SimpleANPR - Image Manipulation

Martyn Sweet
Computer Science BSc – Part 1

m.b.sweet@student.reading.ac.uk

Abstract - This report discusses the implementation, testing and lessons learnt while implementing a simple numberplate extraction system in C++ using methods found during extensive research and development. The report discusses potential improvements which could be made to the program using additional methods or refinement of the existing implementation.

Table of Contents

1. Introduction

SimpleANPR is a program which can be used to extract the characters from an image containing a numberplate. The program uses connected-component analysis to detect potential regions of characters, before refining the search and outputting the characters if possible. If no characters can be successfully located (possibly due to noise), the general region of the numberplate is outputted. The outputted image can then be run though an existing OCR system in order to derive the cars numberplate in text form.

2. Usage

```
Command line execution: ./SimpleANPR image1.jpg [image2.jpg image2.jpg ...] [--debug] [--cases 12] [--buffer 2] [--spacing 5] [--sdl] [--nosave] [--help]
Images: Relative or absolute paths allowed, images must be of correct orientation
--debug: Displays more SDL steps and outputs debug information to the console
--cases: Amount of sample area iterations to try on each image
--buffer: Amount of pixels around the character to be extracted
--spacing: Amount of spacing to the right of each character in the output image
--sdl: Display the SDL window for the result
--nosave: Don't save the final result, otherwised saved as plate_image1.bmp to current working directory
--help: This information message
```

3. Sample inputs / outputs

Input Image	*Output Image*
	
	
	

Table 1: *Input and Output Images*

4. Implementation

After much trial and error (see review) the program used connected-component analysis at it's core to find a numberplate region, and then extract the characters. At the necessary steps, the input image is cleaned by techniques of normalisation and histogram equalisation.

The first step is to locate a rough numberplate region, this was done to improve the probability of successful extraction, as the isolated region helps improve image cleansing techniques resulting in better results for each adaptive monochrome attempt and character extraction. This step finds a potential group of characters using the connected-component analysis, the first of these characters is used as a reference to draw a rough box for further analysis.

Once the region has been located, the same algorithm is run to extract the characters. While this may seem inefficient, the main difference here is the sample area when preforming the *adaptive monochroming* step, this step allows images with varying levels of light intensity and noise to be handled and still get reasonable results. The adaptive monochrome provides a binary image (black and white), which is used to effectively to determine isolated white blobs of pixels, which we call regions of interest (ROI). Once each region is located, it is either accepted or rejected by the *ratio analysis* stage, which simply checks the ratio of width and height and expects it to be in a defined range.

These potential blobs of pixels may have successfully passed *ratio analysis* however not be characters, as an odd blob of pixels will quite probably fall under the dimensions allowed. This issue is handled by the l*ocating neighbouring characters* stage. This process involves *disjoint sets* to group together boxes which lay to the right of each other, this is possible as all the characters on a numberplate are spaced uniformly as required by law. It would be highly improbable for another group of more than 4 regions of acceptable ratio to be found within an image. The process first orders the set of potential characters by x, characters which are then close to each other are unioned into a single set, the result of this is lots of characters all being in one set, the most common of these sets is then found and the matching characters identified.

The above process is repeated multiple times increasing the adaptive monochroming area by 3 for every `--cases X` specified, this creates a *vector* of *vector<ROI>*. The most common character count is then found, voting and weighting occurs to determine the best match. Once this match has been found, the located regions are drawn onto a new canvas and saved and/or outputted to the screen depending on the users requirements.

The program makes use of a few C++11 standards, mainly lambda functions, these inline functions which occur in many sort clauses allow the code to be much more readable and aid in clarity as the datasets often contain nested datatypes. Traditionally, the sort function would need to call another function which would have to be specified, this often created code which was not as clear as the below implementation.

```
std::sort(SetVote.begin(), SetVote.end(), [](const std::pair<int,int> &Element1, const std::pair<int,int> &Element2) {
                                              return Element1.second > Element2.second;
                                          });
```

Figure 1: *Sorting a pair with a lambda function*

4.1. Debug Mode

In *Figure 2* we can see the stage of character location before extracting the general plate region. This step shows located blobs (green border), blobs which passed *ratio analysis* (blue border) and blobs which have been grouped together and identified as characters (pink border). Each potential character will draw an orange horizontal line, if this falls within another character, they are paired. All pixels identified as part of a region are turned red for easy identification of regions.

Figure 3 shows CCA after the first iteration of monochroming, as can be seen 3 characters (GBJ) were identified as potential character candidates, this result will be taken into consideration, but further iterations will provide better results and overrule this one during the voting stage.

Figure 4 shows a much more successful character extraction after multiple monochroming iterations, several more iterations will produce a similar result, and thus this will be used as the result.

The located regions are then drawn onto a new canvas as shown in *Figure 5*, which can be saved or displayed on the screen as required.

Figure 2: *Initial connected-component analysis*

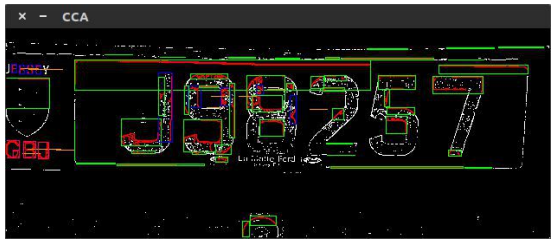


Figure 3: *First iteration of connected-component analysis in located region*

Figure 4: *Iteration of connected-component analysis*

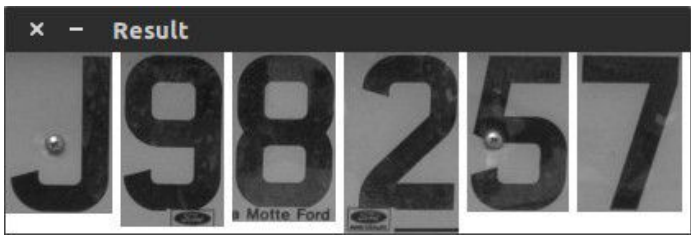


Figure 5: *Final result of the program*

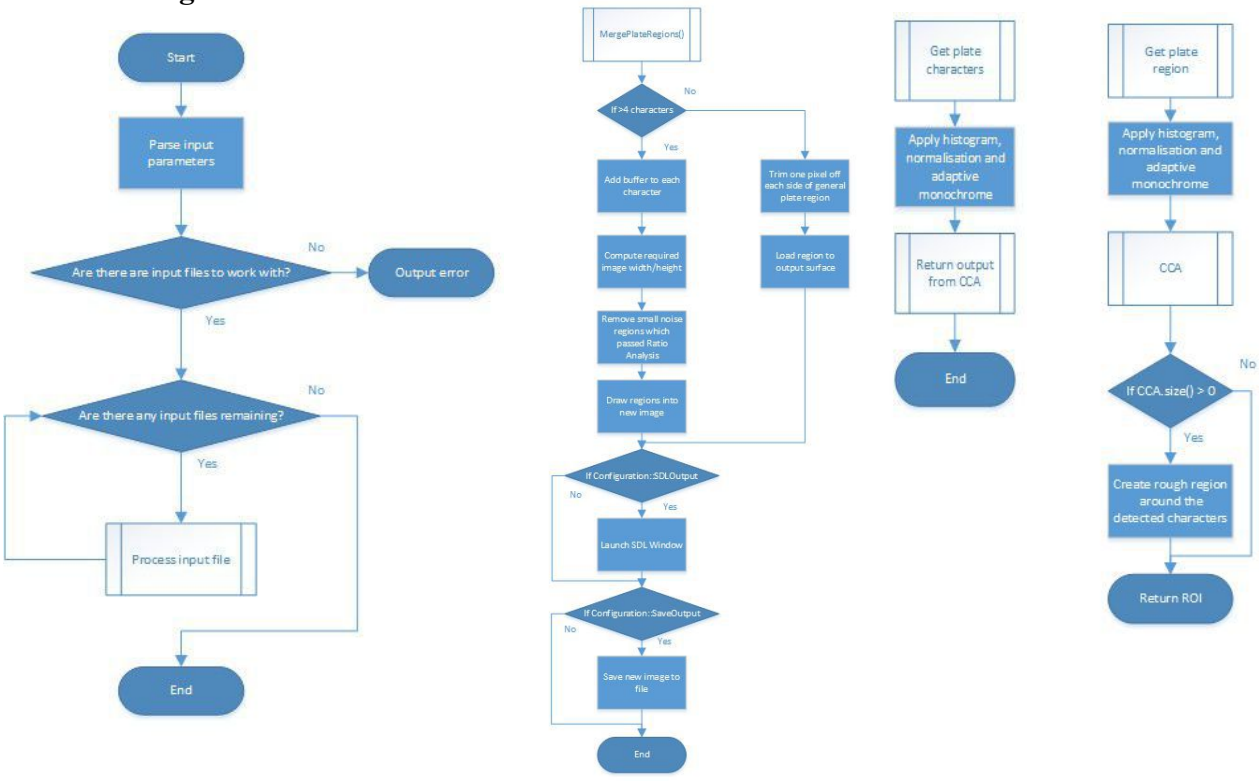
5. Program Flow

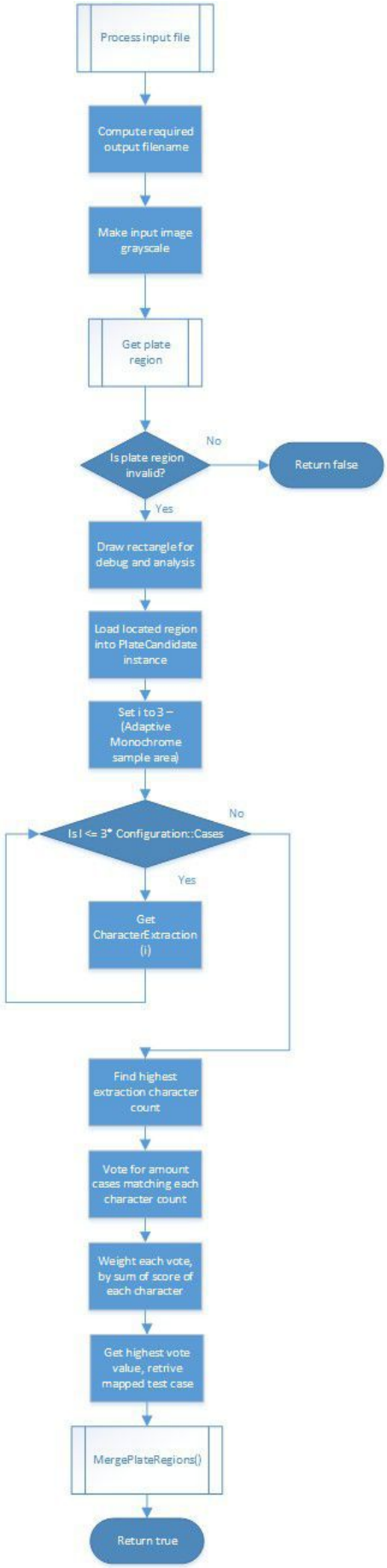
5.1 High-level Pseudocode

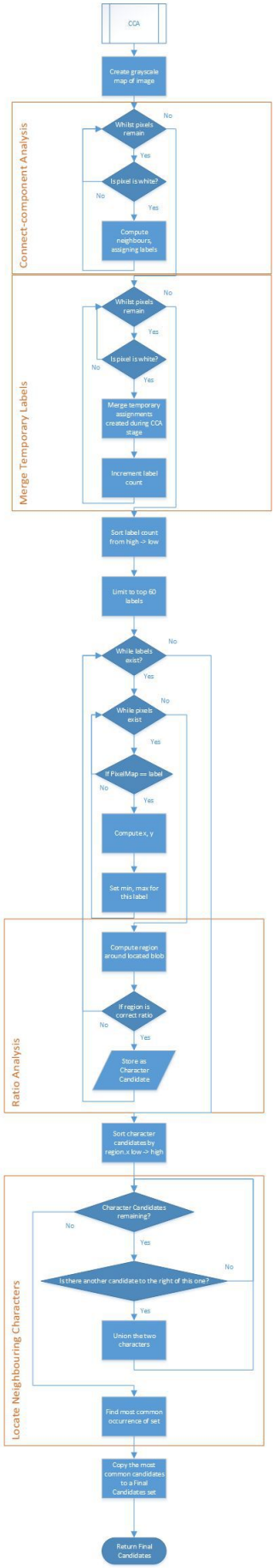
The program follows the following high-level pseudocode:

```
Set Configuration Variables
For each image
     Get plate region
     for i=3; i<= Configuration::Cases*3; i+=3
          Get potential character blob regions
     end
     Find the highest amount of characters found
     Vote + weight and score test cases
     Use highest scored test case for final output
     Merge character regions to new SDL_Surface
     if Configuration::SDLOutput
          Display in SDL Window
end
     if Configuration::SaveOutput
          Save image to working directory
     end
end
```

5.2 Flow Diagram




Process input file
Compute required output filename
Make input image grayscale
Get plate region
Is plate region invalid?
No
Return false
Yes
Draw rectangle for debug and analysis
Load located region into PlateCandidate instance
Set i to 3 – (Adaptive Monochrome sample area)
Is I <= 3* Configuration::Cases
No
Yes
Get CharacterExtraction (i)
Find highest extraction character count
Vote for amount cases matching each character count
Weight each vote, by sum of score of each character
Get highest vote value, retrive mapped test case
MergePlateRegions()
Return true


CCA
Create grayscale map of image
Connect-component Analysis
Whilst pixels remain
Is pixel is white?
Compute neighbours, assigning labels
Merge Temporary Labels
Merge temporary assignments created during CCA stage
Increment label count
Sort label count from high -> low
Limit to top 60 labels
While labels exist?
While pixels exist
If PixelMap == label
Compute x, y
Set min, max for this label
Ratio Analysis
Compute region around located blob
If region is correct ratio
Store as Character Candidate
Sort character candidates by region.x low -> high
Locate Neighbouring Characters
Character Candidates remaining?
Is there another candidate to the right of this one?
Union the two characters
Find most common occurrence of set
Copy the most common candidates to a Final Candidates set
Return Final Candidates

6. Differences from design

The final program from this project has a few differences from the initial design, this is due to the difficulties faced while extracting the numberplate characters. It was decided that good extraction was preferred, as a result there was no time to preform the OCR analysis of an extracted character. The program has a wealth of command line options compared to the initial design, which allows for multiple images and tweaking of other parameters which can effect the output of the program.

Mainly, the program now uses connected-component analysis for detecting potential characters in the image, originally row profiles was intended however it was seen to have a number of drawbacks (see review).

7. Data Structures

```
/* Rectangle */
struct Rectangle {
    int x;
    int y;
    int Width;
    int Height;
};

/* Region of Interest */
struct ROI {
    Rectangle Rect;
    int Score = 0;
    int Pixels = 0;
};

/* 2D Point */
struct Point {
    int x;
    int y;
};

/* Global Configuration */
namespace Configuration {
    extern bool Debug;
    extern bool SDLOutput;
    extern bool SaveOutput;
    extern int  Cases;
    extern int  Buffer;
    extern int  Spacing;
};
```

Figure 6: *Used datastructures*

The program makes use of three structs and one namespace in order to simplify the movement of data through the programs execution, these can be seen in *Figure 6*.

The Rectangle structure is used extensively to draw and keep track of regions throughout the program, it is mainly accessed via the Region of Interest (ROI) structure as a nested element. This provides a nice container for all the information required for each potential character candidate.

The Point structure is used to represent a point in 2D space, allowing the data to be stored in a related, contained fashion, as opposed to having multiple variables.

The Configuration namespace is used as a global variable container in order to allow all files of the program to access and check the configuration variables, this gives a wide range of flexibility and is much neater than passing constant variables around as function arguments.

The program makes extensive use of vectors to hold groups of regions (ROI), this allows for easy dynamic management and processing of elements as each image will produce a different amount of regions which need processing.

Pairs are also used extensively throughout the program in collaboration with maps in order to sort datasets where the key is important, for example keeping the label ID, but sorting the amount of references the ID has. This would require a lot of work around for traditional vectors or arrays as the keys are changed during the sorting process.

Maps are used in the program for providing dynamic key allocations without the need to initialise the dataset before hand. This allows only very specific keys to be used without first having to expand the array to a value which may not be known, these keys are then dynamically ordered for later processing by calling the value by its key.

8. Class Documentation

SimpleANPR consists of 4 classes which are used during the processing of the image.

BitmapImage – This class contains the core functions for image loading and manipulation, such as drawing rectangles or apply adaptive monochroming.

ANPRImage – This class inherits *BitmapImage* and is responsible for managing instances of PlateCandidate in order to locate the region, characters, and determine the best output case before merging the results into a new surface.

PlateCandidate – This class inherits *BitmapImage* and contains the functions required for locating characters in the image using connected-component analysis (CCA) and either returning all the characters or just a potential region.

DisjointSet – This class provides the functionality to implement disjoint sets, allowing grouping of objects within the connected-component analysis.

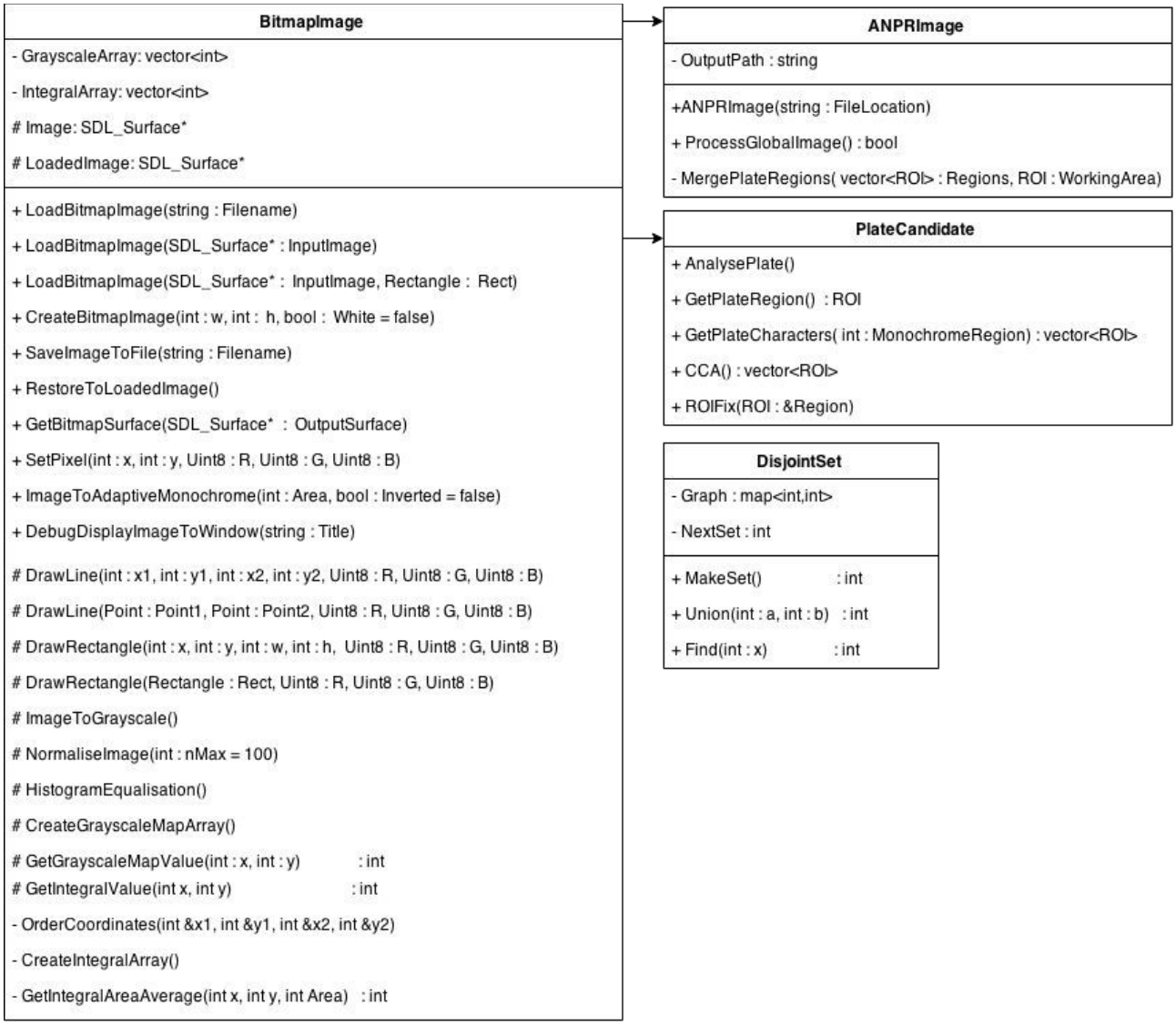


Figure 7: *Final UML Diagram*

9. Testing

9.1 Command line Input

All command line inputs were tested. The program successfully works with relative or absolute paths to image locations and with all the stated command line flags.

It was noted that the program does not work with EXIF rotation data, as this was not implemented by design, as a result the program sees the image in the wrong orientation during processing, resulting in an incorrect output.

It was also noted that the program preforms slightly worse in a Windows enviroment compared to a Linux environment, the reason for this seems to be related to the *adaptive monochroming* step or a cleanup step before which is affecting the results.

9.2 Output Testing

It was important to see how well the program preformed on real world images, it was noted that these images were not of ANPR quality (focused at the number plate with little background) and often have poor lighting causing the PARTIAL or FAIL of some tests. This being said however, the program preformed well under most cases. The tests allowed reoccurring issues to arise and were run within a Linux environment. Incorrect results were then analysed using the `--debug` flag.

```
$ ./ANPR ../testcases/{1..35}.jpg
```

The results of the test are shown in *Table 3*, on the next page, a summary of the results are shown in *Table 2*.

	Count	***Percentage***
PASS	25	71.4%
PARTIAL	8	22.8%
FAIL	2	5.71%

Table 2: *Percentage outcome from test cases*

As can be seen from T*able 3 (see next page)*, the majority of test cases worked successfully, these were often high quality images with a good clear number plates. The system most commonly failed under the following scenarios:
- Poor quality image
- Shadowing on the numberplate
- Character I/1 being excluded from ratio analysis

The first two issues calls for better image preprocessing, which is difficult to archive in short computation times. Future work would definitely include looking into this for improving the quality of an image before the connected-component analysis.

As a character I/1 is only the width of a stroke (and not the normal width of a character) it falls under different range during ratio analysis. On the failed images, the ratio was manually calculated for this character and falls in the range of *4.3-6.0*. Unfortunately applying this ratio into the *ratio analysis* logic results in lots of unacceptable regions being classed as characters. It would therefore be more intelligent to locate potential characters which have been missed during the character grouping phase – possibly finding characters which have failed *ratio analysis* between two valid confirmed characters.

Image	*Number Plate*	*Detected*	*Output*	*Notes*
1	J98 257	PASS	J98 257	
2	AR02 ONR	PASS	AR02 0NR	
3	ESS 3X	PASS	ESS 3X	
4	WOR 516K	PASS	WOR 516K	
5	B20 DVW	PASS	D20 DVW	
6	Y494 TSA	PASS	Y494 TA	
7	NODDY 1	PARTIAL	NODDY	The character 1 is too small to be chosen by *ratio analysis*
8	RN52 KVS	PASS	RN52 KVS	
9	KN52 KVS	PASS	RN52 KVS	
10	FL1 CKR	PASS	FL1 CKR	
11	HX51 NWT	PARTIAL	HX5 NWT	Image is very small, producing a low quality output and missing 1 during ratio analysis
12	HX51 NWT	PASS	HX51 NWT	
13	BJ62 URY	PASS	BJ62 URY	
14	BJ62 URY	FAIL	----------	Shadowing on number plate causes merging during *adaptive thresholding*
15	V343 DYD	PASS	V343 DYD	
16	V343 DYD	PARTIAL	43 DYD	The first 3 has significant noise, causing it to be rejected by ratio analysis
17	W806 SLG	PASS	W806 SLG	
18	X297 LEB	PARTIAL	X2 97E	Image angle is poor, resulting in merging of characters with the surroundings during *adaptive thresholding*
19	X297 LEB	PASS	X297 LEB	
20	V344 DLT	PASS	V344 DLT	
21	P92 OUR	PASS	P92 OUR	
22	P92 OUR	PASS	P92 OUR	
23	W334 OES	PASS	W334 OES	
24	RV55 CXL	PASS	RV55 CXL	
25	RV55 CXL	PASS	RV55 CXL	
26	W833 UOC	PARTIAL	833UC	Noise on character O, caused it to be excluded during *ratio analysis*

27	HG62 PVA	PASS	HG62 PVA	
28	HG62 PVA	FAIL	----------	Reflections picked up by *CCA*, resulting in the wrong plate region to be chosen
29	W517 PDS	PARTIAL	W57 PDS	*Ratio analysis* failed to get character 1
30	VI SEO	PARTIAL	Region Detected	*Ratio analysis* failed to get character 1, resulting in only 4 characters
31	V93 FES	PASS	V93 FES	
32	V562 KWK	PARTIAL	V52 KWK	Small image resolution resulting in issues with *ratio analysis*
33	V562 KWK	PASS	V562 KWK	
34	RV14 PVX	PASS	V562 KWK	
35	RV14 PVX	PASS	V562 KWK	

Table 3: *Input test cases*

10. Review

The project has provided a great opportunity to learn about common image editing techniques and theories such as Hough Space, Projection Analysis, Edge Detection and Convolution Matrices. The usage of connected-component analysis came about by trail and error and testing and development of the techniques mentioned which were not providing a satisfactory result. Connected-component analysis worked well enough (and much better than other techniques) at detecting possible characters, allowing for detection of a plate. The following describes the techniques and methods used and further dismissed during the development phase as they were not reliable at detecting potential plates.

The first attempt at locating a numberplate was via the theory of edge and intensity projections. These allowed analysis of the image via numerical analysis, by finding peaks of either edges or intensity, predictions could be made of potential regions of the numberplate. This worked reasonably well, however failed terribly when the image was not entirely car, for example there was a significant background. It could however be used in conjunction with the current CCA implementation to provide extra heuristics in order to improve the extraction process.

The success of this method relied heavily upon the preprocessing of the image, a whole method of intelligently managing and apply convolution matrices was created in order to allow simple implementations of blurring the image and apply horizontal ranking to exaggerate edges for the projection analysis.

The second attempt at locating number plates was to break the image into sections, and then preform edge or intensity projections on the more isolated sections. This was achieved via the theory of hough space. By computing potential lines in an edge based image (using canny edge detection) a number of horizontal lines could be computed, merged and finally used to break the image into more manageable regions. While this worked most of the time, some images simply did not have enough distinguishable lines in order to segment the image or the process would end up cutting the numberplate in half before the projection analysis stage.

In conclusion, due to the amount of time invested in investigating the methodologies above during the learning process, the project still has plenty of room for improvement, namely improving the image cleaning techniques to provide ways of removing noise and creating consistently good results. It was also seen in testing that the ratio analysis could be improved to increase the probability of catching characters. The project has provided a great experience to explore the core concepts of image analysis and manipulation with real world images.

Simple ANPR Detection

This final project will be an Image Processing task, written in C++. The program will be capable of taking an input image (.jpg), and extracting the number plate. Due to the complexity of the task, only UK number plates which have minimum skew and a clear boundary should be extracted consistently and accurately. The program will have to extract a number plate, extract the characters then preform heuristic analysis in order to determine the characters in sequence.

Review of existing programs

OpenALPR

While ANPR is widely used around the world, there is only a few OpenSource ANPR system easily accessible on the web. Using the demo at www.openalpr.com/demo.html, images can be uploaded to detect number plates. The majority of the test images uploaded worked successfully, a boundary was located and the number plate displayed, however one did not, which was surprising. As a result, I will not use this image for my test cases, as this will hamper the success of the development.

ALPR includes a command line utility which allows it to be used easily with any system (such as calling it from another program), which allows multiple country codes, multiple number plate candidates and seeking into video files, as well as an option to automatically detect which type of number plate to detect. Having this program as a command line interface is a major advantage to the program as it can be used by anything.

JavaANPR

JavaANPR is software produced by Ondrej Martinsky at Brno University of Technology. The program is far from stable compared to OpenALPR but does okay in recognising number plates by adjusting skew and light conditions adaptively. The website javaanpr.sourceforge.net contains a documentation PDF which states key concepts and methods used in the project (however unfortunately lacks some significant details), including edge and intensity detection and other common algorithm which will assist the development of the project to be developed.

CARMEN ® ANPR Software

CARMEN ® software provides ANPR software which focuses on certain applications of number plate detection including 'FreeFlow', which can detect fast moving vehicles at a long range to 'Parking' or 'Parking Lane' which can assist in parking control, vehicle entry and access control in multiple lanes if required. The 'FreeFlow' software can quite successfully detect fast moving (up to 250km/h) vehicles over multiple lanes, with recognition for a wide range of characters, also deriving the vehicles speed with good accuracy however the software is costly and requires very high quality images in order to preform the task accurately.

Requirements

Functional Requirements

The program will take a command line argument to open a .JPG file in order for a number plate to be extracted. If the .JPG does not exist, a number plate cannot be found or characters cannot be determined reliable, exceptions will be called to terminate the program properly.

- Allow input of string to choose input image via command line
- Read JPG Image from disk
- Locate a number plate with reasonable confidence
- Extract characters with reasonable confidence
- Output number plate characters to command line
- If chosen, output an image with a draw boundary and characters to screen

Non-Functional Requirements

- The system should terminate within 1.5 seconds
- The system should be able to successfully detect >80% of number plate test cases (which also are successful with OpenALPR)

Specification

The program will be run via a command line interface, in the following format:

```
$ ./SimpleANPR [--display] <image1.jpg> [image2.jpg] [..]
```

The executable binary will take an optional variable of `--display`, which will show the images at certain stages of processing when using a graphical interface. Multiple images can be defined using a space seperated list. The output of the program will be as follows:

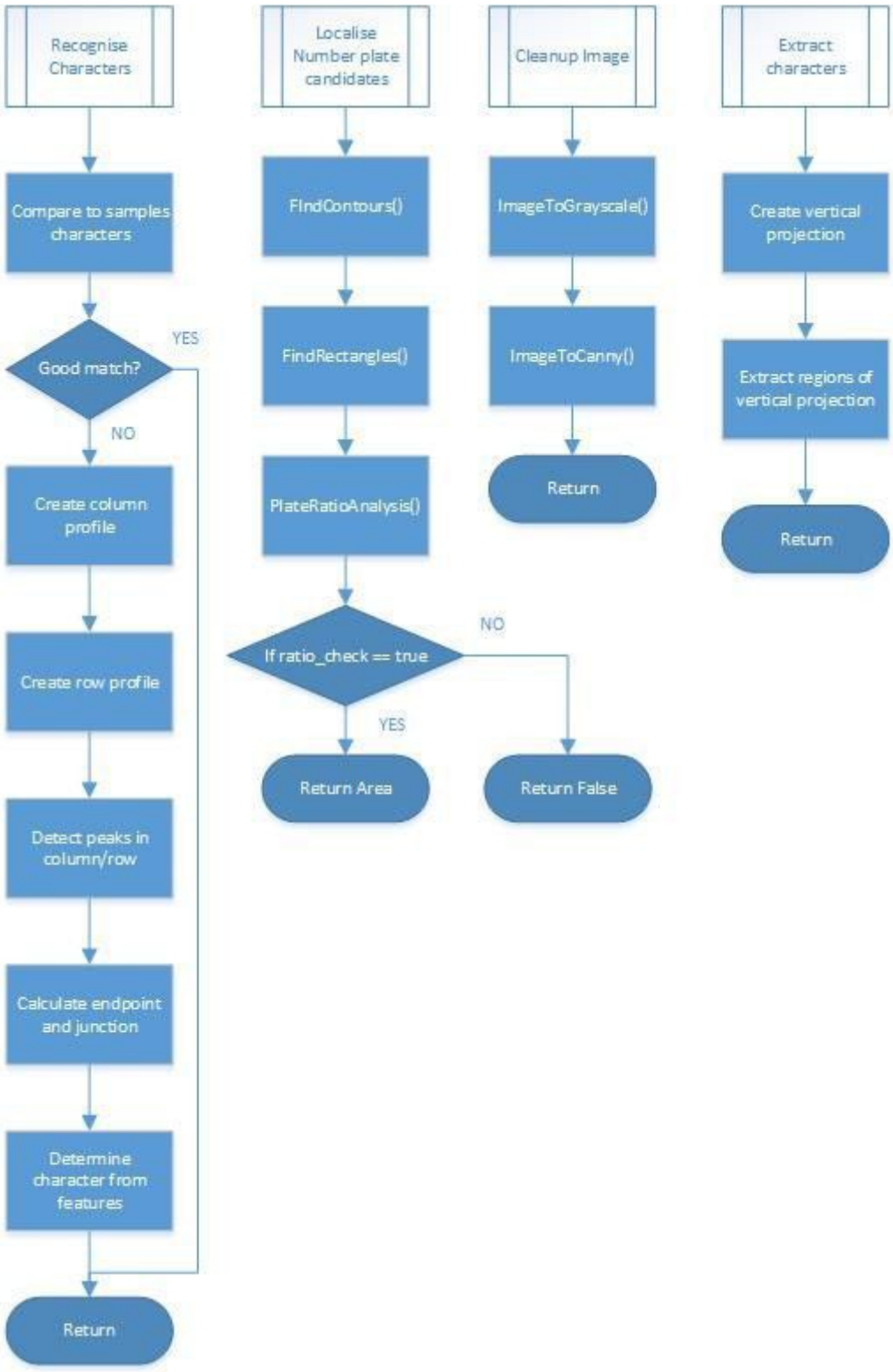
```
$ ./SimpleANPR car.jpg car2.jpg car3.jpg
car.jpg: WR05 CVO
car2.jpg: J98 257
car3.jpg: ESS 3X
```

Comparison to existing programs

The functionality of the program is very similar to OpenALPR due to the nature of the topic. This project however will use SDL to output images direct to the user if required, allowing for a more graphical interface compared to OpenALPR. The project also allows for a far simpler implementation compared to the CARMEN ® software which can be advantageous to people looking at learning about computer vision and detection.

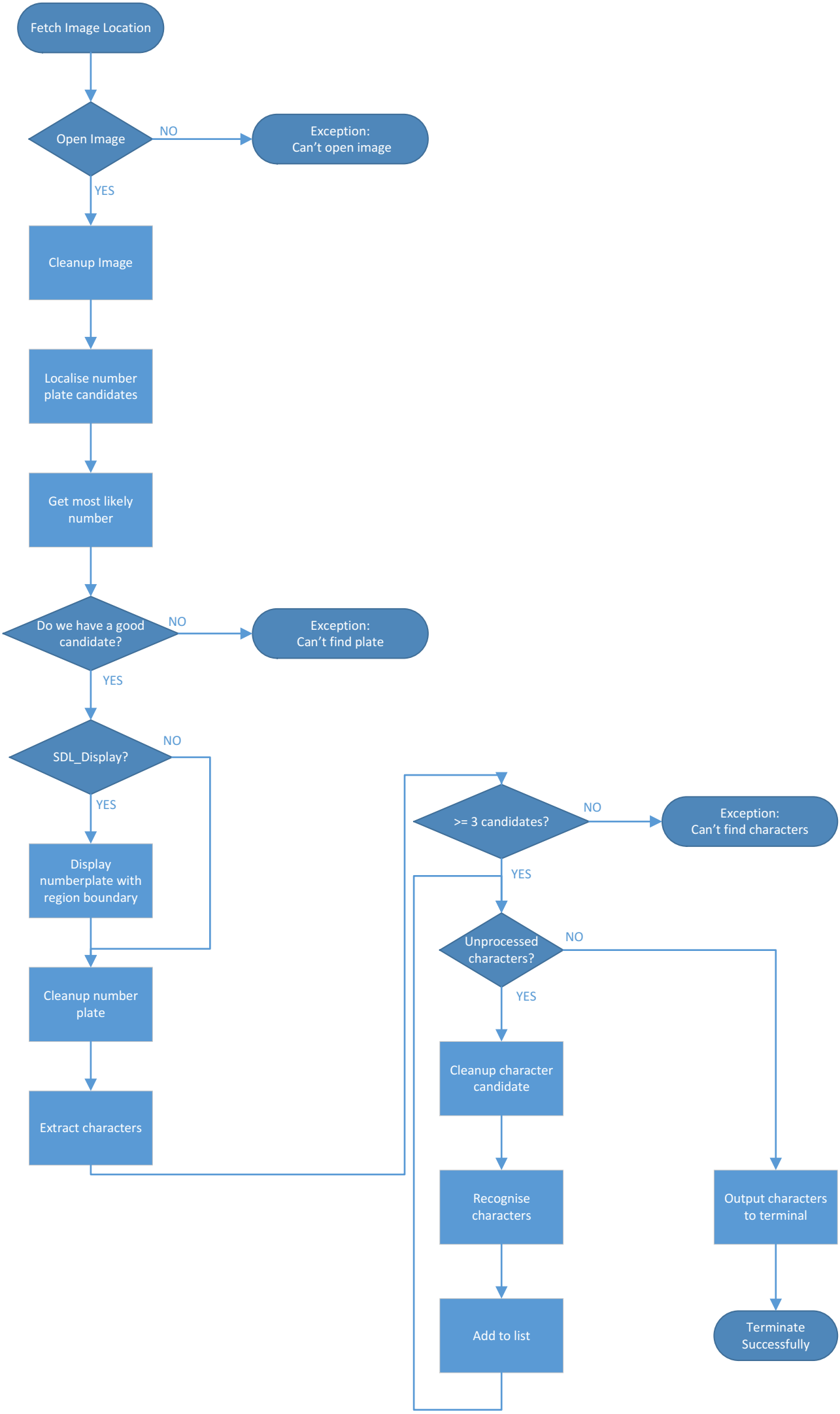
Design

The software is required to take a colour image, convert it to grayscale and preform analysis on the image in order to extract data such as number plate boundaries and characters in order to derive the number plate characters in ASCII form. Below shows a decomposition of some core tasks as shown in the flowchart further in this document.



High-level pseudocode

```
file = command line argument

if !open(file)

     Throw Exception

end

Cleanup Image
Localise numberplate candidates
Verify numberplate

if !numberplate_is_present

     Throw Exception

else

     if SDL_Display

      Display numberplate with region boundary

     end

end

Cleanup numberplate region
Extract characters

if (characters < 3)
     Throw Exception
end
for each character
     Cleanup character
     Recognise characters

     if SDL_Display
      Display characters with ASCII translation
     end

     Add ASCII character to list
end

Output character list to command line
```


Fetch Image Location
Open Image
NO
Exception:
Can't open image
YES
Cleanup Image
Localise number plate candidates
Get most likely number
Do we have a good candidate?
NO
Exception:
Can't find plate
YES
SDL_Display?
NO
YES
Display numberplate with region boundary
Cleanup number plate
Extract characters
>= 3 candidates?
NO
Exception:
Can't find characters
YES
Unprocessed characters?
NO
YES
Cleanup character candidate
Recognise characters
Add to list
Output characters to terminal
Terminate Successfully