Spring 2014

IMPROVING THE EFFICIENCY OF TESSERACT OCR ENGINE

Sahil Badla

IMPROVING THE EFFICIENCY OF TESSERACT OCR ENGINE

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Sahil Badla

May 2014

SAN JOSÉ STATE UNIVERSITY

The Undersigned Project Committee Approves the Project Titled

Improving the efficiency of Tesseract OCR engine

by

Sahil Badla

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Teng Moh — Date
Department of Computer Science

Dr. Melody Moh — Date
Department of Computer Science

Prof. Ronald Mak — Date
Department of Computer Science

APPROVED FOR THE UNIVERSITY

Associate Dean Office of Graduate Studies and Research — Date

ABSTRACT

IMPROVING THE EFFICIENCY OF TESSERACT OCR ENGINE

By Sahil Badla

This project investigates the principles of optical character recognition used in the Tesseract OCR engine and techniques to improve its efficiency and runtime. Optical character recognition (OCR) method has been used in converting printed text into editable text in various applications over a variety of devices such as Scanners, computers, tablets etc. But now Mobile is taking over the computer in all the domains but OCR still remains one not so conquered field. So programmers need to improve the efficiency of the OCR system to make it run properly on Mobile devices. This paper focuses on improving the Tesseract OCR efficiency for Hindi language to run on Mobile devices as there a not many applications for the same and most of them are either not open source or not for mobile devices. Improving Hindi text extraction will increase Tesseract's performance for Mobile phone apps and in turn will draw developers to contribute towards Hindi OCR . This paper presents a preprocessing technique being applied to the Tesseract Engine to improve the recognition of the characters keeping the runtime low. Hence the system runs smoothly and efficiently on mobile devices(Android) as it does on the bigger machines.

ACKNOWLEDGEMENTS

I would like to thank Dr. Teng Moh for his guidance. His insight, advice and guidance throughout the project was invaluable. I also thank Dr. Melody Moh and Prof. Ronald Mak for serving on my defense committee.

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

Now that we have entered Web 3.0, the main players in it are mobile and more mobile devices. Anything you could imagine is available for mobile. Similarly mobile application pool has grown tremendously because of high availability and also the portable nature of mobile phones and public API's. It's easier to carry them all the time instead of carrying your computer or laptop.

Consider a scenario when you are out in some country with your family and the native language there is not English, how would sightseeing look like? A person cannot carry a translation book with himself to look up for signs and notice boards. That would ruin the joy of sightseeing. What if we have a device which acts as a ready translator and has a camera so that you just point it on the text and you get the translated text right away, wouldn't it make several lives easier.

The inspiration for this project is something similar to this scenario. People who belong to India face a similar issue. In India, which has many regional languages, travelling to other states and roaming around has always been a problem. One can't always hire a guide for the whole trip. Hindi language is the focus as it is the national language of India. With more than a billion people living and more than half a billion mobile phones, India lacks mobile phone applications focusing on Hindi and related applications.

This led to the research for applications that people could use as ready translators in the handheld devices and we all know that device is none other than a Smartphone. Almost everyone has a Smartphone these days. Now the problem was to find the missing pieces and address the problem to a solution. Next step is to research about the applications which could extract text from images and that came out to be "Optical Character Recognition" applications also know as OCR. This is the start to the building of a system which will then be ported over to Android platform where the Hindi (national language of India, originated from Sanskrit) text could be extracted. This would be "the" application for the visitors travelling to India. One more important thing is that it would be available on the mobile devices as an open source project.

The inspiration for this project is based on the fact that there are not many applications for Hindi OCR and very few of them are available for mobile phones. What makes this application stand apart from other OCR applications apps is that it is open source, runs on mobile phones and addresses a very common problem which more than a million people in India are facing.

This application is a starting point for a lot of similar applications that require OCR and it is dedicated for Hindi. It has numerous use cases such as translation applications for visitors and tourists, educational applications for students/teachers/kids, regional applications etc.

OCRs for personal computers is a fairly common thing and is being used in various domains. Making it open source allows this potential field to be more discovered and contributed by developers from all over the world. However the scope of this project is limited to efficient extraction of the text only and translation would be out of scope. Let us start from the various components involved in this system.

Optical character Recognition (OCR) is used in conversion of scanned, printed text images [1] or handwritten text into editable text for further processing and analysis. OCR allows the machine to recognize the text automatically. OCR has been used since early 90's in various types of machines and is improving gradually. We could experience various problems while developing an OCR system. First: Computer has to know what characters look like. There is little visible difference between letters and digits. For instance it is difficult to differentiate between digit “0” and letter “o” for the machine. Second: It is even difficult to differentiate foreground text from background text and other content. Let's look back in the history where it all started. The first OCR system was installed in 1955 at the reader’s digest, which used to OCR input sales report into a computer. After that the OCR became very helpful in computerizing any manual or physical task relating to documents [Patel 2012]. OCR is being used widely for various purposes which includes: License plate recognition systems in various countries at toll stations, roads & CCTV's, image text extraction from natural scene images [21], scanners and extracting text from scanned documents, cards, printers [12], etc [21]. The proposed system is a faster OCR system which has another step added as a pre processor which could increase the efficiency and the time to complete the task. We can see a lot of variations of OCR systems such as check scanning applications for Bank of America and Chase Bank which are used to add checks to your account by just taking a picture of them. The application extracts the details and sends it over to the

server for transaction. Similar application for tax payments was developed by Turbo Tax to easily pay tax via Smartphone. Various language translation applications are there on the application store(Android, Apple & Windows) which extract and translate & then narrate the text.

There is a whole range of OCR software available today in the markets like: Desktop OCR, Server OCR, Web OCR, Mobile OCR etc. Accuracy of extracting text of any of these OCR tool varies from 71% to 95% [Patel 2012]. Many OCR tools are available as paid and work really well but only few of them are open source and free. Tesseract is basically Java code, so that makes it platform independent. Just like any other open source package, this can be forked easily. This is the best part about being open source. In the subsequent sections discuss more about Tesseract and its architecture and the groundwork for this project.

Chapter 2 - Tesseract OCR overview

2.1 Introduction to Tesseract OCR

An Overview of the Tesseract OCR Engine describes Tesseract as: "Tesseract is an open source optical character recognition(OCR) engine [7]. HP originally was originally started it as a project [7]. Later it was modified, improved and taken over by Google and later released as open source in year 2005. It is now available at [8]" (Smith, 2007) . It is very portable as compared to others and supports various platforms. Its focus is more towards providing less rejection and improved accuracy. Currently only command base version is available but there are many projects with UI built on top of it which could be forked. As of now Tesseract version 3.02 is released and available for use. Now Tesseract is developed and maintained by Google. It provides support for around 139 languages [7].

2.2 Type

Tesseract is an example based system. This makes it efficient and flexible. By example based systems we mean that the engine works on a set of example rules defined in the system and results depend on this data. So in simpler words to get good results we need to define these set of rules properly which is called "Training the engine". The reason to flexibility of Tesseract is the fact that we could always change or modify the rules depending on the requirements.

2.3 Architecture

Tesseract OCR is an elegant engine with various layers. It works in step by step manner as shown in the block diagram in fig. 1. The first step in the cycle is to sense the color intensities of the image, named as adaptive thresholding [9], and converts the image into binary images. Second step is to do the connected component analysis [7] of the image, which does the task of extracting character outlines. This step is the main process of this cycle as it does the OCR of image with white text and blacks rest of the image [21].

Tesseract was probably the first [7] to use these cycles to process the input image. After this the outlines extracted from image are converted into Blobs(Binary Long Objects). It is then

organized as lines and regions and further analysis is for some fixed area [7]. After extraction the extracted components are chopped into words and delimited with spaces. Recognition in text then starts which is a two pass process. As shown in fig 1, the first part is when attempt to recognize each word is made. Each satisfactory word is accepted and second pass is started to gather remaining words. This brings in the role of adaptive classifier. The adaptive classifier then will classify text in more accurate manner. The adaptive classifier needs to be trained beforehand to work accurately. When the classifier receives some data, it has to resolve the issues and assign the proper place of the text. More details regarding every step is available at [7][21].

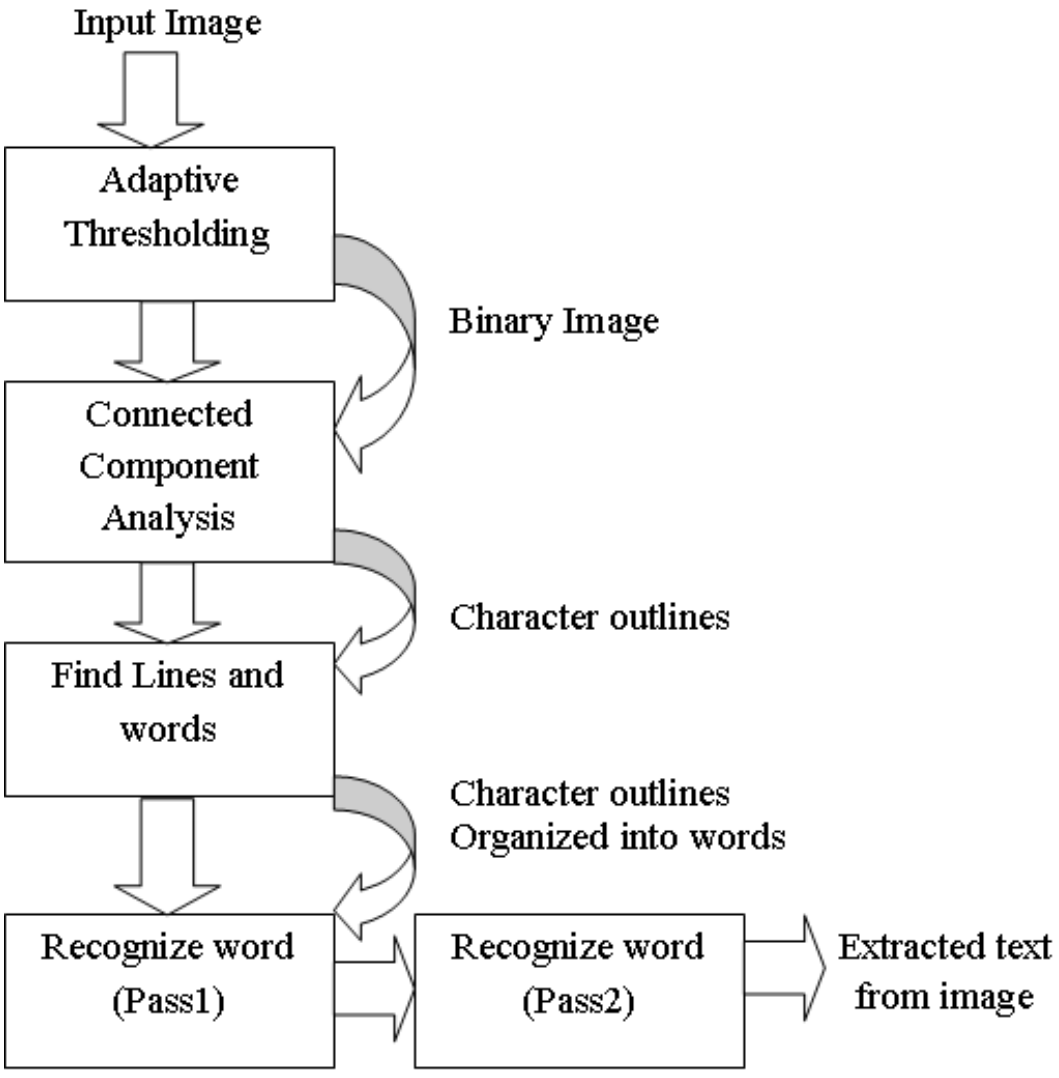

Fig 1: Tesseract flow

2.4 Working of Tesseract

Tesseract works pretty much as a scanner. Its interface is pretty simple as it takes input o the command lines with very basic commands. We need to input any image with text in it. The image is shown in fig.2 [10] for example. Then it is processed by Tesseract. The command to do that is shown in fig.3. The basic Tesseract command takes only two arguments: First is input image that contains text and second argument is output text file which is usually text file [10]. Tesseract by default picks the extension of output file as .txt. There no need to specify the explicitly the output file extension [21].

Fig 2: Image having text[10]

Tesseract supports various languages. Each language comes with a trained language data file. The language file must be kept in a location Tesseract knows. When using in the project it is advised to keep it within the project folder. This folder is Tesseract home folder in your machine. In this research, we are aiming to extract English and Hindi characters from the images so we have to keep both Hindi and English data files. After processing step is completed , the output file gets generated as shown in fig 3. In simple images with or without color (gray scale).

Experiments show that Tesseract is capable of achieving high accuracy such as 95% but in the case of some complex images having multi layered backgrounds or fancy text, Tesseract provides better accuracy in results if the pictures are in the gray scale mode as instead of color. To prove this hypothesis, we ran Tesseract for same images in color and gray scale mode and in both cases different result were achieved [21].

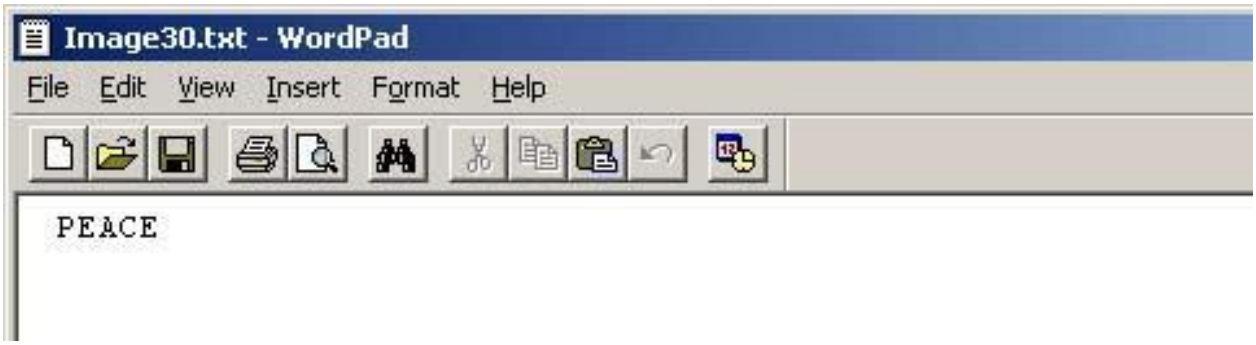

Fig 3: Output of the OCR in text file[10]

Chapter 3 - Previous Work

3.1 Literature review

There were three main considerations in conducting a literature review for the topic "Improving the efficiency of Tesseract OCR engine". The first consideration is that of current technology and limitations. There are a number of applications which are similar to the experiment being conducted. The second is the type of architecture being used in the current applications. The third consideration is the various types of preprocessing steps that could be added without affecting the throughput of the system. The literature review attempts to address these three issues, and show reasonable feasibility in this approach, both theoretically and experimentally. Here the main focus is to add a preprocessing which increases the accuracy and then build this system for Smart Phone platform which, in our case, would be Android. Various publications where gone through and ultimately gave a start to address all these issues to the extent that further research, and possibly actual experimentation, is warranted.

3.2 Current technology and limitations

OCR is widely used in web, desktop and graphic applications. OCR based applications are most privately owned such as World Lens, Card Scan etc. There are much fewer applications specifically for mobile. There are a few open source frameworks for mobile OCR.

Examples of private libraries

RICOH Library:

This library was introduced in late 90's. One of the best libraries of the time, this had a lot of applications such as Ricoh's own printer and scanner products, book scanner algorithms, online picture to document converters. The architecture of this library was ruled based machine translation. It had various rules and patterns defined at character level.

A shortcoming of this system is that it work with perfectly lighted media and processing also

takes a lot of time.

Kusachi et al (2004):

The system was developed in 2004 and incorporated hybrid translation system. By hybrid we mean that it is a combination of statistical and example based system.

The shortcomings of this system are that it supports fewer languages. The system is not flexible enough to add more languages. Also it works on block letters. It does not support cursive letters at all. Again like the previous one it has slower processing.

Open Source Libraries

Tesseract (Google):

The Tesseract was ranked one of the top three engines in 1995 OCR accuracy test conducted by University of Nevada, Las Vegas. The best thing about Tesseract is that it started back in 1995 and is ever improving. When it was taken over by Google it made a transition. It is probably one of the most accurate open source OCR engines available and it is growing ever since then. Tesseract works on Linux, Windows and Mac OSX. The source can also be compiled for other platforms, including Android and iPhone. It supports around 149 languages which come as different packages. Tesseract is an example based text detection system so whatever language we want to work with we need to either download that package or train the engine with our own training data.

The benefits of using this engine are that it is flexible in terms of supporting different languages and could be compiled to run on different platforms.

GOCR:

This is also amongst the top three best rated open source OCR engines. GOCR's best feature is that unlike Tesseract it has graphical user interface that can be used with different frontends. This makes it very easy to work with different languages and architectures. It supports many

different image file formats, and its quality has improved since then.

JAVAocr :

This is yet another OCR engine targeting mobile devices. Most of the features are same as Tesseract and GOCR but the one main feature that stands it apart is that it has very small memory footprint. Very few external dependencies makes it suitable for mobile development. It has a modular structure for easier development and deployment on system. It is built to run on cross platform applications.

There are some disadvantages in this OCR engine. First, it's not fast in processing. Some of the portions work really well but others are lousy. Second, it's not very well documented and supported.

The paper entitled, "Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study"[21], provided a good reference for the capabilities and applications of the Tesseract OCR. The authors have used gray scaling as a preprocessor to the Tesseract OCR engine. The most interesting thing about this paper was the subject matter and the test data. They presented overview of the subject and then the experiment they took out to OCR the car license plates with an added preprocessing. This article provided relevant, useful, and up-to-date information regarding Tesseract OCR and its use. This paper is a very good source, giving this paper a good comparative background and introduction to Tesseract. Then there is a comparison of the results using the new system they built with the normal Tesseract OCR.

The applications based on these open source Frameworks are very few especially for Hindi. The below literature review gives an idea about various research works on Hindi and English OCR applications and various types of implementations.

Some other papers[28],[29] and [30] that provide good insight on Hindi OCR are as the initial steps in the systems architecture. These research works for Hindi OCR apply to both desktop and

mobile applications. Based on these studies we will lay out the framework for our application. They also serve as a benchmark for the result and conclusion of the effectiveness of my experiment.

The setup for Hindi OCR within Tesseract is another important task. The way you train the engine and input the dataset matters a lot.

Training Tesseract for Hindi includes four steps mainly and they are as follows:

Generating training images:

Tesseract official website explains these steps very well "The first step is to determine the full character set to be used, and prepare a text or word processor file containing a set of examples" [31]. The two important points to remember for a training file are: Firstly make sure the file contains all the characters that we are expecting. Secondly there should be at least 5 samples. There should be more samples of the more frequent characters - at least 20 [31]. Keeping all this in mind let's explore how do we create box files.

Make box files:

Box file is defined as a sample to match with the characters. We need a 'box' file for each training image. Tesseract official website defines the box file as "The box file is a text file that lists the characters in the training image, in order, one per line, with the coordinates of the bounding box around the image" [31]. Tesseract is not so intuitive about the sample data. Inconsistencies may lead to wrong interpretation of data.

Run Tesseract:

In this step we run the Tesseract engines with the new training files. The purpose is to create the trained dataset which works as a rule engine. Also it creates log files.

Compute character set:

Guidelines from Tesseract website state that "Tesseract needs to know the set of possible characters it can output. To generate the unichar set data file, use the unicharset_extractor program on the box files generated above" [31]:

extractor language . fontname. box

One more requirement in this is, Tesseract needs to have access to character properties i.e. isalpha, isdigit, isupper, islower, ispunctuation [31].

The system described in [30],[31] have classified Hindi OCR and its setup, training very nicely. The only limitation is that the system is for desktop. We have to build the similar system for mobile devices.

3.3 Various types of architectures

The paper "An Automatic sign recognition and translation system" [22], provided an in-depth look at the various types of architectures that we could use. This presented the working of Tesseract on the Android system. The attributes analyzed in this system were the accuracy and efficiency of Tesseract OCR on a smart phone. The scope of my research is also towards smart phones so this is a good starting step for my experiment. The objective is to determine if Tesseract would be feasible to be used on smaller machines.

ATMA[30]:

This application uses the original OCR engine without any enhancement. The system is built on an Android device. Architecture is simple and is implemented in Android operating system. OCR Engine used is Tesseract 3.01 which is the latest stable version. The device is not mentioned but it is for sure a smart phone with Android. Other tools used are: Android NDK r7 and Android SDK r16 for compiling and building the android project. Capture mode used is "still camera".

Complete OCR for Hindi Text[28]:

This application also uses Tesseract OCR engine. This version is not specified though. Application is mainly for printed Hindi text and classifies Hindi characters very nicely. The system is built for Desktop computers. Capture mode is still camera. It uses segmentation of characters as a technique and post processing for error detection.

Shirorekha chopping[29]:

This application uses Tesseract 3.01 OCR engine. The target operating system mentioned is Ubuntu. Main feature described in this paper is chopping the image character by character. The system trained the Hindi data in one of the new font's. The system does a comparative study against Google's stock Hindi training data. Various pre-processing steps were applied.

TranslatAR[32]:

The device used in this system is Nokia n900. This paper demonstrates one of the newest architectures i.e. video augmentation capture mode. That means that this system deals with videos frames in the real time. Other features included are foreground-background color extraction technique.

Looking at all these applications and their architecture, one thing that is realized is that when there are a lot of preprocessing steps a lot of time is spent before even the extraction begins. So preprocessing step needs to kept minimal to save time. On the architecture side Tesseract 3.01 is the most stable form , so that will be the target version of OCR engine. To keep the things simple, the system won't implement the OCR in video mode. The aim of this project is to extract the characters efficiently in least amount of time.

3.4 Types of preprocessing steps available

The article "Transcription for the OpenPlaques project" [23], provides various types of preprocessing methods on the images. The article gave good insight on the computer graphics rendering and filtering. In this paper, individual methods are analyzed and their results identified and studied. The methodologies presented will be utilized to determine the feasibility of using this method as a pre processor in the solution. The article also presented advanced features such that text filtering using a sliding window on the target image [23].

Steps available in Tesseract:

Binarization:

Binarization of the image means to convert the image of up to 256 gray levels to a black and white image. Binarization is used mostly as a preprocessing step to image processing tools. This works by choosing a threshold value and classify all pixels above this and below this value. Then we just normalize the images to that threshold value. Now how to select the correct threshold is the main question. Due to this uncertainty, adaptive image binarization is commonly used because it adapts a threshold value depending on the image color levels.

Grayscaling:

Grayscaling means to transform the image into black and white. Usually colored images when converted to black and white reveal a lot of information. Most of the computers can normally represent up to 256 levels of gray color. Grayscaling is a process which converts a continuous tone image to gray scale levels. The process is used in a lot of real time image processing tools. Because of this most of the CCTV's and traffic light camera's are grayscale. It makes the separation of various parts of the images easier. We need to take care of the DPI along with the image's resolution. Different capturing devices capture images in different resolutions. This incurs a lot of inconsistency. This problem can be tackled effectively using data compression techniques, but still grayscaling technique is bound to use a lot of memory [24].

The need for further preprocessing

Current implementation on Tesseract works fine for desktop or laptop computers. But for mobile devices we need something lighter and efficient. The preprocessing steps prepare the input image to be almost ready for extraction by the Tesseract Engine. Also the preprocessing aims to remove noise, light variations etc which impair the task of recognition. Let's look at other algorithms available to achieve this.

More steps available are:

Luminosity

Luminosity method is a technique which uses grayscaling as the base. For converting an image into Luminosity we convert it into grayscale then preserve some light intensities.

Luminosity is a almost like the grayscaling method, but more sophisticated to take the human perception of color into account [24]. As a matter of fact the human eye is functions bit differently than computer graphics. It is more sensitive to green and least sensitive to blue color intensities. So in the process we adjust these frequencies to be preserved.

OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions and algorithms [36]. Their main focus is to provide API mainly aimed at real-time computer vision. It was originally developed by Intel. OpenCV library is free for use for development (under the open source BSD license) [36]. The best feature is that the library is cross-platform .

Linearization

Linearization is basically adjusting the image from blurs and fuzzy edges. Sometimes due to motion while capturing the edges bleed into each other and gives fuzzy image. Linearization technique handles this very nicely and takes care edge by edge.

Pixelation

Used for individual character segmentation. Pixelation is defined as displaying the individual pixels of the digitized images. In this we display block for each pixel at a distance to each other which is apparent to the users. This can happen unintentionally too when a low-resolution image get stretched on large images.

3.5 Existing Applications on App Store

World Lens:

World is one of the leading applications available on the web for both Android and IOS smart phones. It is one of its kind and uses augmented reality to translate the captured content. Developed by Quest Visual, world lens uses built in camera of the phone to quickly scan and identify text and foreign languages. The processing is fast and utilizes video mode. So there is not actual saving of images in the phone memory. The words are displayed in context to the

original text after translation.. World lens is available as trial version as well as paid version for the Apple's IOS as well as for a selection of Android smart phones.

This is the best rated applications available in market. This is a close source applications which supports more than 40 languages. It extracts as well as translates the characters.

Some of the cons for this application with respect to the proposed system are that it needs internet connection to work. The application is heavy in terms of processing because it is dealing in the video mode. Works best with high end devices. It also needs a lot of memory.

Mobile OCR:

This is a mobile application which makes possible obtaining text and working with it from pictures taken from a camera hardware. It has a simple and direct multilingual interface, which lets us access its features in a fast and effective way. The customized interface allows us to adapt the characteristics of the camera for an optimal image preprocessing that will improve the results obtained by the OCR. Also, the advanced text post processing techniques developed by the Smart Mobile Software increase the effectiveness until the achievement of almost perfect results.

It is not an open source application. It is compatible with IOS and Android devices. The application supports many languages. It works for digital as well as printed text. The application is lighter than other competitors.

One of the cons for this application with respect to the proposed system is that the app needs to download app data for each language. The processing of the overall system is slow.

Image to text OCR:

The ImageToText app is a free app which allows us to extract characters text from images, and share the results over the web. The app is fairly simple to use. Working is as simple as taking a picture, from the camera, of a document that you would like to OCR, and e-mail the image to yourself or share it. You will receive the image as well as the text file that contains the editable text that is extracted from the image. This currently supports English documents only. It is

developed by Ricoh Innovations. It is a closed source application and is compatible with IOS and Android devices.

One of the cons for this application with respect to the proposed system are that it has no support for languages apart from English. Also the accuracy is much lesser than the other applications.

After getting enough ideas and instructions from the above presented research work we'll start implementing the proposed system using Tesseract OCR for Android devices. The subsequent sections include the implementation and working of the system.

Chapter 4 - Implementation

4.1 Setup

To conduct this experiment there are a few things that must be configured onto the system. The device used for this experiment is primarily Samsung Galaxy S3 running on Android 4.1 version. To install Tesseract onto the system we have to follow certain steps on the machine being used for development[8].

Installation steps include downloading the source from the Tesseract website(installer) [8]. Installer depends on the type of operating system being used. After that download Android SDK r19 and Android NDK r7c(open source) to open and build the code. We need Android NDK because of some native code in Tesseract. This code is basically the Leptonica image processing library which is written in C. Then we need to build the project using the following commands in the IDE:

```
cd <project-directory>/tess-two
ndk-build
android update project --path .
ant release
```

The initial setup is the most crucial part. If it is not done properly one may keep getting weird errors during execution. Once this is done then the system could be installed on Android system.

4.1.1 Running Tesseract

Running Tesseract on Android system is one of the biggest challenges experienced in the project. After successful installation we can verify whether the Tesseract OCR engine is working fine by calling the below steps [25]:

Tesseract does not have any graphic user interface and works on command-line. We have to open

a terminal window or command prompt window in the computer. A very simple example of the command is shown below [8] [25]:

```
tesseract image_name output [-l lang] [-psm page_seg_mode] [config_file...]
```

So in simpler words, a basic command to do OCR on an image named 'input_image1.png' and saving the extracted result to 'output_image1.txt' would be [25]:

```
tesseract input_image1.png output_image1
```

Or to do the same thing with some other language say Hindi [25]:

```
tesseract myinput.png out -l hin
```

Training Tesseract for Hindi

Steps to train Tesseract for a particular language were presented in Section 3.2. We would be following the same steps to train Tesseract again for Hindi character set in 4 fonts. It is very important for accurately extracting characters.

Examples of the font files are as follows:

Guḍākeśa 99	आसीद्राजा नलो नाम वीरसेनसुतो बली। उपपन्नो गुणैरिष्टै रूपवानश्वकोविद्: ॥
Śāntipur 99	आसीद्राजा नलो नाम वीरसेनसुतो बली। उपपन्नो गुणैरिष्टै रूपवानश्वकोविद्: ॥

Fig 4: Hindi fonts

Example of the box file is as follows:

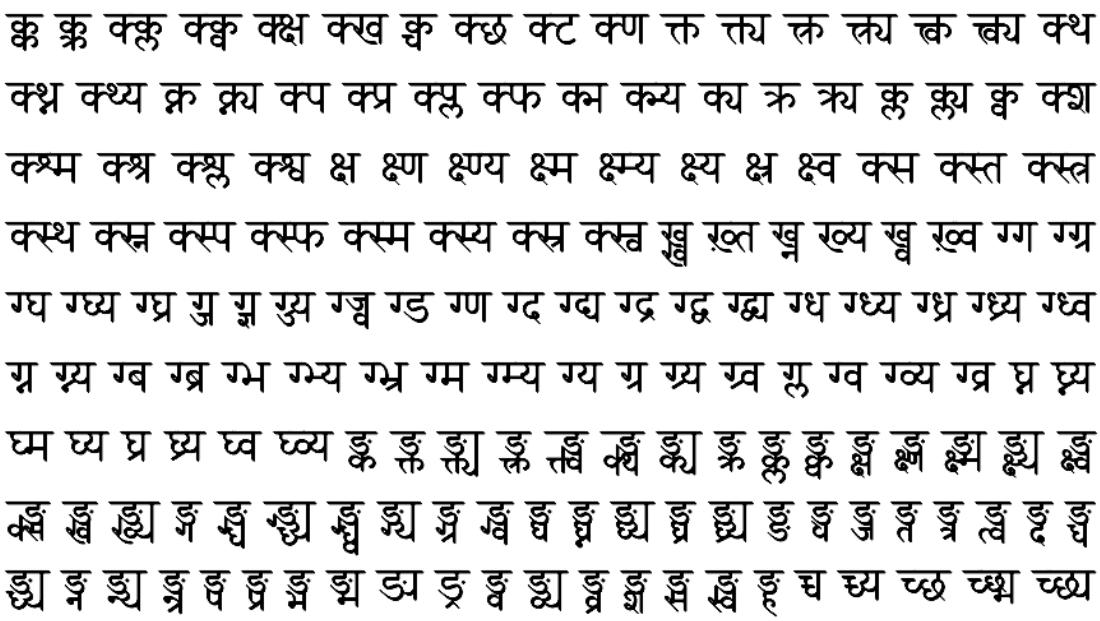

Fig 5: Box file

4.2 Architecture

In the previous sections we did an extensive search on the existing application architectures. One thing common to all these applications is that to get better accuracy preprocessing is required. Also most of the application implement still camera capture mode to image input. Let's start laying out the overall architectural diagram of the proposed system. By now various pieces(Tesseract engine, preprocessing, Android migration) of the system are clear and we'll start putting them up in order.

Overall Flow:

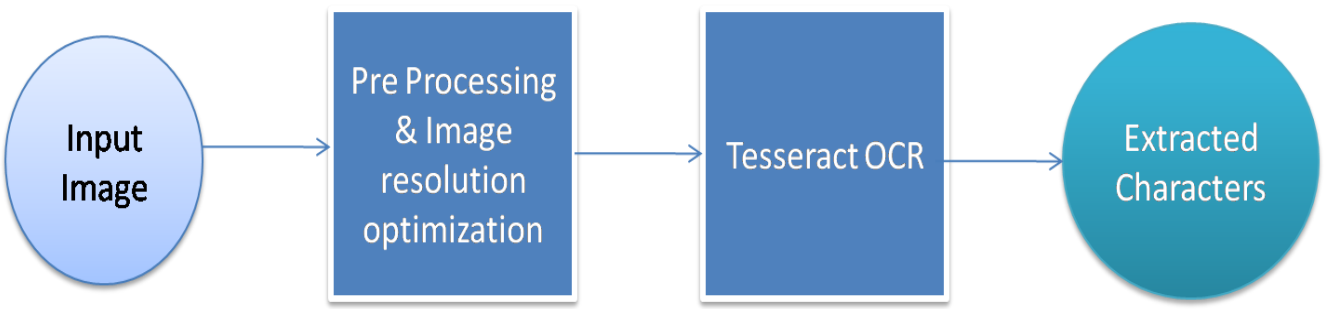

Fig 6: Architectural flow

The above image shows the overall architecture of the system. Basically it has two main subsystems. First is the Preprocessing and second is Tesseract API. The project's main focus area in this report is the preprocessing step which works on the input image to make it ready for Tesseract engine. One thing to note in this is that there is a tradeoff between processing and accuracy. The more time you spend on preprocessing gets more accuracy but it increases the runtime too. The figure below explains the relationship very nicely.

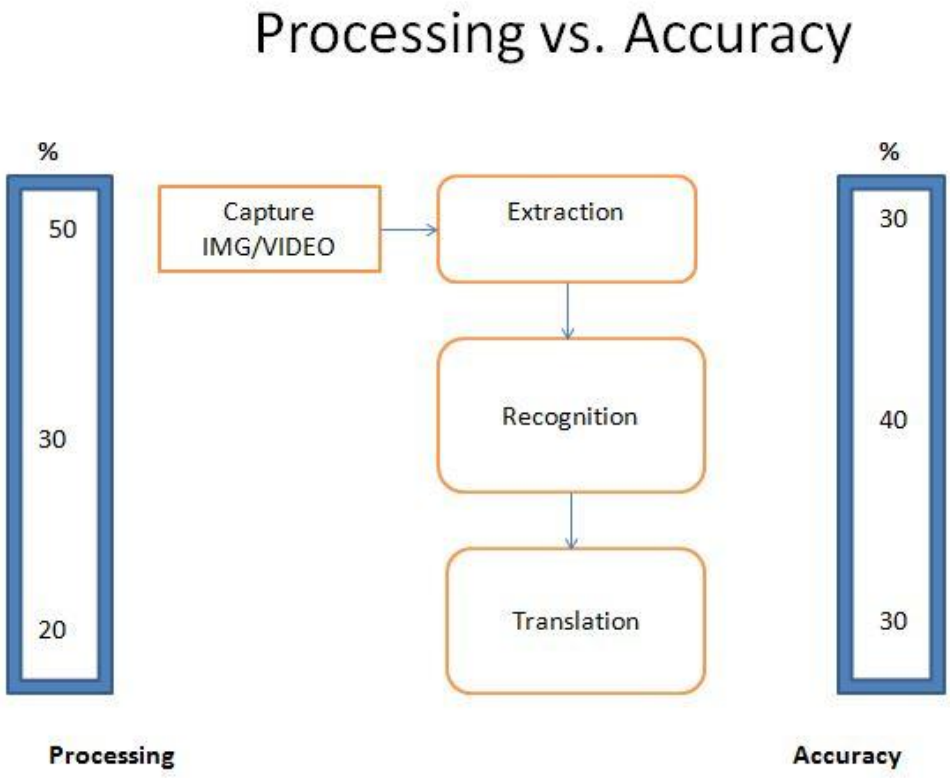

Fig 7 : Processing vs. Accuracy

The image explains the distribution/weight for different steps in OCR with respect to processing and accuracy. On the processing side most of the time is spent in the extraction step. It is because how well the characters are extracted better it is for the recognition step. Then recognition and translation are almost equally weighted. On the accuracy part most of the accuracy is incurred in the recognition phase. It is on the part of rule matching subsystem to recognize and match characters efficiently.

Preprocessing Step:

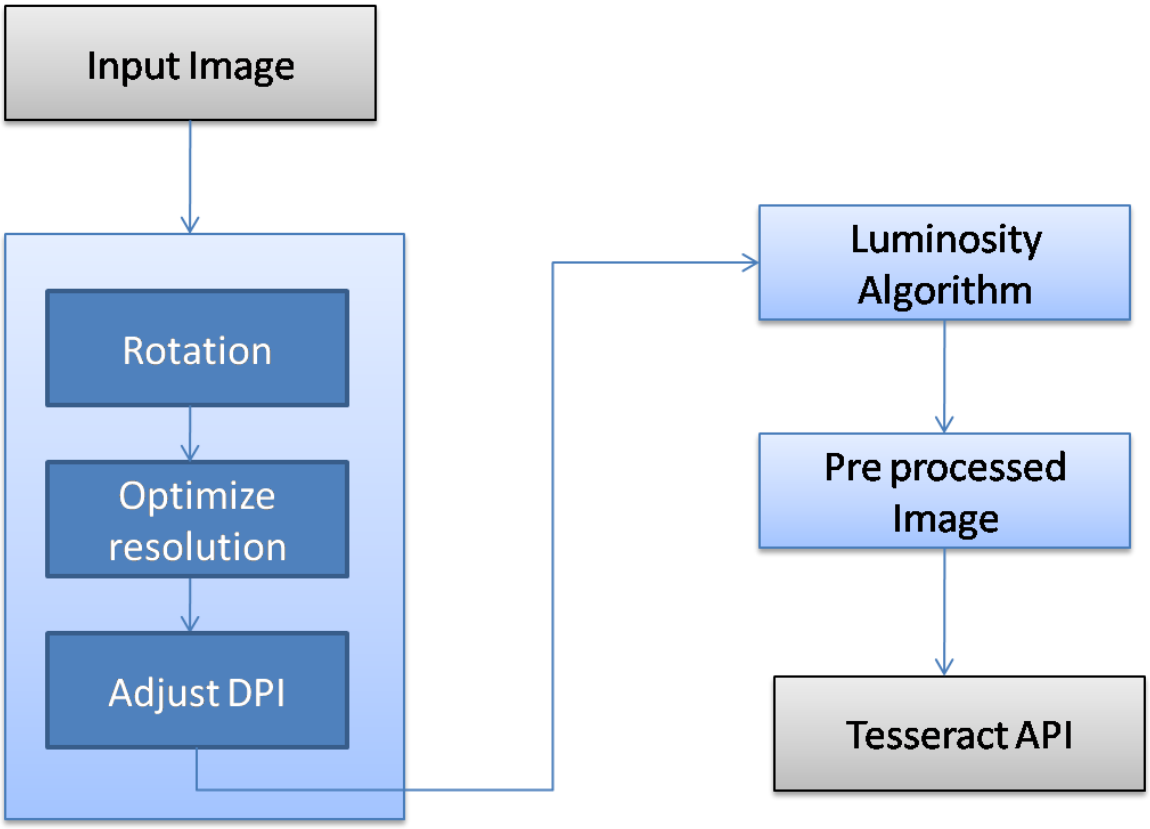

Fig 8: Preprocessing Architecture and steps

The above image explains the Preprocessing subsystem of the proposed application. The various steps in the pre-processing phase are: Rotation, optimize resolution, adjust the DPI's and lastly the Luminosity grayscaling algorithm is applied to the images. The processed image is then fed to Tesseract as input. Rotation step rotates the image if while capturing the camera was not in Zero degree angle. It's a relatively smaller step. Resolution optimization is also comparatively smaller step which compresses the larger images to best resolution for Tesseract. Most of the time is consumed in Dpi adjustment and grayscaling algorithm. They are the main steps which makes the image ready for Tesseract.

Android App architecture:

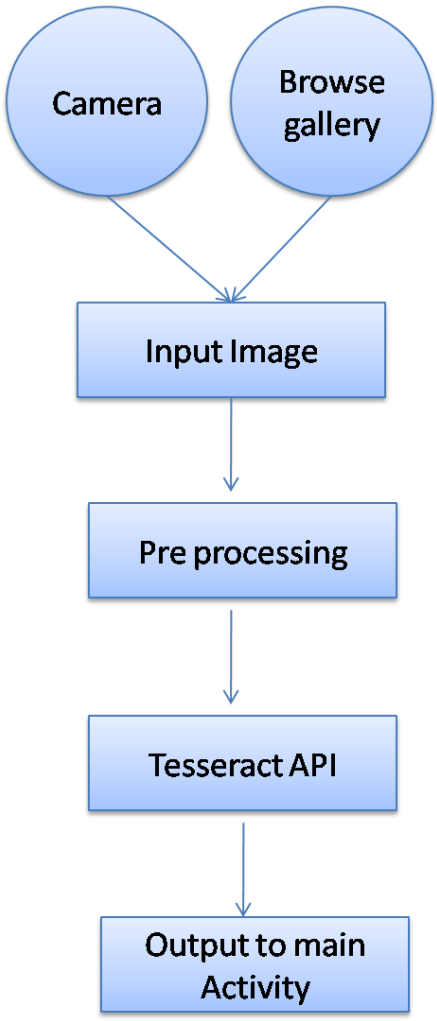

Fig 9: Android application flow

The image above explains the architecture from the Android point of view. The application flow is as follows: The first step is to choose the input image, which could be either clicked from the camera of selected from the image gallery. After selection the image is fed to the preprocessing subsystem. The processed image is fed to Tesseract API which extracts the characters from the image and the result is displayed to the screen.

4.3 Tesseract Android Tools

Tesseract Android tools is another project which helps in compiling the Tesseract and Leptonica libraries so that they could be used as an API, for example, on the Android platform. So it is structured as a service which exposes API calls to native code. The API is written in JAVA for works fine with Android projects. This step is very important to port the system built so far onto Android. The Tesseract library will behave as an API to the project's front end. We need to compile the project using Android tools to make it available as an Android project.

4.4 Preprocessing step

Luminosity Technique

Luminosity theshold grayscaling is a method for converting an image into grayscale but preserving some of the color intensities. The Luminosity technique is almost like the average color method, but more sophisticated to take the human perception of color into account [24]. The human eye is more sensitive to certain color. For instance it is more sensitive to green and least sensitive to blue. Below is the equation for getting the color of a pixel [24]:

$$20 + 70 + 150 over 3$$

Each pixel is converting using the above formula. Below is the illustration of converted images using both methods of gray scaling.

Result images:

Fig 10: Colored Image

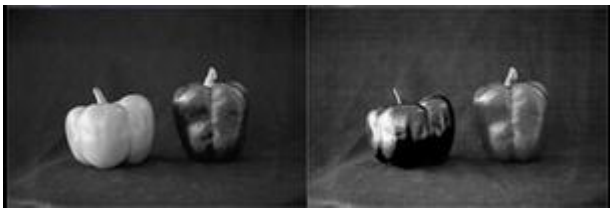

Fig 11: Output of Luminosity vs. simple gray scale

Algorithm below defines how we process image luminosity technique:

```
// Get buffered image from input file;

// iterate all the pixels in the image with width=w and height=h
for int w=0 to w=width
{
        For int h=0 to h=height
        {
                // call BufferedImage.getRGB() saves the color of the pixel

                // call Color(int) to grab the RGB value in pixel

                Color= new color();

                // now use red,green,black components to calc average.

                int luminosity = (int)(0.2126 * red + 0.7152 *green + 0.0722 *blue;

                // now create new values

                Color lum = new ColorLum
```

```
                    Image.set(lum)

                    // set the pixel in the new formed object
                }
            }
```

In this algorithm we traverse the image pixel by pixel and then store it in an array. Then the formula for luminosity as shown below is applied.

```
luminosity =  (0.2126 * red portion + 0.7152 * green portion + 0.0722 * blue potion);
```

This is the main step where we transform each pixel to get the real details out of it. Finally we again pack all the pixels to form the image again.

4.5 DPI Enhancement

To get the best results out of the image we need to fix the DPI too. Grayscaling alone would only work in the case when there is no distortion, light effects in the image [26]. We won't be experiencing such ideal images every time. Some steps we need to consider in DPI enhancement are: fix DPI (if needed) 300 DPI is minimum acceptable for Tesseract. Better range of DPI results in better extraction process. The reason why we need this step is we'll be using this application on different smart phones. Each one has different camera specification and pixel density. So it is better to normalize the picture before saving it in the gallery and make it consistent to the Tesseract engine.

Since in our case the images are not perfectly clicked and ideal, and also we want to keep the process light and suitable for mobile phones , we'll keep the scope limited to DPI enhancement only. For our images we fix the DPI to 300 (needed by Tesseract) [26]. If the images are larger than this we compress the images to the size where we can achieve 300 dots per inches.

The figure below describes the effect of DPI enhancement

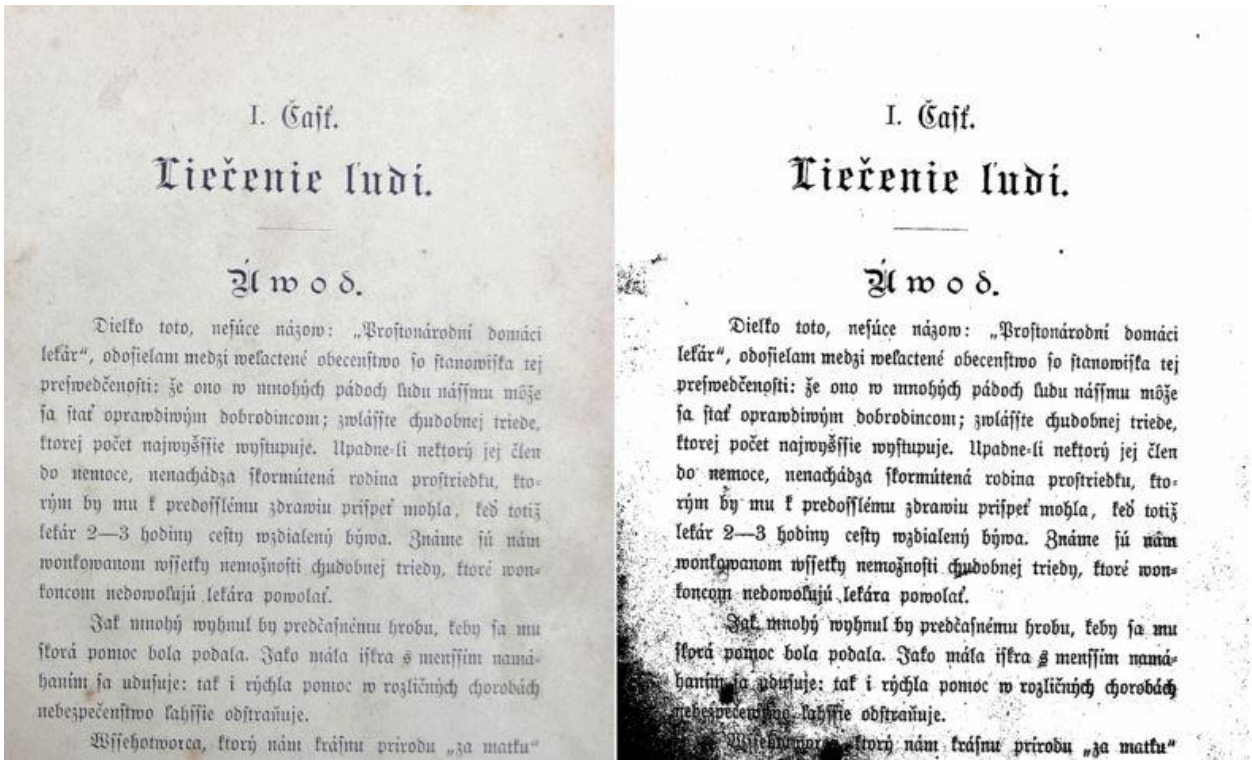

Fig 12: DPI enhancement

The figure below shows how the algorithm works to refine the edges.

Fig 13: edge enhancement

Algorithm for DPI enhancement:

```
start edge extract (low, high){

            // define edge
               Edge edge;
            //  form image matrix
                Int imgx[3][3]={
               }
               Int imgy[3][3]={
               }
               Img height;
               Img width;

               //Get diff in dpi on X edge

              // get diff in dpi on y edge
               diffx= height* width;
               diffy=r_Height*r_Width;
    img magnitude= sizeof(int)* r_Height*r_Width);
    memset(diffx, 0, sizeof(int)* r_Height*r_Width);
    memset(diffy, 0, sizeof(int)* r_Height*r_Width);
    memset(mag, 0, sizeof(int)* r_Height*r_Width);

             // this computes the angles
             // and magnitude in input img
               For ( int y=0 to y=height)
                   For (int x=0 to x=width)
                         Result_xside +=pixel*x[dy][dx];
                         Result_yside=pixel*y[dy][dx];

              // return recreated image
               result=new Image(edge, r_Height, r_Width)
     return result;

}
```

In a nutshell this algorithm works to normalize the pixel density. Each pixel has its size and depth of the image is denoted by number of pixels packed in it. As already discussed , the system can encounter image with lesser density than minimum required or a very large picture.

4.6 Combined picture

Following are the screenshots from the working Android application. The system on which the project is running is Samsung Galaxy S3 and operating system is Android 4.1 jelly bean.

Figure 14 shows the main activity which is the main landing page of the application. When we install and run the application this is the page that opens. It has two options to choose the image for OCR. First is the camera button which opens the camera and lets us click pictures. Second option is to choose existing image from the gallery. In this case the image could reside anywhere in the app memory or SD card.

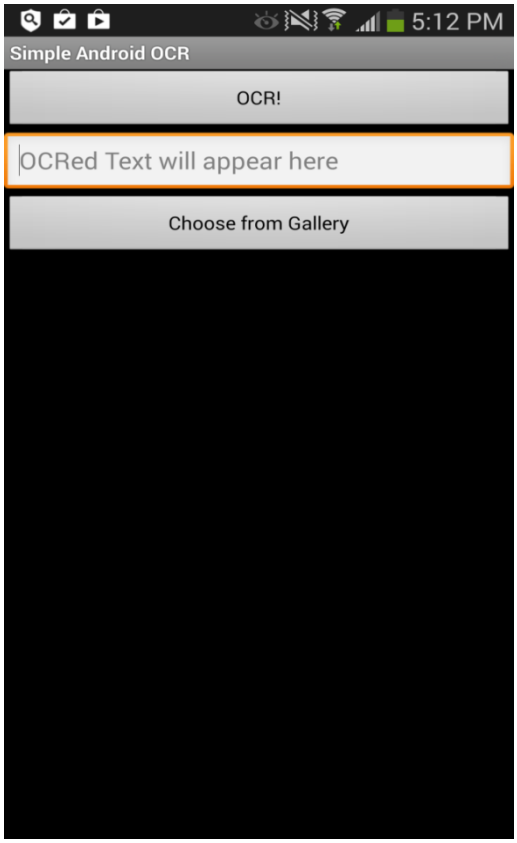

Fig 14: Main Activity

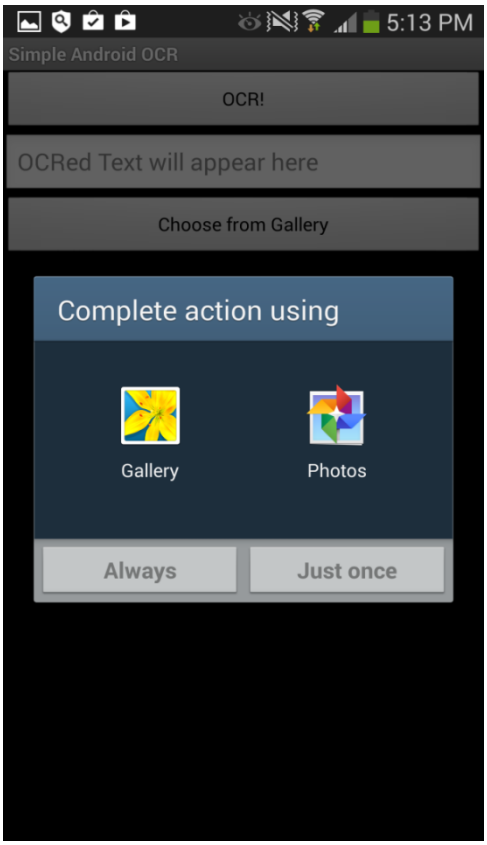

Fig 15: Choose from gallery

Figure 15 shows the popup that opens when we click the button "choose from gallery" and takes us to the phone's image gallery. Figure 16 shows the output text appears in the text box after the OCR. Figure 17 and 18 show the application data inside the app. This folder is created once the application is installed and stores the trained data files. Tesseract uses this location to process the

input images and to compare the extracted characters against the trained data.

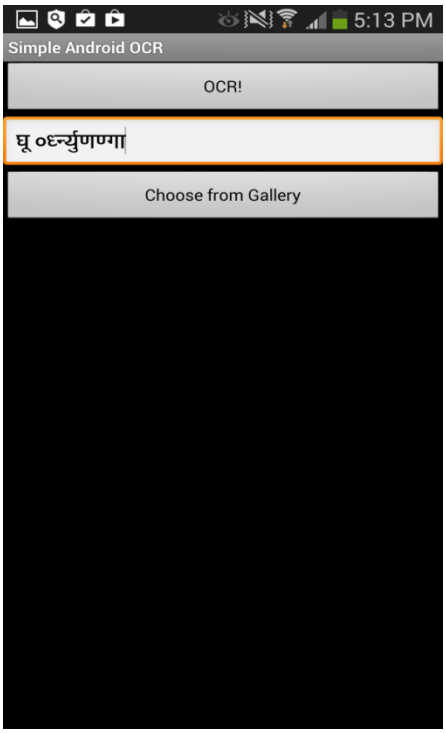

Fig 16: OCR result

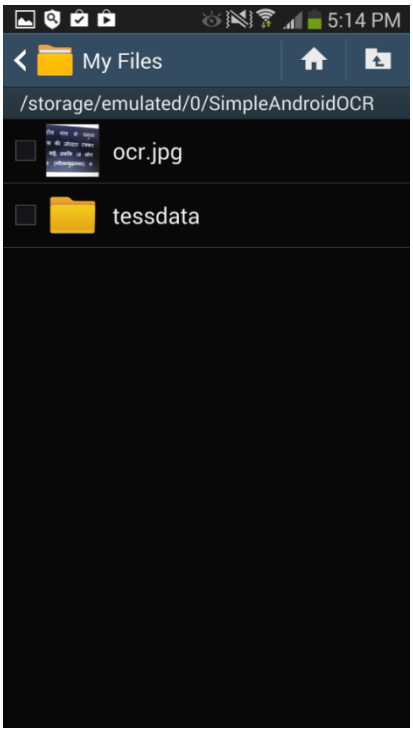

Fig 17: App data

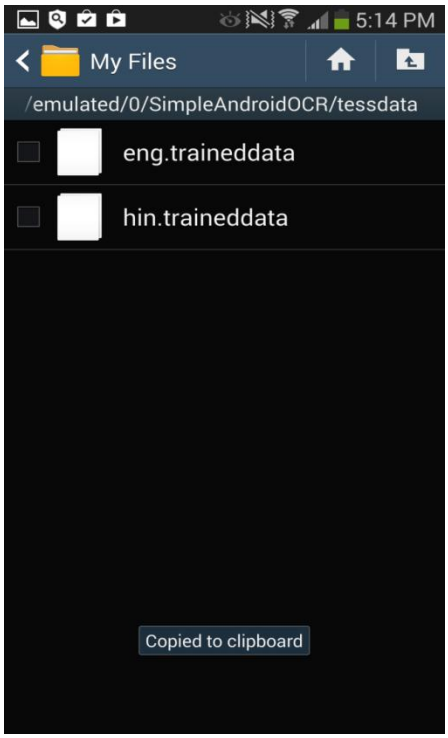

Fig 18: Trained Data in SD Card

As presented in the above images the fully functional Android application that was proposed in the beginning. In the subsequent section results of the experiment would be compared against other research works.

Chapter 5 - Results

5.1 Experiment results

Following are the results from the experiment conducted based on the data set from previous research works. Results were gathered separately on Hindi and English data.

Comparison with Hindi OCR apps:

ATMA [30]:

Android Travel Mate Application is the application which runs Tesseract in its core just like our system and it extracts Hindi text too. So it the first research work to compare our system with. The data provided was a set of random images , logos and text in Hindi. The system was run several times for this data set. Following is the table showing the results of the experiment with this data set:

Image no	Image Type	Accuracy	Runtime
1	3 words	70	1745
2	4 words	70	2304
3	2 words	90	1845
4	2 words	100	2372
5	2 words	95	1599
6	10 words	50	2478
7	2 words	50	1597
8	3 words	30	2000
9	8 words	70	3568
10	12 words	75	3448
11	3 words	90	1709
12	15 words	90	5629
13	13 words	70	6179
14	9 words	85	4723
15	8 words	80	2966
16	3 words	90	1172
17	2 words	90	1347
Average	**101 words**	**76.17%**	**46.4 sec**

Table 1: result 1

Our experiment results(Table 1) shows that **Average time taken per word is** 0.459 sec. The original paper results are as follows. Let me now compare the results of both.

Language	English	Hindi
Avg. Mean Confidence per word	69	24
Avg. Time taken per word	153 ms	681 ms
Light and Noise susceptibility	LOW	HIGH
Average Accuracy	**97.9%**	**79.2%**

Fig 19: ATMA[30] results

The experiment clearly shows percentage decrease in runtime than the original paper i.e. the time taken to extracts the words. The average time taken per word decreased from 681 ms to 459 ms.

For few images such as image no 6, 7 and 8 (highlighted in gray color), the system showed very less accuracy percentage. The reasons for getting less accuracy were found out to be: special symbols overlapping with character boundaries and light distortions.

The new system keeps the accuracy almost the same as the original paper results.

Shirorekha Chopping Integrated Tesseract OCR[29]:

The experiment was conducted against the data set from the above research paper[29]. The data set consisted of images with large no of Hindi characters. The results from the experiment are as follows:

Image no	Image Type	Accuracy	Runtime
1.	21	90	1079ms
2.	29	91	1100ms
3.	20	92	1054ms
4.	24	89	1066ms

Table 2: result 2

The results from the experiment depict that the Average time to process each image is 1074 ms . Average accuracy was measured to be about 90.5%.

Original Paper results are as follows:

	Processing Time	Total Characters in Test Image
Google's hin.traineddata	2000 ms	94
Parichit's hin.traineddata	1500 ms	94
Proposed hin.traineddata	1000 ms	94

Fig 20: Shirorekha results[29]

The results from both the experiments are nearly the same. The results in the figure were conducted on Desktop machine whereas the new system runs the experiment on Mobile phone while keeping the runtime and accuracy same on lower power machine.

Comparison with English OCR

Let us know compare the results for English character recognition. The new system was ran against data set from recent research works on English OCR as well.

The experiment was conducted against the data set of the research paper[21].

Image no	Image Type	Accuracy	Runtime
1	5 WORDS	100	2202
2	6	97	1455
3	3	100	1127
4	3	100	1539
5	3	100	1247
6	4	98	3674
7	1	100	1165
8	7	95	1752
9	7	100	1281

	39 words	98.8%	15 sec

Table 3: result 3

The results from the experiment show that Average time taken per word is 0.36 sec per word and Average accuracy is about 98.8 %. The results from the original paper are as follows:

gray scale	5	56	0.402	20
gray scale	4	40	0.548	75
gray scale	7	70	0.402	42.86
gray scale	4	40	0.701	25
gray scale	4	44	0.705	100
gray scale	2	22	0.7	100
gray scale	8	73	1.717	25
gray scale	6	67	0.806	16.67
gray scale	6	55	0.596	16.67
color	9	100	3.048	
color	9	100	1.007	
	Average Accuracy	**70**		

Fig 21: result from paper[21]

The results clearly show the increase in efficiency in the results from the new system.

5.2 Conclusion

The experiment results show a significant increase in efficiency specially in Hindi character recognition. Keeping the efficiency same, the system is able to process the image in lesser time.

Original paper[30] has average runtime of 681 ms and the experiment results show the runtime of 459 ms keeping the accuracy almost the same. The consistency of system for English OCR is also the same. We are able to demonstrate significant increase in efficiency and of English OCR vs. Hindi OCR.

Original Paper	**Our Experiment**
Ratio of Eng to Hindi runtime: 153ms: 681ms	**Ratio of Eng to Hindi runtime**: 360ms: 459ms

Table 4: Conclusion

Hindi to English recognition rate in paper[30] is about 153 ms: 681 ms. The results from new system show the decrease in this ratio. This shows that we are able to improve character recognition for Hindi language on the whole by improving the process for recognition and that too on mobile device.

5.3 Examples

Input Images for the experiment:

Fig 22: test image

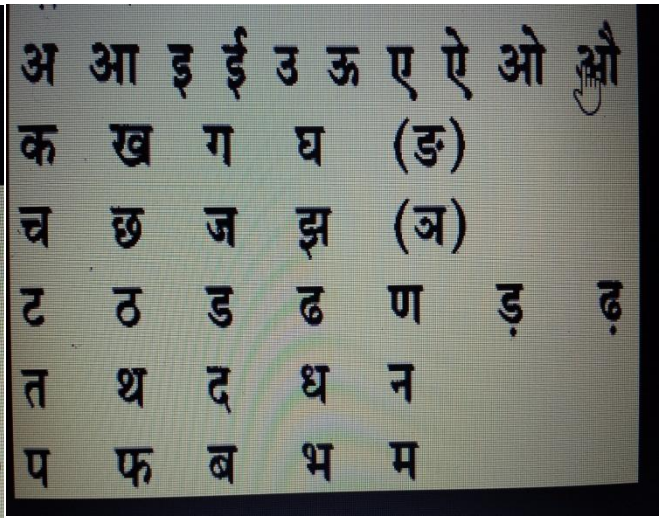

Fig 23: test image

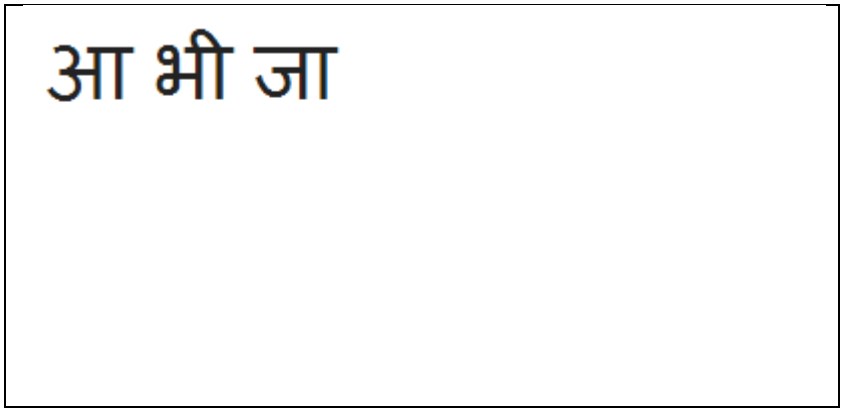

Fig 24: test image

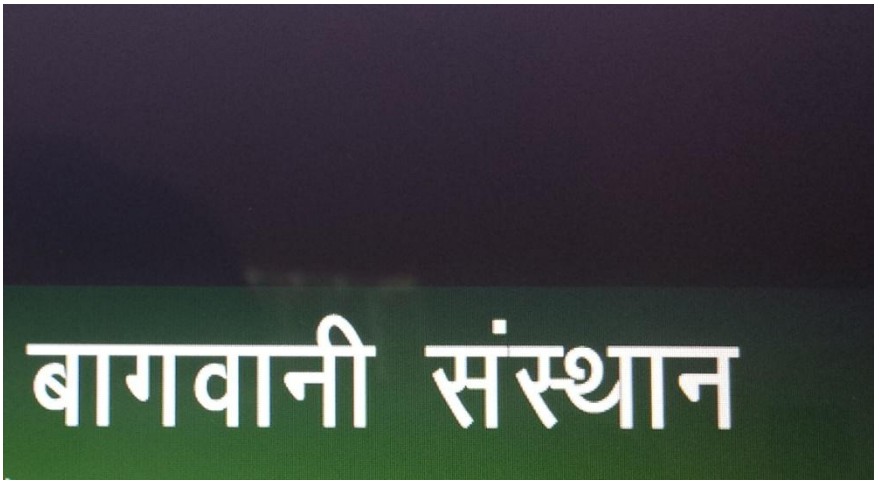

Fig 25: test image

Output Images:

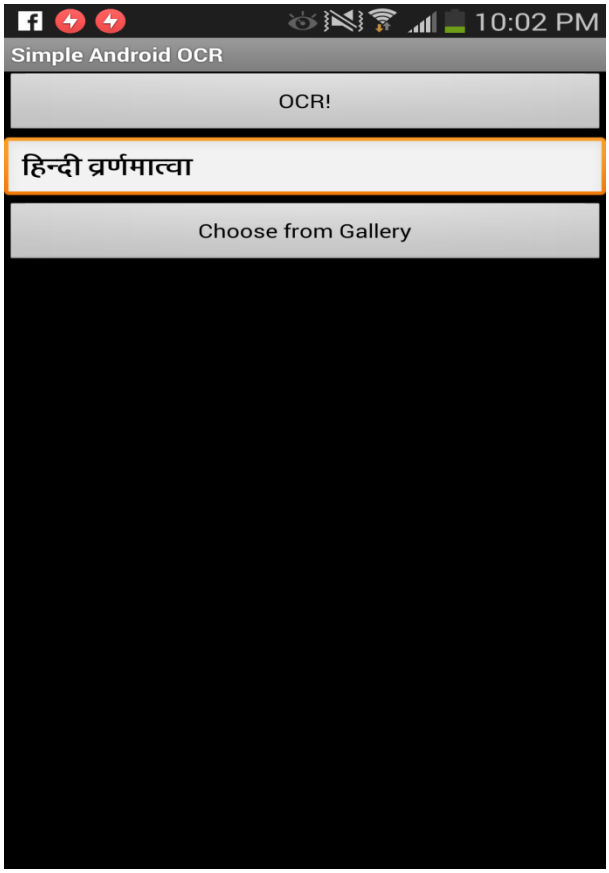

Fig 26: output image

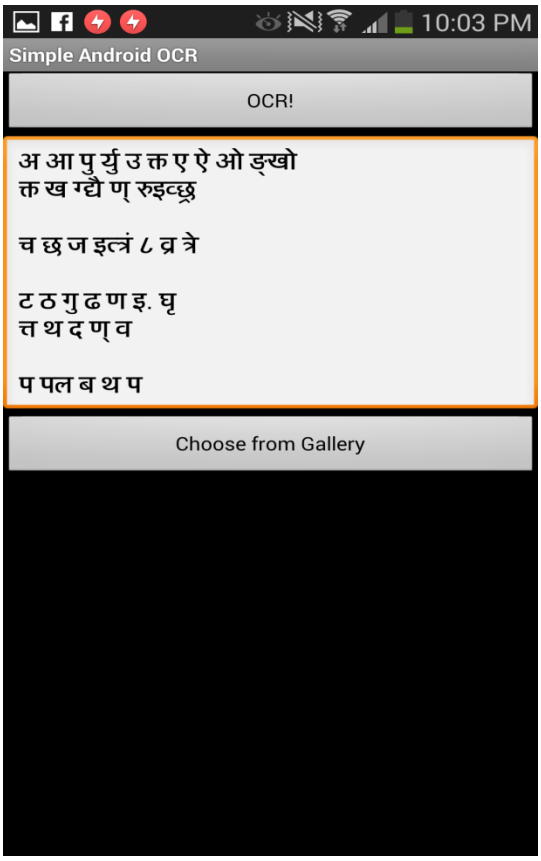

Fig 27: output image

5.4 Future Work and Scope

OCR is a very useful and popular application. It is being used in a lot of domains currently. The idea of improving and improvising the OCR for Hindi text is a very different aspect. India is a country with more than a billion people. Hindi being the national language is the used and required everywhere. Hindi OCR and translation application for mobile phones solves a greater piece of the problem stated in the beginning of the report. The experiment could be used to build translation apps for Hindi language. This in turn could be used in various departments such as educational institutions, transport, research and development.

References

[1] ARCHANA A. SHINDE, D**.** 2012.Text Pre-processing and Text Segmentation for OCR. International Journal of Computer Science Engineering and Technology, pp. 810-812.

[2] ANAGNOSTOPOULOS,C.,ANAGNOSTOPOULOS, I., LOUMOS, V, & KAYAFAS, E. 2006. A License Plate Recognition Algorithm for Intelligent Transportation System Applications**.,** IEEE Transactions on Intelligent Transportation Systems, pp. 377- 399.

[3] Y. WEN, Y. L. 2011. An Algorithm for License Plate Recognition Applied to Intelligent Transportation System., IEEE Transactions on Intelligent Systems, pp. 1-16.

[4] XIN FAN, G. L. 2009. Graphical Models for Joint Segmentation and Recognition of License Plate Characters. IEEE Signal Processing Letters, pp. 10-13.

[5] HUI WU, B. L. 2011. License Plate Recognition system. International Conference on Multimedia Technology (ICMT). pp. 5425 - 5427.

[6] PAN, Y.-F., HOU, X., & LIU, C.-L. 2008.A Robust System to Detect and Localize Texts in Natural Scene Images.The Eighth IAPR International Workshop on Document Analysis Systems.

[7] SMITH, R. 2007. An Overview of the Tesseract OCR Engine. In proceedings of Document analysis and Recognition.. ICDAR 2007. IEEE Ninth International Conference.

[8] GOOGLE. Google Code. google code. [Online] 2012. http://code.google.com/p/tesseract-ocr/.

[9] F. SHAFAIT, D. K. San Jose, CA : s.n., 2008.Efficient Implementation of Local Adaptive Thresholding Techniques Using Integral Images.. In Document Recognition and Retrieval XV, S&T/SPIE Annual Symposium on Electronic Imaging.

[10] 1stwebdesigner. 1stwebdesigner. [Online] 2012. http://www.1stwebdesigner.com/wp-content/uploads/2009/11/typography-tutorial/text1-how-to-create-typographic-wallpaper.jpg.

[11] dsigninspire. Desing Inspire. [Online] 2012. http://dsigninspire.com/wpcontent/uploads/2011/ 09/moon-shine.jpg.

[12] Geometric Rectification of Camera-Captured Document Images. Jian Liang; DeMenthon, D.; Doermann, D.; April 2008. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.30, no.4, pp.591-605.

[13] Y. WEN, Y. L. 2011.,An Algorithm for License Plate Recognition Applied to Intelligent Transportation System. IEEE Transactions on Intelligent Systems, pp. 1-16.

[14] Lihong Zheng, Xiangjian He, Bijan Samali, Laurence T. Yang. An algorithm for accuracy enhancement of license plate recognition. Journal of Computer and System Sciences, Available online 9 May 2012.

[15] Deselaers, T.; Gass, T.; Heigold, G.; Ney, H.; Latent Log-Linear Models for Handwritten Digit Classification. June 2012. IEEE Transactions on Pattern Analysis and Machine Intelligence, , vol.34, no.6, pp.1105-1117, doi: 10.1109/TPAMI.2011.218.

[16] Jianbin Jiao, Qixiang Ye, Qingming Huang, A configurable method for multi-style license plate recognition. 2009. Pattern Recognition, Volume 42, Issue 3, , Pages 358-369.

[17] H. Erdinc Kocer, K. Kursat Cevik. 2011.Artificial neural networks based vehicle license plate recognition. Procedia Computer Science, Volume 3, Pages 1033-1037.

[18] Apurva A. Desai. 2010. Gujarati handwritten numeral optical character reorganization through neural network, Pattern Recognition, Volume 43, Issue 7 Pages 2582-2589, ISSN 0031-3203, 10.1016/j.patcog.2010.01.008.

[19] Roy, A.; Ghoshal, D.P. 2011. Number Plate Recognition for use in different countries using an improved segmentation, 2nd National Conference on Emerging Trends and Applications in Computer Science (NCETACS),vol., no., pp.1-5, 4-5. doi: .1109/NCETACS.2011.5751407.

[20] Fink, Gernot.2009. Markov models for offline handwriting recognition: a survey. International Journal on Document Analysis and Recognition.Springer Berlin / Heidelberg pp. 269-298,volume: 12,Doi: 10.1007/s10032-009-0098-4.

[21] Chirag Patel, Atul Patel and Dharmendra Patel. Article: Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study. *International Journal of Computer Applications* 55(10):50-56, October 2012. Published by Foundation of Computer Science, New York, USA.

[22] Jie Yang, Jiang Gao, Ying Zhang, Xilin Chen, and Alex Waibel. 2001. An automatic sign recognition and translation system. In *Proceedings of the 2001 workshop on Perceptive user interfaces* (PUI '01). ACM, New York, NY, USA, 1-8. DOI=10.1145/971478.971490 http://doi.acm.org/10.1145/971478.971490

[23] Jonathan.[Online].Available: http://jonathanstreet.com/blog/tag/tesseract/

[24] Sav.(2012).[Online].Available: http://www.savthecoder.com/blog/index.php?page=9

[25] GOOGLE. Google Code. google code. [Online] 2012.https://code.google.com/p/tesseract-ocr/source/browse/wiki/ReadMe.wiki

[26] GOOGLE. Google Code. google code. [Online] 2012.https://code.google.com/p/tesseract-ocr/wiki/ImproveQuality

[27] Blog. http://www.fujixerox.com/eng/company/technology/image_en/

[28] Bansal, V.; Sinha, R. M K, "A complete OCR for printed Hindi text in Devanagari script," *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on* , vol., no., pp.800,804, 2001

[29] Nitin Mishra, C Patvardhan, Vasantha C Lakshmi and Sarika Singh. Article: Shirorekha Chopping Integrated Tesseract OCR Engine for Enhanced Hindi Language Recognition. *International Journal of Computer Applications* 39(6):19-23, February 2012. Published by Foundation of Computer Science, New York, USA.

[30] Nitin Mishra and C Patvardhan. Article: ATMA: Android Travel Mate Application. *International Journal of Computer Applications*50(16):1-8, July 2012. Published by Foundation of Computer Science, New York, USA.

[31] GOOGLE. Google Code. google code. [Online] 2012.https://code.google.com/p/tesseract-ocr/wiki/TrainingTesseract3

[32] Fragoso, V.; Gauglitz, S.; Zamora, S.; Kleban, J.; Turk, M., "TranslatAR: A mobile augmented reality translator," *Applications of Computer Vision (WACV), 2011 IEEE Workshop on* , vol., no., pp.497,502, 5-7 Jan. 2011

[33] TECH CRUCH. http://techcrunch.com/2012/08/11/analysis-web-3-0-the-mobile-era/

[34] IBM. www.haifa.il.ibm.com/projects/image/glt/binar.html

[35] Website .www.webopedia.com/TERM/G/gray_scaling.html

[36] Website. en.wikipedia.org/wiki/OpenCV