UNIVERSITY OF DUBLIN

TRINITY COLLEGE

Mobile Phone Bus Time App

by

Brendan Hegarty

B.A. (Mod.) Computer Science
Final Year Project April 2015
Supervisor: Dr. Kenneth Dawson-Howe

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration of Authorship

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at any other university

Name: _____

Date: _____

"Solving big problems is easier than solving little problems."

Sergey Bin, Co-founder of Google

UNIVERSITY OF DUBLIN, TRINITY COLLEGE

Abstract

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland

B.A. (Mod.) Computer Science

by Brendan Hegarty

The aim of this project was to investigate the use of Android and computer vision together by developing an Android application which made use of computer vision. The specific function of the application was to allow a Dublin Bus passenger to take a picture of the stop sign containing the stop number and then quickly fetch the real time information for that bus stop.

The technologies used in this project are the Android Software Development Kit (SDK) and OpenCV. The Android SDK is used to take care of general Android application development and OpenCV is a computer vision library which is used for the image manipulation.

Acknowledgements

I would like to thank my supervisor, Dr. Kenneth Dawson-Howe, for giving me the opportunity to undertake this project. His constant support and guidance was an invaluable resource throughout the project.

I would also like to thank my friends and classmates for all the help and support they have given me during this project and over the past four years.

Finally I would like to thank my parents, Colm and Susan, who have been great to me throughout my whole life, and without whom none of this would have been possible.

Contents

Declaration of Authorship i

Abstract iii

Acknowledgements iv

List of Figures vii

1 Introduction 1
- 1.1 Aims . . . 1
- 1.2 Motivation . . . 1
- 1.3 Project Overview . . . 2
- 1.4 Overview of Report . . . 2

2 Background 4
- 2.1 Android . . . 4
  - 2.1.1 History . . . 4
  - 2.1.2 Development . . . 5
- 2.2 Computer Vision . . . 6
- 2.3 OpenCV . . . 7
- 2.4 OpenCV4Android . . . 8
- 2.5 Optical Character Recognition . . . 9
- 2.6 Tesseract . . . 9
- 2.7 Conclusion . . . 10

3 Design and Implementation 11
- 3.1 Android Development . . . 11
- 3.2 Android Activities . . . 11
  - 3.2.1 Main Activity . . . 12
  - 3.2.2 Image Activity . . . 12
  - 3.2.3 Results Activity . . . 13
- 3.3 Image Processing . . . 14
- 3.4 Sign Recognition . . . 14
  - 3.4.1 Back Projection . . . 15

3.4.2 Find Largest Contour . . . 16
3.4.3 Difference Image . . . 17
3.5 Digit Reading . . . 18
3.6 Obtaining RTPI . . . 18
3.7 Alternate Techniques Explored . . . 19
3.7.1 Sign Recognition . . . 19
3.7.2 Digit Reading . . . 20
3.7.3 Obtaining RTPI . . . 20
3.8 Conclusion . . . 21

4 Future Work **22**
4.1 GPS . . . 22
4.2 City Center Signs . . . 22
4.3 Luas/DART Functionality . . . 23
4.4 Conclusion . . . 23

5 Conclusion **25**

Bibliography **26**

Electronic Sources and Resources (Attached CD) **26**

List of Figures

2.1 Number of app downloads from Google Play from August 2010 to July 2013 5
2.2 Distribution of Android versions throughout the world between December 2009 and March 2015 6
2.3 Number of OpenCV downloads from 15/3/2001 until 31/3/2015 7
2.4 Number of OpenCV4Android downloads from 1/1/2012 until 31/3/2015 . 9

3.1 Screenshot of the Main activity 12
3.2 Screenshot of the Image activity 13
3.3 Screenshot of the Results activity 14
3.4 Sample image of a Dublin Bus stop sign 15
3.5 Resulting image from back projection - white represents yellow present in the original image 16
3.6 Largest shape in the image 17
3.7 Difference between the back projected image and the largest contour image 18
3.8 Example of a sign used for template matching in sign recognition 19

4.1 Two different types of stop signs: the ones found in the suburbs (left), and the one found in the city centre (right) 23

Chapter 1

Introduction

This chapter will outline the aims and motivation behind the project and a general overview of both the project itself and this report.

1.1 Aims

The aim of this project was to investigate OpenCV4Android, which is a relatively new technology. This is an Android port of computer vision libraries to do image manipulation.

In order to do this investigation, I developed an Android application which uses techniques from the previously mentioned computer vision libraries to read the stop number from a Dublin Bus stop sign and present the user with the Real Time Passenger Information (RTPI) for that stop.

1.2 Motivation

A lot of modern research goes into maximizing connection speeds in developed countries so that

In public transportation systems all over the world, there exists signs to signify where passengers can get on or off. Whilst these signs exists everywhere, they have very little use as other than when someone new to that system (people who recently moved residence, tourists, etc.) needs them to tell whether they have reached their destination yet or not.

In cases where there exists a Real Time Passenger Information (RTPI) system, information could be read from these signs, which could then be used to obtain the real time information for the passenger. Nowadays, more and more people carry smartphones with them wherever they go, so developing an application which would do this is quite feasible.

Computer Vision has many techniques which allows this to be possible. Even moreso now because it is now possible to use Computer Vision techniques in both Android and iOS applications.

1.3 Project Overview

For this particular project I looked at the real time information for Dublin Bus. Dublin Bus have real time information for all of its stops in Dublin. In the city center, it even has a digital display at most stops which displays this for passengers.

However, outside of the city there exists no such displays, and if passengers would like to know when the next bus is coming, they would have to open up the Dublin Bus app, go to the relevant search option, and input the route number or stop number. This can take a while as, depending on what the passenger does, the Dublin Bus app could end up making multiple calls to the Dublin Bus API.

I created an Android application which would allow the passenger to simply take a picture of the Dublin Bus stop sign, and the application would read the stop number from it and fetch the real time information. Once it reads the stop number from the image, it makes a call to the Dublin Bus API which returns the information for that stop. This returned information gets parsed and the relevant bus stop details are presented to the passenger in a simple table.

1.4 Overview of Report

The next chapter will describe the different technologies used for this project. It will also talk about similar apps like this on the market.

Then, Chapter 3 will talk about design choices for each step of the application and how each of these steps work with each other.

Chapter 4 will briefly describe some possiblities for future development of the application, including possible features that it is currently lacking.

Finally, there will be a ‘Conclusion’ chapter which will summarise the results of my project and what I was able to achieve from doing it.

Chapter 2

Background

In this chapter I will talk about the technologies required to create the application for which this report is about, specifically about Android development itself, and libraries required for computer vision.

2.1 Android

In terms of this project, Android refers to the mobile operating system which is developed by Google and is based on the Linux Kernel. Its main use is for touchscreen mobile devices such as smartphones and tablets, but it can also be used to make specialised user interfaces for televisions and wrist watches.

In July 2013, the Google Play store hit the one million mark for the number Android applications published, with over 50 billion downloadeds of these apps (see figure 2.1).[1] In 2014, Google announced that there were over one billion active monthly Android users, which is approximately double what it was the previous year.[2]

Google have released Android's source code under open source licenses, which means that it is free for everybody to access. This has resulted in a larger community of developers using the open-source code code as a foundation for community-driven projects. Android's success has gotten it involved with the "smartphone wars" between technology companies.

2.1.1 History

The Android operating system originally started its life with Android, Inc., which was a small company founded by Andy Rubin, Rich Miney, Nick Sears, and Chris White,

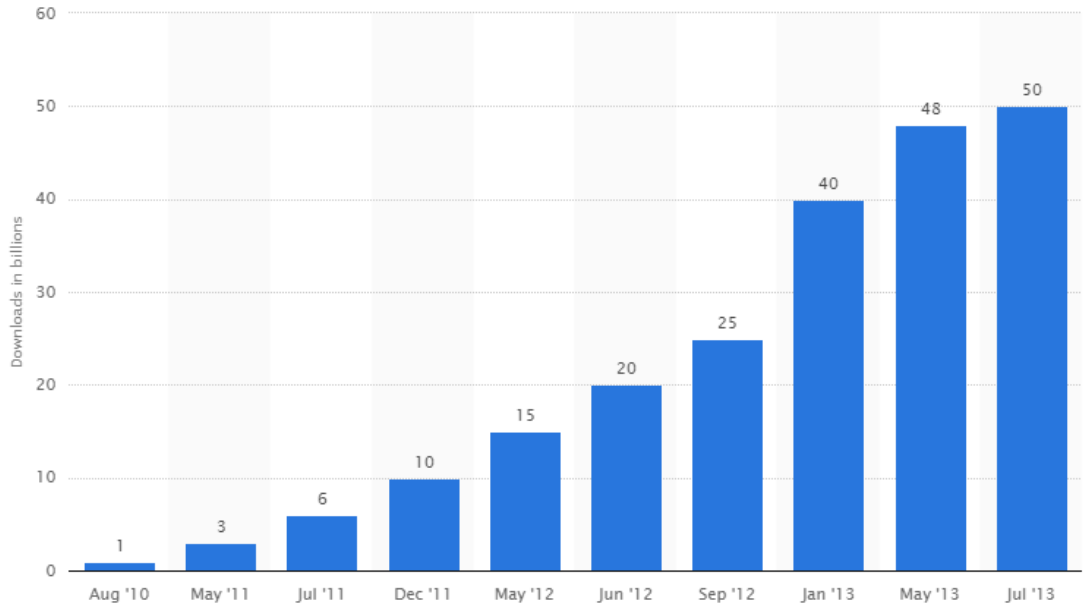

FIGURE 2.1: Number of app downloads from Google Play from August 2010 to July 2013

in October 2003. The company's original aims where to make an advanced operating system for digital cameras to help improve the connection between cameras and PCs.[3] Despite the fact that the founders of Android, Inc. had been very successful in the past, they decided to operate the company in secret, only saying that they were working on software for mobile phones.

In August 2005, Google bought Android, Inc. with a number of the original founders staying with the company. Rubin then led a team which developed a mobile device platform using the Linux Kernel. Later, towards the end of 2007, Android was finally unveiled by the Open Handset Alliance[4], with the first smartphone running Android being the HTC Dream which was released the following year.

There have been many updates to Android since its release in 2008, both for bug fixes and for the addition of new features. So far there have been 10 major versions of Android, with the latest one being Android 5.0 "Lollipop". Figure 2.2 shows the distribution of Android versions being used at a given point between December 2009 and March 2015. Every major version of Android has been named in alphabetical order after a dessert or sugary treat.

2.1.2 Development

The development of Android versions and updates are done in private by the team at Google. It is only when it is ready to be release to the public that the source code is

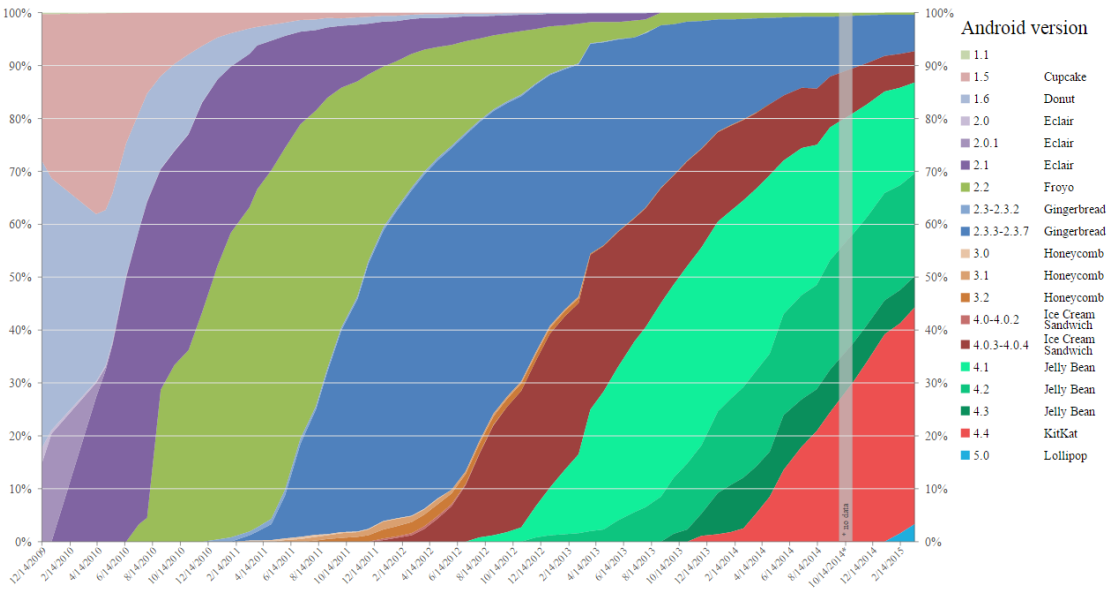

FIGURE 2.2: Distribution of Android versions throughout the world between December 2009 and March 2015

made available to the public. Google aims to have a major update to Android every six to nine months.

However, depending on the device in question updates can be slow to reach it. Being Google's own product and the flagship device for Android, the Nexus brand of phone and tablet receive updates as soon as they are released. For other brands, it can take a couple of months until the update is available. The reasoning for this is that there can be an extensive variation in hardware between different Android devices, so each update needs to be tailored for that device.

The kernel used by Android is based on the Linux kernel's long-term support branches, generally being either version 3.4 or 3.10. The kerner version varies depending on the actual Android device.

2.2 Computer Vision

Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from either a single image, or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding. The goal of computer vision is to model, replicate, and exceed human vision using computer software and hardware at different levels. It is concerned with the theory behind artificial systems that extract information from images, which can take many forms such as video sequences or views from multiple cameras.

The first attempt to solve the problem of computer vision was made by Seymour Papert in MIT.[5] It was believed that it would be a relatively easy problem to solve so this first attempt was referred to as "The Summer Vision Project" which took place during the summer of 1966. The aim of this project was to develop a vision system that would dividea a vidisector picture into multiple regions such as likely objects and likely background areas.

Unfortunately this project wasn't as successful as everybody thought it would have been. The reason for this is that computer vision is a lot more complicated that everybody thought it would be. The human visual system is a very complex system. According to cognitive research the human brain is devoting between 40-70% of its capacity into processing the visual signal that it receives from the eye.

People spent the next few decades trying to solve all the problems which they were being faced with. It wasn't until the 1990's that real progress was made. Suddenly there were many methods being proposed for these computer vision problems and what we know to be modern computer vision was born.

2.3 OpenCV

OpenCV is an open source computer vision and machine learning library. It is free for use under the open-source BSD license. The OpenCV library currently has over 2,500 optimized algorithms, which includes sets of state-of-the-art computer vision and machine learning algorithms. It is used by over 47,000 people and has approximately 6.23 millions downloads since it was released in 2001 (see figure 2.3).

OpenCV sees use from companies of all sorts, ranging from well-established companies like Google, Intel, Microsoft and IBM, to many startups like VideoSurf, Applied Minds, and Zeitera. Research groups and government bodies also make extensive of OpenCV.

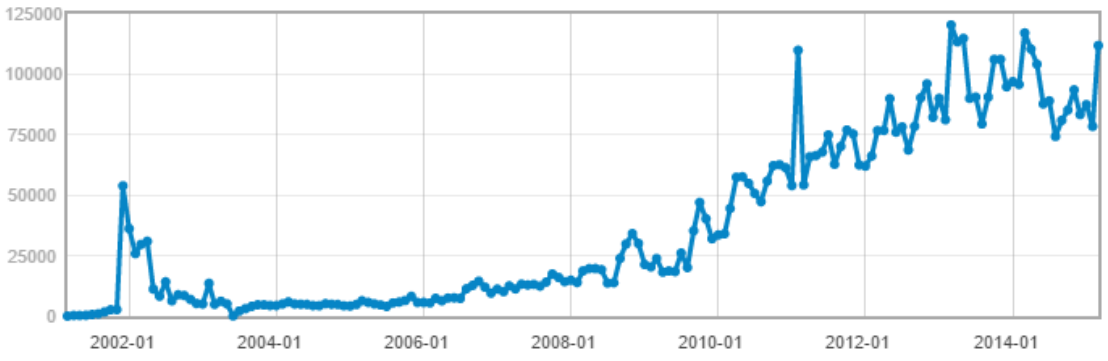

FIGURE 2.3: Number of OpenCV downloads from 15/3/2001 until 31/3/2015

The OpenCV project began its life in 1999 as an Intel Research initiative to advance CPU-intensive applications. It was part of a series of projects including real-time ray tracing and 3D display walls.

The library is written in C++, with its primary interface being for C++ and C. It also contains interfaces for Java, Python and MATLAB. It currently has support for Windows, Android, Linux, OS X, iOS, FreeBSD, OpenBSD, Maemo and Blackberry 10.

The first alpha version was released at the IEEE Conference on Computer Vision and Pattern Recognition in June of 2000. It then went through five beta test periods between 2001 and 2005, with the first 1.0 version being released in 2006. Willow Garage, a robotics research lab and technology incubator, began to support OpenCV in 2008 meaning it was under active deployment again. A verion 1.1 "pre-release" was then released in October 2008.

2009 saw the second major release of OpenCV. This included major changes to the C++ interface which added some new implementations and the improvement of existing ones. Development of the libraries is now taken care of by a Russian team who push for official releases every six months or so. A non-profit organisation known as OpenCV.org took over support of the OpenCV project in August 2012 and currently maintain both a developer and user site for the library.

2.4 OpenCV4Android

OpenCV4Android is the name given to the Android port of OpenCV. In 2010 Ethan Rublee, a member of Willow Garage, developed the first build of the Android port. This first "alpha" version was not available to the public however. In June of 2011, NVIDIA began to support this new project. The OpenCV Android port was greatly improved upon with this new support, including the addition of full camera support.

The port was ready to be entered into its first beta test period that July. During this period, a number of more improvements were made. The Android Java API was made for OpenCV meaning that approximately 330 OpenCV functions could now be used inside an application. August of the same year saw the Android port's second beta test period. This brought the number of OpenCV functions that could be used up to 700, included some bug fixes and included OpenCV native camera support to more Android devices.

April 2012 saw the first full release of OpenCV4Android. Since it's release the Android port has seen 7 minor updates, with the last of these being in November 2013. Each

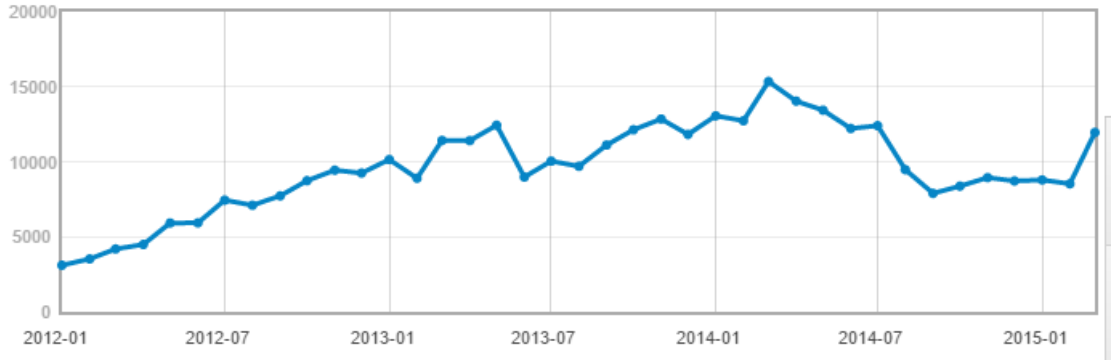

FIGURE 2.4: Number of OpenCV4Android downloads from 1/1/2012 until 31/3/2015

of these updates were for minor issues such as bugs, adding camera support for more Android devices, etc.

Since 2012 there have been over 370,000 downloads of OpenCV4Android. As popularity of Android increases, so does the interest in developing applications with computer vision.

2.5 Optical Character Recognition

Optical character recognition (OCR) is the process of converting images containing text into machine-encoded text. It's main use is as a form of data entry from printed records to that they may be edited electronically, searched, and stored more compactly. OCR is a field of research in pattern recognition, artifical intelligence and computer vision. Optical character recognition can be traced back to the early 20th century. Since then, there has existed OCR technologies for use with telegraphy and for creating reading devices for the blind.[6]

Back in 1914 a Russian physicist and inventor by the name of Emanuel Goldberg developed a machine which could read text and convert the characters into standard telegraph code. During the late 1920s and early 1930s he was developing a machine which could be used to search microfilm archives using an optical code recognition system.

2.6 Tesseract

There exists a large number of OCR engines today. The one being used for this project is known as Tesseract. Tesseract is a free software that is released under the Apache License, Version 2.0. Since 2006, Google have sponsored its development. The reason I

chose this was because Tesseract is generally considered to be the most accurate open source OCR engine currently available.[7]

Tesseract started off life as proprietary software at Hewlett Packard labs during the mid 1980s, with it being finished in 1994. Between then and 1998 it had gone through some improvements including a port to Windows, and migration from C to C++. It then stayed in that state for nearly a decade until it was released as open source in 2005.

2.7 Conclusion

To summarize, there are a number of different technologies required to produce this application. Even though it hasn't been around for too long, the Android operating system continues to grow more and more every year. With this increase, so too does the interest in the development of applications for the operating system. The same could be said for the use of computer vision, even if it has been around for slightly longer. Despite the fact that Android development and computer vision are two completely different fields, a community exists to try and bring the two together, as shown by the fact that the Android version of the computer vision libraries have been downloaded over 370,000 times in the last 3 years.

Chapter 3

Design and Implementation

This chapter will outline design choices and how I implemented the application. First I will talk about the development of the Android application itself, then I will move onto individual techniques used for different parts of it.

3.1 Android Development

The development of Android applications is done using the Android Software Development Kit (SDK). This SDK comes with all the necessary development tools needed, including all the relevant libraries, a debugger and an Android emulator. The main platforms which are currently supported for development or Android are any modern desktop Linux distribution, Windows XP or later, and Mac OS X 10.5.8 or later.

The languages used in Android development are Java and XML. The entire back-end functionality, including the use of the computer vision libraries, is done in Java. XML is used for both the front-end design & layout of the pages, and for the general configuration of the application.

3.2 Android Activities

Android applications have components known as "activities". Activities are essentially the different pages that you would see. Each of these have both a Java file which handles all of the back-end functionality, and a corresponding XML file which handles the front-end design & layout.

3.2.1 Main Activity

This is a very basic activity which acts as a landing page. It is what gets displayed when the user first opens the app. When the user selects the Take Picture button, the activity will launch a behind-the-scenes activity which loads the default camera and saves the resulting picture. After this, it will return to the main activity with the result image. This will then call the camera result activity and the image will then get passed into it.

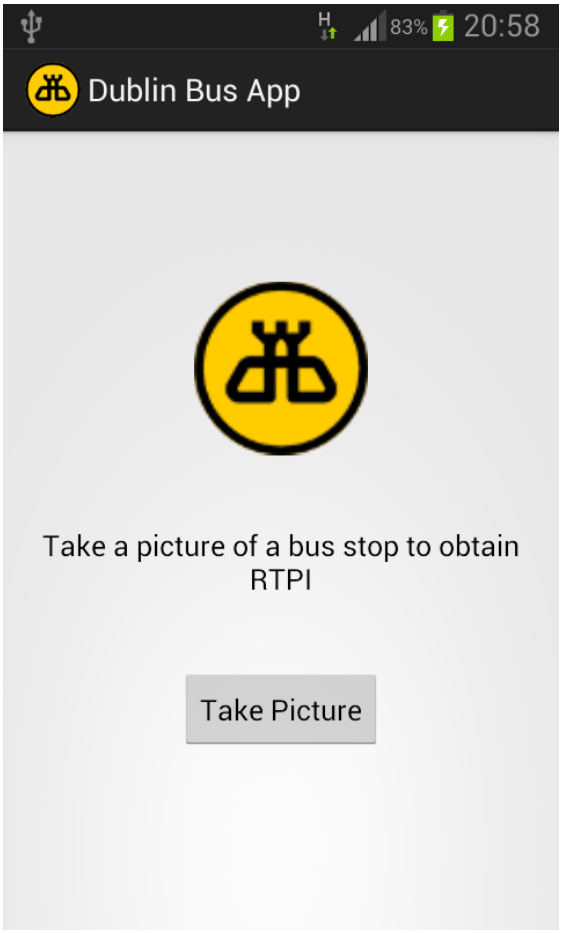

FIGURE 3.1: Screenshot of the Main activity

3.2.2 Image Activity

As I mentioned previously, this activity gets launched after the user takes a picture with a camera. All of the image processing (which I will talk about in a second) takes place in this activity.

Before doing this, the activity sets the camera image to be displayed on screen, just so the user can confirm the image is correct. It will also show the user what number the app obtained from the image, also letting the user confirm it. If either of these are wrong,

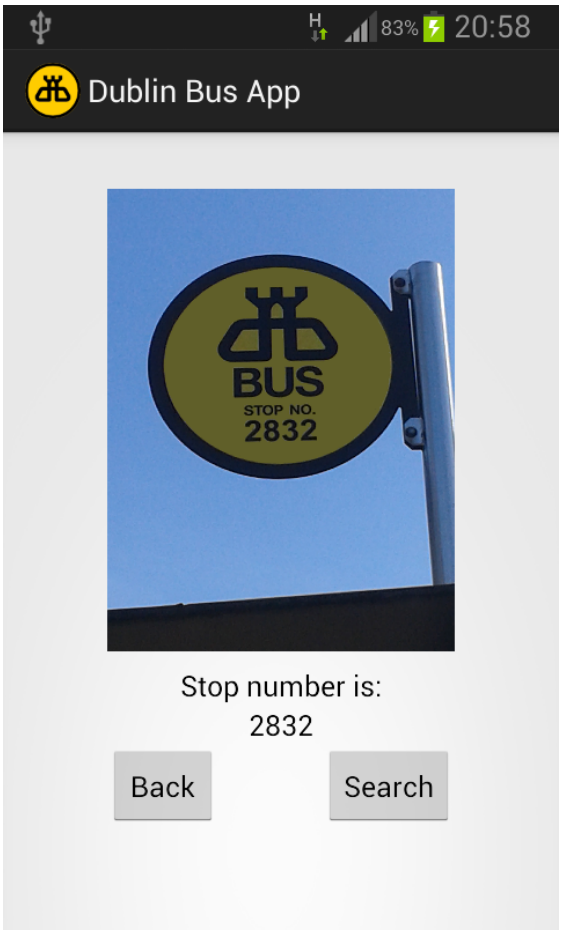

FIGURE 3.2: Screenshot of the Image activity

the user can simply use the Back button which will take them back to the landing page, and they can re-take the image.

3.2.3 Results Activity

Finally, we have the RTPI results activity. Similarly to the landing page, this activity is very basic. It only contains a table, which gets populated based on the results of the Dublin Bus API call. It will only display the first 10 results (if applicable). The user can then return back to the previous activity, allowing them to go through the whole process again.

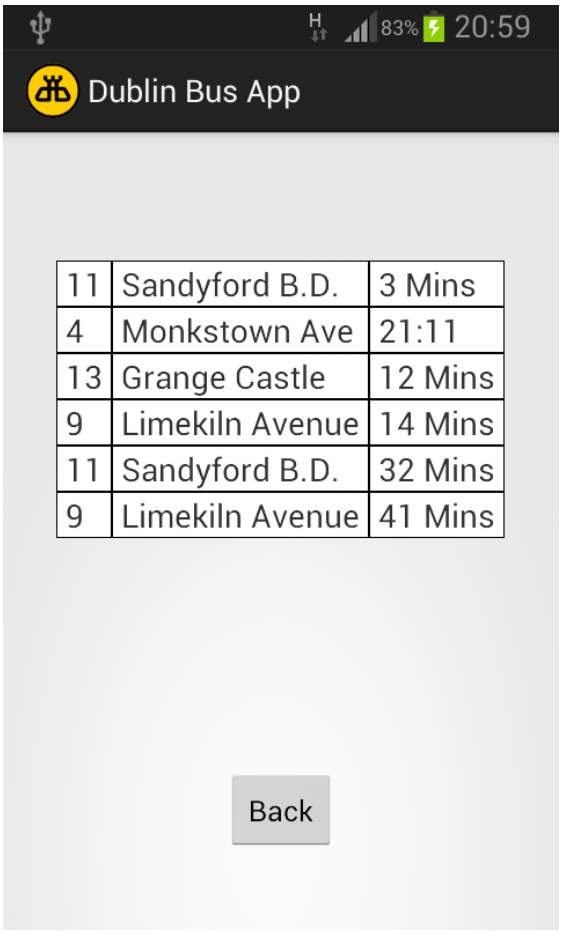

FIGURE 3.3: Screenshot of the Results activity

3.3 Image Processing

Image processing, or image manipulation, is used to the stop number from an image of a Dublin Bus stop sign. As mentioned in the previous chapter, OpenCV is the library used for doing this. The image processing for this application is essentially split into two parts. The first one is the process of actually recognising a Dublin Bus stop sign and getting a proper image down to a binary image which just has the relevant text. Once this is done, the second process of actually reading that text and converting it into a string can begin.

3.4 Sign Recognition

As mentioned previously, the first part of the image processing is to actually recognise a Dublin Bus stop sign. This part has three steps to it, all of which are described below in detail.

FIGURE 3.4: Sample image of a Dublin Bus stop sign

3.4.1 Back Projection

Back projection is a computer vision technique to record how well the pixels of a given image fit the distribution of pixels in a histogram model. It is essentially used to find a certain colour in an image. For back projection, a sample set of images is required. In this case, it would be images containing different shades of yellow. A histogram is generated based on the colours of each pixel in the image. This histogram then gets normalized with a maximum value of 1.0, which allows the values to be treated as probabilities. The now-normalized histogram then gets projected onto the image taken my the camera providing a probability image where the probability of each pixel is the similarity between the two images.

This process will result in an image where the whiter the pixel, the higher the probability is of that pixel being yellow. Once this image has been obtained, it goes through a binary thresholding where any pixel that is not black is set to be white. This gives a binary image where white represents any shades of yellow present in the original image, and where black represents everything else. You can see this resulting image in figure 3.4.

My reasoning for using back projection is that it doesn't narrow down the application to be used with only these signs. As you will see later on in chapter 4, I list a future feature as adapting the application or use with the Luas and/or the DART. Back projection

FIGURE 3.5: Resulting image from back projection - white represents yellow present in the original image

can be used to find any colour that is wanted, so that it could potentially be used for these two in future development.

3.4.2 Find Largest Contour

After this is all done, the app will find all contours in the image. In the context of computer vision, a contour is actually a shape and not an edge. Going through each of these, it will keep track of the area of each one, and at the end it will remember what the largest one was. Assuming the user of the application aimed directly at the image and close enough for it to be clear, the largest contour should be just the circle of the bus stop sign. This largest contour then gets drawn as a binary image which you can see in figure 3.5.

The idea behind this technique is that, assuming the user is making sure the picture they take of the bus stop is clear, the largest thing in the image should be the sign. The OCR engine needs the text to be in white not black (the opposite to what it is after back projection), so this image can be used to swap around the colours within the area of the sign.

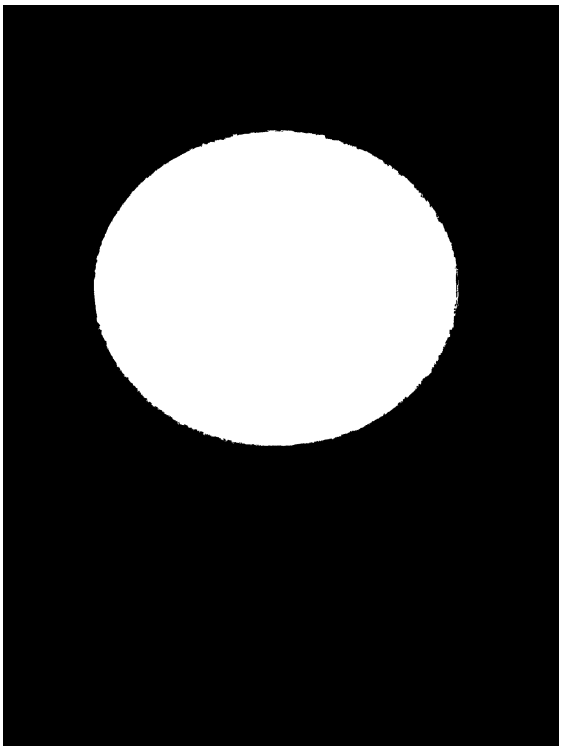

FIGURE 3.6: Largest shape in the image

3.4.3 Difference Image

This next process is as simple as it sounds. It is just a simple comparison between the two images from the previous steps, and getting the differences between them. The two images are compared pixel-by-pixel, and in the new image the corresponding pixel will be white if there is a difference between the images, or black if they are the same. The idea of this is that it will eliminate the yellow of the sign and present us with just the text that is present on the sign, as seen in figure 3.7. This new image is now ready to be processed by the OCR engine which will return a string of all the text in the image.

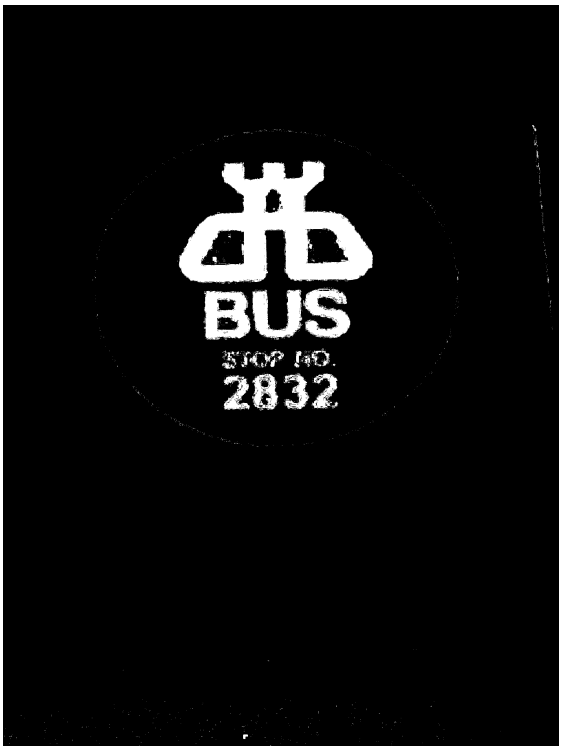

FIGURE 3.7: Difference between the back projected image and the largest contour image

3.5 Digit Reading

The second step of the image processing is to actually process the binary text image and pull the text out of it. This is done using an optical character recognition engine called Tesseract. Tesseract is an open source OCR engine. It is generally conceived as being the most accurate open source engine currently available. Using the Leptonica Image Processing Library, Tesseract can read images in a wide variety of image formats and convert them to text in over 60 languages. Tesseract works on Windows, Linux and Mac OS. It can be compiled for other operating systems, including Android and iOS. Unfortunately these are lacking in some features that Tesseract has, and they arent as well tested the main PC operating systems.

3.6 Obtaining RTPI

Dublin Bus have their API open for all to use. It is a simple process of sending a HTTP GET request to the APIs URL. This gives back a bunch of XML, which then gets processed and the RTPI information gets parsed into a 2 dimensional String array. This 2D array then populates an Android table simply displaying the route, destination and time remaining.

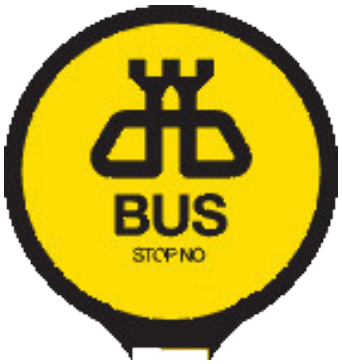

FIGURE 3.8: Example of a sign used for template matching in sign recognition

3.7 Alternate Techniques Explored

During this project, I had been investigating multiple different techniques for each step of the project, for both the image processing and the process of obtaining the real time information.

3.7.1 Sign Recognition

For recognising a Dublin Bus sign there was two techniques which I had looked at, other than back projection which I used. These two techniques are template matching and making use of a HSV image.

Template Matching

Template matching is a process where, given a sample image, it searches a second image for anything which matches this. In the case of this project, the sample image would be a sample Dublin Bus stop sign, such as the one in figure 3.8. The idea would be that it would simply know the location in the image where the sign is and only search for text in this region. Unfortunately this process did not work for me. If the sample image does not contain a stop number then the application would search for a sign that also did not have a stop number. Alternatively, if a stop number was present in the sample image then it would look for a stop with that number present in the main image.

Using HSV Image

A method which I found while browsing online was to make use of an image in the HSV (hue-saturation-value) colour space. With OpenCV, converting to this from the RGB colour space is a simple process. With this HSV image, it's possible to remove

everything from the image that are outside of a certain range of HSV values, resulting in only the yellow being present. The big problem with this method is finding an ideal range. Depending on current lighting a sign could easily fall outside of the set range even though it should be accepted.

3.7.2 Digit Reading

For this project, the only method I had actually tried to implement was the use of an OCR engine. However, in case it ended up not working out for me I had looked into a couple of other methods: using SPR and template matching like before. The main problem with these two methods is that while they would work to get individual digits, more work would possibly be needed to make sure all the obtained digits are in the correct order.

Statistical Pattern Recognition (SPR)

Statistical pattern recognition is the process of recognising patterns based on the probablilty that a certain feature, or features, occur. Something that could be an issue with SPR is that some numbers which are similar to each other, like 1 and 7, could end up getting mixed up, resulting in the user being presented with real time information for the wrong bus stop.

Template Matching

In a similar way to looking for a sign, a possiblity for recognising the stop number could be template matching. Instead of just one image, it could go through 10 images, starting at 0 and going to 9, until it finds a match in the image. A possible problem with this is that, similarly to using SPR, it could get mixed up with some digits, such as 1 and 7.

3.7.3 Obtaining RTPI

Originally when making the application, my plan was to use a SOAP request to obtain the real time information from Dublin Bus. Unfortunately, this turned out to not be possible inside an Android application.

SOAP Request

During inital development of the Java back-end for the application I had been obtaining the Dublin Bus RTPI through a SOAP request. I ran into a problem when moving it all over to the Android problem. The Java libraries which exist for Android are actually stripped down versions of the standard Java library. One of the missing packages from the Android version was one that is required to make SOAP requests. This meant that I had to scrap that idea and move to a simple HTTP request instead.

3.8 Conclusion

In this chapter, the various methods I implemented for the development of the application were described. To start off, the application's activities, which are the actual inner workings of the application, have been described. These can be seen in figures 3.1, 3.2 and 3.3. Following this, the computer vision techniques used and reasons for choosing them are then discussed, with images showing each step of the image processing step (see figures 3.5, 3.6 and 3.7). Finally, there are brief descriptions on techniques which I had been investigating initally but then decided to drop from the project and reasons why.

Chapter 4

Future Work

There are many ways in which this application could be improved upon. In this chapter, I will talk about some of the features for the application which are not yet present, but could be implemented in future development.

4.1 GPS

Initially, I had been looking into using the mobile phone's GPS location in relation to narrowing the what the possible bus stop could be. The reason for this feature not being done as part of the actual project is that the GPS co-ordinates are not readily accessible to the public with the calls for the real time information. The GPS co-ordinates for each bus stop must exist in some form because online mapping services, such as Google Maps, have the location of all the bus stops on the map. It may be possible to obtain this information by contacting Dublin Bus and asking about it.

Once the GPS information can be attained, the application itself could pretty much be transformed. Instead of the user taking a picture of the bus stop, they could simply make the application search for what bus stops are nearby to the phone's GPS location and present the user with a list of these stops. Assuming that the stop the user wants is listed, they can then simply tap that stop on the list and they will then be presented with the real time information for that stop.

4.2 City Center Signs

There currently exists two different types of Dublin Bus stop signs, depending on if it is in the city centre or in the suburbs. Figure 4.1 shows these two different signs.

FIGURE 4.1: Two different types of stop signs: the ones found in the suburbs (left), and the one found in the city centre (right)

At the moment, the application is designed to only work with the type of stop found in the suburbs. The application needs further work to be carried out on it to support the city centre bus stop signs. Whilst the Computer Vision side of things would be largely the same, the resulting string from the Tesseract OCR engine would need to be parsed differently because of the different layout of the sign.

4.3 Luas/DART Functionality

While this application only works for Dublin Bus, the technology behind it could easily be adapted into either a separate application for the Luas and the DART, or even to implement these two public transport systems into the current application. In a similar way to the bus stops, both the Luas and the DART have signs at every stop, and real time information available for their respective stops. This would involve some previous investigating to see whether or not the real time service for these two systems is publicly available or not.

4.4 Conclusion

In this chapter, you can see some features which could be done in future development of the application. The features are both ideas which I had originally hoped to implement

as part of the project from the beginning, and ideas which are just possibilities based on the current setup of the application.

Chapter 5

Conclusion

In this chapter I will talk about the results of the project. The overall aim of this project was to investigate OpenCV4Android by developing an Android application which also made use of computer vision.

The main result of the project is that there is in fact a working Android application that uses computer vision. A user can take a picture of a Dublin Bus stop sign and they will get presented with the real time information for that stop.

However, it works on a very basic level. There are a few things that would need to be tidied up if the application were to be release-ready. For example, as I mentioned in section 4.2 of the previous chapter, the current state of application is only designed to work with the type of signs found in the suburbs of Dublin and beyond. The application would also need to undergo some general tidying up, both to clean up the user interface and to properly make sure the application works no matter what the external lighting conditions are.

From this project, I was able to learn about OpenCV4Android. As I have mentioned previously, it is quite a new technology. Despite this, it has come along way in its short lifetime. Compared to the main versions of OpenCV, the Android port can be used for nearly anything. It is missing some of the lesser used functions and techniques, but it does contain all of the common functions which are used most often.

Bibliography

[1] Victor H. Android's google play beats app store with over 1 million apps, now officially largest. July 2013. URL http://www.phonearena.com/news/Androids-Google-Play-beats-App-Store-with-over-1-million-apps-now-officially-largest_id45680.

[2] Justin Kahn. Google shows off new version of android, announces 1 billion active monthly users. June 2014. URL http://www.techspot.com/news/57228-google-shows-off-new-version-of-android-announces-1-billion-active-monthly-users.html.

[3] Chris Welch. Before it took over smartphones, android was originally destined for cameras. April 2013. URL http://www.theverge.com/2013/4/16/4230468/android-originally-designed-for-cameras-before-smartphones.

[4] Industry leaders announce open platform for mobile devices. November 2007. URL http://www.openhandsetalliance.com/press_110507.html.

[5] Efstratios Gavves. December 2014. URL http://www.egavves.com/a-brief-history-of-computer-vision/#sthash.DZGX7QMC.dpbs.

[6] Herbert F. Schantz. The history of ocr, optical character recognition. 1982.

[7] Nathan Willis. Google's tesseract ocr engine is a quantum leap forward. September 2006. URL http://archive09.linux.com/articles/57222.