License Plate Recognition at Close Distances

Report for BTech 451

by

Muhammad Saad Malik

Supervisor: Professor Reinhard Klette

Tamaki Campus
Department of Computer Science
The University of Auckland
New Zealand
October 2014

I dedicate all my hard work and success to my beautiful mum and little sister.

Abstract

This is a report for my final year BTech project. The project given to me is to do with developing a license plate recognition system. This report covers the research, methods, implementation and design of a license plate recognition system. It discusses 3 algorithms that were implemented during this project. The algorithms achieve respectable success rates and run times. The report also presents experimentational results. Lastly, the report proposes a different approach to solving the license plate recognition problem.

Keywords: License plate, number plate, localisation, detection, segmentation, character recognition, recognition system.

Acknowledgments

I will firstly thank my supervisor Professor Reinhard Klette. I could not have done this project without his help. I thank Terry Sun, my industry mentor for giving me this project. I thank Dr. Mano for his support throughout this project and throughout my BTech degree. I also thank my beautiful partner Shagoofa Tahir for her help and support throughout this year. Last but never least I will thank almighty God, as without him none of this would be possible.

Muhammad Saad Malik
Auckland
October 28, 2014

Contents

“

Chapter 1

Introduction

This introductory chapter informs about the motivation and goal of this project. This chapter will also inform about the test data to be used for the working prototype. The last section of this chapter will give an overview on the structure of this report.

1.1 Motivation and Goal

This report covers the year long project given to me by SnapInspect. SnapInspect is a business based in Auckland, New Zealand. SnapInspect focus on mobile application development and are well known for their property inspection app.

Project Description. Automatically identify vehicles by reading their number plates. This is a useful application for the automotive and transport industry. The goal of this project is to produce an application which is able to recognize license plates at a close distance.

Problem. The company has a small parking lot where the parking spaces have been allocated for staff and visitors. The current problem is that some visitors park their cars for more than the time allowed. These parking violations are reported to visitors by staff. The staff currently monitors the parked cars manually without following any fixed procedure. This method however, does not always work correctly and requires unnecessary time and effort.

Solution. To address this problem the company has proposed a solution of an automatic license plate recognition system. The system is aimed at reducing the time and effort required to inform a visitor about their parking violation. The system will handle and make decisions on whether a car has been parked for too long or not. I have been given the task of designing and implementing such a system to solve the company's problem.

Requirements. SnapInspect's main concern for this project is the budget. Due to this fact SnapInspect is not making use of any of the other available license plate

Figure 1.1: Sample from project scenario, provided by SnapInspect.

recognition systems. The company has requested the system to be developed for an android based device. An android device is much cheaper in comparison to the other license plate recognition systems. There are also some technical requirements such as the system's performance and success rate. The performance of the system is concerned with the runtime of the algorithm and the success rate is based on the percentage of correctly recognized license plates. SnapInspect has not given any strict guidelines for both these requirements but they are expecting a system that will work smoothly.

How it Works. The system will work based on a given scenario. For this project, the scenario is the parking lot (See Fig. 1.1). The system will simply recognize license plates in the parking lot and store the results into a database. Initially, the plan is to place or mount an android device in the parking lot where it can recieve a good view of the parked cars. The camera of the android device will provide live footage from which license plates will be recognized. It is suggested that the algorithm runs at least every 5 minutes in order to keep track of the activity in the parking lot. Results in the database will be compared to previous results every time the algorithm is executed. The comparison allows to check how long a car has been parked for. If a license plate is recognized (i.e. in every execution of the algorithm) for a period longer than the permitted parking time, the system will know the car has exceeded

the parking time limit. The next step will be to notify the staff about the parking violation. This may be done by sending a simple message to the staff or through some other similar method. Once the notification phase is complete the system will return to executing the recognition algorithm.

Technical Details. The most important part of this project is the license plate recognition algorithms. Other parts such as android application development are not as vital for the success of this project (i.e. the android application can always be completed later in the near future). Therefore, this report will heavily consist of technical material concerning license plate recognition methods. A license plate recognition system is made up of several different parts. Hence, before heading into the research and design of a license plate recognition system, in the following I will outline the primary steps which are required to be taken for the overall system to work.

Step 1. Capture a frame (image) from the live video feed of the android device.

Step 2. Pre-process image.

Step 3. Apply license plate localisation algorithm.

Step 4. Apply license plate tilt correction and segmentation algorithm.

Step 5. Apply optical character recognition algorithm.

Step 6. Save results into database.

Step 7. Repeat process.

1.2 Chosen Test Data

The test data to be used in this project will be provided by SnapInspect. Apart from that test data I will personally be taking photos of parked cars in similar situations to get a start on the project.

1.3 Structure of Report

The report is structured in order of the steps described in section 1.1. Chapter 2 discusses license plate detection and segmentation algorithms. Chapter 3 goes over the experimentations and results. Chapter 4 discusses optical character recognition and its implementation and results. Chapter 5 discusses the identified limitations and the last chapter, chapter 6 talks about the android application design.

Chapter 2

Localisation Basics

This chapter briefly recalls some basics of image processing and analysis as used in this report, it reviews the license plate detection and segmentation (i.e. into characters) subjects, and outlines the approach followed in this project.

2.1 Image Processing Basics

In this section I will briefly recall basics of image processing as of relevance for my project.

Histogram. A histogram of an image shows tabulated frequencies of image pixel values. See Fig. 2.1 for an example. A histogram gives useful information about the tonal distribution in an image. This information can be used to adjust an image accordingly. Contrast and exposure adjustment are some examples of using a histogram. In computer vision a histogram is a compelling tool for image analysis. The tabulated frequencies can be studied in several ways to help with certain problems. For example, a histogram can be studied for pattern recognition and image segmentation. For pattern recognition a histogram helps because patterns can be easily identified in the tabulated frequencies. For image segmentation an image can be segmented by finding high or low areas of contrast through the image histogram. A histogram has several other applications and in section 2.2 we will see a few of them in use.

Horizontal and Vertical Projection Histograms. The horizontal and vertical projections are simple histograms concerned with calculating the number of pixels (of some intensity) in each row and column of an image.This technique is useful for image segmentation. Please note that horizontal and vertical projections are different to normal image histograms.

Connected Component Analysis (CCA). CCA also known as blob extraction is a very useful and an essential technique used to find objects in an image. The method

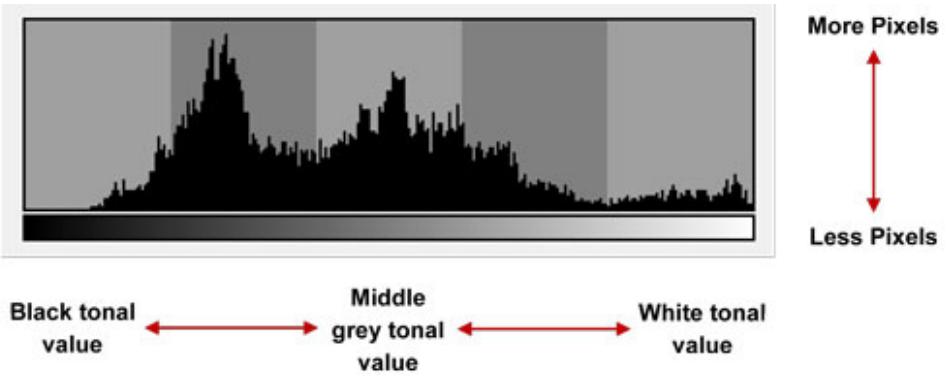

Figure 2.1: Example of an image histogram [2].

simply scans an image looking for regions of pixels that have similar intensities [1]. The detected regions are known as connected components or blobs. The blobs have several useful properties such as area, shape and location. Due to these properties the blobs can be processed and recognised as a certain object and then be easily extracted out of the image. However, the downside of CCA is that it can not detect specific objects on its own. This is because CCA is only able to detect objects that have some form of pixel connectivity (such as similar pixel intensities). Therefore, using the CCA method requires major pre-processing as well as post-processing to be able to recognise and extract objects. In the case of license plate detection, CCA is a popular technique. This is because a license plate forms into a simple rectilinear shape which CCA can detect easily if pre-processing is done correctly. In the next section we will see examples of CCA in use.

Kernel. A kernel is typically a small matrix of numbers. It is used in image convolutions [3]. An image convolution is a simple process of multiplying two matrices and producing an output matrix. The two matrices are the input image and the kernel. The kernel can be used to have different set of values to produce different effects for the input image. Kernels are fundemental in image processing and several filters and operators make use of them[3]. See Fig. 2.3 for an example of a kernel.

Basic Edge Detection. Edges are important features in an image, which are used for image analysis purposes. Edge detection applies mathematical methods to identify pixels at brightness discontinuities. Such pixels define edges. Edge detection is an example of feature detection. There are a number of edge detectors in image processing but I will only briefly explain the Sobel operator and the Canny edge

Figure 2.2: *Left:* Grey-level input image. *Right:* Result after applying the Sobel operator.

detector.

Sobel Operator. The *Sobel operator* is a discrete differentiation operator [4]. It detects edges by finding the approximate absolute gradient magnitude at each point of an input grayscale image. The sobel operator is applied by using the two kernels shown in Fig. 2.3. The two kernels are applied seperately at each pixel and their results are combined to compute the approximate absolute gradient magnitude. Applying the kernels seperately also means that seperate gradient measurements for vertical and horizontal edges can also be produced [5].

The value of the Sobel operator at pixel location (x, y) is simply a first-order partial derivative calculation, and therefore equals [6]:

$$S(x, y) = |S_x(x, y)| + |S_y(x, y)| \tag{2.1}$$

Canny Operator. The *Canny operator* detects edges and produces thin edge segments as the final output. The thin edge segments only have a width of one pixel [7]. The operator goes through multiple stages to process an image. An image is firstly converted into grayscale, then it is smoothed by Gaussian smoothing. An operator such as *Roberts Cross* or *Sobel* is then applied to the image to provide estimates of the edges. The next step known as *non-maxima suppression* tracks along the ridges of the estimated edges and thins them by setting pixels that are not part of the ridge to 0. The tracking process makes use of two thresholds T_1 and T_2 where T_1 is greater than T_2. The tracking begins on a ridge at a pixel with a value greater than T_1. Tracking stops when a pixel with a value less than T_2 is found.This process applies hysteresis with the use of the two thresholds. This means the outcome of a pixel is not only de-

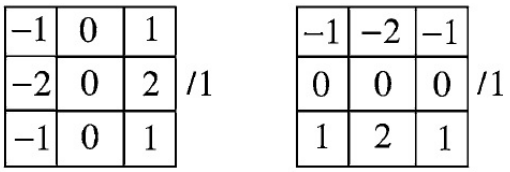

Figure 2.3: Kernels of the Sobel operator [6]. *Left:* S_x . *Right:* S_y.

cided by the previous tracked pixel value, but pixels before that are also considered [7] [8].

Hough Line Transform. *Hough line transform* is a technique which is used to detect straight lines. The *standard Hough line transform* expresses lines in the polar system. These lines are given by equation 2.2.

$$r = x \cos\theta + y \sin\theta \tag{2.2}$$

In the equation 2.2 r is the length (distance to origin) of the line in the interval [0, r_{max}] with r_{max} as given in 2.3 [9]. The θ is the angle of the line with respect to the x-axis, in the interval [0, 2π). Using some point from an image with the equation 2.2 produces sinusoid curves. Lines can be detected when the curves for all points in an image are produced and analyzed for intersections. This is because the intersections between the curves suggest the points of the possible lines. Therefore, in general standard Hough line transform detects lines by counting the number of intersections in the curves produced by each point in the image[9] [10] [11].

$$r_{max} = \sqrt{N_{cols}^2 + N_{rows}^2} \tag{2.3}$$

Spatial Domain. The *spatial domain* is the normal image space. If an image is projected to some scene and changes are made to the image, the changes will directly project to the scene. This means that distances in the image correspond to real distances in the scene [12].

Discrete Fourier Transform. A *Fourier transform* decomposes an image into sine and cosine components. It is used to convert images in the spatial domain into the *Fourier domain*. Hence, the Fourier domain consists of the output transformed image. Points in the Fourier domain image represent certain frequencies in the spatial domain image. A *discrete Fourier transform* is a sampled Fourier transform. Therefore, not all frequencies from the spatial domain image are taken for the transforma-

tion. The sample taken is just large enough to represent the entire spatial domain image. This type of transform has several applications in image processing. One main reason for using this transform is to be able to study the geometric characteristics of a spatial domain image [13].

Discrete Wavelet Transform. A *discrete wavelet transform* is an alternative to the Fourier transform.

Gaussian Smoothing Operator. *Gaussian smoothing operator* is an essential and widely used operator in image processing. It is used to blur images. The purpose of blurring is to reduce extra details and noise. The Gaussian smoothing operator works by applying a special kernel filter. This kernel is defined by the samples of the 2D *Gauss function* which is the product of two 1D Gauss functions. The Gauss function is defined as in equation 2.4.

$$G_{\sigma,\mu_x,\mu_y}(x,y) = \frac{1}{2\pi\sigma^2} \times e^{\frac{(x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}} \tag{2.4}$$

Where σ is the standard deviation and μ_x and μ_y are the expected values for x and y [14].

Structuring Element. A structuring element is very similar to a kernel. It is basically a matrix of binary values. The difference between a structuring element and a kernel is that structuring elements are used for morphological operations, not for image convolution. The structuring element has the properties of size, shape and origin. The size of the structuring element is given by the dimension of the matrix, the shape is given by the pattern of ones and zeros in the matrix and the origin is usually a pixel within the matrix such as the center pixel [15]. When the structuring element is applied to an image it compares its pixel values with the corresponding pixel values of the image. The result depends on the type of operation being applied. A structuring element is said to be *fitting* when all its values are exactly the same as the underlying image pixels. It is said to be *hitting* if at least one of its values are the same [15].

Morphological Operations. Morphological operations are used to process an image based on its shapes or morphology of its features. The operations are especially suited to work with binary images. All morphological operations are applied by using a structuring element. Following will be an overview of some of the types of morphological operations.

Erosion. Erosion is a very basic morphological operation that erodes away the foreground pixels. The effect of erosion shrinks or reduces the regions (i.e. objects) in the foreground. This means the background grows, holes and gaps between re-

gions become larger. Small details and noise is also eliminated. In erosion, for a pixel to be eroded the structuring element must fit the underlying image pixels.

Dilation. Dilation is another very basic morphological operation which works completely opposite to erosion. Dilation increases the foreground pixels. Hence, the effect of dilation grows or expands the regions (i.e. objects) in the foreground. The background reduces, holes and gaps between regions become smaller. In dilation for a pixel to be dilated the structuring element only has to hit the underlying image pixels [15].

Opening. The opening operation is a composite function. It is simply an erosion operation followed by a dilation operation. It helps open up gaps between regions by eliminating thin pixel connections [15].

Closing. Similar to the opening operation, the closing operation is also a composite function. Closing however is a dilation operation followed by an erosion operation. It helps close up gaps between regions by adding pixels in small gaps [15].

Thinning. Thinning in image processing is a morphological operation that simplifies images by producing very thin curves of the input image. The operation works on binary images and produces binary images as output. The thinning operation is also applied to an image by using a structuring element.

Thresholding. In image processing the simplest method of image segmentation is thresholding. Thresholding on grayscale images produces binary images. There are several methods to thresholding an image, here I will only give an overview of basic binary thresholding and also Otsu's thresholding performed on grayscale images.

Binary thresholding. Sets the pixels to zero if the value of the current pixel is less than a given threshold. If the value of the current pixel is greater than the selected threshold the pixel's value is set to a maximum value such as 255.

Otsu's Binarization. Otsu binarization assumes the image consists of two classes of pixels, the foreground (object) and the background. Otsu's method calculates the optimal threshold of the image by separating the two classes of pixels. The selected threshold maximizes the intra-class variance [16]. The method works by the formula shown in equation 2.5, where σ_b^2 is the intra-class variance, P_1 and P_2 are class probabilities and μ_1 and μ_2 are the means of the object and background classes respectively [16]. See Fig. 2.4.

$$\sigma_b^2 = P_1 * P_2(\mu_1 - \mu_2)^2 \tag{2.5}$$

Figure 2.4: Example of Otsu binarization. *Left:* Original image [25]. *Right:* Result of Otsu's binarization.

2.2 Methods for License Plate Detection

The first step in the license plate reading system is license plate detection. In this section I will give an overview of different methods of license plate detection.

Method 1. [17] employs two different methods using CCA to detect license plates. The first method works by using prior knowledge of license plates. Detecting of a white frame is carried out by using CCA and some candidate frames are collected. Then a few steps are taken to validate if a certain region is a license plate. The second method is used when the first method is not able to detect a license plate. The second method works by searching for large numerals or characters in the image seqeunce. This search is also carried out by using CCA. Candidate regions are collected and are classified as characters based on prior knowledge. This includes the properties of the characters such as their average area and average distance between each character on a license plate. The success rate of this algorithm reported in [17] is 97.16%.

Method 2. The approach taken by [18] to detect license plates involves thresholding and analysis of black to white pixel transitions. [18]'s method firstly applies a threshold of 170 to the input image. This threshold value is predetermined from experimentations with their particular scenario. Then some additional thresholding is again performed to remove further unwanted data. This additional thresholding is done through an analysis of black to white pixel transitions. The focus is to count the number of transitions in each row of the remaining data. If the count is less than 14 for any row, the pixels in the row are all set to 0 (i.e. black). The count of 14

transitions is calculated through the assumption that each license plate has at least 7 characters. The additional thresholding results in the license plate being somewhere in between completely black rows. To complete the localisation the remaining data is searched for the longest vertical line of white pixels. This means that when searching the image from left to right, the first longest vertical line of white pixels is the left edge of the license plate. Similarly the last longest vertical line of white pixels is the right edge of the license plate. By finding the vertical lines there is enough information gathered to extract out the license plate. Some checks regarding the aspect ratio are also performed for more accuracy.

Method 3. [19] detects license plates by studying the variance of the input image. A few pre-processing steps are taken to reduce noise and increase contrast before localisation. Sobel edge detection is performed on the image and the image is split into 25 blocks each of the same size. The size ofcourse is large enough to fit a license plate. Variance in each block is calculated and a threshold variance value is obtained by the simple formula $V_{th} = (V_{max} + V_{min})/2$, where V is variance. Each block is then compared with the obtained threshold variance value. Any block below the threshold variance is removed. Hough line transform is performed on the remaining blocks to remove any unnesscary long lines. The last step consists of some morphological operations and CCA to extract out the license plate from the remaining blocks.

Method 4. In [20] the approach to detecting license plates is mainly concerned with the sobel filter and CCA. After some pre-processing of the input image a vertical sobel filter is applied to the image. The gradient of the filtered image is obtained for the purpose of validating the possible license plate regions. The filtered image is then thresholded using Otsu's binarization and analyzed using CCA. Hence the candidate regions are extracted and validated through the formula $S_G = w_1 \times S_{Grad} + w_2 \times S_{Ratio} + w_3 \times S_{Shape} + w_4 \times S_{GTrack}$. Where S_G is the sum of priors including the gradient, aspect ratio, shape and tracking criterion for the candidate. The candidate which has the best value of S_G is processed and the rest are stored for future considerations.

Method 5. [21] uses discrete wavelet transform with a sliding window to detect license plates. Firstly, a grayscale image is transformed by using a 1-level 5/3 discrete wavelet transform (DWT). The HL and LH subbands are used to locate the license plate as they provide vertical and horizontal information about the image. Using the HL subband a histogram of heights of all vertical lines is calculated. This is followed by calculating the average of the heights. Then noise in the HL subband is reduced by removing any vertical lines that are too long or below the average line height. The next step reduces noise in the LH subband. This is done by calculating

the average of the coefficients in the LH subband and then removing all coefficients below this average. Then a transition analysis is carried out in the HL subband to give an approximate location of the license plate. Along with this step the LH subband is also scanned for horizontal lines to provide a more accurate approximation. These steps result in a detected candidate region. Since the candidate region is only an approximation further processing is required. A block around the candidate region is created to be sure that the entire license plate is inside this area (i.e. the block's area). Then to finalize the detection a sliding window is applied within the block. The sliding window is created to have an average size of a license plate. The last step of this process is to convert the HL frequency domain back to the spatial domain. [21] has reported this method to have a success rate of 97.33%. The success rate is obtained from testing the algorithm on 292 images.

2.3 Methods for Segmenting License Plates

After a license plate has been successfully detected, the next step is segmentation. Segmentation is performed to seperate and extract each character out from the license plate. This step is crucial because character recognition algorithms depend on the quality of license plate segmentation. The segmentation process can be challenging in most cases. This is because most real world situations result in license plate images that are either tilted or out of perspective. The different types of illumination and noise also needs to be considered. Therefore, for license plates to be segmented correctly the process must first deal with these problems. Following are different proposed methods of license plate segmentation.

Method 1. [17]'s segmentation process starts by resizing all detected plates to 100 x 200 pixels. The next steps are taken to correct the horizontal tilt. They use CCA to find large numerals in the license plate and find the center point of each detected numeral. Then calculating of the tilt angle between two central points is done and the average tilt angle is obtained. Using the average tilt angle a 2d rotation method is applied to horizontally correct the license plate. To fix the vertical slant of the license plate the area of the large numerals is extracted. The width of the horizontal projection is calculated from the extraction. Then the extracted area is rotated from -45° to 45° while the projection value and rotation angle is recorded. The projection minimum is found from the projection data and it is used to correct the vertical slant. The correction method used is the horizontal pixel movement method. This method basically shifts pixels horizontally according to the formula in equation 2.6. Where d is the distance each pixel moves horizontally, x and y are the pixel coordinates, H is the height of the image and ζ is the vertical slant angle.

$$d_x = (y - \frac{1}{2}H)/\tan(\zeta) \tag{2.6}$$

After the license plate has been put into correct perspective, it is processed to enhance its contrast. To do this [17] uses a modified version of the gray statistical approach (GSA). Details of this GSA are omitted here. Once the enhancement has been complete, the plate is binarized using an improved *bersen* binarization algorithm. Lastly, the simple projection technique is used to segment the license plate into seperate characters.

Method 2. [20]'s approach to segmentation is CCA. CCA is applied to the license plate and prior knowledge is used to extract the character. The main priors include linearity, parallelism and the distance between characters. The tilt of the license plate is corrected by using metric rectification. The details of the algorithm are not provided in [20] but the basic idea provides sufficient information on how it works.

Method 3. [22] uses vertical and horizontal projections to segment the license plate. The difference in their approach from other papers that use projections, is in the pre-processing of the license plate. They use an object enhancement algorithm to enhance the license plate characters. The effect of the object enhancement algorithm weakens background pixels while strenghening the foreground pixels. [22] estimate that 20% of the license plate image are character pixels. The goal is to enhance these pixels which in their object enhancement algorithm takes two steps. The first step is to scale the gray level of all pixels into the range of 0 to 100. Then second step sorts all the pixels by their gray level in a descending order and multiplies the top 20% of the pixels by 2.55. This way the object pixels are enhanced whereas background pixels remain weak. From this enhancement the characters can be more accurately segmented using the projection method.

Other Methods. Other popular methods include using hough line transform for tilt correction and connected component analysis to extract out the characters.

2.4 Selected Approach

In this section I will discuss my selected method of license plate detection and how it works.

License Plate Localisation. My approach to license plate detection is somewhat similar to the approach in [20]. My approach however involves mathematical morphology. In the following I will go over the steps taken in my approach to detect license plates.

Step 1. My algorithm starts off with an input image which is firstly converted to grayscale. Since all images consist of some form of noise, the focus of this step is to reduce noise and smoothen the image. This is necessary as it improves the results of further processing on the image. Therefore, to reduce noise I apply a 5×5 gaussian blur to the image.

Step 2. In this step I process the image to detect edges. To do this I use the Sobel operator. License plate characters have more dominant and meaningful vertical edges in comparsion to horizontal edges. Hence, I apply the Sobel operator in the vertical direction. This means d_x is set to 1 whereas d_y is set to 0 (i.e. kept as a constant). The kernel size I used is 3×3. Note that the gaussian blur applied in step 1 also helps this step of edge detection.

Step 3. After edge detection I apply a closing morphological operation to the image. The closing operation helps dilate white regions as well as erode away small unwanted details. The structuring element used to apply this operation is given a similar aspect ratio to a license plate. This is so the resulting image produces rectilinear shapes (i.e. like the shape of a license plate). Once this is done one can realise grayscale information is no longer required. Therefore to simplify the image I process it using Otsu's binarization. When the binary image is obtained I apply two more morphological operations. These are an erosion operation followed by a dilation operation. One could simply apply an opening operation but the structuring element will then have to be the same for both operations. For my approach I require two different structuring elements so the opening operation on its own would not work. For erosion I use a structuring element of a size slightly larger than the one used for the closing operation. Using erosion again further helps in removing shapes that are too small to be a license plate. It also helps break apart large shapes which may contain the license plate (i.e. due to the application of these morphological operations the license plate may sometimes join into other objects). Then for dilation I use a larger and more square shaped structuring element. There are two reasons I applied dilation after the erosion operation. First reason is to simply bring back important removed information. For example, a license plate may already be in perfect shape but by applying erosion it may lose its shape. This is where dilation can be used to recover the shape. The second reason is because I needed the remaining shapes to gain more area around them. This is because if some shape is eventually a license plate it should cover as much of the license plate as possible. This reason also explains why I used a larger structuring element for dilation.

Step 4. By the end of step 3 an image of several regions which may consist of a license plate is produced. The focus in this step is to extract out regions which are most likely to be license plates. To do this I have used connected component analysis

(CCA). Firstly I calculate the minimum rectangular area for each region from which I obtain rectangles that represent each region. Then I compare the properties of the rectangles to the properties of a license plate. Based on the comparison I eliminate regions which are unlikely to be a license plate. The properties I use to determine if a region is a license plate or not include the area, the angle and the aspect ratio of an average license plate. If the area of a rectangle is either too big or too small the corresponding region is ignored. If the angle of a rectangle suggests the corresponding region is tilted too much (e.g. 75°), then that region is also ignored. Lastly, if the aspect ratio of a rectangle is quite different from a license plate's aspect ratio then that region is also ignored. In the end regions that correctly meet the properties of a license plate are extracted from the original grayscale image using CCA.

License Plate Validation. License plate validation is a process of validating images as a license plate. In this case these images are the candidate regions extracted in the localisation process. My approach to validate a region as a license plate involves running a simple test. The candidate region must pass this test in order to be classified as a license plate. This test is derived from [17].

Validation Test. A candidate region must contain at least two same or similar sized rectangles. I use this condition for validation because all characters on a license plate have the same width and height. If two similar sized rectangles are found then they are most likely to be license plate characters. On average license plates have at least five characters which means two characters should be detected if the candidate region is a license plate. Once such rectangles are detected, I calculate the center points of the rectangles. I then check if the center points lie in sensible positions to be considered as license plate characters. In the end the test has two conditions that must pass. The first I have already mentioned, and the second condition is that the candidate region must have a character sized rectangle somewhere near the center. To check this I again make use of the calculated center points.

On-going Validation. Apart from the validation test, the candidate regions are tested throughout the segmentation process to ensure they are in fact license plates. The segmentation algorithm is discussed in the next subsection.

License Plate Tilt Correction. The next step after validation is to correct any undesired tilt that the license plates may have. This is part of the segmentation algorithm (the segmenetation algorithm actually starts here). Once again I was inspired by [17]. Therefore, my approach is very similar but not exactly the same. To fix the tilt I simply gather all the calculated center points from the validation process and sort them in an ascending order according to their x coordinates. After sorting the center points I calculate the tilt angle of the license plate using the function available

in OpenCV. See equation 2.7, where *cvFastArctan* is a function that takes in a x and y value and calculates the associated angle. The provided x value is the absolute difference between the x coordinates of the lowest and highest center points (with respect to x). Similarly, the provided y value is the absolute difference between the y coordinates of the lowest and highest center points (with respect to y). Once the angle is found, I use a 2d rotation matrix to rotate the license plate accordingly.

$$angle = cvFastArctan(||y_{diff}||, ||x_{diff}||); \tag{2.7}$$

License Plate Pre-processing. Pre-processing the license plates is required to get better results during character segmentation. License plates not pre-processed will have noise and other undesired conditions that may produce incorrect segmented characters. The main focus of this pre-processing is to cleanly convert the license plates into a binary image. The first step is to reduce noise. For this I apply a 3×3 gaussian blur to the license plate image. Then I use a simple contrast enhancement method to give more dominance to the darker pixels in the region. I do this to make the characters more dominant. This approach is similar to the object enhancement algorithm described in [22]. The contrast enhancement I apply is given by the formula in equation 2.8.

$$g_{(x,y)} = \alpha \cdot f(x, y) + \beta \tag{2.8}$$

Where g is the output and f is the input. α and β are the gain and bias which are known to control the contrast and brightness. I have predefined α as 2 and β as -100. I obtained these values through experimentations with different license plates. Finally to convert the candidate regions into a binary image I use Otsu's binarization as I did in the localisation process.

Character Extraction. This is the last stage of the segmentation algorithm. This step simply involves using connected component analysis to extract the characters out of the license plate. This is so that they can be processed individually during the character recognition algorithm. Due to the way I designed the character recognition algorithm, this step is not carried out in the segmentation algorithm. The characters are extracted, processed and recognized one by one in the character recognition algorithm itself.

Chapter 3

Localisation and Segmentation Experiments

This chapter goes through the license plate detection and segmentation experiments.

3.1 Experiments

In this section I will discuss and present results from the experiments done using the license plate detection and segmentation algorithms. All images used for these experiments are personally taken by me (except one which was provided to me by SnapInspect). All images have a high resolution of 3264x2448. All figures are displayed below the following subsections.

Discussion of Detection Experiments. The outcome from the detection experiments suggests that the detection success rate is high. Figure 3.1 is a sample from the actual project scenario. Figures 3.2, 3.3, 3.4, 3.7, 3.8 and 3.9 are images taken in a similar scenario to the actual project scenario. The other images are taken in different scenarios to test the detection algorithm in various situations. I will first discuss the experiments done on frontal views of cars. In Fig. 3.1 it can be seen that all 3 license plates have been detected. Other detected regions in the image are false positives. The false positives are dealt with during the license plate validation process. Similarly, in figures 3.2, 3.3 and 3.4 all license plates are detected correctly. Even situations where there is high exposure to sunlight the detections are still successful. See Fig. 3.4. However, problems do occur when distance of license plates from the camera is changed. From the experiments I found that license plates closer to the camera fail to be detected correctly. The experiments suggest that the detection algorithm has a limitation of only optimally working on a certain range of distances. See chapter 5 for more details on the limitations. In figures 3.5 and 3.6 the detections fail if no alteration to the algorithm is done. Detection is only successful if the algorithm is adjusted to support the new distances. See figures 3.5 and 3.6.

The system is mainly focused for frontal views of parked cars. This is due to the nature of the given project scenario. However, a few tests were also considered

using tilted views of parked cars. Figures 3.7, 3.8 and 3.9 show successful detections of license plates from a tilted view of the camera. Just as in frontal views, the detections begin to misbehave when the distance of the camera is changed. The algorithm once again has to be adjusted to support the new distances. In figures 3.10 and 3.11 it can be seen that the detections are successful after the algorithm has been altered. One important aspect to notice is that the detections are still not completely as accurate as for images further away from the camera. The detections consist of extra information that is not associated with the plates (i.e. noise). The reason for this may be due to the various angles that are present in the images. See figures 3.10 and 3.11.

Discussion of Segmentation Experiments. Segmentation experiments also produced good results. I tested the segmentation algorithm on several license plates that were detected using the detection algorithm. The experiments were firstly carried out on frontal view license plates as they have a higher priority. Plates with a range of tilt angles were used in the experiments. Plates with no or very minimal tilt were also considered to ensure the algorithm only fixes tilt on plates that require it. The sample of plates also included plates in different lighting conditions. Some plates were under ideal lighting, whereas others were under poor lighting conditions (e.g. very bright or dark). Different plate sizes were also used, some of which were detections closer to the camera. See Fig. 3.12 which shows a collection of such frontal view license plates before the segmentation algorithm is applied. Figure 3.13 shows the results of applying the segmentation algorithm on the plates in Fig. 3.12. As it can be seen the tilt of all plates in Fig. 3.12 was corrected successfully. From the experiments I understood that the segmentation algorithm would only fail if the detected license plate was dirty or there was some other form of noise around the license plate characters.

I also carried out some experiments on the detections from tilted views. The segmentation algorithm again produced good results. However, a limitation in the algorithm was identified. Even though the tilt of the plates was corrected, the perspective of the characters was not. See Fig. 3.14. This limitation increases the chance of poor character extraction and character recognition. Details of this limitation are discussed in chapter 5. The last two license plates in Fig. 3.14 illustrate an example of characters out of perspective.

Results and Overall Performance. Overall the detection and segmentation algorithms performed well, especially in conditions that matched the given project scenario. The major advantage of the detection algorithm was that it was able to detect several plates in one execution. This makes the algorithm very efficient in comparison to other methods. The detection algorithm was also robust to different

lighting conditions. Similarly, the segmentation algorithm was robust to different lighting conditions as well as various tilt angles and plate sizes.

However, there were a couple of identified limitations which need to be addressed. From the experimentations it was evident that the detection algorithm works best only for a certain range of distances and the segmentation algorithm lacks the perspective correction feature.

To evaluate the performance of the algorithms I only considered images that matched the given project scenario. The detection algorithm was tested on 13 images, which consisted of in total 45 license plates (from 45 parked cars). Please note that the license plates were not all different in each image. For the purpose of testing it was not important for the license plates to be different in every sample. What was important however was the different conditions and situations the images are taken in. The sample of 13 images covers different conditions (i.e. such as lighting) and situations (i.e. such as number of cars parked and how they are parked). The segmentation algorithm was tested on 11 license plates images. In both tests the sample size was rather low. This was due to the lack of test data. I was only able to take a few images on my own and SnapInspect could also not supply test data as it was planned. Either way the results were obtained using the samples mentioned earlier. See table 3.1 which shows the results of the experiments.

Algorithm	Sample Size	Success Rate
Detection	45	100%
Segmentation	11	100%
Overall	56	100%

Table 3.1: Experimental Results.

The experimental results in table 3.1 show that algorithms have an overall success rate of 100%. This is obviously very optimistic and unreal. The clear reason for this is the low sample size. Apart from that, by looking at the results I can conclude the algorithms will produce high success rates in real life situations.

Run times. The average run time for the detection algorithm was 2.6 seconds. This was regardless of how many plates were detected. The average run time for the segmentation algorithm was 0.31 seconds. This run time was for segmenting a single license plate. These run times are very good especially considering the project scenario, where the algorithms will only run once every 5 minutes.

Figure 3.1: Successful detections from actual project scenario.

Figure 3.2: Succesful multiple plate detections.

Figure 3.3: More successful detections.

Figure 3.4: Successful detections with high exposure to sunlight.

Figure 3.5: Detection of plate closer to the camera once algorithm is altered.

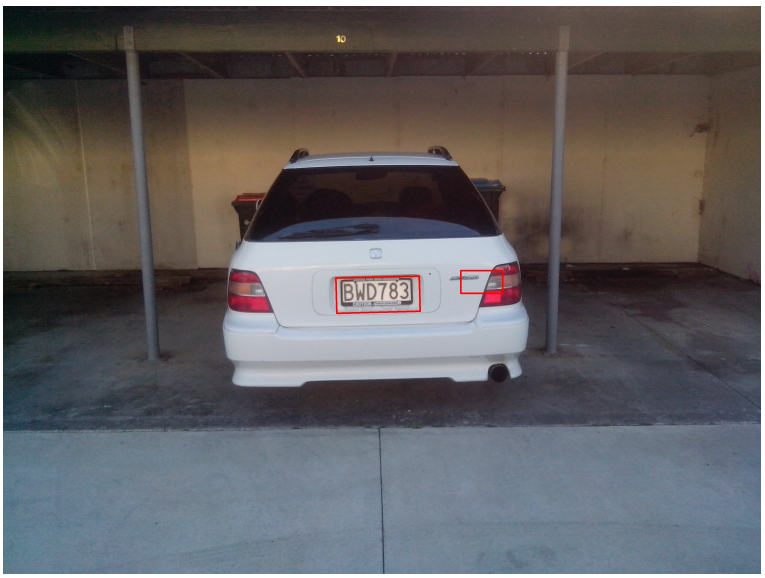

Figure 3.6: Another detection of plate closer to the camera.

Figure 3.7: Tilted view detection.

Figure 3.8: Tilted view detection.

Figure 3.9: Tilted view detection.

Figure 3.10: Tilted view detection.

Figure 3.11: Tilted view detection.

Figure 3.12: Frontal view plates before tilt correction.

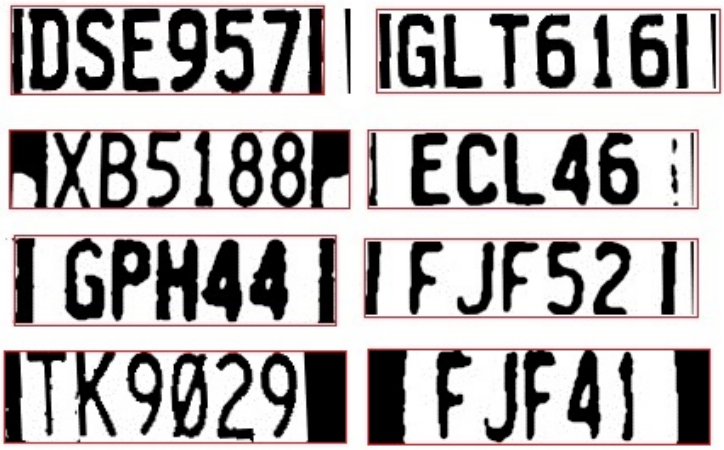

Figure 3.13: Plates tilt corrected and ready for character extraction.

Figure 3.14: *Left:* Tilted view plates. *Right:* Tilt corrected, ready for extraction.

Chapter 4

Reading of Plates

This chapter informs about different methods of character recognition as well as my approach to character recognition. The last section of this chapter discusses an alternative to character recognition.

4.1 Optical Character Recognition

Character recognition is the final stage of the license plate recognition system. In this stage the license plate characters are read and are used according to the purpose of the system. For this project, results of character recognition are just stored so they can be used for future reference. Before discussing my approach, in this section I will overview methods of character recognition from which I was inspired.

Method 1. [19] uses the very common method for character recognition; template matching. [19] first creates templates using prior knowlegde of license plate characters. They give each template a size of 42x24 pixels for uniformity. Then they put the templates into categories according to the number of holes in the characters. To recognize how many holes a certain character (whether it is a template or a segmented character region) has, [19] makes use of Euler numbers. Connected component analysis is used to obtain the Euler numbers. Characters without holes are given Euler number $E_0 = 1$ and all since odd indexed Euler numbers are zero, characters with one hole are given Euler number $E_1 = 0$. The last set of characters with two holes are given Euler number $E_2 = -1$. Before carrying out the matching [19] binarizes all the templates and all the segmented character regions. Then all segmented character regions are resized to 42x24 pixels and matched with templates that have the same number of holes. The method of template matching used by [19] is calculating of the correlation coefficient between the template and the segmented character region. The template which gives best correlation results is selected as the recognized character.

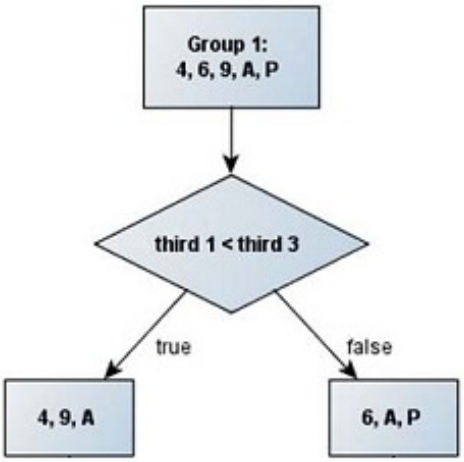

Figure 4.1: Decision making based on pixel concentration analysis [18].

Method 2. [22] also uses template matching but not in the same way as [19]. [22] uses 10 samples of each character to create templates. The process of creating the templates starts with an empty template (i.e. a matrix with all values set to zero). Then using each sample of a certain character, addition of white pixels is done into empty template. So wherever there is a white pixel in the sample, a value of 1 is added to the corresponding position of the empty template. This way a template is created with the weight (sum) of all white pixels from the samples of a certain character. The matching process is similar to the creating of the templates. Each segmented character region is matched pixelwise with the templates. A *score* is calculated according to the number of matched pixels. The *score* is incremented by 1 for every match and decremented by 1 for every mismatch. The template which produces the highest score is selected as the recognized character.

Method 3. [18] recognizes characters by testing the segmented character regions with several conditions. The first condition is based on how many end points a region as. Before finding the end points [18] makes use of a thinning algorithm to thin the regions to only 1 pixel thin. Thinning is important as it makes the process of finding the end points much simpler. The end points are then found by doing a pixelwise search. The algorithm moves on to the next condition only considering results that have the same number of end points as found for the region. The other conditions are based on how the pixels are distributed in the region. Such conditions are tested repeatedly until only one outcome is possible. Pixel distribution in

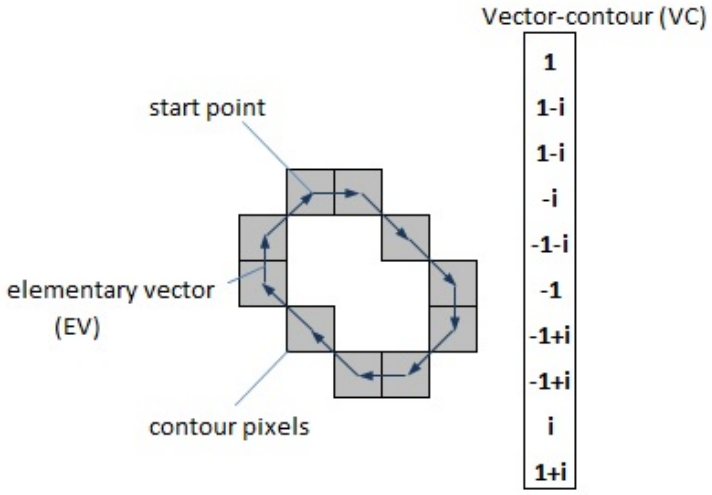

Figure 4.2: Vector-contour from thinned templates [23].

the regions is analyzed by dividing the regions into parts (i.e. into thirds or ninths). Pixel concentration in each part is calculated and decisions are made concerning the next condition. Prior knowlegde of the characters is used along with the pixel concentration calculations to determine the outcome of a condition. Once all relevant conditions are evaluated only one outcome remains and that outcome is selected as the recognized character. See Fig. 4.1.

Method 4. [23] uses a unique method of template matching and pixelwise contour analysis to recognize characters. [23] starts with templates based on prior knowledge. Then the templates are thinned to 1 pixel thin similar to [18]'s approach. For every pixel in the thinned templates, positions of neighbouring pixels are recorded into a *vector-contour*. See Fig. 4.2. The segmented character regions are then matched with every recorded *vector-contour*. If a segmented character region results to be a scalar multiple of a certain vector-contour, the template associated with the vector-contour is selected as the recognized character.

4.2 Chosen Method

In this section I will discuss my selected method of reading the license plate characters and how it works.

Optical Character Recognition (OCR). My approach to character recognition is very simple but effective. The process involves template matching with the use of a bitwise operation. This is a method I am proposing myself with inspiration and ideas from the research I did. In the following I will go over the steps taken in my approach to recognize license plate characters.

Step 1. This step is actually carried out when the application first starts. In this step all the necessary templates are loaded into the application and are also pre-processed so they are ready to be used. I load the templates into the application in the start so unnecessary file reading does not occur. All templates are obtained from [24]. Each template has a size of 23x48. The pre-processing of the templates involves some simple binary thresholding and inverting. To all the template characters I apply a binary threshold with values 127 and 255 as the minimum and maximum respectively. The template characters are then inverted as it is required for the later steps. For better results and simplification I arranged the templates into 3 categories as done by [19]. These categories are *no holes*, *one hole* and *two holes*. The *no holes* category contains all the characters that have no holes such as the letters C, E, F, G, H and so on. Similarly, the *one hole* and *two holes* categories include characters such as O, P, B and other characters containing holes. Dividing the characters into these categories helps simplify the process of template matching and it also improves the efficiency of the algorithm. This is because it will reduce the number of times a segmented character region (*SCR*) is matched to a template. In other words, it will not be required to compare the *SCRs* to every template during the matching phase (i.e.*SCRs* will only be matched with templates that have the same number of holes).

Step 2. The character recognition algorithm starts with applying an Otsu threshold to the cropped license plate obtained from the previous tilt correction algorithm. The license plate is also inverted (i.e. from black on white to white on black) during the Otsu thresholding. The inversion is required so that in the next step only the characters are detected as contours and not the background.

Step 3. In this step I start segmenting the cropped license plate into characters and putting the characters into categories that I mentioned in step 1. Only one character is processed at one time. To segment the license plate and categorize the characters I find contours and then process each contour individually. Only contours that have certain properties (e.g. area or position) are considered as there can still be some regions in the license plate which are not characters. Whenever a correct contour is found the algorithm proceeds by counting the number of holes and switching to the right category to perform template matching.

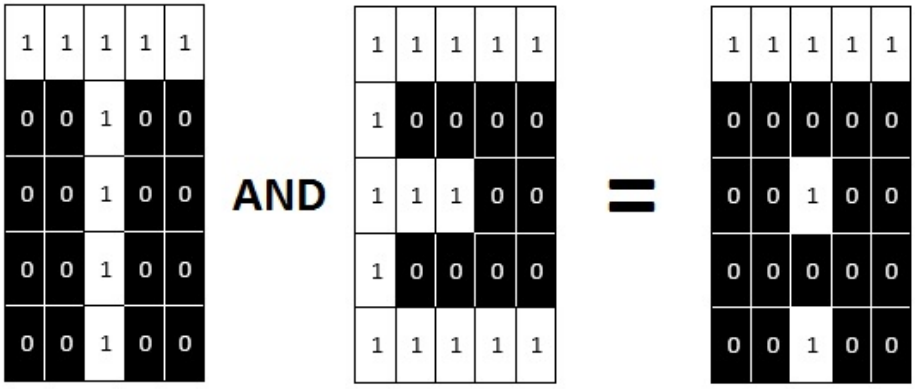

Figure 4.3: *Left:* Segmented character region "T". *Center:* Template for letter "E". *Right:* Result of AND operation.

Step 4. Before a *SCR* is matched with a template I resize it to a size of 23x48. This is so both the template and the *SCR* have the same size. As mentioned earlier, the method I use to perform template matching is something I am proposing myself. This method I propose uses a logical AND bitwise operation between the *SCR* and the template. Before stating the purpose of this bitwise operation I will recall how the logical AND works.

$$100111 \ \ AND \ \ 111010 = 100010 \tag{4.1}$$

The logical AND bitwise operator is applied to binary values. The operator returns true (or a one in binary terms) wherever it finds two ones in the same position of the two input binary values as shown in equation 4.1. This function will work well for template matching because essentially the *SCRs* and the templates are just binary values. This is due to the thresholding that is applied to them beforehand. Therefore, using this operator I can count how many of the pixels match between a *SCR* and a certain template. Then the template which matches the best can be concluded as the recognized character. So to recognize the characters I apply the AND operator using the *SCRs* and the templates as the two input binary values. The output produced is a matrix of binary values containing the value *one* at pixel locations that matched and containing the value *zero* at pixel locations that did not match (See Fig. 4.3). Let this matrix equal *AM*.

Step 5. In this step I select the best match from the template matching for the *SCRs*. I first calculate the number of non-zeros in the template, in the *SCR* and in

Figure 4.4: Example of problem faced using AND operator. Letter 'L' detected as an 'E'.

the matrix produced by the AND operator. Let these values equal to *zt*, *zc* and *za* respectively. Then I calculate the differences as shown in equations 4.2 and 4.3. To explain the purpose and use of the two differences I will demonstrate how the letter 'L' would be recognized using my algorithm.

$$diffOne = \|zc - za\| \tag{4.2}$$

$$diffTwo = \|zt - za\| \tag{4.3}$$

The letter 'L' would fall in the category of *no holes* and then will be matched with all the templates in that category using the AND operator. At times it would turn out that the letter 'E' is the best match for 'L' because 'L' can be created using the pixels in the template for 'E'. See Fig. 4.4. Since the letter 'E' can cover all of 'L' a cross check must be done on the match using the matrix *AM* for both the template and the *SCR*. This is where the calculation of the two differences shown in 4.2 and 4.3 are used. The difference *diffOne* estimates how well the match is between the *SCR* and matrix *AM*. The difference *diffTwo* estimates how well the match is between the template and matrix *AM*. In this example *diffTwo* would be greater than *diffOne* because the template does not match matrix *AM* very well. If the template was letter 'L' then *diffOne* and *diffTwo* would be quite similar. Therefore, to recognize the SCR the lowest sum of the two differences is found and the template associated with that sum is concluded as the recognized character. This works because even though 'L' can be created using 'E', 'E' cannot be created using 'L'. The two differences help in ensuring that the template matches the *SCR* and the *SCR* also matches the template (i.e. the match is not only one way but both ways). Please note that these checks are only necessary in my approach of template matching.

Figure 4.5: Slanted license plate characters.

4.3 Experiments

In this section I will discuss and present the results of character recognition experiments.

Character Recognition Results Discussion. The character recognition algorithm was tested on 11 license plates. Plates from frontal and tilted views were considered. In total there was 61 characters to be recognized. The end result was that 54 out of the 61 characters were detected correctly. This gave a success rate of 89%. A success rate of 90% and above would have been more favourable. The success rate was affected due to a few different reasons. One area the algorithm performed poorly was on the titled view license plates. This is an expected behaviour because the characters in the tilted view plates are slanted and out of perspective. See Fig. 4.5. Pixelwise template matching in that case is not very robust. Incorrect recognitions included slanted letter "F" being detected as an "E" and slanted number 9 being detected as a 6. Another area the algorithm lacked success was in detecting characters which were quite similar in shape to other characters. This was commonly the case with characters such as the letter "B", the number 0 (zero) and also the number 8.

Another experiment was also conducted. This time no tilted view license plates were considered. The license plates also had only one appearance of the following three characters "B", 0 (zero), and 8. The results were far better than the initial experiment. Out of 38 characters 37 were detected correctly, giving a respectable success rate of 97%. It is important to note here that the sample size is not very large.

From the experiments I learnt that my character recognition algorithm has certain limitations. It is clear that my algorithm is not completely robust to out of perspective license plates as well as characters that are similar in shape. To achieve a higher success rate correction of the perspective is essential and a more detailed analysis of the characters is also important. A positive outcome however is the fact that my algorithm promises substantial results when tested on plates that do not

trigger the limitations.

Run time. The average run time of the character recognition algorithm was 0.19 seconds. This time was calculated when using the algorithm on a single license plate. Just like the other two algorithms this run time is really good.

4.4 Alternative Approach

In this section I will talk about another approach that I came up with to recognize license plates. Please note that this approach may not work in every scenario but it works for the given project scenario.

Project Scenario Recall. SnapInspect has a parking lot where visitors sometimes exceed the time limit allowed to park. The purpose of the license plate recognition system is to monitor cars in the parking lot and report any cars to staff that violate the time limit. The initial approach is to read the license plates of cars over certain time periods. If a license plate is repeatedly recognized for a time longer than allowed, the system reports it to the staff.

Alternative Approach. In the scenario described above, I discovered that reading of the license plates is not really necessary. Therefore, I propose that rather than reading the license plates, just match the detected plates with previous detections. In the following I will go over an example with a few steps to further explain this approach.

Step 1. We can take an example of three parked cars and their detected license plates. See Fig. 4.6.

Step 2. When plates are detected the following are saved into the application: time of detection, locations of plates and a binary version of the detected plate. Saving the plate in binary form is important because binarization eliminates small changes that can occur to the plates (e.g. lighting). If coloured plates are stored and different lighting conditions occur (i.e. sunshine), then matching the plates would suggest they are different plates. However, after binarization this is not the case.

Step 3. Whenever the algorithm runs again, all saved data in the application is updated. If a detected plate is already in the saved data then only the time is updated (i.e. the new time will be the difference between the current time and the time of previous detection). If plates in the saved data are not detected again, the plates are deleted from the saved data (this means the car has left).

Figure 4.6: *Left:* Sample image of project scenario. *Right:* Detected license plates.

Figure 4.7: Using AND operator to match detected license plates.

Step 4. To identify cars that have been parked for too long, the system can simply refer to the times recorded for the plates that are being detected repeatedly.

Step 5. Last step is to report the staff about the cars recognized to be violating the parking time limit.

Matching Binary Plates. To match the detected plates to previous detections, I simply make use of the method I used for character recognition. See Fig. 4.7.

Efficiency and Success Rate. Using this alternative approach gives a positive significant change to both the system's efficiency and success rate. The efficiency

of the system is improved due to reduced processing. Using this alternative approach means that the only major algorithm executed is the license plate detection algorithm. License plate tilt correction, license plate segmentation and license plate character recognition algorithms are all skipped. The success rate is also improved because of only doing simple comparsions between binarized license plates rather than complex segmentation and reading of characters. Since license plate segmentation and character recognition can give erroneous results if conditions are not good to a certain degree, this approach is a useful alternative. By using this approach the success rate is also not affected by license plate tilt or translation.

Chapter 5

Limitations

This chapter informs about the identified limitations of the license plate recognition system developed for this project.

5.1 System Limitations

Here I will discuss the limitations that my license plate recognition system is subjected to.

License Plate Detection Limitations. The license plate detection algorithm is limited to a certain range of distances. The range is not very small so running the algorithm with variable distances will still produce good results. However, distances outside this range can produce erroneous results. Therefore, the range of distances must be known before using the system. Usually license plates that are too close or too far from the camera are known to be not detected correctly. Apart from the range of distances, another limitation is that the algorithm only works on a given image resolution. Therefore, the image resolution must be known as well before using the system. Knowing the resolution is important because the algorithm works pixelwise on the input image. Which means different resolutions produce different results. A positive however is that the variables in the algorithm can be easily adjusted to suit different types of image resolutions so the system is able to work on most devices.

Another limitation occurs due to custom license plates in the real world. Custom license plates often consist of wide gaps between the characters. My license plate detection algorithm uses mathematical morphology to morph together all the characters on the plate. Since custom plates have gaps the morphing of the characters sometimes fails (depending on how wide the gaps are). This results in parital detections of plates and at times the plates are not detected at all.

License Plate Segmentation Limitations. The license plate segmentation algorithm has a limitation of only performing well on detected license plates with good perspective. License plates that are out of perspective are likely to have a lower success rate of being segmented correctly. This limitation exists because my algorithm only considers and fixes the tilt of the detected license plates. The correction of the perspective is not developed in this algorithm. This limitation can also affect the results of character recognition. Character recognition is done through template matching and slanted segmented character regions increase the rate of incorrect recognitions. A step to solving this problem is to make use of the method proposed in [17], where the issue of slanted characters is addressed and solved.

Optical Character Recognition Limitations. The one major limitation of the character recognition algorithm is that it is heavily dependant on the success rate of the license plate segmentation algorithm. This is a common limitation when it comes to automatic license plate recognition systems. This limitation comes into effect in my algorithm because it works by performing pixelwise template matching to recognize characters. The explanation for this is quite clear; poorly segmented regions result in poor recognition.

Other Limitations. Another limitation worth mentioning is my license plate recognition system is only suitable for use during the daytime. There have been no experiments or tests conducted using my system at night time. SnapInspect was not interested in the system working at night time because the work hours at their company are only throughout the day.

There may be many other unidentified limitations of the system. The reason for this is mainly because the system has not been tested sufficiently. This is due to the lack of test data. Further limitations and flaws can only be discovered once the system is tested against plenty more of test data.

Chapter 6

Android Application Design

This chapter goes over the android application design of the license plate recognition system.

6.1 Application Design

Due to a unforeseen event which lead to unmanageable time constraints I was unable to complete an android application out of my license plate recognition system. However, in this chapter I will briefly go over the application design which is likely to be used in the near future.

Design Overview. The application's design is very straightforward and basic. The application's primary source of data comes from the camera of the android device. Other data sources come from data stored locally on the device and also in some cases data stored remotely. The application will require some form of network connection to allow communication with the end user (e.g. SnapInspect staff). Apart from one way messages being sent to the company staff the application will have no other interaction with its end users.

User Interface. There will be no real user interface of the application. This is because the application is supposed to work automatically. When the application starts a camera view will be opened and the algorithms will start executing in the background. Algorithms will be executed every 5 minutes to have information constantly updated.

Data Storage. Results from running the algorithms will most likely be stored locally on the device. The results will only take a few megabytes of storage space. The storage space will also be recycled quite frequently, so data storage will not be a concern if stored locally. If some reason is discovered to store results remotely the runtimes of the algorithms may be affected.

Android Development Specifications. Currently no specifications are given to me regarding the Android application. However, the application is most likely to

be developed for devices with Android version 4.0 and above. The application will be written in C++ and will use the OpenCV library for Android.

Chapter 7

Conclusions

This project was given to me by SnapInspect and my task was to design and implement a license plate recognition system. This report covered the entire year long project. Research, approaches, implementation and design were all discussed extensively in this report.

The positives of this project came mainly from the technical side. The project went quite well with all major algorithms implemented. Several experiments and tests were also conducted throughtout the year. There were on going discoveries of limitations and new developments had to take place every now and then. Various methods were also researched and implemented to test which approach works best. In this report only the final decided approaches were discussed. The results showed promising success rates and run times as well as certain areas that have room for improvement.

There were also a few negatives that affected the quality of this project. One of them was an unfortunate event that took place which really slowed down the progress of this project. This was the reason for which certain areas of the project were not given enough attention and the quality was discouraged due to time constraints. One example of this is the fact that I was unable to develop an Android application out of my license plate recognition system. Another negative was the lack of communication with my industry mentor. Throughout the project there were only two formal meetings with my industry mentor. So the project could not take into account their feedback and opinions. Another major negative was the absence of test data. My industry mentor was supposed to supply me with test data. However, they were unable to arrange this in time. I had to resort to using my own images, which was not enough to make solid conclusions about my algorithms.

The project had many other ups and downs but overall it can be concluded as a successful project. Personally, I learnt a whole lot of new things and also gained good experience in managing and completing a real life project.

Bibliography

[1] HIPR2. homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm, last visit: 11 August 2014

[2] Go Digital. godigitalslr.com/wp-content/uploads/2012/02/HistogramDiagram.jpg, last visit: 7 August 2014.

[3] HIPR2. homepages.inf.ed.ac.uk/rbf/HIPR2/kernel.htm, last visit: 11 August 2014

[4] Open CV 2.4.9.0 Documentation. docs.opencv.org/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html, last visit: 11 August 2014

[5] HIPR2. homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm, last visit: 11 August 2014

[6] Klette, R.: Concise Computer Vision. Page 63, Springer, 2014

[7] Klette, R.: Concise Computer Vision. Page 64, Springer, 2014

[8] HIPR2. homepages.inf.ed.ac.uk/rbf/HIPR2/canny.htm, last visit: 11 August 2014

[9] Klette, R.: Concise Computer Vision. Page 123, Springer, 2014

[10] HIPR2. homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm, last visit: 11 August 2014

[11] Open CV 2.4.9.0 Documentation. docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html, last visit: 11 August 2014

[12] HIPR2. homepages.inf.ed.ac.uk/rbf/HIPR2/spatdom.htm, last visit: 11 August 2014

[13] HIPR2. homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm, last visit: 11 August 2014

[14] Klette, R.: Concise Computer Vision. Page 57, Springer, 2014

[15] COMPSCI 373 Computer Graphics and Image Processing. cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/2013/CS373-IP-02.pdf, last visit: 11 August 2014

[16] Klette, R.: Concise Computer Vision. Page 170, Springer, 2014

[17] Wen, Y., Lu, Y., Yan, J., Zhou, Z., Deneen, K., Shi, P.:An Algorithm for License Plate Recognition Applied to Intelligent Transportation System. IEEE Transactions, 2011

[18] Romic, K., Galic, I., Baumgartner, A.: Character Recognition Based On Region Pixel Concentration For License Plate Identification. 2012

[19] Kodwani, L., Meher, S.: Automatic License Plate Recognition in Real Time Videos using Visual Surveillance Techniques.

[20] Thome, N., Vacavant, A., Robinault, L., Miguet, S.: A Cognitive and Video-based Approach for Multinational License Plate Recognition. Machine Vision and Applications, 2011

[21] Wang, Y., Lin, W., Horng, S.:A Sliding Window Technique for Efficient License Plate Localization Based on Discrete Wavelet Transform. Expert Systems with Applications, 2011

[22] Kranthi, S., Pranathi, K., Srisaila, A.: Automatic Number Plate Recognition. International Journal of Advancements in Technology, 2011

[23] Kumary, S., Angel, B., Laya, T.: License Plate Detection and Character Recognition Using Contour Analysis. International Journal of Advanced Trends in Computer Science and Engineering, 2014

[24] Personalised Plates. http://www.plates.co.nz, last visit: 17 October 2014

[25] Image Library: i126.photobucket.com/albums/p105/DoZZa1/P1010414.jpg, last visit: last visit: 11 August 2014