Camera Phone Sign Translator

Darren Mason
B.A.I. Engineering
Final Year Project April 2009
Supervisor: Dr. Kenneth Dawson-Howe

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

I hereby declare that this project, in part or in whole, has not been submitted to any University as an exercise for a degree. Furthermore, this project is entirely my own work, except where reference is given in the text.

__________________________________

Darren Mason

24th April 2009
Acknowledgements

I would like to thank my supervisor, Dr. Kenneth Dawson-Howe, for all of his advice throughout this project.

I would also like to thank all of my friends and family for taking such an interest and supporting me along the way.
Abstract

With the relatively recent introduction of powerful smart-phones, the general public has become aware of the usefulness of mobile phone applications. These were not as popular in previous years even though the phones were perfectly capable of supporting such applications. Even with the current rise in mobile phone development there is a lack of applications that utilize the integrated camera as an input device.

The aim of this project is to develop a working system that allows a standard camera phone user to take a picture of a sign and translate its text into another language.

The system consists of software for the mobile phone to interface with the user and take a picture, while all of the resource intensive processing is performed on a server. The server extracts text from the image and translates it into a different language.

The camera phone sign translator fulfils the aims of this project, capable of translating signs from English into a number of different languages. With a good success rate already achieved and suggestions for further development included, this project has the potential to be a robust and practical mobile phone application.
Table of Contents

DECLARATION ........................................................................................................II

ACKNOWLEDGEMENTS .....................................................................................III

ABSTRACT ..............................................................................................................IV

TABLE OF CONTENTS ............................................................................................V

1. INTRODUCTION ..............................................................................................1
   1.1 AIMS ...........................................................................................................1
   1.2 OVERVIEW ...............................................................................................2
       1.2.1 Capturing Image .................................................................................3
       1.2.2 Performing OCR ..............................................................................3
       1.2.3 Translating Text ..............................................................................3
       1.2.4 Displaying Result ............................................................................3
       1.2.5 Performance ....................................................................................4
   1.3 MOTIVATION ..............................................................................................4

2. BACKGROUND ..................................................................................................5
   2.1 SIMILAR PRODUCTS ...............................................................................5
   2.2 AVAILABLE OCR PACKAGES ...............................................................7
   2.3 AVAILABLE TRANSLATION PACKAGES .............................................7

3. SYSTEM ARCHITECTURE ..............................................................................9
   3.1 MOBILE PHONE APPLICATION ............................................................9
   3.2 THE WEB SERVER ..................................................................................10
   3.3 THE OCR ENGINE ...............................................................................11
   3.4 THE TRANSLATION PACKAGE .............................................................12

4. MOBILE PHONE APPLICATION - CAMERA ...........................................13
   4.1 USER INTERFACE ..................................................................................13
   4.2 MULTI-THREADING ..............................................................................14
   4.3 USING THE CAMERA ..........................................................................15
   4.4 SENDING DATA TO THE SERVER .........................................................16

5. WEB SERVER ..................................................................................................17
   5.1 USING TESSERACT .............................................................................17
   5.2 POST PROCESSING ................................................................................19
1. Introduction

The integrated camera of a mobile phone is rarely used as an input device for mobile applications. By using computer vision methods while analyzing images acquired from the camera this task can be made possible, which is the basis for the design of this project. This chapter will set out the aims and motivation for this project, and give an overview including a guide to the chapters that follow.

1.1 Aims

The aim of this project is to design and implement a system that allows the user to capture an image using the integrated camera on their mobile phone and translate any text found in that image to another language of their choosing.

It should incorporate a custom made application designed for a camera phone as the user interface to the system. This application will allow the user to choose their preferred language and provide the means for capturing an image from the integrated camera and submitting it for processing.

The system should facilitate the use of low-end mobile phones, which have little processing power and low resolution cameras. Therefore, most of the image processing shall be implemented on a server accessible to the phone over the internet. In this way specific software and libraries may be used which are not available on the mobile phone. The server should also make use of any existing free software to aid in performing the Optical Character Recognition (OCR) and language translation.
1.2 Overview

The following diagram, Figure 1-2, defines how the entire system will be implemented. It shows the use of both a mobile phone application and a server, which uses existing packages to perform both the OCR and translation. Four main processes are highlighted:

1) Capture the image and send it to the web server.
2) Use Tesseract OCR to find any text in the image.
3) Translate the language of the text using Google Translate.
4) Return the translation to the phone and display it to the user.

Figure 1-2: Overview of entire system, highlighting 4 main processes; capturing an image and sending to server, performing OCR to find text, translating text, and sending result back to phone to be displayed.

Parts 1 and 4 both execute on the mobile phone while parts 2 and 3, while using third party software packages, both execute on the server side. The platforms used for the mobile phone application and server as well as information about Tesseract and Google Translate will be explained in Chapter 3, System Architecture. Some background on similar products currently being developed as well as available OCR and translation software will be given in Chapter 2, Background.
1.2.1 Capturing Image

In part 1 the user takes a picture using the integrated camera and sends it to the web server over a HTTP connection. The functionality for taking a picture is built into the mobile phone application and can be selected from the main menu. Once captured, the application submits the image to the server including necessary information about the required language. The application then waits for a response. A thorough explanation of part 1 will be given in Chapter 4, Mobile Phone Application - Camera.

1.2.2 Performing OCR

Once the image is received on the server side it uses an OCR package called Tesseract to extract any text. Tesseract returns any characters that it finds including vital information about their location and formatting. These results are then analyzed and any mistakes found are rectified. Chapter 5, Web Server, will give details about the process of using Tesseract and cleaning up its results.

1.2.3 Translating Text

Once the results from Tesseract have been finalized a string is created which contains all of the English text. The required language was sent to the server along with the image data so both this and the English string are passed to the Google Translate servers for translation. This topic is covered in Section 5.3, Translating Text.

1.2.4 Displaying Result

The translated text, along with pixel coordinates for the bounding box of where the original text was found, is sent back to the mobile phone application. The application still has a copy of the captured image, which it fades to white, and uses the bounding box information to place the translated text in the appropriate area of the image. This creates a graphical representation of the result while there is also an option to view a textual version. All of this is explained in Chapter 6, Mobile Phone Application - Displaying Result.
1.2.5 Performance

The entire system operates successfully with perfect execution on 22 out of 50 test images. Successful aspects of the project and difficulties encountered, as well as areas needing improvement, are discussed in Chapter 7, Evaluation.

1.3 Motivation

It is common knowledge that mobile phone usage has exploded over the past few years. In 2007 there were 121 mobile phones for every 100 people in the UK. Italy had an even higher number than this with 154, which shows how immensely popular the mobile phone has become in the developed world [1]. With the somewhat recent introduction of smart-phones, people have begun to realise the potential of the mobile phone to do more than just make calls or send messages. The extremely popular iPhone appeals to users through the promotion of customized applications, which can be bought via the online App Store. It is actually the method of delivery of these applications that has brought them to users’ attentions and made them so popular. Apart from the increase in internet speeds that came with 3G, mobile phones have been capable of supporting such applications for years. It is the intention of this project to use a Sony Ericsson W200i, which was released in the first quarter of 2007. This is a basic camera phone marketed as a music player and is a good representation of a standard quality mobile phone in wide use today.

Although the main concept of this project is to translate any text found in the image to another language, it will be designed for the main purpose of translating public signs. The intention is to create an easy to use application where the user needs only to take a picture of the sign and submit it for translation. In this way, tourists can avail of the resources of their mobile phone to understand any foreign language signs they encounter while travelling abroad. This could turn out to be a very successful product among frequent business travellers and holidaymakers who would see it as a far easier option than searching through a dictionary or even typing the text into a translation device.
2. Background

There are similar products emerging in this area of using computer vision on mobile phone applications, which are either in development or have only recently been released. Section one will give an overview of such products, while the next sections will give an overview of available OCR and translation software packages that could be used in this project.

2.1 Similar Products

Nokia are currently developing a product with very similar aims to this project. It is called Nokia Multiscanner and is the product of years of research at the Nokia Research Centre Beijing under the name Shoot-to-Translate [2]. It is evident from their publication that Nokia Multiscanner is designed to work on state-of-the-art smart phones such as the Nokia E66 and E71. The Nokia E66 for example has a 3.2 million pixel camera with autofocus and flash as well as 128MB memory and a 369MHz processor to support processing the image [3]. These features make it possible for the Nokia E66 to take an extremely clear image and handle all of the required processing on the phone, but it is a very expensive smart phone which would not be as popular among basic mobile phone users. This type of product would be restricting its base of potential users to ones who can afford the latest in mobile phone technology such as business persons.

Rafa Maya writes on his blog [4] about his experiences developing the kReader, which is another example of a mobile phone application that extracts text from an image. Its main feature is to read aloud a printed document after taking a picture of it, but it also supports translating the text [5]. Maya stresses the fact that the mobile phone’s hardware must be of adequate quality to support the software requirements and get consistently good results. He
has opted to use the Nokia N82 to run kReader as it has sufficient memory and a very good camera with autofocus and a powerful Xenon flash. This is another example of an application that can only run on state-of-the-art smart phones.

Figure 2-2: kReader on Nokia’s N82, [5].

ScanR is a service that makes use of server-side processing in order to extract any text from an image [6]. It allows users to send images to the server via multimedia messaging, e-mail or upload through the use of their special mobile phone application. The user can then view their documents online. This product is the most similar in concept to this project as it uses a server to do all of its processing. There are limitations, however, on the resolution of images accepted. For saving documents the recommended image size is 2 megapixels whereas the camera used in this project produces images with a resolution of only 0.3 megapixels.

The eyePhone is a concept by Ernst Pechtl and Hans Geiger of SuperWise Technologies AB and is currently in development [7]. This application will allow the user to take an image of any object in the world and, with the aid of satellite navigation and image processing, provide information about that object to the user.

All of the above examples that process on the phone make use of state-of-the-art smart phones in order to achieve the desired results. Even the online service scanR requires a high quality camera. This project will be able to run on some of the most basic mobile phones around today and so will not require the user to purchase an extremely expensive camera phone in order to use the service. The products described above show what is currently available and what lies ahead for applications that use the integrated camera of a mobile phone to provide information in a way that mobile phone users had never before thought possible.
2.2 Available OCR Packages

There is a very small selection of free OCR programs available online, and not all are open source or include an API for developers. While trying to find suitable software to be used in this project the following free programs were encountered.

GOOCR [8] is a free OCR program that includes a front-end for personal use. There is not much documentation available to support using the software as an OCR engine for a client program and there was no online community of developers using the software. It is being developed on Linux and only recently, in March 2009, a Windows version was developed. It is still early in development and no good reviews were found online so it was not used in this project.

Ocrad [9] is developed on Linux and can be used as a back-end to other programs. It is not available on Windows and has no online community so has not been used in this project. ClaraOCR [10] is another program available to Linux users but has no documentation or online community, and there has been no activity on the project site since 2002. The domain claraocr.org was found as a reference to the project but is not even in use anymore.

Tesseract OCR [11] is the software of choice for this project due to its high reputation online. It has a large community of developers using the software and there is a group page at Google Groups [12] for people to discuss their developments and any problems encountered. The software is open source and an API is also provided so that it may be used as a back-end for other programs. This makes Tesseract the most suitable free software to perform the OCR for this project.

2.3 Available Translation Packages

A similar situation exists when trying to find free translation software that can be used by a client program. There are many websites currently offering free translation where the interface is via a web page, which allows the user to paste text into a form and then submit it for translation. Once the text is processed a webpage is displayed in the browser containing the translated text. An example is shown in Figure 2-3.
Some of these websites include Google Translate [13], Microsoft’s Live Search Translator [14], Yahoo’s Babel Fish [15], SDL Free Translation [16] and Prompt Translator [17]. There are also offline services such as Lingoes [18], which provides free software that can be used to translate documents on a desktop. But this software does not provide an API and so cannot be used by another program.

Harnessing the power of these online services for use in a client program can be very difficult. It would involve studying the source code of the online form to discover where and how it is submitting the text for translation. It should be possible to reproduce the method used to submit text to the server and receive the reply in a Java program, but the webpage returned would need to be parsed in order to extract the translated text.

Google Translate is the most suitable software for this project due to the number of supported languages and the availability of a Java API. This API provides access to the online service from within a Java program, removing all of the overhead described in the paragraph above. Therefore it is the best option to be used in this project.
3. System Architecture

Figure 3-1: Overview of system architecture.

This diagram, Figure 3-1, provides an illustration as to how each part of the system operates together. The 4 main parts are as follows:

1. The mobile phone application.
2. The web server.
3. The OCR engine.
4. The machine translation package.

The mobile phone application communicates with the server application over the internet while the server application invokes both of the OCR and translation software. The server application and the OCR software both run on the server/laptop while the translation software is an online service hosted on Google’s servers. Each of these elements will be explained in the following sections.

3.1 Mobile Phone Application

The mobile phone application is implemented using the Java Platform, Micro Edition (Java ME) [19]. This is a specification of a subset of the standard Java Platform designed to provide a certified collection of Java APIs for the development of software for resource constrained devices, such as mobile phones.

The application is designed for mobile phones implementing the Connected Limited Device Configuration (CLDC) 1.1 and the Mobile Information Device Profile (MIDP) 2.0. The
CLDC contains the minimum subset of Java libraries needed for the Java Virtual Machine to operate on a mobile phone, while MIDP provides a higher level programming interface. MIDP 1.0 provided the Graphical User Interface (GUI) API but it wasn’t until MIDP 2.0 was released that the Mobile Media API was implemented, allowing video playback and access to the integrated camera, which is necessary for this project. MIDP 2.0 became standard on mobile phones around 4-5 years ago.

The I/O API implemented in MIDP 1.0 allows the application to create HTTP connections in order to upload or download information from the internet. This is assuming, of course, that the mobile phone user is with a network that provides internet access. In general, network providers use the General Packet Radio Service (GPRS) when connecting to the internet as it has become a standard protocol for using data services on GSM networks. This is also a vital part of this project, as the mobile phone application must connect to the internet in order to communicate with the server.

**3.2 The Web Server**

An Apache Tomcat [20] server installed on a laptop allows for the hosting of an HTTP Servlet, which accepts incoming HTTP connections from the mobile phone application. A single server can host many different Servlets so a different URL must be used for each one. If the IP address of the computer is 89.204.237.51 and the server is configured to operate on port 8080 and the Servlet for this project is installed in the path /midp/project then the specific URL used is:

http://89.204.237.51:8080/midp/project

The Servlet extends the HTTPServlet class defined in the Java Platform, Enterprise Edition (Java EE) [21], which provides for the implementation of functions to handle both POST and GET requests. The server decides which one to execute depending on the method used by the client when setting up the connection. The mobile phone application uses the POST method when sending data so it is this function that performs all of the necessary operations and finally sends the translation back to the phone.
3.3 The OCR Engine

Google’s Tesseract [11] is the chosen OCR package for this project as it is considered one of the best that is freely available. A dynamic linked library (DLL) is provided along with the Tesseract source code so that developers can use it from within their own code. One major setback was encountered while trying to integrate the Tesseract software with this project. It is coded in C and C++ while the web server is implemented in Java, but the server has to be able to execute the Tesseract software on the received image and access its results. Two solutions were discovered:

- Execute Tesseract from the command line and read the results from a file.
- Use the Java Native Interface [22] to create a wrapper class for the C code.

Executing the software from the command line would involve saving the image to disk, running Tesseract and then reading in the file output from Tesseract and parsing it to create the required objects. If there was some way to call the C code from within Java then this would save a lot of overhead.

The Java Native Interface (JNI) is a framework that allows Java programs to invoke libraries written in another programming language such as C or C++ [23]. It can be very difficult to encapsulate complex code into a Java interface, and it is also vital that no mistakes are made as debugging of the native code is not well supported.

Tesjeract [24] is another project being developed at Google Code which aims to create JNI wrapper code for the Tesseract DLL. In this way the Tesseract DLL may be used within Java programs, which is exactly what is needed for this project. The Tesjeract classes are then used in the Java program and behave as an interface to the Tesseract software. It is still in development but the latest version released works perfectly as part of this project.
### 3.4 The Translation Package

Machine translation is never going to provide a perfect fluent translation, but all that is necessary for this project is that the user grasps the meaning of the text in the sign. Therefore, it is not required that the translation have perfect grammar as the user will be able to understand it as long as the vocabulary is correct. The alternative to on-the-fly machine translation is to employ a human translator which would be slow and unnecessary.

Google Translate [13] is a machine translation service hosted online and accessible via a JavaScript form on their website. The user pastes the text into the text area and selects the source and required languages, then submits the form. The page is refreshed with the translated text contained within it. This is an ideal user interface for individual users but not very useful when it comes to using it within a program.

Upon inspection of the source code for the web page it is not immediately obvious what method is used when submitting the text for translation. This would have been immediately apparent if a basic HTML form had been used. Therefore, it would be a difficult task to read through all of the JavaScript code and attempt to reproduce its functionality in a Java program. Even if this was done, the result would consist of a HTML page that would need to be parsed in order to extract the translation from it.

An unofficial Java API [25] has become available on Google Code which seems to implement all of what has been described in the previous paragraph. The API consists of a simple function call, which communicates with the Google servers to retrieve the translation. The source text, the source language and the required language are all passed as arguments to the function and it returns a string consisting of the translated text.
4. Mobile Phone Application - Camera

The application on the mobile phone is the user interface to the entire system. In this chapter it will be explained how the application allows the user to capture images and send them to the web server to be processed, as illustrated in Figure 4-1.

### 4.1 User Interface

When the application is started the user is presented with the main menu as can be seen in Figure 4-2. This menu has four options:

*Start Camera*

This is the main function of the application. It starts the camera and allows the user to take a picture and submit it to the web server for translation.

*Select Saved Picture*

This feature is meant to allow the user to select an image saved in the phone’s memory and submit it for translation, but it is not yet implemented.

*Options*

This allows the user to set such options as the required language to translate text into, and the IP address of the server. This is a temporary requirement as the web server is hosted on a laptop which has a dynamic IP address. Every time the laptop is connected to the internet the
IP address will have changed and needs to be set in the application. Once set, the options are stored in the phone’s memory and will remain in the same state the next time the application starts up.

About
This option displays information about the application to the user including some simple instructions and contact details of the author.

The screenshots below in Figure 4-3 shows what the user encounters when following through with the main function of the application, ‘Start Camera’.

![Figure 4-3: Image on left shows live feed from camera, while image on right shows a preview of the captured image.](image)

When the user selects ‘Start Camera’ they are presented with a live video feed from the integrated camera. There are commands to cancel and return to the main menu, or capture a snapshot from the camera. After capturing the image it is previewed to the user so that they have a chance to inspect it. This gives the user a chance to make sure that the image is sharp, well lit and that the text is horizontal. There is an option to retry, which will bring the user back to the live video stream to take another picture, or to submit the current image to the web server for processing.

### 4.2 Multi-Threaded

Some operations implemented on the mobile phone may take a long time to execute, such as capturing an image from the camera or communicating with the server. Mobile phones generally use GPRS when connecting to the internet which is considerably slow compared to the standard broadband speeds available today. These operations are executed in a new thread which keeps the system thread idle so that it can respond to other events, preventing the phone from appearing frozen to the user.
During these lengthy operations an animated wait screen can be displayed using another thread which can be updated with information to keep the user informed as to the state of the application. This wait screen contains a cancel command which can be used to cancel, for example, the network connection. If the application was not multi-threaded then the animation would be frozen and the cancel command would be unresponsive. The wait screen in Figure 4-4 to the right is displayed once the user presses ‘Submit’ and all of the necessary data is being sent to the web server.

Even though the use of threads is necessary for the reasons specified above, creating new threads on such small platforms will consume a lot of processing power and memory. The solution to this problem is to create only one new thread when the application starts. This thread can then be used for both capturing images and communicating with the server as neither of these operations will ever occur concurrently. When the thread is created, and every time an operation completes, it is put to sleep and not woken up again until it is needed.

### 4.3 Using the Camera

The Mobile Media API (MMAPI) provides methods to display a video stream from the camera on the phone’s display. By querying the dimensions of the source stream the necessary width and height may be calculated so that it all fits on the screen and is kept proportional. The new dimensions can be passed to the method and the stream will be automatically resized to fit these new dimensions. This resizing of the source stream, however, does not affect the size of the image captured from the camera.

When capturing an image from the camera the MMAPI method `getSnapshot(..)` is used. This returns a byte array containing encoded image data. The encoding is specified using a string as an argument to the method. By passing null as the encoding the system default encoding is used which is usually JPEG but the size of the image may not be as large as possible. In order to ensure that the largest image resolution is captured the string passed to the method must be something like the following:

```
“encoding=jpeg&width=640&height=480”
```
This is the encoding used for the application when being run on the Sony Ericsson W200i, but the emulator will not accept this encoding. In reality, every device will accept a different set of encodings and the particular set allowed for a specific device may be found by querying the system properties. In this way an encoding may be selected which uses the JPEG format and maximum resolution possible.

An image object may then be created from this JPEG byte array and displayed on the screen for the user to analyze. The image must first be resized so that it is all visible on the screen and this is done manually by re-sampling the pixels to give the image the desired resolution. The new dimensions are calculated in a similar way to the video stream, by ensuring that both the height and width fit within the screen dimensions and keeping them proportional at the same time.

4.4 Sending Data to the Server

The original JPEG byte array and the language to translate the text into are sent to the web server over a HTTP connection. The standard Java ME I/O interface is used to create the HTTP connection and the URL of the web server is set in the options. The request method of the HTTP connection is set to ‘POST’ and an output stream opened up to write data to. The stream facilitates writing bytes so this is ideal for the image byte array. The first bytes to be written are the required language string followed by a semi-colon to signal the end of the string and then the image data. An input stream is then opened to receive the reply and the application waits until it has received everything. During all of this communication with the web server the user is presented with the animated wait screen as seen in Figure 4-4, which is updated to let the user know when sending data is complete.
5. Web Server

The web server accepts incoming POST requests from the mobile phone application and runs the appropriate function of the Servlet to handle them. An input stream is provided containing all of the data that was sent to the server and an output stream to send back a reply. The bytes in the input stream are read until a semi-colon is encountered and then the required language string is created from them. After this all of the bytes are saved into an array to contain all of the image data. A copy of the image data is saved onto the laptop as a JPEG file for testing purposes.

The next few sections will give an insight into the processing done by the Servlet, from extracting text from the image to translating the text found. The graphical user interface for the server will also be explained, which shows what characters have been found in the image.

5.1 Using Tesseract

Tesseract requires an uncompressed TIFF image as its input so the first action that needs to be done is to convert the compressed JPEG byte array into that format. The Java Advanced Imaging API [26] provides functionality to convert images of many formats that the standard Java Platform cannot. It is used to decode the JPEG array and create an image object, which can then be encoded as a TIFF image.
An instance of Tesjeract is created with the language set to English. For now, the system is restricted to use on English signs so this is passed as an argument when creating the object. Tesjeract provides the exact same functionality as if the Tesseract API was being used directly. The TIFF image is then passed to the Tesjeract method to find any characters in it and an array is returned containing each character as well as extra information about each one, including:

- The bounding box of the character in pixel coordinates.
- The number of spaces before the character.
- Whether the character is at the end of a line, or paragraph.
- The confidence value for the entire word that the character is in.
- Codes for the font and point size of the character.

The following image, Figure 5-2, represents the results from Tesseract. It features the bounding box of each character interpreted by Tesjeract showing the location where it was found, while also displaying the actual character recognized.

![Figure 5-2: The bounding box of each character found is drawn using alternating colours while the interpreted characters are drawn in red.](image)

The bounding box coordinates for each character is very useful information and is used when validating the results and calculating the bounding box of all the text found in the image. The spaces and new line attributes are used in forming words from the character array. The confidence value ranges from 0-255, 0 being extremely confident and 255 being the least confident. This value is the same for every character in a word and so is effectively how confident Tesseract is in that word. This is taken into account and any words with a confidence value of 255 are ignored, which leads to improved results. A lot of stray characters found in the background of the image would have this confidence value and are hence removed. Although there was one case where a word was assigned a confidence value of 255 and was actually part of the text in the sign. This shows that although improving results in most images, it will also have negative effects on some others.
5.2 Post Processing

In many cases the results from Tesseract were very messy, with false negatives where it should have found text and false positives where there were no characters in the sign. An example of a sign with false positives detected throughout it can be seen in Figure 5-3.

![Figure 5-3: Example of false positives misinterpreted by Tesseract.](image)

This is mostly due to Tesseract misinterpreting diagrams or marks on the sign and textures in the background as characters. When the resulting characters are compiled into a sentence these false positives are mixed throughout. This may not pose much trouble if being read by a human but it can cause severe problems when attempting to translate the sentence using a machine translator. In order to achieve consistently good results a method of removing these false positives would need to be implemented before constructing the sentence.

The easiest way to remove false positives is to use their location in the image to differentiate them from the main text. In order to do this an assumption would have to be made that there is one main body of text in the sign. If a concentration of text is found in the image then any characters above, below and either side of this text can be ignored. False positives found above and below the main body of text will be classified as vertical outliers while ones either side of the text shall be classified horizontal outliers. The next few sections will explain certain methods implemented in this project.
5.2.1 Vertical Outliers

In order to find vertical outliers a histogram of all characters found in the image is projected onto the y axis. This histogram features peaks where there are lines of text and is empty in the spaces between these lines.

![Figure 5-4: Histogram of characters projected onto y-axis.](image)

The first attempt made at cleaning up the results is to remove any lines with only one character in them. This method takes care of some random characters spread throughout the image but patterns in the background or around the sign may produce a number of false positives in the same line.

The next step ensures that an upper and lower limit may be found in the image from which we can ignore characters above and below respectively. Some test images have resulted in Tesseract finding extremely large characters who’s bounding box go beyond the edges of the image. In order to ignore these, the number of gaps between lines is counted along the histogram and the entire histogram reduced by one until there are at least two gaps. This worked effectively to ignore those extremely large characters and aid in finding the main body of text.

To find the upper and lower limits a threshold had to be set for the allowable line spacing. By starting at the point in the histogram with the largest peak and moving away in both directions, the distance of the gaps in the histogram can be measured. Once a large enough
gap is reached the limit can be set at that point. Therefore, every line of text between the upper and lower limits has appropriate line spacing and can be considered the main body of text in the sign. The threshold used was that the gap between lines must be less than twice the average height of all the characters. Figure 5-5 illustrates the upper and lower limits for the image in Figure 5-4, calculated using this method. Characters with a top coordinate above the upper limit or a bottom coordinate below the lower limit are then ignored.

![Figure 5-5: The two long horizontal lines extending out of the histogram denote the upper and lower limits while the shorter one between them denotes the peak in the histogram.](image)

### 5.2.2 Extremely Small Characters

Tesseract has a habit of finding extremely small characters in the image even though they do not exist. These come about from textures in the background of the image or even a rivet in the sign may be interpreted as an ‘O’. In order to remove such characters yet another threshold would need to be used, that the smallest character must be larger than half the average character size. The method used iteratively finds the smallest character, removes it if too small, and recalculates the average size of all the remaining characters. This continues until the size of the smallest character found is an acceptable size. As characters deemed too small are removed, the average size of the remaining characters will gradually become more accurate. Figure 5-6, below, provides an example of how small these false positives can be.
5.2.3 **Horizontal Outliers**

Having completed the other methods for cleaning up the image there can still be false positives either side of the main body of text. These ones cause the most trouble when attempting translation as they get mixed into the sentence before and after each line. This will have a large impact on the flow of the sentence and the translator will either translate a few of the words separately or maybe leave them all the same.

The method for removing horizontal outliers uses the spacing between characters in a line. If the space between characters is greater than three times the width of the character then it is believed that the character is not part of the main body of text and is a false positive. At the moment this method iterates through each character in the list and, ignoring the first one in every line, tests the gap between that character and the previous one. This method has produced good results with the test images but could be vastly improved. By finding a point in the line that is guaranteed to be within the main body of text, characters in either direction could be tested and a left and right limit found for each line of text. This method would perform similarly to the vertical outliers’ method and would produce much more consistent results.

![Figure 5-6: An example of extremely small characters misinterpreted by Tesseract.](image)
5.2.4 Additional Methods

On some occasions Tesseract may confuse the digit ‘0’ with the character ‘o’. If a word were to have zeros in it then the translation package will not be able to understand it, leaving it unmodified in the translation. This would have a massive impact on the translated text and is not acceptable. In order to rectify the occurrence of this mistake the characters either side of the zero are analyzed. It is assumed that the digit zero will not occur next to an alphabetic character in the same word, therefore if the character either side of the zero is not a numeric digit then the zero is changed to an ‘o’.

Due to several characters being removed from the original list some formatting may have been lost, such as new lines or spaces. When compiling the sentence from the final list of characters measures must be taken so as not to join words together where there should be a space or new line. The bounding box of the characters is used when checking to see if there is substantial space between characters or whether a character is on a new line.

5.3 Translating Text

Once the list of characters is finalized and an English string created it is ready for translation. The Google Translate Java API is used to access Google’s online machine translation service which provides a single function for it. The required language to translate the text into was sent from the mobile phone application to the server along with the image data so this string is passed directly to the translate function along with the English text. The source language is restricted to English as has been stated earlier in this report. The translated text is returned as a string and can then be sent back to the mobile phone application to be displayed.

5.4 Graphical User Interface

The server has a Graphical User Interface (GUI), designed using the Java Swing framework, which shows what is happening in the background. This is useful for demonstration purposes as well as testing and debugging. The GUI is displayed on the computer on which the server is running every time a connection is made to it using the POST request method. It consists of a pane to display the image received from the mobile phone and a text area for outputting information about what steps have been completed. These can be seen in Figure 5-7 below.
As soon as the server accepts a connection the GUI shows up and states that it is waiting to receive the image. This is the longest process due to the slow internet speeds of the mobile phone but once completed the image is displayed. At this point the OCR is executed and the results show up on the image in the following format. The image is faded and the bounding box of every character discovered by Tesseract, as well as the character itself, is drawn on top of the image. The English string is output to the GUI text area and once the translation is complete it is also outputted. In Figure 5-7 every character found by Tesseract is drawn onto the image but the English string outputted to the text area is compiled from the refined results after post processing is complete.

![Graphical User Interface after processing is complete and the text has been translated.](image)

*Figure 5-7: Graphical User Interface after processing is complete and the text has been translated.*
The GUI provides a couple of options to the administrator to help with analyzing the image. The colour of the characters drawn on top of the image may be changed using a selection of available colours so that the best contrast with the background image may be obtained. There is also a button to modify how the results are highlighted on the image. It toggles between three states to show:

- The bounding box of the characters.
- Both the bounding box and the character found.
- Neither of the two so that the sign can be seen clearly.

When there is a high volume of connections made to the server it can run out of memory from opening a new window for each image. In order to make this more efficient the GUI can be set to not display. The method used is to send a GET request to the server which toggles whether or not the GUI will be shown for all of the following connections. The easiest way to do this is to visit the URL of the server in a web browser and the text returned will inform the administrator as to whether it is set on or off.
6. Mobile Phone Application - Displaying Result

While the server is performing OCR and translation the mobile phone application waits for it to send back results. It soon receives pixel coordinates for the bounding box of the text found in the image followed by a string containing both the English text and the translated text. The bounding box coordinates must be scaled so that they relate to the resized image which fits on the phone’s display and then the image is faded before drawing the translated text within these coordinates.

6.1 Fading the Image

The image is faded so that the translated text drawn on top of it is legible. This retains any other information in the sign, such as pictures or arrows, when the user views the translation. By placing the translated text in the same location as where the original text was found it is hoped that the translation will be understood in the correct context.

The image is faded by setting each of the red, green and blue colour components of each pixel 40 percent closer to the value 255. Therefore each colour component is increased in proportion to its original value so the resultant pixel still retains some of its colour information. The resultant image is partly faded to white, as can be seen in Figure 6-2, making it a lot easier to read black text drawn on top of the image.
6.2 Drawing Translated Text onto the Image

The display on a mobile phone is usually quite small, for the Sony Ericsson W200i it is only 128 x 160 pixels. Therefore it is preferable to draw using the largest font size possible. In the J2ME specification there are three different font sizes available; small, medium and large, which can be either plain or bold. The algorithm used for choosing the appropriate font size iterates through each one from bold large to plain small until the text fits within the bounding box. For each font it counts the number of lines required to display the string, each line fitting within the width of the bounding box, and determines if that number of lines will fit within the height of the bounding box.

If at the smallest font size the text will not fit, then the bounding box is ignored and the screen dimensions used instead. If at this point the text still does not fit, then it is allowed to run off the bottom of the screen. A pseudo-code representation of this algorithm can be seen in Appendix A.
In order for the user to see the entire translation the ‘Text’ command can be used to switch the display from graphical to text only, which has a scroll option to view the rest of the text. This view shows the entire string received from the server containing both the English and translated text. The user can toggle back and forth between the graphical and textual views of the result. Once the user is satisfied the ‘Back’ command takes them back to the main menu.

![Image](image.jpg)

Figure 6-4: Screen capture of mobile phone on left shows graphical view including bounding box of text found in image while screen on right shows text-only view. The right soft key toggles between graphical and text-only views.
7. Evaluation

This chapter will evaluate the success of the system by giving a review of the results from testing. The results will be described graphically, detailing the effectiveness of the OCR package and showing the final translation on the display of the mobile phone. Successes will be explained and difficulties encountered during development outlined. Areas that may benefit from further development will also be discussed.

7.1 Results

In order to evaluate how effective the system is at translating public signs a collection of fifty test images were collected around Dublin City Centre using the mobile phone application. The application was running on the Sony Ericsson W200i mobile phone which uses a standard VGA quality integrated camera. This camera produces 0.3 megapixel images with a resolution of 640 x 480 pixels. These images are included on the CD attached to this report, while a subset can be seen in Figure 7-1 below.

![Figure 7-1: An example of test images used for evaluation.](image)

The following examples display how successful the system performed for different test images. Each example will include the original image, a copy displaying all of the characters found by Tesseract, and finally the graphical display of the result on the mobile phone. The image displaying the result of Tesseract will include any false positives which would have been removed during post-processing.
In Figure 7-2 Tesseract discovered all of the text in the sign without any false positives. Therefore, post-processing is not necessary and an accurate translation can be performed by Google Translate. All of the text fits within the bounding box with a small font size.

In this test Tesseract finds a few false positives along the top of the image but these vertical outliers are successfully removed before translating the text into German. The text on the phone display can fit inside the bounding box with a medium font size.
In the above two examples Tesseract has returned some false positives which have been successfully filtered out of the text before translation. In Figure 7-4 the font size must be small in order to fit within the bounding box on the phone’s display, while in Figure 7-5 the text is at the largest possible font size that the phone can display. In this case there is an issue with the text displayed on the phone as both headings have been merged into a single sentence. However the translated text still makes sense and will not cause any problems for the reader.
7.2 Successes

The aim of this project, to create a camera phone sign translator, has been achieved. The entire system has been constructed from many separate applications and software packages, which all work together to enable a basic camera phone user to translate a public sign from English into a number of other languages.

Success was achieved in 22 of the 50 test images, therefore it can be said that the system has a success rate of 44%. Each of these 22 successful test images produced a sensible translation on the mobile phone display; therefore the OCR found all of the text in the sign and any false positives were successfully filtered out of the results before translation.

7.2.1 Software Integration

This project makes use of four third-party software packages. These include:

- Tesjeract JNI wrapper for Tesseract OCR [24].
- Google Translate [13].
- Java API for Google Translate [25].

It was an aim of this project to make use of existing software to perform the OCR and translation, and this has been achieved. In both cases, however, additional software was required to integrate the required OCR and translation packages with the system.

Integrating all of the different elements of the system, over 2 platforms, has been a success. The mobile phone application communicates with the Servlet over a HTTP connection, while the Servlet communicates with Tesseract via a DLL and the use of Tesjeract, and the online Google Translate service is accessed from the Servlet with the use of a Java API. At no point during the testing of the system did these communications break down, and any errors that may occur at any stage in the system are effectively communicated back to the mobile phone application to be notified to the user.
7.2.2 Post-Processing

The processing of results from Tesseract implemented in this project has lead to much more consistent results and a far higher success rate among the test images. Without implementing these methods the success rate for the same 50 images was only 16%. A lot of these tests failed due to false positives cluttering up the text and preventing it from being translated properly. By successfully removing these false positives in most images the success rate increased to 44%. The reason for other tests not being improved is mostly due to false negatives. However, this is yet another example of how improving performance on most signs also reduces performance on others. All of these methods rely on assumptions to find ways to improve results and as expected these assumptions are not always correct.

The main assumption that every sign will have a single body of text will immediately rule out any signs with a number of paragraphs, or even text with very large line spacing. Some of the other assumptions include the space between lines, the space between words and the size of characters. Although causing these errors is unsatisfactory, the number of tests that improved severely outweighs the number degraded.

7.2.3 Graphical Display on Mobile Phone

The aims of this project included displaying the result of the translation to the user on the mobile phone. Initially this was done textually but it was later decided that the display should be more appealing to the user. The best way to display the translation would be to replace the text in the sign with the translated text, but this would be an immense task and would not be feasible due to limited resources on the mobile phone and the time constraints of this project.

The next best solution is to draw the translated text on top of the original image and attempt to keep it in the same area as the source text. This has been achieved by fading the image to white and drawing black text on top so that it is legible. The bounding box coordinates of the characters found by Tesseract act are used to place the translation in the same area as the source text, keeping the context of any images or diagrams in the sign the same. This feature has successfully improved the user interface by making it more appealing to the user.
7.3 Difficulties

As with any project of this size many difficulties are encountered, ranging from software to hardware and even networking problems. An important part of this project was to integrate existing software into the system, which can be very difficult. The platforms used include a mobile phone and a web server which create problems regarding use of special libraries and difficulties with debugging.

7.3.1 Integrating Tesseract

The most difficult software to integrate with the system was Tesseract. It provides a DLL for use in client programs, but these would have to be written in C++ in order to access the Tesseract functions directly. The program that needed to use Tesseract in this project was the server application, the Java Servlet. This created an immediate barrier between integrating the Tesseract DLL and the Servlet as the Java program would not be able to invoke the C++ functions from the DLL directly.

After attempts were made to create a JNI wrapper for the Tesseract DLL it became apparent that this was a very big task. The Java Native Interface is difficult to work with and even more difficult to debug. This problem with using Tesseract in a Java program had been raised in the Google Groups discussions and it was discovered that developers with experience in using the JNI were attempting to create a wrapper class for the Tesseract DLL. The project, called Tesjeract, was made available at Google Code [24] and so was used in this project. It successfully acted as an intermediary between the Java Servlet and the C++ DLL.

7.3.2 Portability of Mobile Phone Application

The problem with the Mobile Media API is that only a subset is included in the MIDP 2.0 standard. Whether a specific device implementing MIDP 2.0 is capable of capturing an image from the camera or not is completely up to the manufacturers and even some recently released mobile phones will not allow it. This leads to confusion as to whether a certain phone is able to support the mobile phone application as even though the manufacturers say that it implements the Mobile Media API, it may still refuse access to the camera.

The platforms that this application is being developed on are the Sun Wireless Toolkit Emulator and the Sony Ericsson W200i. Another complication is that each different device
accepts different encoding arguments when taking a snapshot with the camera, as explained in Section 4.3 Using the Camera. In order to expand the portability of the application research would need to be conducted into the accepted encodings allowed for different devices and the application changed to select an appropriate one.

Other than these two complications there isn’t anything else affecting the portability of the mobile phone application across devices satisfying the basic requirements, these being that the device be MIDP 2.0 enabled and have internet access. The application does not require a good camera and has no special memory or processing power requirements.

### 7.3.3 Server Hosting

The program used to perform both of the OCR and translation needed to be accessible to the mobile phone application over the internet. There are many websites online that provide free hosting for people to put up their own web pages, such as www.110mb.com and www.zymic.com. These sites may allow customers to use PHP and MySQL but only commercial hosting websites will allow customers to use the Java Virtual Machine, which is required to host Java Servlets. The best solution to this problem was to install a web server on a personal computer, in this case a laptop, and host the Servlet there.

The Apache Tomcat Server [20] was installed on the laptop and it required some router and firewall configuration before it was accessible over the internet. By default it operates on port 8080 of the host computer so once the IP address of the computer was known it was possible for the mobile phone application to connect to it. This created a small problem considering that every time the laptop connects to the internet it is given a new IP address. So the URL used by the mobile phone application would need to be updated to reflect this. An option was added to the application so that the user could manually enter the IP address of the laptop, and this is saved in memory so that it remains the same the next time the application is started.

This method worked perfectly while on a home network that could be configured to facilitate the server but it would not work while connected to the college network. The Trinity College proxy blocks the laptop from being accessible over the internet. The only way around this problem was to use a 3G modem that offers a direct connection to the internet that could be used while in college.
7.3.4 Miscellaneous

Memory problems were encountered while running Tesseract from the Servlet, which is running in the Java Virtual Machine (JVM). The Servlet was only allocated a certain amount of memory which was not sufficient for Tesseract to execute on some images. This resulted in the server crashing but a solution was found for the JVM to allocate a larger portion of heap memory to the Servlet.

Another problem with the Apache server, which is specific to Windows Vista, is that a program running as a Windows Service does not have permission to access the desktop of the user currently logged in. The symptom of this problem was that the server GUI would not display on the desktop. After a long time researching the problem the author succeeded in allowing the server GUI to display, but not on the desktop of the currently logged in user. Windows would alert the user that a program was trying to display and the desktop could be switched to a new one to view it. This was very inconvenient and eventually it was discovered that if the server was run manually and not as a Windows Service then the GUI would display on the current user’s desktop.

7.4 Future Work

As stated previously, the system currently has a success rate of 44%. This rate has the potential to increase if further development were carried out on this project. A major reason for failure is the quality of the image captured by the camera phone. Poor image quality is the primary cause of false negatives in the OCR results. These are characters in the sign that Tesseract does not find, mostly due to blurring, noise and distortion in the image. In order to improve the detection rate of Tesseract, it would have to be provided with higher quality images. A higher resolution camera with a powerful flash could be employed, but this would be contradictory to the aims of this project. A more appropriate solution would be to pre-process the images before performing OCR on them.

The ideal situation would be to have a binary image with pure black text on a pure white background. This may be attempted by analyzing the primary colours found in the image and identifying the text and background. It may not be possible to produce a binary image due to multiple colours in the sign and the inclusion of the background in the image. But if the quantization of the image was reduced appropriately then the regions of text and background
would be easier to identify and the detection rate of Tesseract would be improved. This method would also remove noise from the image, which is a cause of false positives.

Another way to optimize OCR performance would be to geometrically transform the image to remove the effects of perspective and tilt. Tesseract requires that the text be in horizontal lines for optimum performance but perspective can distort the lines of text and the camera may not have been held horizontal with the text in the first place. Edge detection could be used to find the orientation of the text and then the image can be transformed to correct this.

By performing some of this pre-processing on the mobile phone before sending the image to the server the amount of data being sent can be greatly reduced. Sending data over the GPRS network is an extremely slow process and is the main factor determining how long the system takes to translate a sign. Therefore speeding up this part of the process will be the most effective way of speeding up the entire system. The best way to do this would be to reduce the number of bits per pixel of the image, hence reducing the quantization of the image, or even to reduce the number of pixels being sent to the server. This could be achieved by cropping the image to include only the text and exclude most of the background. This modification alone would also improve the accuracy of the OCR by removing possible false positives that could have been detected in the background.

The system is currently restricted to translation from English to a number of different languages. The languages currently available are German, French, Spanish, Polish and Italian, although the system is capable of supporting much more. Through the utilization of Google Translate this list can easily be expanded to include over 40 languages. The source language, however, is restricted by the use of Tesseract OCR when reading the image of the sign. Tesseract can recognize six languages and also has the ability to be trained to learn new ones. These factors can be taken into account when attempting to make a robust system that covers as many languages as possible.
8. Conclusion

It has been proven that the integrated camera of a mobile phone can be used as an input device for mobile applications. Such an application may include data entry that can be a tedious task for the user, especially using the keypad of a mobile phone. By simply taking a picture of a piece of text the mobile phone application now has the ability to understand the image and extract the text from it, thanks to advances made in the area of Computer Vision.

This project has successfully implemented a mobile phone sign translator, which uses the integrated camera to take a picture of a public sign and then translate any text found in that image to another language. It makes use of existing Optical Character Recognition software to find text in the image and an existing online machine translation service for translating the text into a number of languages. It incorporates the use of a web server to perform most of the resource intensive operations, including the OCR and translation, which allows the system to facilitate the use of a very large range of camera phones. Any mobile phone that allows an application to access its integrated camera should be capable of using the system, which creates an extremely large base of potential users worldwide.

The intended use of this system as a sign translator has great potential to be a successful application on standard mobile phones. Tourism is a massive market, providing a solid user base of mobile phone users who would be interested in such an application. With a solid foundation already implemented, and an outline of further development provided, this project has the potential to exceed the measured 44% success rate and become a truly robust camera phone sign translator.
References


Appendices

Appendix A: Text Drawing Algorithm

This is the algorithm used when attempting to fit the translated text within the bounding box of where the original text was found. The aim of this algorithm is to place all of the text within the bounding box while making the best use of available space.

```plaintext
While(height of paragraph > height of bounding box) {
    Decrease_font_size();
    Calculate_height_of_paragraph();
}

Decrease_font_size() {
    If(font is null) font = Large|Bold;
    Else if(font is Large) font = Medium|Bold;
    Else if(font is Medium) font = Small|Bold;
    Else if(font is Small) {
        If(font is Bold) font = Small|Plain;
        Else {
            If(bounding box height < screen height) {
                Set Bounding box width to screen width;
                Set bounding box height to screen height;
                Font = Large|Bold;
            }
            Else {
                Set bounding box height to 9999;
            }
        }
    }
}

Calculate_height_of_paragraph() {
    For(all chars in text) {
        Add char to current word;
        If(char is a space or last char) {
            If(length of line + current word > bounding box width) {
                Increase number of lines;
                Set new line to current word;
            }
            Else {
                Add current word to end of line;
            }
        }
    }
}
```
Appendix B: Electronic Sources

The attached CD contains all of the source code for the mobile phone application and the web server, as well as any libraries required for them to run. Also included is the collection of test images used for evaluating the system.

There are some requirements for Tesseract to execute successfully:

- The paths for tessdll.dll and tesjeract.dll must be set in the loading functions of the camera server source code.
- The tessdata folder must reside in the same location as tessdll.

The following packages, located in `Server\dist\lib`, must also be added to the project:

- google-api-translate-java-0.52.jar for Google Translate.
- jai_codec.jar and jai_core.jar for the Java Advanced Imaging methods.
- servlet-api.jar for implementing a Java EE Servlet.

In order to test the server program without using the mobile phone application to supply image data, another project called TestServer was created to process image files on the computer. This is the reason for saving the JPEG images which the server receives. The source code for this project is also included on the CD so that the functionality of the server can be demonstrated without the need to make it accessible over the internet.