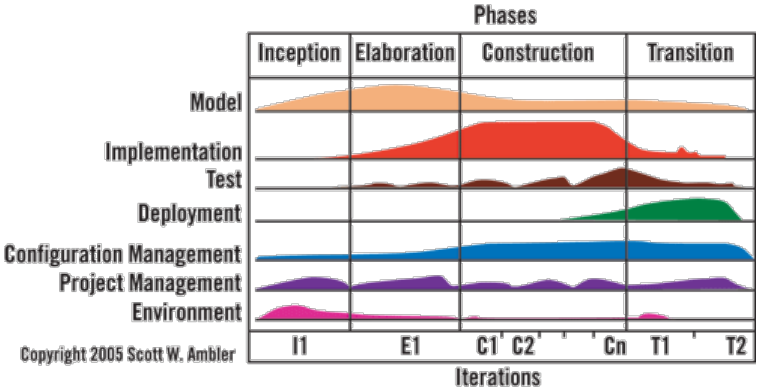

Figure 29: AUP Lifecycle

The principle of the Agile Unified Process is caught in its 4 main stage [22]:

1. **Inception**. The objective is to distinguish the underlying extent of a task, a potential architecture for the system.
2. **Elaboration**. The objective is to demonstrate the design of the system
3. **Construction**. The objective is to produce working software consistently which meets the most noteworthy needs of the project "stakeholders".
4. **Transition**. The objective is to approve and be ready to move the developed system into a production environment.

Workflows progress through iterations which transitions the process across the 4 phases. These iterations can be done on a schedule that the system designer decides.

1. **Model**. The project scope, risks, costs and the feasibility of the project are assessed and the project environment is prepared.
2. **Implementation**. The plans developed in the model phases are turned into code and thoroughly tested with unit tests.
3. **Test**. Every aspect of the system is tested to ensure it meets the quality required. This is done through unit testing, usability testing and user-acceptance testing.
4. **Deployment**. Ensures that the proper steps are taking when deploying the workflow.
5. **Configuration Management**. In this phase any system artefacts, such as configuration files are kept up to date.
6. **Project Management**. Controls activities related to the project but not directly involved are managed.
7. **Environment**. Makes sure that the tools, processes and standards that the system requires are kept up to date and kept relevant

AUP provides the balance between the structure and agility that will be necessary in the development of the system.

4.7 Conclusion

The research of methodologies that was carried out in this chapter helped choose the most suitable one for this project. The design of the front and and back end in this section will greatly help with the development of the whole system. The prototype of the user interface helps visualise what it will look like and helped to create the changes that were shown in *Figure 20* to improve the user interface of the ALPR. The explanation of the image processing functions will be helpful in the development stage of the project and how it can be improved using the methods such as neutrals networks and Harris corner detector algorithm

This chapter has documented some important front end prototypes, changes, design methodologies and back end details. The next chapter will look at how these information was used in the implementation of the system.

Chapter 5. Implementation

5.1 Introduction

In this chapter the designs discussed in chapter 4 will be implemented and and overview of the whole system will be done. A walk through of the final result and explanation of how it was achieved. Details about why certain implementations were done and how they could have been improved.

The design and implantation follows the research from previous chapters. The final system will be illustrated in this chapter with the use of screenshots and code snippets of important algorithms and techniques used to achieve the final result. All of the coding was developed in MATLAB R2015a as mentioned in the technology research chapter.

5.2 System Overview

5.2.1 User interface

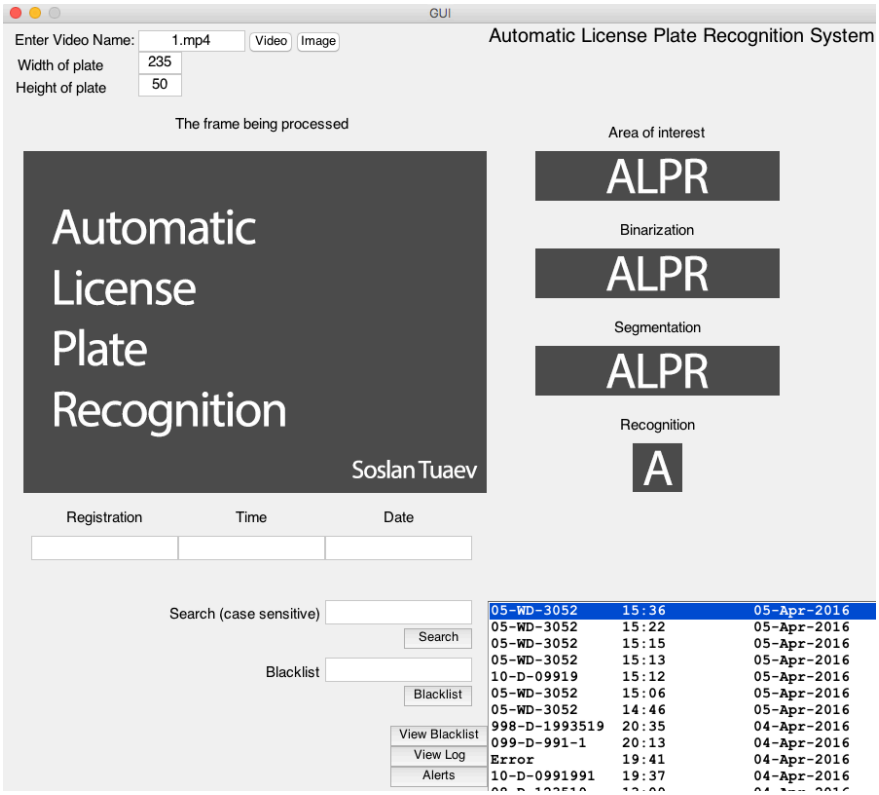

Figure 30: ALPR interface

The ALPR interface was briefly discussed in the design chapter under the subheading "changes to the design" but a more in-depth overview will be done here.
The interface was build in MATLAB "GUIDE" GUI builder using the drag and drop feature to add edit boxes, images and lists to the interface. *Figure 31* is what the GUI builder looks like when editing the interface.

There is one unnecessary feature of the ALPR interface and that is the 4 images on the right. They are unnecessary because they are only used to demonstrate the back end image processing functions of the ALPR. In a real world ALPR these wont be of any use except the main image. The bottom right box with the license plate log data is updated every time the interface is open or a license plate has been processed. The log data can be searched using the search box to search data by the license plate number, time or date. Lastly, the plate width and height is there for the user to decide what size of a license plate is to be searched.

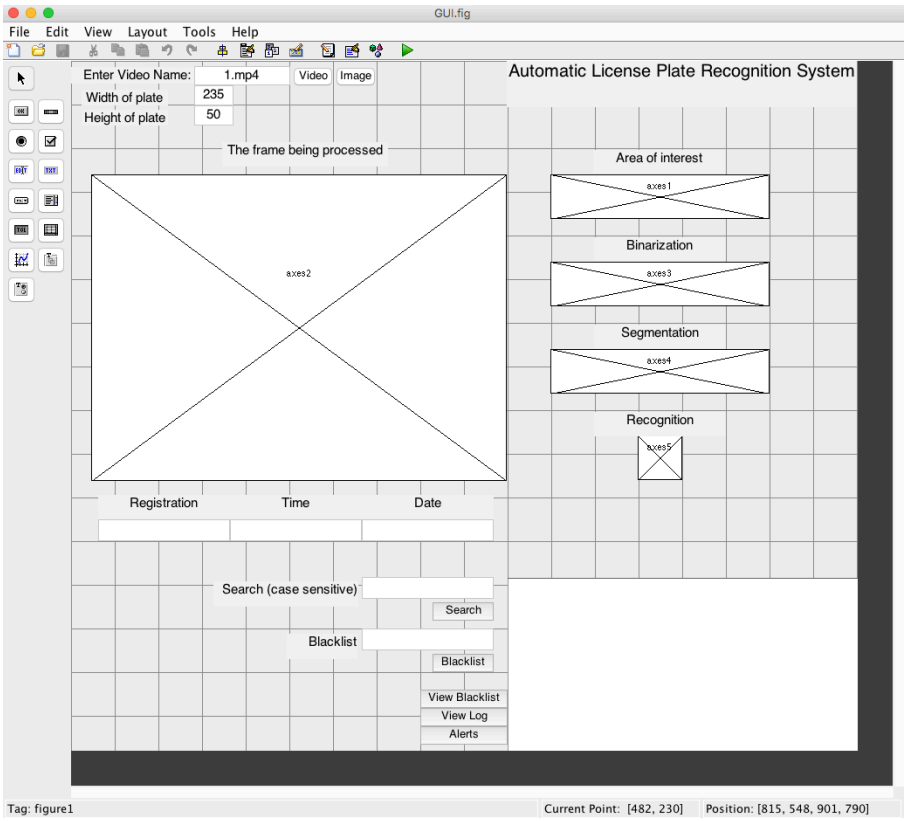

Figure 31: MATLAB GUIDE GUI builder

It is very clean and easy to work with to create a simple user friendly interface. Each object that is dragged into the interface creates a function in the main file of the system (GUI.m) that is then edited to perform actions.

5.2.2 Importing video and image

The current ALPR system in this project is only able to work with already recorded video footage and still images. Real time processing was not implemented during the course of this project.

Once a user has imported a recorded video and pressed the `"Video"` button. The program will call the `ProcessVideo` function that will count every frame in the video source so that a for loop can be initiated that loops through all the frames in the video. For demonstration purposes, the program only get the 10^{th} frame of every video. *Figure 32* is the implementation of counting the amount of frames in the video and reading every frame and storing them into array an array.

```
130 -   frameCount=videoSource.NumberOfFrames;    %#ok<VIDREAD> % The number of frames in the video file
131
132     % Reading a range of frames (numbered from 1)
133 -   videoFrameRange=[1 frameCount]; % frame range
134 -   capturedFrames=read(videoSource,videoFrameRange); %#ok<VIDREAD> % Reading frames
135
136 -   startingLine=1;          % Beginning of line
137 -   timeString
138 -   for i=10:10 % Processing frame 10-10
139
140 -       frameLoopCount=i; % frame from array.
141 -       singleFrame=capturedFrames(:,:,:,frameLoopCount); %  single frame from array
142         % Call license plate detection function to locate  the license plate region
143 -       [~,Number,Letter,~,Win]=AreaOfInterest(singleFrame,startingLine,handles);
```

Figure 32: Reading frames from video in MATLAB

In this example the for loop is only going to loop one time because of demonstration purposes as already mentioned. `"capturedFrames"` is the array holding every frame of the video and in line 141 a single frame is being taken from the capturedFrames array and stored in a new variable called `singleFrame` which is then sent to the `"AreaOfInterest"` function where the real image processing begins by locating the license plate in the captured frame. If the user was to insert a static image and press the "Image" button, then the `"ProcessImageButton"` function will be called where the image will be read into a variable and sent straight to the `AreaOfInterest` function

5.2.3 Locating the License Plate

In the previous section, a video frame image has been sent to the `AreaOfInterest` function to locate the license plate region. The first process in this function is to convert the image into grayscale and adjust the adjust the contrast using the built-in `imadjust` function that corrects the contrast of the image if it is too high. Images are usually of the class uint8 and have to be converted to double precision values mainly to avoid any problems that might arise when using unit8. Next is selecting a smaller region of the image by creating an offset of for the image. *Figure 33* explains it better with visuals.

Figure 33: selecting smaller region from input image

Which is done with the following code in the `AreaOfInterest` function where startingLine is equal to 200 and `I1` is the original inputted image. This code is simply predefining an area that is expect to have a vehicle driven past and then that frame cropping out of the original image.

```
%-----------------Selection of the region of interest-----------------
sm=450;                         % offset(moving down)
point1=[1 startingLine];        % From Column to Row [C R]
point2=[768 startingLine+sm]; % From Colum to Row [C R]

rmin=round(point1(1,1));          % rounding to the nearest numbers
rmax=round(point2(1,1));
cmin=round(point1(1,2));
cmax=round(point2(1,2));

I2=I1(cmin:cmax,rmin:rmax); % storing region of interest in I2
%       figure,imshow(I2);
```

Figure 34: selecting smaller region of input image

Creating this smaller region allows for a better chance to locate the license plate in the image. Now the implementation of Harris corner detection [10] can begin.

5.2.3.1 Mathematical Calculate of Corners using Harris Detector

Corners are represented by a partial change in the gradient of an image and that is what this algorithm is looking for. This is done by considering a local window in an image so that an average change of image intensity will determine a result by shifting the window by a small amount in various directions [10]. A corner is found when the all shifts have a large change which can be detected if the minimum change produced by any of the shifts is large.
The Harris algorithm is an improvement of Moravec's corner detector algorithm. Moravec's algorithm suffer because of three problems [10]:

1. T*he response is anisotropic because only a discrete set of shifts at every 45 degrees is considered*
2. *The response is noisy because the window is binary and rectangular*
3. *The operator responds too readily to edges because only the minimum of E is taken into account*

The mathematical specification of Moravec's algorithm is in *Figure 35* where *I* is the intensity of an inputted grayscale image, *w(u,v)* is the image window, *x* is the the shift in the *u* direction, *y* in the right direction. In simply terms, Moravec's algorithm is looking for a maxima in the min {E} that is above a threshold [10]:

$$E_{x,y} = \sum_{u,v} w_{u,v}[I_{x+u,y+v} - I_{u,v}]^2$$

Figure 35: Moravec's corner detector

The Harris algorithm improves the problems that Moravec's algorithm suffers from, by using a gradient formulation that will detect the response of any shift in *(x,y)* with the formulas shown in *Figure 36.*

Harris corner detector improved formula

$$E_{x,y} = \sum_{u,v} w_{u,v}[I_{x+u,y+v} - I_{u,v}]^2$$

$$= \sum_{u,v} w_{u,v}[xX + yY + O(x^2, y^2)]^2 \quad \textit{Approximation}$$

$$X = I \otimes (-1, 0, 1) = \partial I / \partial x \quad \textit{Gradient of x}$$
$$Y = I \otimes (-1, 0, 1)^T = \partial I / \partial y \quad \textit{Gradient of y}$$

$$E_(x, y) = Ax^2 + 2Cxy + By^2 \quad \textit{E, Small shifts}$$

Where

$$A = X^2 \otimes w$$
$$B = Y^2 \otimes w$$
$$C = (XY) \otimes w$$

$$w(u, v) = exp - (u^2 + v^2)/2\sigma^2 \quad \textit{use a smooth circular window, for example a Gaussian:}$$

Reformulation

$$E(x, y) = (x, y)M(x, y)^T$$

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \quad \textit{The eigenvalues of M correspond to principal curvatures of the local autocorrelation function}$$

$$Tr(M) = \alpha + \beta = A + B \quad \textit{Trace}$$
$$Det(M) = \alpha\beta = AB - C^2 \quad \textit{Determinant}$$
$$R = Det(M) - kTr(M)^2 \quad \textit{Response}$$

R is positive for corners, negative for edges, and small for flat regions.

Figure 36: Harris corner detector formula

5.2.3.2 Implementing Harris Corner Detector in MATLAB

The Harris corner detector was implemented in this project to locate the coordinate of corners and then selecting an area where the highest amount of corners was located. This was mentioned in previous sections in better detail. The implementation of the algorithm in MATLAB is as follows.

Continuing from the last two sections where the smaller region of interest was selected and Harris formula was shown in *Figure 36*: *Figure 37* will show the corresponding formula in MATLAB's language. First thing is to apply a sobel mask to get the gradient of `x` and `y` followed by apply a Gaussian filter.

```
23    %--------Applying sobel edge detector--------
24 -  dx=[-1 0 1; -1 0 1; -1 0 1];    % Sobel mask horizontal
25 -  dy=dx';                         % Sobel mask vertical
26    % Applying mask to image
27 -  Ix=conv2(double(I2),dx,'same'); % Gradient of x
28 -  Iy=conv2(double(I2),dy,'same'); % Gradient of y
29
30    %--------Applying gausian filter on computed values--------
31 -  sigma=2;
32 -  g=fspecial('gaussian',max(1,fix(6*sigma)),sigma);
33 -  Ix2=conv2(Ix.^2,g,'same');
34 -  Iy2=conv2(Iy.^2,g,'same');
35 -  Ixy=conv2(Ix.*Iy,g,'same');
36
```

$$X = I \otimes (-1, 0, 1) = \partial I / \partial x$$
$$Y = I \otimes (-1, 0, 1)^T = \partial I / \partial y$$

$$A = X^2 \otimes w$$
$$B = Y^2 \otimes w$$
$$C = (XY) \otimes w$$

Figure 37: Applying Sobel mask and Gaussian filter to image

From line 24-28 in *Figure 37*, a Sobel mask is being applied to the image matrix horizontally and vertically so that the mask is applied to the whole image, followed by creating a Gaussian filter that is of size 6*sigma and of a minimum size 1x1. This Gaussian filter is applied to get the "Ix2, Iy2, Ixy" (A, B, C) which is basically the eigenvalues of the M [2x2] matrix in *Figure 36*. Next is to calculate the trace, determinant and response.

```
37    %--------------Harris Corner Detector------------
38 -  k=0.04;      % Harris parameter
39 -  Tr = (Ix2 + Iy2);                  % Trace
40 -  Det = (Ix2.*Iy2 - Ixy.^2);         % Determinant
41 -  Respon=Det - k*Tr.^2;              % Reponse
42 -  R=(1000/max(max(Respon)))*Respon; % Rationing weights
43
44    %--------Non-maximal suppression and threshold--------
45 -  radius=6;                         % radius of region considered in non-maximal suppression.
46 -  sze=2*radius+1;                   % The size of the mask
47 -  MX=ordfilt2(R,sze^2,ones(sze));  % dilation of gray scale
48
49 -  thresHold=150;                    % threshold -- 150
50 -  R11=(R==MX)&(R>thresHold);        % Singular points that are greater than the threshold
51
52 -  [r1,c1]=find(R11);   % The coordinates of the points in the region of interest
53 -  cordRC=[r1 c1];      % Storing coordinates
```

Figure 38: Harris corner detector and threshold

The response is calculated on line 41 in *Figure 37*. The response has weights that determine if there is a corner, edge or flat area in the image. Positives are for corners, negatives are for edges and 0's are for flat regions of the image. In line 42, the weights are being rationed with a threshold of 1000 so that a non-maximal suppression and threshold can be applied in lines 47-50 where the extraction of the local maxima is done by getting the gray scale dilation of the response. If the corner strength matches the dilated image and the threshold is greater than specified (150), that would mean a corner is located. The coordinates of the the located corners is stored in line 52-53 where r1 and c1 correspond to the row and column pixel of the located corner.

5.2.3.3 Locating License Plate Region

Following from the previous section, where the coordinates of corners was stored in the variable `cordRC`. A license plate size needs to be specified, this is taken from the width and height input boxes in the interface. Next, the image dimensions of the image being searched for a license plate needs to add the dimensions of the license plate to the width and height of the image. This is done to prevent any errors that might occur when image is being scanned and the corner of the license plate in the image is near the edge of the image. An error would occur because the scan starts from the center of the detected corners and if the center is near and edge of the image then the scanning box would go outside the image which will result in a computational error. That's why extra space is added to the width and height of the image.

```
64      % Increasing image window field by adding black space to edges (To avoid errors)
65      % Plotting coordinate points on blank matrix
66 -    blackImCoor=zeros(length(R11(:,1))+2*heightSize,length(R11(1,:))+2*widthSize);
67 -    blackImCoor(heightSize:length(blackImCoor(:,1))-heightSize-1,widthSize:length(blackImCoor(1,:))-widthSize-1)=R11;
68      % Plotting coordinate points on original image
69 -    origImCoor=zeros(length(R11(:,1))+2*heightSize,length(R11(1,:))+2*widthSize);
70 -    origImCoor=uint8(origImCoor);
71 -    origImCoor(heightSize:length(origImCoor(:,1))-heightSize-1,widthSize:length(origImCoor(1,:))-widthSize-1)=I2;
```

Figure 39: Increasing the size of image by adding weight and height of license plate

Line 66 is creating a blank image with the size of the original and adding the new width and height to it. Line 67 plotting the coordinates of located corners in the blank image `blackImCoor`. The same thing is done to the original image in lines 69-71 with the results being shown in *Figure 39, 40*.

Figure 40: plotted coordinates on blank matrix

Figure 41: resized original image with plotted coordinates

The coordinates are clearly visible in *Figure 40* and the added height and width is clearly visible in *Figure 41*. Now to find the area that is thought to be the license plate. The new height and width needs to be added to the previously located coordinates of corners by simply taking the row and column from line 52 (*Figure 38*) and adding the new height and width.

Scanning through the image to locate the highest amount of corners in a specified license plate box size is done by looping through all the single corner coordinates, summing the total amount of points found in that box and storing the sum of points found for every loop in an array.

```
79      % Scanning through all the plotted points and getting the sum
80 -    for r=1:numCords
81 -        scanBlack=blackImCoor(newcordRC(r,1)-s1:newcordRC(r,1)+heightSize-s1,newcordRC(r,2)-s2:newcordRC(r,2)+widthSize-s2);
82 -        sumCoor(r)=sum(sum(scanBlack));       % The number of dots (pixels Sum in columns and rows)
83 -    end
```

Figure 42: Loop to sum coordinates found in box size

The code in *Figure 42* is looping through every corner coordinate and with the license plate box size and summing it up in the array `sumCoor`. This is simple to do because blackImCoor is a matrix of 0's and 1's only. 1's being the coordinate of corners and 0's blank space. The highest value in sumCoor is the most likely region that the license plate is located. So that area is cropped out from the original image.

Figure 43: blank matrix highest area with corner coordinates

Figure 44: original image with highest area of corners

5.3 Character segmentation

The image in *Figure 44* is sent to the segmentation function where it is first converted to binary, then searched for connected objects that could be a character and finally segmented and sent to classification.

5.3.1 Locating Objects for Segmentation

This image of the license plate is now in the segmentation function. The first thing that is needs to be done is the binarization of the image, which is turning the image matrix into 1's and 0's. For this specific project, the cropped license plate is a grayscale image with matrix values ranging from 0-255. Binarization is turning these grayscale values into binary. The binarization in this instance is done by dividing the image into 45x45 square blocks and calculating the mean intensity of each block and adding adding 10 to the mean. The mean result is used as the threshold so that everything above the mean threshold is turned to black (0's) and everything else is turned white (1's). In MATLAB this is done using the `blockproc` function in MATLAB, or manually (Lines 4-31 in the Segmentation function). The resulting binary image is shown in the "Binarization" image box in the user interface of the ALPR *Figure 45*.

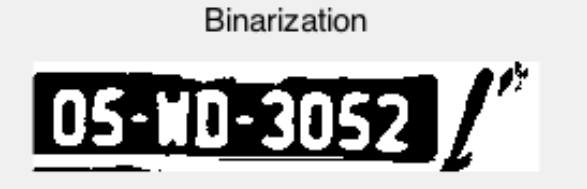

Figure 45: Binarization of license plate

The next step is to find all the objects that are connected in the image. A connected object in a binary image is one that is made up of a single 1 value or multiple 1's connected together like the characters in the binary license plate *Figure 45*. These connected objects are known as blobs. To locate these blobs, the MATLAB `regionprops` function is used to get the area and the bounding box of a blob.

```
51    % Getting the area and boundingbox from the binaryPlate matrix
52 -  blobValues=regionprops(binaryPlate,'Area', 'BoundingBox');
53 -  Length=length(blobValues);    % Number of objects
```

Figure 46: MATLAB regionprops function

A visual example of what `regionprops` is doing is shown in *Figure 47*

0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	1	1	1	1	1	0	0	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	0	0	1	1	0	0
0	0	1	1	1	1	1	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 47: Locating bounding box and area with regionprops function

In the `regionprops` function, the area and bounding box is what was specified to look for. The blob *Figure 47* is of values 1 that represent an image of zero in binary. The bounding box is outlined by the red square and the area that is being counted is marked with red 1's. The bounding box returns the coordinates from where it begins in the format [Column, Row, Width, Height]. These values can then be used to draw a line around the blobs just to show where the blob is located. Outlining the bounding box is unnecessary for an ALPR in the real world but it was used in this project for demonstration purposes of what blobs were being selected.

Once the blob values have been saved in an array, they can be used to decide if the blob is a character or just a random object in the image. This is done by looping through the array and to see if the area and the bounding box is within a range of specified values. A for loop is used to cycle through the array of blob values to in *Figure 48* and if the values are in the specified range of area and bounding box, then the blob bounding box value will be stored in a new array called `bBox`.

```
%------------Filtering to only select characters------------
counter=0;    % licenseBox for rows
for i = 1:Length
    areaL = blobValues(i).Area;         % Area of number/letters
    boxL = blobValues(i).BoundingBox;   % Frame objects

    % Selecting the number and letters blobs by only choosing the areas
    % where the Area and BoundingBox values are > and < than specified
    if areaL > 25 && areaL < 300 && boxL(4)>10 && boxL(3) ...
            >= 2 && boxL(3) < 25
        counter=counter+1; % incrementing the row of the array
        % Box is a 2D array that holds the rounded BoundingBox in every
        % row that is incremented
        bBox(counter,:)=round(blobValues(i).BoundingBox);
    end
end
```

Figure 48: Filtering to select character bounding box coordinates

A blob is classified as a character if the value of the area is within the range 25-300, the bounding box height is greater than 10 pixels and the width is between 2-25 pixels. These values were decided after testing multiple image blobs from license plates. `bBox` is an array of `counter` rows and bounding box values that are going to be sent to image recognition where a neural network will be used to classify the image.

5.4 Character Recognition

Character recognition is done using feed forward neural network that was discussed in the design chapter. Before implementing the neural network in the ALPR, the neural network needs to be trained so that it will be able to classify the inputted segmentation image.

5.4.1.1 Training the Neural Network

In the design chapter, training a neural network was already explained. So in this section, the implementation of training a neural network using back propagation will be explained using MATLAB code snippets.

To train a neural network, a set of sample images needs to collected. These sample images are called training sets *Figure 49*. For this project, 100 24x14 images of each numbers and letters that occur in the Irish license plate were collected and stored in individual folders with the name of the number and letter. Every image in every folder is named from 1-100. The images are grayscale and in bitmap (.bmp) format.

Figure 49: Training set images

The larger the training set of images, the better the neural network will be able to recognize input images.

These training sets are first imported into MATLAB by specifying the path to the folder and looping through the folder to import and store every image as a "slice" in a 3D array.

The first 90 images in every folder is used as the target image that the neural network will use to train itself to recognize and the remaining images 90-100 will be used to test the neural network against. The 3D array of every number is then added to one whole 3D array that will hold every training image in the matrix, so for number 0, the 90 training images will be in the 3D array slices (:,:,0) to (:,:,90) and so on for every number until the array is full with 900 slices. This is done to make it easier to cycle and process the images.

```
106     % Creating a 1 whole matrix for the training images
107 -   for i=1:N   % Training images
108         % taking the previously created 3D image matrix and storing it in one
109         % whole matrix like so: (0)1-90, (1)91-180, (2)181-270 etc
110 -       im(:,:,i)=im0(:,:,i);
111 -       im(:,:,i+N)=im1(:,:,i);
112 -       im(:,:,i+2*N)=im2(:,:,i);
113 -       im(:,:,i+3*N)=im3(:,:,i);
114 -       im(:,:,i+4*N)=im4(:,:,i);
115 -       im(:,:,i+5*N)=im5(:,:,i);
116 -       im(:,:,i+6*N)=im6(:,:,i);
117 -       im(:,:,i+7*N)=im7(:,:,i);
118 -       im(:,:,i+8*N)=im8(:,:,i);
119 -       im(:,:,i+9*N)=im9(:,:,i);
120
121         %     fprintf('Now inserting planes %d, %d, and %d\n', i, i+N, i+2*N);
122 -   end
```

Figure 50: Storing single number 3D images into one 3D matrix

Next the images need to be converted to double precision values from their original uint8 class. This is done for calculation purposes because working with double precision values prevents errors can could occur during calculations and also the fact that the images that are sent to recognition in the ALPR are also double precision values.

```
138      % The transition of values from (0:255) to (0:1.0)
139 -    for i=1:10*N   % Training images
140 -        imd(:,:,i)=(im2double(im(:,:,i)));
141 -    end
```

Figure 51: Converting to matrix values of images to double precision values

Once the values are double's the next step is to convert every slice of the 3D array into vectors, which is done so that when the neural network is being trained, it can cycle through the arrays without any problems.

```
147      %------------------Input and test images----------------
148      % Converting 24x14x900 matrix to a linear 1x336(900x336)
149      % 1 5 9  13  = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
150      % 2 6 10 14
151      % 3 7 11 15
152      % 4 8 12 16
153 -    for s=1:10*N % 0-900
154 -        for i=1:14 % every column
155 -            for j=1:24  % every row
156                  % taking every row(j) value in every column(i)
157                  % and storing in a new single row
158 -                Xv(s,(i-1)*24+j)=imd(j,i,s);
159 -            end
160 -        end
161 -    end
```

Figure 52: Converting every slice of the 3D array of image values into a 2D vector array

Another vector T is created as the target out for the output weight of the neural network. This target vector will be used to calculate the error of every training cycle of the neural network. The error will be calculated by getting the difference between the output value of the neural network and the target vector so as to configure the weights of the neural network to better fit the target output. This is done during the back propagation where the goal is to update each of the output weights from every training cycle to be nearer to the actual target output.

```
176      %----------------------Classification-----------------------
177      % 10 dimensional column vector. For example, if a particular training image, x,
178      % depicts a 6, then y(x)=(0,0,0,0,0,0,1,0,0,0)T is the desired output
179      % from the network
180      T = 1:10;
181 -    for i=1:N
182 -        T(i,:)=[1 0 0 0 0 0 0 0 0 0];
183 -        T(i+N,:)=[0 1 0 0 0 0 0 0 0 0];
184 -        T(i+2*N,:)=[0 0 1 0 0 0 0 0 0 0];
185 -        T(i+3*N,:)=[0 0 0 1 0 0 0 0 0 0];
186 -        T(i+4*N,:)=[0 0 0 0 1 0 0 0 0 0];
187 -        T(i+5*N,:)=[0 0 0 0 0 1 0 0 0 0];
188 -        T(i+6*N,:)=[0 0 0 0 0 0 1 0 0 0];
189 -        T(i+7*N,:)=[0 0 0 0 0 0 0 1 0 0];
190 -        T(i+8*N,:)=[0 0 0 0 0 0 0 0 1 0];
191 -        T(i+9*N,:)=[0 0 0 0 0 0 0 0 0 1];
192 -    end
```

Figure 53: Creating the target output for the neural network

The final step before starting to train a neural network is to set the amount of input hidden and output neurons in the network. The input neurons are going to be 336 because the images that are being used are of the size 24x14 and the vector that was created with them is 336 columns long. The output neurons are going to be 10 because there are 10 numbers to be trained (0-9). The hidden layer neurons can vary but for 336 input neurons and 10 output neurons, 50 is a good amount of hidden neurons so that the network wont undertrain which will lead to high prediction error and or over fit the network with too many neurons which will lead to the network not generalizing well. There could be more hidden neurons but 50 does the job. The learning rate is going to be 0.5 and the training period will be 100 cycles.

The last step before the neural network starts to train is that a weight matrix needs to be created for hidden and output layers that will be changed to be closer to the actual target weight `T`.

```
206     % Randomising the weight matrix. (Extra row added that it
207     % can multiply against the threshold value and not modify the threshold.
208     % We simply want the threshold value added with the
209     % other values.
210 -   w1=randint(inNeu+1,hidnNeu,[-2,2])/10+rand(inNeu+1,hidnNeu)/100;   % The weights in the range [-0.3 0.3]
211 -   w2=randint(hidnNeu+1,outNeu,[-2,2])/10+rand(hidnNeu+1,outNeu)/100;
```

Figure 54: Randomising weights for the neural network to train

Now the training can begin for the neural network. The first image vector is taken and set as the input neurons and fed forward to the neural network. The feed forward process is taking the input neurons and multiplying every column of the input neurons by row of the hidden layer weight matrix to create a connection weight which is used in the pre-activation function to calculate the total net input for every hidden neuron and then squashing it using the logistic function to get the output weight of the hidden.

```
213 -   for e=1:cycle          % Cycle for cycle for all epoch(100)
214 -       for I=1:10*N      % The cycle one epoch (900)
215             % Feed forward
216 -           x(1:inNeu)=Xv(I,:); % Passing single row (image)
217             %------------The output value of the hidden layer----
218             % preallocating [1 50] zeros to store the output value of the hidden layer
219 -           net=zeros(1,hidnNeu);
220 -           o=zeros(1,hidnNeu);
221 -           for i=1:hidnNeu
222 -               xw=0;
223 -               for j=1:inNeu
224                     % Connection weights
225                     % multiplying every column of x(image I) by every
226                     % row+1 of the w1 matrix to create a hidden layer connection weight
227 -                   xw=xw+(x(j)*w1(j+1,i));
228 -               end
229                 % Pre-activation
230                 % The total net inputs of the hidden layer
231 -               net(i)=(1*w1(1,i))+xw;
232
233                 % Logistic activation function: squahes the neurons
234                 % pre-activation between 0 and 1
235                 % Creating the sum output weight of the hidden layer by getting the
236                 % exponential of net. The activation function compressing
237                 % values in the range 0-1
238 -               o(i)=1/(1+exp(-net(i)));
239 -           end
```

Figure 55: Feed forward, hidden layer

Once the hidden layer outputs weights have been calculated and stored in `o(i)`, the next step is to calculate the output neurons of the output layer by performing the same process using the output from the hidden layer neurons as the input for the output layer.

```
241             %------------The output value of the output layer---
242             % preallocating [1 10] zeros to store the output value of the output layer
243 -           netvih=zeros(1,outNeu);
244 -           ovih=zeros(1,outNeu);
245 -           for i=1:outNeu
246 -               ow=0;
247 -               for j=1:hidnNeu
248                     % Connection weights
249                     % multiplying every column of o(Hidden layers) by every
250                     % row+1 of the w2 matrix to create an output layer connection weight
251 -                   ow=ow+(o(j)*w2(j+1,i));
252 -               end
253                 % Pre-activation
254                 % The total net inputs of the output layer
255 -               netvih(i)=(1*w2(1,i))+ow;
256                 % Logistic activation function: squahes the neurons
257                 % pre-activation between 0 and 1
258                 % Creating the sum output weight of the output layer by getting the
259                 % exponential of netvih. The activation function compressing
260                 % values in the range 0-1
261 -               ovih(i)=1/(1+exp(-netvih(i)));
262 -           end
```

Figure 56: Feed forward, output layer

Now that the output neurons have been calculated, the feed forward process is over and back propagation begins. Back propagation is used to update each of the weights in the network, starting from the output and going back, so that the actual output is closer to the target output.

The first step in back propagation is to compare the actual output from the output layer against the target output using the delta rule. The delta rule is used to minimize the error in the output of the neural network through a gradient decent. Once delta is calculated, the actual output error can be decreased by subtracting this value from the current output neuron weights and multiplying by the learning rate, which is 0.5 in this neural network.

The actual updates of the output neuron weights is done after the same calculate is done for the hidden layer. *Figure 57* is the English of what is happening in back propagation and *Figure 58* is the MATLAB implementation of back propagation.

Output layer

Delta = (TargetO - ActualO) * ActualO * (1 - ActualO)
Weight = Weight + LearningRate * Delta * Input

Hidden Layer

Delta = ActualO * (1-ActualO) * Summation(Weight_from_current_to_next AND Delta_of_next)
Weight = Weight + LearningRate * Delta * Input

Figure 57: Back propagation formula in English

TargetO = T (The expected output set to recognize the digits)
ActualO = Output from the hidden/output layer during feed forward
Delta = Bias of Neuron
LearningRate = The rate at which the weights are being updated.

```
273          %------------output layer error------------
274 -        dvih=zeros(1,outNeu); % [1 10] matrix
275 -        for i=1:outNeu
276 -            dvih(i)=(T(I,i)-ovih(i))*ovih(i)*(1-ovih(i));
277 -        end
278          %------------hidden layer error------------
279 -        d=zeros(1,hidnNeu); % [1 50] matrix
280 -        for i=1:hidnNeu
281 -            dw=0;
282 -            for j=1:outNeu
283 -                dw=dw+dvih(j)*w2(i+1,j);
284 -            end
285 -            d(i)=o(i)*(1-o(i))*dw;
286
287 -        end
288          %-----Updating weight matrix for the output layer
289 -        for i=1:outNeu
290 -            w2(1,i)=w2(1,i)+learnRate*dvih(i)*1;
291 -        end
292
293 -        for i=1:outNeu
294 -            for j=1:hidnNeu
295 -                w2(j+1,i)=w2(j+1,i)+learnRate*dvih(i)*o(j);
296 -            end
297 -        end
298          %-----updating weight matrix for the hidden layer
299 -        for i=1:hidnNeu
300 -            w1(1,i)=w1(1,i)+learnRate*d(i)*1;
301 -        end
302
303 -        for i=1:hidnNeu
304 -            for j=1:inNeu
305 -                w1(j+1,i)=w1(j+1,i)+learnRate*d(i)*x(j);
306 -            end
307 -        end
308 -    end
309      % The total error for the neural network
310 -    Ep(I)=sum(dvih.^2);     % The error for the image
311 -    Epoch(e)=sqrt(sum(Ep)/N);   % mean-square error
312
313  %     Epoch
314 - end
```

Figure 58: Back propagation in MATLAB

The feed forward and back propagation is done 100 times (Epochs) and the total error is of the network is calculated using the squared root error function in line 311.

5.4.1.2 Character Recognition using Neural Network

Now that the neural network is trained and knows how to classify numbers: the segmented characters from the license plate can be fed to the neural network. First the bounding box values need to be sent to the `NN_Digit_Recognition` function where they will be used to crop out the actual character from the license plate image.

```
84      %-------------Recognition of digits-----------------
85      % Sending the outlined images to recognition
86 -    Number=-1;
87 -    if counter>0
88 -        for i=1:counter
89 -            [imgOutlined,Number(i)]=NN_Digit_Recognition(imgOutlined,bBox(i,:),handles);
90 -        end
91
92 -    elseif counter==0;
93 -        Number=-1;
94 -    end
```

Figure 59: Sending bounding box coordinates to recognition

`imgOutlined` is the license plate image converted into a 3D matrix where every slice is the same as `licenseBox` its used to make a colour image appear as grayscale because all 3 colour channels are identical. In the `NN_Digit_Recognition` function the `imgOutlined` is stored in a new variable `outImg` which will be used to highlight every character found in the license plate.

```
1   function [outImg,N]=NN_Digit_Recognition(inImg,bbox,handles)
2   % Auxiliary function that draws a specified bounding box in the image
3   outImg=inImg;
4   % Frame coordinates from bounding box
5   x1=bbox(:,1);       % Column
6   y1=bbox(:,2);       % Row
7   x2=x1+bbox(:,3);    % Width
8   y2=y1+bbox(:,4);    % Height
9
10
11  %------------Framing every number for display(unnecessary------------
12  outImg(y1:y2,x1:x2,2)=1;
13  outImg(y1+1:y2-1,x1+1:x2-1,2)=inImg(y1+1:y2-1,x1+1:x2-1,2);    % To frame
14  % Sending image to GUI
15  axes(handles.axes4);
16  imshow(outImg);
```

Figure 60: Highlighting characters in license plate image

`x1, y1, x2, y2` is the coordinates of the bounding box for the blob that is a character. These values are used to create an outline of bounding box in the the original license plate image. This is unnecessary but its used for demonstration purposes to show what blobs are being selected.

```
24  segImg=inImg(y1:y2-1,x1:x2);   % cropping image from license plate
25  % read a sample image
26  segImgA = segImg;
27  [row,col] = size(segImgA);
28  scale = [24/row 14/col];        % size to scale from
29
30  %# Initializations:
31  oldSize = size(segImgA);                  %# Get the size of your image
32  newSize = max(floor(scale.*oldSize(1:2)),1);  %# Compute the new image size
33
34  %# Compute an upsampled set of indices:
35  rowIndex = min(round(((1:newSize(1))-0.5)./scale(1)+0.5),oldSize(1));
36  colIndex = min(round(((1:newSize(2))-0.5)./scale(2)+0.5),oldSize(2));
37
38  %# Index old image to get new image:
39
40  outputImage = segImgA(rowIndex,colIndex,:);
```

Figure 61: Resizing cropped image blob

The coordinates are used to crop blob from the original bounding box and then resized to 24x14. This is done by setting a scale factor for the cropped image by taking its height and width. The resize is done by multiplying the row/column sizes by the scale factor and the rounding down the result to the nearest integer. Max is used in case and the rounded down result is less than 1 which would result in 1. Next the new row and column indices are calculated in `rowIndex` and `columnIndex`. First, a set of indices is computed with `1:newSize(n)` so that the width pixels span from 0-1 for the first pixel, 1-2 for the second pixel and so on. The coordinate of this pixel is treated as the centre, which is why 0.5 is subtracted and then the coordinates are divided by the scale size specified in line 28 and 0.5 added to the calculated centre coordinates of the centre pixels for the original image. These are then rounded to get the indices for the image. The min is used to make sure that the indices do not exceed the size of the original image. The last step is to apply the indexing to the original image.

The image is then converted to double precision so that it can be fed into the neural network. The image is fed into the neural network and the feed forward process starts by taking the matrix value of the segmented image and storing it in a vector. The hidden and output layer

output neurons are calculated the same way they were previously explained. Except this time the actual output neurons are going through a threshold to be classified.

```
%------------The output value of the output layer---
netvih=zeros(1,v);
ovih=zeros(1,v);
for i=1:v
    ow=0;
    for j=1:n
        % Connection weights
        % multiplying every column of o(Hidden layers) by every
        % row+1 of the w2 matrix to create an output layer connection weight
        ow=ow+(o(j)*w2(j+1,i));
    end
    % Pre-activation
    % The total net inputs of the output layer
    netvih(i)=(1*w2(1,i))+ow;
    % Logistic activation function: squahes the neurons
    % pre-activation between 0 and 1
    % Creating the sum output weight of the output layer by getting the
    % exponential of netvih. The activation function compressing
    % values in the range 0-1
    ovih(i)=1/(1+exp(-netvih(i)));
    % Classification using threshold
    if ovih(i)<0.1
        outR(i)=0;
    elseif ovih(i)>0.30
        outR(i)=1;
    else outR(i)=ovih(i);
    end
end
if outR==[1 0 0 0 0 0 0 0 0 0]
    N=0;
elseif outR==[0 1 0 0 0 0 0 0 0 0]
    N=1;
elseif outR==[0 0 1 0 0 0 0 0 0 0]
    N=2;
elseif outR==[0 0 0 1 0 0 0 0 0 0]
    N=3;
elseif outR==[0 0 0 0 1 0 0 0 0 0]
    N=4;
elseif outR==[0 0 0 0 0 1 0 0 0 0]
    N=5;
```

Figure 62: Feed forward neural network character classification

The same method of classification is used for letters. The result values are then sent to the main file where they are stored in a text log file and displayed in the user interface.

5.4.2 Storing Results

Once the result of the numbers and letters from the license plate have been extracted, they are stored in a text file log with the date and time it was captured. The numbers and letters first have to be put into the correct order of NN-LL-NNNN which is done in the main as shown in *Figure 63*.

```
147 -   [~, b] = size(Number);
148 -   [~, d] = size(Letter);
149     % Formatting and printing results to a text file.
150 -   if b>3 % if more than 3 characters in the license plate.
151 -       for num = 1:2 % writing first 2 numbers
152 -           fprintf(fileID,'%d',Number(num));
153 -       end
154 -       if d==2 % if there is two letters in the license plate
155 -           fprintf(fileID,'-%s-',Letter);
156 -           for num = 5:b % continue writing numbers from 5th position
157 -               fprintf(fileID,'%d',Number(num));
158 -           end
159 -           fprintf(fileID,'\t%s\t%s\t',timeString, dateString);
160 -       else %else if there is only one letter in the license plate
161 -           fprintf(fileID,'-%s-',Letter);
162 -           for num = 4:b % continue writing numbers from 4th position
163 -               fprintf(fileID,'%d',Number(num));
164 -           end
165             % add time and date.
166 -           fprintf(fileID,'\t%s\t%s\t',timeString, dateString);
167 -       end
168 -   else
169 -       errStr = 'Error'; % if no numbers/letters found. Give error with time and date
170 -       fprintf(fileID,'%s\t%s\t%s\t', errStr, timeString, dateString);
171
172 -   end
```

Figure 63: Formating result and writing to text file

Lines 147 and 148 are getting the size of the arrays of numbers and letters which is used in the formatting. If there is less than 3 numbers in the array `Number`, then that isn't a license plate and an error will be written to the file with the time and date. If there are greater than 3 values in the `Number`, then that is considered to be a license plate and the first two values are written to the text file `fileID`. `fileID` is a writable text file "CurrentPlate.txt". The first two values are written and then the letters are checked to see if there are one or two letters. If there are two letters in the array `Letter`, then they are written into the text file followed by the continuation of the number starting from position the fifth position or else if there is only one letter, the number continues from fourth position. The date and time is always added after a license plate number is written or if there was an error in the recognition of the license plate characters.

```
174     % Saving image of license plate
175 -   imgLname = sprintf('LicensePlates/%s_%s.jpg',timeString2,dateString); % naming license plate image
176 -   imwrite(licenseBox,imgLname);  % Saving license plate image
177
178     % outputting the text file values to the GUI
179 -   [regnum,readDate,readTime] = textread('Output/CurrentPlate.txt','%s %s %s');
180 -   set(handles.edit2,'String',regnum);
181 -   set(handles.edit3,'String',readDate);
182 -   set(handles.edit4,'String',readTime);
183 -   fclose(fileID);
```

Figure 64: Storing license plate image and displaying extracted license plate values in the interface

The license plate images are stored in the "LicensePlates" folder and the file name of the images is the time and date they were captured. The CurrentPlate.txt file is used only for storing the last recognition of a license plate and it is used to read the values from so that they can be shown in the interface. This is done by taking the 3 strings that are stored in it, which is the license plate number, time, date captured (`regnum, readDate, readTime`). These values are then sent to the 3 interface boxes labelled Registration, Time and Date as shown in *Figure 65*.

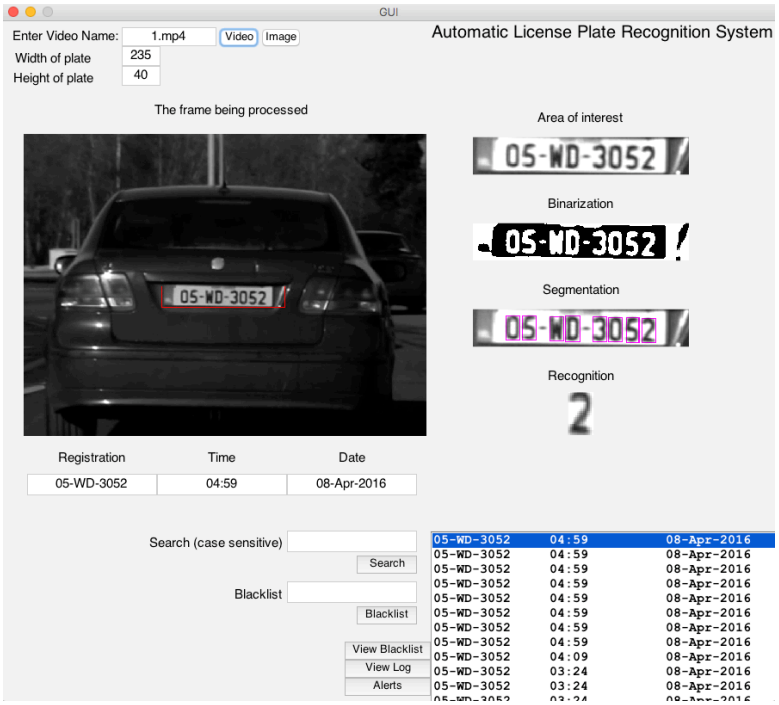

Figure 65: User interface

The values that are in the bottom right box is the log box of all the previous license plate numbers captured. These are the values that used to be in the CurrentPlate.txt file but were moved to the LicenseLog.txt file. The values are copied by opening the LicenseLog.txt file in append mode and the CurrentPlate.txt file in read mode. The first line of the CurrentPlate.txt file is read, which is the only line in that file and then it is stored appended to the LicenseLog.txt file. Once the log text file is updated, it needs to be reopened to update the log box in the interface. The log box has a specific format of string-tab-string-tab which is how the text is appended to the log file. Since the data being appended to the log file, that means that the most current data is at the bottom of the file. To fix this a function called `flipud` is print the data in the log file from bottom to top. The implementation of this is in *Figure 66.*

```
213    % Copying output values to log file
214 -  fInput = fopen('Output/LicenseLog.txt', 'A');
215 -  fileID = fopen('Output/CurrentPlate.txt','r+');
216 -  thisLine = fgetl(fileID);
217 -  fprintf(fInput,'\n%s',thisLine);
218 -  fclose(fInput);
219 -  fclose(fileID);
220    % updating the list in GUI with new values
221 -  fInput = fopen('Output/LicenseLog.txt','rt');
222 -  tScan = textscan(fInput, '%s','Delimiter','');
223
224    % formatting the read text to have a "tab" space between every string
225 -  newScan = cellfun(@(x)sprintf('%-15s', x{:}), regexp(tScan{1}, '\t+', 'split'), 'uni', 0);
226    % flipping the file to read bottom to top
227 -  newScan = flipud(newScan);
228
229 -  set(handles.listbox1,'String',newScan);
230 -  set(handles.listbox1,'FontName','FixedWidth');
231 -  fclose(fInput);
```

Figure 66: Copying data to log and displaying log data to interface

The last implementation of the ALPR system that is worth noting, is the blacklisting feature. This feature is used to insert a license plate number into the blacklist that is checked every time a license plate is read. *Figure 67* shows the implementation of the blacklisting functionality.

```
% Searching extracted license plate number against blacklisted vehicles
regnum = char(regnum); % last license plate
sSearch = fopen('Output/BlackList.txt', 'r'); % opening blacklists
tScan = textscan(sSearch, '%s','Delimiter',''); % scanning text
newScan = cellfun(@(x)sprintf('%-15s', x{:}), regexp(tScan{1}, '\t+', 'split'), 'uni', 0); % formatting/converting text to string
licenseP = ~cellfun(@isempty, strfind(newScan, regnum));    % searching text
blackisted = max(licenseP); % searching for max value. if 1 then something is found. if 0 then nothing is found
% Alert box
if blackisted >=1
    % writing number and time of alert.
    fileID = fopen('Output/BlackListAlerts.txt', 'A');
    fprintf(fileID,'\n%s\t%s\t%s',regnum,timeString2,dateString);
    fclose(fileID);
    bBoxmsg = sprintf('Blacklist ALERT: %s',regnum);
    BlackBox = questdlg(bBoxmsg, ...
        'BLACKLISTED', ...
        'Ok','Ok');
end
output = flipud(vertcat(newScan{licenseP}));
set(handles.listbox1,'String',output);
fclose(sSearch);
```

Figure 67: Blacklisting functionality implementation

regnum is the license plate number that was previously extracted from the CurrentLog.txt file. It is a string that is searched for in the BlackList.txt file by first scanning the blacklist file using `textscan` and storing every string a cell `tScan`. The cells `tScan` are then formatted to strings in `newScan` and then `regnum` is searched in newScan which will return an array of rows with a value of 0 if `regnum` is not found in that row or 1 if `regnum` is found in a row. The `max` function is used to see if there is a 1 in any of the rows `license.` If there is a 1 in any of the rows, a warning message will pop up and details of the occurrence are recorded the BlackListAlerts.txt.

5.5 Conclusion

The implementation chapter covered all the main functionalities in the ALPR system that was developed. The overview of how the license plate region is being extracted using Harris corner detector was covered, the interface functionalities, how character segmentation was performed on the extracted license plate image was also explained. An explanation and implementation of the neural network classifier of characters, which was one of the hardest parts of the ALPR system. Storing results of the ALPR in a text file and manipulating the data to display is a readable format and the final section that explained how blacklisting is done.

The next chapter will cover the different types of software testing and the tests that have been done to the APLR system to improve its performance.

Chapter 6. Testing and Evaluation

6.1 Introduction

This chapter will cover testing and the evaluation techniques that could be utilised during the development of the ALPR system. The first section will give and overview of what software testing is followed by what white box and black box testing is. White box and black box will be explained and how they are used during tests. The second section will cover how black box testing was used to test the neural network and how white box tests were carried out on the ALPR system.

6.2 What is Software Testing?

Software testing is the technical task that ensures if a system meets its end user and system requirements. A series of tests are done in relation to the functional requirements that need to be ensures that they provide the necessary functionality to have function system and non-functional requirements to ensure that the performance of the system is efficient and secure. In an ideal world, a software tester would be able to perform tests on every aspect of a system to find any errors that might occur. But this usually isn't possible because even a small program can have multiple input and out combinations. It would be very time consuming and resourceful to create test cases for every aspect of a system [23]. That is why for this project mostly the important aspects of the system will be tested.

6.2.1 White Box Testing

White-box testing is a software testing technique used by software testers to examine the back end structure and derive test data from the logic (source code) of a program [23]. This requires a thorough understanding of workings of the system being tested. This is the main method used to ALRP system as at a unit level as each function was developed and tested. The main purpose of white box testing is to [24]:

- Identify if the logic paths are broken or poorly structured in the execution of a program
- The flow of specific inputs
- The expected output
- The functionality of conditional loops
- Testing of each statement, object and function on an individual basis

Testing requires a series of predefined inputs with corresponding outputs that will be expected from the predefined inputs. If the expected output is not the desired output, then that could mean that there is a bug in the program. White box unit testing was used in the development of the ALPR system to try and catch bugs early in the development to have an advantage of being able fix them before moving on to the next step in the development phase.

6.2.2 Black Box Testing

Black box testing is the testing of the functionality of software to test its behavior. Black box testing can be done by individuals other than the software developer [25] and without prior knowledge of the logic of system. The aim of black box testing to to ensure that the

functional requirements of the system have been met by the developer. The steps needed to carry out black box testing are:

- The initial requirements are examined
- Input valid values to get expected result
- Input invalid values to see what happens
- Create test cases to determine the expected result for different input
- Compare actual result with the expected.

Black box testing was utilized in this project by using test cases for different angle and lighting of images with vehicles to see if the ALPR will be able to locate the license plate in the image and recognize the license plate characters.

6.3 Testing the ALPR

This section will contain the test cases that were used to test the ALPR system in this project. There will be four main test cases that will the test the importing of video/image and selecting a frame from a video, locating the license plate under different circumstances, segmentation of characters, character recognition and the functionalities of the storing and accessing the extracted license plate files.

6.3.1 Importing Video/Image to ALPR Test Case

The following test is done to see if the ALPR know if a video is a license valid video/image source.

Case	Instruction	Expected result	Pass/Fail
1	Insert correct file name of a video/image with its extension.	Video/Image is imported and license plate locating begins.	Pass
2	Insert incorrect file name of a video/image with and without an extension.	A pop up message box telling the user to enter correct file name	Pass
3	Insert a none video/image file	A pop up message box telling the user to enter correct file name	Pass

6.3.2 Locating License Plate Test Case

The following test is checking to see if the ALPR is able to locate license plates from different angle or lighting environments

Case	Instruction	Expected result	Pass/Fail
1	Import image with a vehicle and visible license plate	License plated located	Pass
2	Import image with with no license plate	Stop further exaction and print out error in the log file	Pass
3	Import image with two license plates	Locate the license plate with the highest amount corner count or first position in the array of sum corner points	Pass

4	Import dark image with low visibility	Locate license plate	Fail
5	Import license at rotated 45 degrees	Locate license plate	Fail

6.3.3 Segmentation Test Case

These tests are checking if certain objects are being filtered out from the license plate and only the characters are being selected

Case	Instruction	Expected result	Pass/Fail
1	Segment characters from clear license plate image	Segmented characters	Pass
2	Segment characters from noisy license plate image	Segmented characters	Pass
3	Segment characters from dark license plate image	Segmented characters	Fail
4	If no characters are segmented	Notify user with error message	Pass

6.3.4 Character Recognition Test

Once objects are segmented from the license plate, they are checked to see if they are characters using a fed forward neural network. This test was done after training the neural network. The diagram in *Figure 68* outlines the root mean square error rate of every character in every epoch, which one full training cycle. This neural network had 100 training cycles and the error rate is very rapidly decreasing with every cycle after the 15th training cycle.

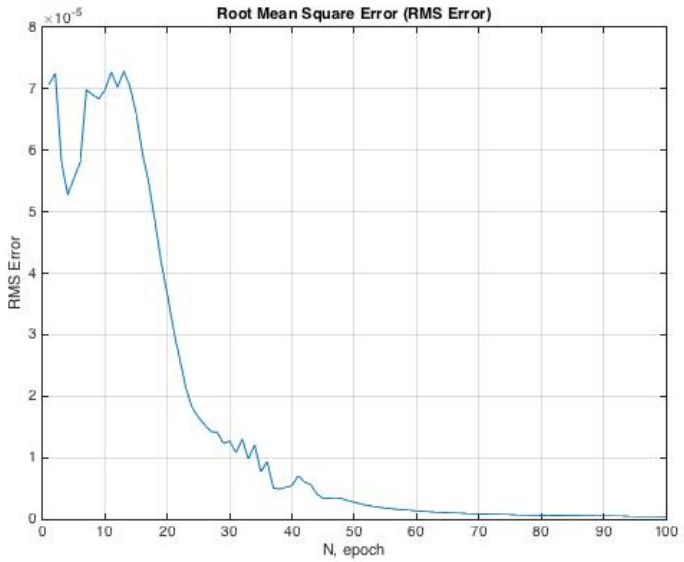

Figure 68: Root mean square error of Neural Network

The neural network was then tested against 100 images of every digit and the result was 99.95% success rate of recognizing the digits. This error rate is different for every training set of images.

6.3.5 ALPR Interface Functionalities

The interface functionalities such as the ability to search previous license plate, blacklisting registration numbers and view different tables.

Case	Instruction	Expected result	Pass/Fail
1	Searching the log file of license plates using license plate number	Display the times and dates that license plate number has been captured if it is in the log.	Pass
2	Searching the log file of license plates using time and date	Display all the license plates that have been captured during that time.	Pass
3	Inserting license plate number into blacklist log	Increment the blacklist log with the new blacklisted vehicle number	Pass
4	Alert the user if a license plate number that was blacklisted is captured	Notify user with a blacklist alert message and store the license plate number, time and date the user was alerted	Pass
5	Click the View Blacklist button	Displays all the blacklisted number	Pass
6	Click the View Log button	Displays all the captured license plate numbers	Pass
7	Click the Alerts button	Display the alert time and date of the alerted license plate numbers	Pass

6.4 Conclusion

The many test cases that were used to test the ALPR were discussed in this chapter. The testing was beneficial to the project to find out what errors are in the ALPR and how they are caused. The main problem with the ALPR is that it is bad at recognizing license plates in the darkness and it doesn't always get the full license plate image when the license plate is not straight and at a big inclined angle. The detection of the license plate in darkness is extremely hard without an infrared camera that real world ALPR use.

The next chapter will be an analysis and review of the project and how it changed from the initial proposal and an explanation of why the changes where made.

Chapter 7. Conclusion

7.1 Introduction

This chapter will compare the final project result against the project goals that were outlined. An explanation of why some features where not implemented from the actual proposal of the project.

7.2 Initial Objective

The initial proposal for this project was to create a network of ALRP systems that will record the license plate numbers of vehicles and the locations they were last seen to predict where the vehicle could be within an area on a map. The project was to include objects such as using a real time video source and making the system work in real time, which was a little bit too ambitious to achieve with no prior knowledge of image processing and the time given to develop the system. But none the less, the image processing skills during the process of researching similar systems and the algorithms that code be used to create an ALPR.

During the time given, the initial objectives change to just creating an ALPR system that would get a high rate of license plate recognition, having the data extracted stored and creating blacklist notifications to the user which was achieved.

7.3 The System

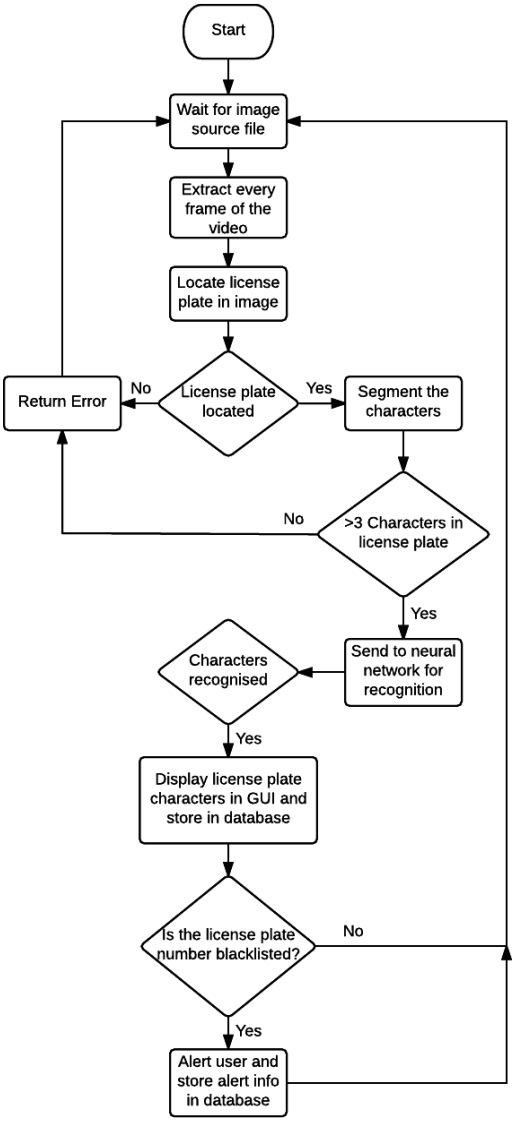

Figure 69: ALPR system process diagram

The flowchart in *Figure 69* represents how the system is working when an image source is added to the ALPR to process. The process of the ALPR starts when an image or video is imported into the ALPR. This diagram is working with a video input source because the first thing that its doing after an input is extracting the frames of the video. The frames are processed individually. Each frame is searched for a license plate and if one is found, the segmentation process starts or else an error is returned. The segmentation function is looking for connected objects that could be a character and once it has more that 3 (because Irish license plates do not have less than 4 characters in the license plate.) characters, these characters are sent to the neural network for recognition. And the result is sent back to the main function where it if formatted in the format NN-LL-NN (N for number, L for letter). Once formatted, it is stored in a database with previously extracted license plate numbers and displayed in the user interface. The result is check against a database of blacklisted license plate numbers and a notification will pop up if a match is found. Once the system is finished processing, it then waits for another input source.

7.4 Final Notes

Having no prior knowledge of image processing algorithms and usage of tools, it is very difficult to build an ALPR system. But the challenge is enjoyable when the right research is being done and the time is being spent working on the project. A lot of time was consumed researching and developing way of extracting the license plate as well as learning image processing. It is very stressful to start working on something you have never done before and learning a new language, but it was worth doing because it teaches many concepts that are required in a work environment such as:

- Researching new information
- Time management.
- Keeping up and matching deadlines.
- Challenge of working on something new
- Deciding whether to implement something new or improving what is already done

Bibliography

[1] David J. Roberts & Meghann Casanova, "Automated License Plate Recognition (ALPR) Systems: Policy and Operational Guidance for Law Enforcement," Department of Justice, National Institute of Justice, Washington DC, 2012. [Online] [Accessed 03 March 2016]. http://www.theiacp.org/Portals/0/pdfs/IACP_ALPR_Policy_Operational_Guidance.pdf

[2] Wikipedia contributors. (2016, January) Wikipedia, The Free Encyclopedia. [Online] [Accessed 03 March 2016]. https://en.wikipedia.org/w/index.php?title=Vehicle_registration_plates_of_the_Republic_of_Ireland&oldid=701308048

[3] Mobile License Plate Recognition Systems Assessment Report. (2008) Mobile License Plate Recognition Systems Assessment Report. [Online] [Accessed 05 March 2016]. https://www.aclu.org/files/FilesPDFs/ALPR/arizona/alprpra_PhoenixPD_PhoenixAZ_4.pdf

[4] TD Duan, "Building an automatic vehicle license plate recognition system.," Intl. Conf. in Computer Science, Can Tho, 2005.

[5] TEC Security. [Online] [Accessed 05 March 2016]. http://www.tecsecurity.ie/js/tinymce/plugins/filemanager/files/ezCCTV-ANPR.pdf

[6] IPConfigure Inc. (2015) IPConfigure. [Online] [Accessed 05 March 2016]. http://www.ipconfigure.com/products/lpr

[7] An Garda Síochána. Automatic Number Plate Recognition. [Online] [Accessed 05 March 2016]. http://garda.ie/Controller.aspx?Page=106&Lang=1

[8] License Plate Recognition Algorithms and Technology. [Online] [Accessed 06 March 2015]. http://www.platerecognition.info/1102.htm

[9] K.M. Sajjad, "Automatic License Plate Recognition using Python and OpenCV," Department of Computer Science and Engineering, M.E.S. College of Engineering, Kuttippuram,.

[10] Chris Harris & Mike Stephens, "A COMBINED CORNER AND EDGE DETECTOR," Plessey Research Roke Manor, United Kingdom, Research 1988.

[11] "What is MATLAB," Cooperative Institute for Meteorological Satellite Studies,.

[12] Shermal Fernando. What is OpenCV. [Online] [Accessed 29 Nov 2015]. http://opencv-srf.blogspot.ie/2010/09/what-is-opencv.html

[13] Neha Chopra, Vaishali Gupta Pratiksha Jain, "Automatic License Plate Recognition using OpenCV," *International Journal of Computer Applications Technology and Research*, vol. 3, no. 12, pp. 756-761, 2014.

[14] Ciarán Hegarty, "Fantasy Football System," Dublin Institute of Technology, Dublin, BA Thesis 2014.

[15] Vivienne Trulock. (2008) Paper Prototyping. [Online] [Accessed 13 March 2016]. http://hci.ilikecake.ie/eval_paperprototyping.htm

[16] M.Deepa S.Namitha, "Automatic License Plate Recognition For Ambigious Character Using Template Matching With Fuzzy Classifiers," *International Journal of Emerging Technology and Advanced Engineering*, vol. 4, no. 3, Febuary 2014.

[17] Hemanth Kumar Mantri. (2013, May) Quora. [Online] [Accessed 24 March 2016]. https://www.quora.com/How-do-you-explain-back-propagation-algorithm-to-a-beginner-in-neural-network

[18] Matt Mazur. (2015, March) A Step by Step Backpropagation Example. [Online] [Accessed 24 March 2016]. http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[19] Michael Nielson. (2016, January) Neural Networks and Deep Learning. [Online] [Accessed 24March 2016]. http://neuralnetworksanddeeplearning.com/chap1.html

[20] Alistair Cockburn, *Agile Software Development: The Cooperative Game* , 2nd ed.: Addison-Wesley Professional, 2006.

[21] ISTQB CERTIFICATION. Spiral Model Advantages and Disadvantages. [Online] [Accessed 06 Dec 2015]. http://istqbexamcertification.com/what-is-spiral-model-advantages-disadvantages-and-when-to-use-it/

[22] Scott W. Ambler. (2009) The Agile Unified Process (AUP). [Online] [Accessed 06 Dec 2015]. http://www.ambysoft.com/unifiedprocess/agileUP.html

[23] Corey Sandler, Tom Badgett Glenford J. Myers, *THE ART OF SOFTWARE TESTING*. Hoboken, New Jersey, United States of America: John Wiley & Sons, Inc, 2012.

[24] (2014) Guru99. [Online] [Accessed 08 April 2016]. http://www.guru99.com/white-box-testing.html

[25] L Williams. (2006) Testing Overview and Black-Box Testing Techniques.

[26] Daniel Lélis Baggio, *Mastering OpenCV with practical computer vision projects*.: Packt Publishing Ltd, 2012.