Nameplate Text Detector using Android Application

A project Submitted to the Computer Science and Engineering Department, East West University In partial fulfillment of the requirements for the award of the degree of Bachelor of Science in Engineering.

By
Siddhartha Sarkar
ID. 2011-2-60-006

Supervised By
Dr. Shaikh Muhammad Allayear,
Assistant Professor,
Department of Computer Science and Engineering,
East West University

May14, 2015

ABSTRACT

“Nameplate Text Detector using Android Application” based on Android platform. It premises is to make our daily life easy. With this application anyone can track to his/her vehicle from any complain on the other hand a traffic surgeon or traffic police can easily update the complaint. This report provides detailed description on how I created the application and also how I eased the process of android development while creating the application.

Declaration:

I hereby declare that this is submission is my own work and that to the best of my knowledge and belief it contains neither materials nor fact previously published or written by another person. Further, it does not contain material or fact which to a substantial extent has been accepted for the award of any degree of a university or any other institution of tertiary education except where an acknowledgment.

Signature of Candidate

……

(Siddhartha Sarkar)

Letter of Acceptance

The project entitled "Nameplate Text Detector using Android Application" submitted by Siddhartha Sarkar, ID No. 2011-2-60-006, to the Department of Computer Science and Engineering, East West University, Dhaka-1212, Bangladesh is accepted by the Department for the partial fulfillment of requirements for the degree of Bachelor of Science in Computer Science and Engineering on May10, 2015.

Board of Examiners

_____

Dr. Shaikh Muhammad Allayear,
Assistant Professor,
Department of Computer Science and Engineering,
East West University, Dhaka-1212, Bangladesh

_____

Dr. Shamim Hasnat Ripon,
Chairperson and Associate Professor,
Department of Computer Science and Engineering,
East West University Dhaka-1212, Bangladesh

Acknowledgements

The application “Nameplate Text Detector using Android Application” would never run successfully without the valuable encouragement and guidance from my supervisor Dr. Shaikh Muhammad Allayear, Assistant Professor, Department of Computer Science and Engineering, East West University. He enlightened, encouraged and provided us with the ingenuity to transform my vision into reality. I am particularly grateful to Dr. Shamim Hasnat Ripon, Chairperson and Associate Professor, Department of Computer Science and Engineering, East West University, for his encouragement. And also grateful to Md. Salahuddin, Teaching Assistant for his guidance and counseling. I am also grateful to all my teachers and many of my friends who provided me with necessary hardware and suggestion to test the software as it developed. I would like to thank Google for creating android platform and keeping it free.

Table of Contents:

Chapter 1

Introduction

1.1 Introduction

A vehicle registration plate, or license plate, is attached to vehicles for official identification purposes. The identifier, often numerical or alphanumerical, can be used for uniquely identifying a vehicle within the issuing regions database. There are numerous reasons why it is necessary for individuals or organizations to identify a vehicle and thus its owner. Examples include law/police enforcement, traffic control, and access to restricted areas, electronic toll collection or checking parking permissions purposes. In some of the applications like traffic law enforcement, road monitoring and expressway toll system, where license plate recognition is used, it is necessary to process a large number of vehicles in a short time. In daily life there is huge traffic on roads, in this scenario application has to do very fast processing. Otherwise, violators and criminals can escape. The detection of a single license plate and the recognition of its characters in a reliable way is an expensive task, since it relies on computation intensive algorithms. Dedicated systems have been developed for this purpose delivering the necessary computational power. Once a license plate has been recognized it needs to be submitted to an, often remote, IT system to match it against a database finally identifying the vehicle and possibly its driver. Then this information can be processed and used for dedicated purposes [8].

My creativity is to make an android based application that would be an efficient app for smart phone and also an entertaining app for user. So I started working to create an android application for traffic police vehicle holders and that's will make life became too easy.

1.2. Motivations

We have usually lot of vehicles. Sometimes we make some mistake which causes violation of traffic rule. So we need to know about vehicle, but the task is not so easy instantly. We have so many nameplate detector machines but they use for particular purpose. So it is difficult to get information from this source. Where if I have a smart phone and have an app which get image and give me information about these vehicles then it will be more interesting and life will be easy. With concerning all of those problems I have motivated to make this application.

1.3. Objectives

The main objectives of my application are:

- Main Activity Interface
- Take input through Camera
- Updating and Show Vehicles from database
- Option for select language and camera type and OCR engine
- Share image with different software and database
- Search and save information with recognizing text
- showing all information of database
- Select image from gallery and detect text

1.4. Contribution

In survey of existing model I discuss the details about the android. I collect all the requirements to develop an android application there. Software and hardware requirements are also discussed in the chapter to initialize an android development environment. I then move into my project to discuss about it in details. I mention proposed model, its database structure, class diagrams, class descriptions, data flow diagram, requirements, testing results and requirements to run my application. At last I provide a user manual in Chapter User Manual that describes the proper way of using my application.

The way I followed to reach my goal –

I collected the necessary information about Android.

I learned Android programming technique.

I also collected requirements for my project.

I used SQLite database system for the application.

I made all necessary diagrams of my project Data flow diagram.

I tested my application and it passed in all the method I applied.

I created a manual for the general user.

1.5 Organization of the Project Report

Chapter 2:

Describes the basic information and structure of the Android platform. It also describes all the related literature and previous work we used in my application.

Chapter 3:

Describes the implementation procedure like setting up android development environment, installing android development tools, configuring IDEs etc. I have used Eclipse IDE to create our application's internal coding and xml based user interface.

This chapter also describes the entire procedure like how I create the app, useful description of elements that the application contains how to run this application and also testing techniques of application.

Chapter 4:

Describes conclusion and future work of my application

Chapter 2

Literature Review and Survey of Existing Models

2.1. What is Android?

Android is one of the most popular as well as most widely used mobile operating system. It is a software stack for mobile devices that includes an operating system, middleware and key applications [9]. The android SDK provides the tools and APIs necessary to create application on the android platform using JAVA programming language. Android is an open source mobile OS platform that's running on top of Linux kernel. It uses a non-standard Java Virtual Machine called Dalvin, specialized to handle mobile device processes. The main programming language used for Android is JAVA with many libraries from J2SE and third party open source projects, such as Apache commons, SQLite, web kit, etc. Google purchased the initial developer of the software, Android Inc., in 2005. The unveiling of the Android distribution on November 5, 2007 was announced with the founding of the Open Handset Alliance, a consortium of 84 hardware, software and telecommunication companies devoted to advancing open standards for mobile devices. Google released most of the Android code under the Apache License, a free software license. The Android Open Source Project (AOSP) is tasked with the maintenance and further development of Android. Android consists of a kernel based on the Linux kernel, with middleware, libraries and APIs written in C and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Developers write primarily in a customized version of Java. Android became the world's leading smart phone platform at the end of 2010. For the first quarter of 2012, Android had a 59% smart phone market share worldwide. At the half of 2012, there were 400 million devices activated and 1 million activations per day. Analysts point to the advantage to Android of being a multi-channel, multi-carrier OS. [10] At the beginning of 2013, Android captured 70% of smart phone market share worldwide.

2.2. History of Android

In this portion I will describe the history of android and I also show relation between my project and android.

2.2.1. Foundation

Android, Inc. was founded in Palo Alto, California, United States in October 2003 by Andy Rubin (co-founder of Danger), Rich Miner (co-founder of Wildfire Communications, Inc.), Nick Sears (once VP at T-Mobile), and Chris White (headed design and interface development at

WebTV) to develop, in Rubin's words "...smarter mobile devices that are more aware of its owner's location and preferences". Despite the obvious past accomplishments of the founders and early employees, Android Inc. operated secretly, revealing only that it was working on software for mobile phones. That same year, Rubin ran out of money. Steve Perlman, a close friend of Rubin, brought him $10,000 in cash in an envelope and refused a stake in the company.

2.2.2. Acquisition by Google

Google acquired Android Inc. on August 17, 2005, making Android Inc. a wholly owned subsidiary of Google. Key employees of Android Inc., including Andy Rubin, Rich Miner and Chris White, stayed at the company after the acquisition. Not much was known about Android Inc. at the time of the acquisition, but many assumed that Google was planning to enter the mobile phone market with this move.

2.2.3. Post-acquisition by Google

At Google, the team led by Rubin developed a mobile device platform powered by the Linux kernel. Google marketed the platform to handset makers and carriers on the promise of providing a flexible, upgradable system. Google had lined up a series of hardware component and software partners and signaled to carriers that it was open to various degrees of cooperation on their part. Speculation about Google's intention to enter the mobile communications market continued to build through December 2006. Reports from the BBC and The Wall Street Journal noted that Google wanted its search and applications on mobile phones and it was working hard to deliver that. Print and online media outlets soon reported rumors that Google was developing a Google-branded handset. Some speculated that as Google was defining technical specifications, it was showing prototypes to cell phone manufacturers and network operators.

In September 2007, InformationWeek covered an Evacuee serve study reporting that Google had filed several patent applications in the area of mobile telephony.

2.2.4. Open Handset Alliance

On November 5, 2007, the Open Handset Alliance, a consortium of several companies which include Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, NVidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile and Texas Instruments unveiled itself. The goal of the Open Handset Alliance is to develop open standards for mobile devices. On the same day, the Open Handset Alliance also unveiled their first product, Android, a mobile device platform built on the Linux kernel version 2.6. On December 9, 2008, 14 new

members joined, including ARM Holdings, Atheros Communications, Asustek Computer Inc., Garmin Ltd, Huawei Technologies, PacketVideo, Softbank, Sony Ericsson, Toshiba Corp, and Vodafone Group Plc. [10]

2.2.5. Android Open Source Project

The Android Open Source Project (AOSP) is led by Google, and is tasked with the maintenance and development of Android. According to the project "The goal of the Android Open Source Project is to create a successful real-world product that improves the mobile experience for end users." AOSP also maintains the Android Compatibility Program, defining an "Android compatible" device "as one that can run any application written by third-party developers using the Android SDK and NDK", to prevent incompatible Android implementations. The compatibility program is also optional and free of charge, with the Compatibility Test Suite also free and open-source.

2.2.6. Version history

Android has been updated frequently since the original release of "Astro", with each fixing bugs and adding new features. Each version is named in alphabetical order, with 1.5 "Cupcake" being the first named after a dessert and every update since following this naming convention. [10]

List of Android version names:

1. Cupcake
2. Donut
3. Eclair
4. Froyo
5. Gingerbread
6. Honeycomb
7. Ice Cream Sandwich
8. Android 4.2 Jelly Bean (API level 17)
9. Android 4.3 Jelly Bean (API level 18)

10. Android 4.4 Kit Kat (API level 19)

11. Android 5 Lollipop (API level 21)

2.3 Gingerbread refined the user interface, improved the soft keyboard and copy/paste features, improved gaming performance, SIP support (VoIP calls), and added support for Near Field Communication.

3.0 Honeycomb was a tablet-oriented release which supports larger screen devices and introduces many new user interface features, and supports multi core processors and hardware acceleration for graphics. The Honeycomb SDK has been released and the first device featuring this version, the Motorola Xoom tablet, went on sale in February 2011.

3.1 Honeycomb was announced at the 2011 Google I/O on 10 May 2011. One feature focuses on allowing Honeycomb devices to directly transfer content from USB devices.

3.2 Honeycomb released at July 15 2011, is "an incremental release that adds several new capabilities for users and developers". Highlights include optimization for a broader range of screen sizes; new "zoom-to-fill" screen compatibility mode; capability to load media files directly from the SD card; and an extended screen support API, providing developers with more precise control over the UI. Android 3.2 Honeycomb is the latest Android version that is available to tablets.

4.0.x Ice Cream Sandwich released at December 16, 2011, it's easy multitasking, rich notifications, customizable home screens, resizable widgets, and deep interactivity and adds powerful new ways of communicating and sharing.

4.1.x Jelly Bean released at July 9, 2012 Based on Linux kernel 3.0.31, Jelly Bean was an incremental update with the primary aim of improving the functionality and performance of the user interface. The performance improvement involved "Project Butter", which uses touch anticipation, triple buffering, and extended vsync timing and a fixed frame rate of 60 fps to create a fluid and "buttery-smooth" UI. Android 4.1 Jelly Bean was released to the Android Open Source Project on 9 July 2012, and the Nexus 7 tablet, the first device to run Jelly Bean.

4.2. x Jelly Bean released at November 13, 2012 its API level is 17.

4.3. x Jelly Bean released at July 24, 2013API level is 18.

4.4 Kit Kat released at October 31, 2013 API level is 19.

2.3. Design and Architecture of Android

Android consists of a kernel based on the Linux kernel, with middleware, libraries and APIs written in C and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Android uses the Dalvik virtual machine with just-in-time compilation to run Dalvikdex-code (Dalvik Executable), which is usually translated from Java byte code.

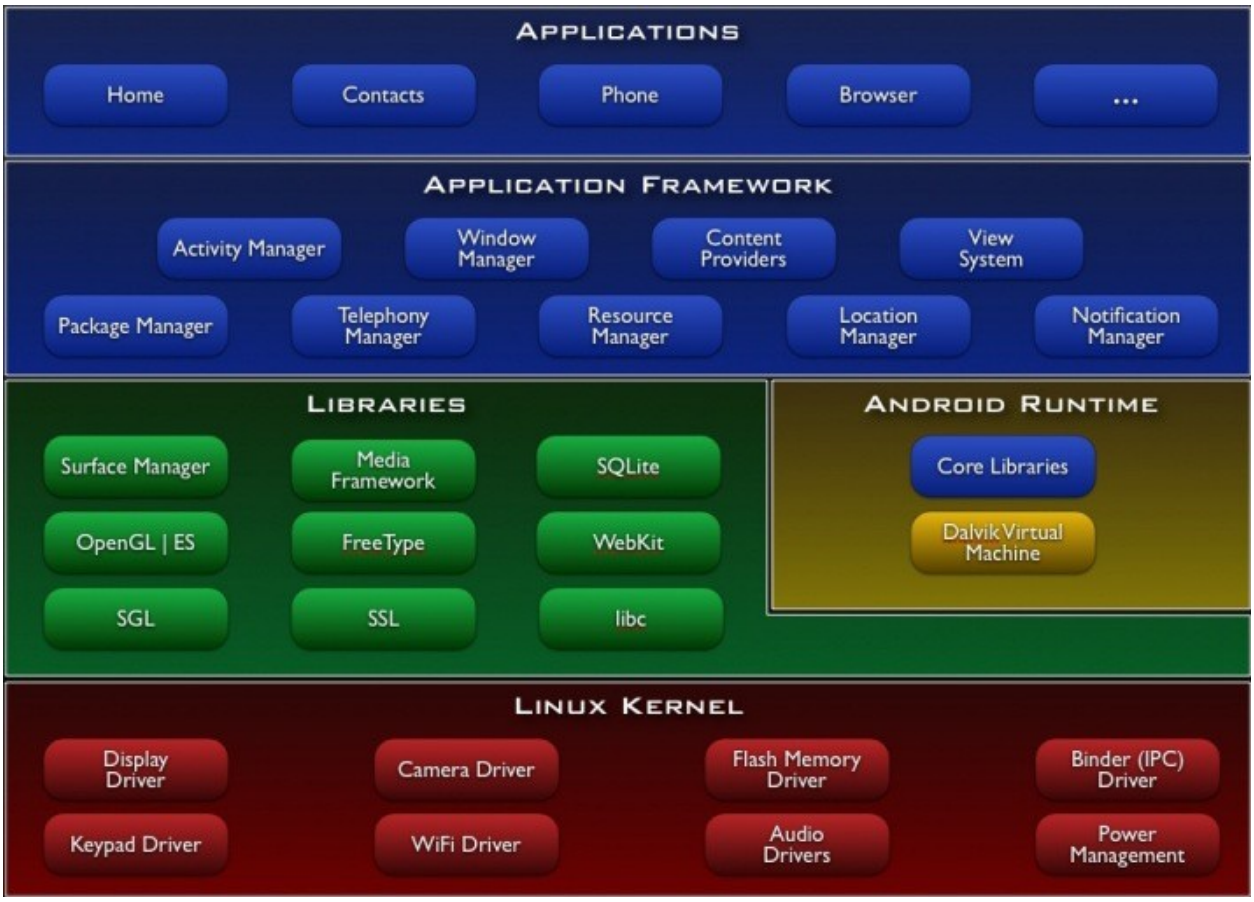
The main hardware platform for Android is the ARM architecture. There is support for x86 from the Android x 86 projects and Google TV uses a special x86 version of Android.

2.4. Linux

Android's kernel is based on the Linux kernel and has further architecture changes by Google outside the typical Linux kernel development cycle. Android does not have a native X Window System nor does it support the full set of standard GNU libraries, and this makes it difficult to port existing Linux applications or libraries to Android. Certain features that Google contributed back to the Linux kernel, notably a power management feature called wake locks, were rejected by mainline kernel developers, partly because kernel maintainers felt that Google did not show any intent to maintain their own code. Even though Google announced in April 2010 that they would hire two employees to work with the Linux kernel community, Greg Kroah-Hartman, the current Linux kernel maintainer for the -stable branch, said in December 2010 that he was concerned that Google was no longer trying to get their code changes included in mainstream Linux. Some Google Android developers hinted that "the Android team was getting fed up with the process", because they were a small team and had more urgent work to do on Android. However, in September 2010, Linux kernel developer Rafael J. Wysocki added a patch that improved the mainline Linux wakeup events framework. He said that Android device drivers that use wake locks can now be easily merged into mainline Linux, but that Android's opportunistic Suspend features should not be included in the mainline kernel. In August 2011, Linus Torvalds said that "eventually Android and Linux would come back to a common kernel, but it will probably not be for four to five years". In December 2011, Greg Kroah-Hartman announced the start of the Android Mainlining Project, which aims to put some Android drivers, patches and features back into the Linux kernel, starting in Linux 3.3.further integration being expected for Linux Kernel 3.4. [12]

2.5 Architecture of Android

The following diagram shows the components of the Android operating system:



2.6 Application

Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language.

Application Framework:

By providing an open development platform, Android offers developers the ability to build extremely rich and innovative applications. Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, and much, much more.

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

Underlying all applications is a set of services and systems, including:

1. A rich and extensible set of Views that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser.

2. Content Providers that enable applications to access data from other applications (such as Contacts), or to share their own data.

3. A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files.

4. A Notification Manager that enables all applications to display custom alerts in the status bar.

5. An Activity Manager that manages the lifecycle of applications and provides a common navigation back stack.

2.7. Libraries

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:

System C library - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices.

Media Libraries - based on Packet Video's Open CORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG.

Surface Manager - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications.

Lib Web Core - a modern web browser engine which powers both the Android browser and an embeddable web view.

SGL - the underlying 2D graphics engine.

3D libraries - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software pasteurizer.

FreeType - bitmap and vector font rendering.

SQLite - a powerful and lightweight relational database engine available to all applications.

2.8. Android Runtime

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently.

The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

2.9. Linux Kernel

Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack. [12]

2.10 Features of Android

Features and current specifications of android are as follow:

- **Handset layouts**

The platform is adaptable to larger, VGA, 2D graphics library, 3D graphics library based on OpenGL ES 2.0 specifications, and traditional Smartphone layouts.

- **Storage**

SQLite, a lightweight relational database, is used for data storage purposes.

- **Connectivity**

Android supports connectivity technologies including GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.

- **Messaging**

SMS and MMS are available forms of messaging, including threaded text messaging and now Android Cloud to Device Messaging Framework (C2DM) is also a part of Android Push Messaging service.

- **Multiple language support**

Android supports multiple human languages. The number of languages more than doubled for the platform 2.3 Gingerbread, 4.0.x Ice Cream Sandwich, 4.1.x Jelly Bean and 4.2.x Jelly Bean

- **Web browser**

The web browser available in Android is based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine. The browser scores a 95/100 on the Acid3 Test.

- **Java support**

While most Android applications are written in Java, there is no Java Virtual Machine in the platform and Java byte code is not executed. Java classes are compiled into Dalvik executables and run on Dalvik, a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU. J2ME support can be provided via third-party applications.

- **Media support**

Android supports the following audio/video/still media formats: WebM, H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, OggVorbis, FLAC, WAV, JPEG, PNG, GIF, BMP.

- **Streaming media support**

RTP/RTSP streaming (3GPP PSS, ISMA), HTML progressive download (HTML5 <video> tag). Adobe Flash Streaming (RTMP) and HTTP Dynamic streaming are supported by the Flash plug-in. Apple HTTP Live Streaming is supported by RealPlayer for Mobile, and by the operating system in Android 3.0 (Honeycomb).

- **Additional hardware support**

Android can use video/still cameras, touch screens, GPS, accelerometers, gyroscopes, magnetometers, dedicated gaming controls, proximity and pressure sensors, thermometers, accelerated 2D bit blitz (with hardware orientation, scaling, pixel format conversion) and accelerated 3D graphics.

- **Multi-touch**

Android has native support for multi-touch which was initially made available in handsets such as the HTC One X. The feature was originally disabled at the kernel level (possibly to avoid infringing Apple's patents on touch-screen technology at the time). Google has since released an update for the Nexus One and the Motorola Droid which enables multi-touch natively.

- **Bluetooth**

Supports A2DP, AVRCP, sending files (OPP), accessing the phone book (PBAP), voice dialing and sending contacts between phones. Keyboard, mouse and joystick (HID) support is available in Android 3.1+, and in earlier versions through manufacturer customizations and third-party applications.

- **Video calling**

Android does not support native video calling, but some handsets have a customized version of the operating system that supports it, either via the UMTS network (like the Samsung Galaxy S) or over IP. Video calling through Google Talk is available in Android 2.3.4 and later. Gingerbread allows Nexus S to place Internet calls with a SIP account. This allows for enhanced VoIP dialing to other SIP accounts and even phone numbers. Skype 2.1 offers video calling in Android 2.3, 4.0.x, 4.1.x, 4.2.x, including front camera support.

- **Multitasking**

Multitasking of applications is available.

- **Voice based features**

Google search through voice has been available since initial release. Voice actions for calling, texting, navigation, etc. are supported on Android 2.2 onwards.

- **Tethering**

Android supports tethering, which allows a phone to be used as a wireless/wired Wi-Fi hotspot. Before Android 2.2 this was supported by third-party applications or manufacturer customizations.

- **Screen capture**

Android does not support screenshot capture as of 2011. This is supported by manufacturer and third-party customizations. Screen Capture is available in 4.0.x+ and through a PC connection using the DDMS developer's tool. [10]

2.11 Applications

Applications are usually developed in the Java language using the Android Software Development Kit, but other development tools are available, including a Native Development Kit for applications or extensions in C or C++, Google App Inventor, a visual environment for novice programmers and various cross platform mobile web applications frameworks. Applications can be acquired by end-users either through a store such as Google Play or the Amazon App store, or by downloading and installing the application's APK files from a third-party site.

2.12. Google Play

Google Play is an online software store developed by Google for Android devices. An application program ("app") called "Play Store" is preinstalled on most Android devices and allows users to browse and download apps published by third-party developers, hosted on Google Play. As of February 2013, there were more than 800,000 apps available for Android, and the estimated number of applications downloaded from the Play Store exceeded 20 billion. The operating system itself is installed on 500 million total devices.

Only devices that comply with Google's compatibility requirements are allowed to preinstall and access the Play Store. The app filters the list of available applications to those that are compatible with the user's device, and developers may restrict their applications to particular carriers or countries for business reasons.

Google offers many free applications in the Play Store including Google Voice, Google Goggles, Gesture Search, Google Translate, Google Shopper, Listen and My Tracks. In August 2010, Google launched "Voice Actions for Android", which allows users to search, write messages, and initiate calls by voice.

2.13 Security of Application

Android applications run in a sandbox, an isolated area of the operating system that does not have access to the rest of the system's resources, unless access permissions are granted by the user when the application is installed. Before installing an application, the Play Store displays all required permissions. A game may need to enable vibration, for example, but should not need to read messages or access the phonebook. After reviewing these permissions, the user can decide whether to install the application. The sandboxing and permissions system weakens the impact of vulnerabilities and bugs in applications, but developer confusion and limited documentation has resulted in applications routinely requesting unnecessary permissions, reducing its effectiveness.

The complexity of inter-application communication implies Android may have opportunities to run unauthorized code. Several security firms have released antivirus software for Android devices, in particular, Lookout Mobile Security, AVG Technologies, Avast!, F-Secure, Kaspersky, McAfee and Symantec. This software is ineffective as sandboxing also applies to such applications, limiting their ability to scan the deeper system for threats. A useful type of security applications program and service, often described as "Find My Phone", is available for Android, as well as for Microsoft Windows Phone and for Apple iPhone, whereby a registered user can find the approximate location of the phone, if switched on, over the Internet. This helps to locate lost or stolen phones. At least one of these can be installed on a phone after it has gone missing.

2.14 Privacy

Android smart phones have the ability to report the location of Wi-Fi access points, encountered as phone users move around, to build databases containing the physical locations of hundreds of millions of such access points. These databases form electronic maps to locate smart phones, allowing them to run apps like Foursquare, Latitude, Places, and to deliver location-based ads. Third party monitoring software such as TaintDroid, an academic research-funded project, can, in some cases, detect when personal information is being sent from applications to remote servers. In March 2012 it was revealed that Android Apps can copy photos without explicit user permission, Google responded they "originally designed the Android photos file system similar to those of other computing platforms like Windows and Mac OS. They're taking another look at this and considering adding permission for apps to access images. They've always had policies in place to remove any apps [on Google Play] that improperly access your data." [11]

2.15. Software Development Tools

In this portion I will describe about android app development tools and the way to development procedure.

2.15.1 Android SDK

The Android software development kit (SDK) includes a comprehensive set of development tools. These include a debugger, libraries, a handset emulator based on QEMU, documentation, sample code, and tutorials. Currently supported development platforms include computers running Linux (any modern desktop Linux distribution), Mac OS X 10.5.8 or later, Windows XP or later. The officially supported integrated development environment (IDE) is Eclipse using the Android Development Tools (ADT) Plug-in, though developers may use any text editor to edit

Java and XML files then use command line tools (Java Development Kit and Apache Ant are required) to create, build and debug Android applications as well as control attached Android devices (e.g., triggering a reboot, installing software package(s) remotely). Enhancements to Android's SDK go hand in hand with the overall Android platform development. The SDK also supports older versions of the Android platform in case developers wish to target their applications at older devices. Development tools are downloadable components, so after one has downloaded the latest version and platform, older platforms and tools can also be downloaded for compatibility testing.

2.15.2 Native Development Kit

Libraries written in C and other languages can be compiled to ARM or x86 native code and installed using the Android Native Development Kit. Native classes can be called from Java code running under the Dalvik VM using the System. LoadLibrary call, which is part of the standard Android Java classes. Complete applications can be compiled and installed using traditional development tools. The ADB debugger gives a root shell under the Android Emulator which allows native ARM code or x 86 codes to be uploaded and executed. ARM or x 86 codes can be compiled using GCC on a standard PC. Running native code is complicated by the fact that Android uses a non-standard C library (libc, known as Bionic). The underlying graphics device is available as a frame buffer at /dev/graphics/fb0. The graphics library that Android uses to arbitrate and control access to this device is called the Skia Graphics Library (SGL), and it has been released under an open source license. Skia has backend for both win32 and UNIX, allowing the development of cross-platform applications, and it is the graphics engine underlying the Google Chrome web browser. Unlike Java App development based on the Eclipse IDE, the NDK is based on command-line tools and requires invoking them manually to build, deploy and debug the apps. Several third-party tools allow integrating the NDK into Eclipse and Visual Studio.

2.15.3 Android Open Accessory Development Kit

The Android 3.1 platform (also back ported to Android 2.3.4) introduces Android Open Accessory support, which allows external USB hardware (an Android USB accessory) to interact with an Android-powered device in a special "accessory" mode.

When an Android-powered device is in accessory mode, the connected accessory acts as the USB host (powers the bus and enumerates devices) and the Android-powered device acts as the USB device. Android USB accessories.

2.15.4 App Inventor for Android

On 12 July 2010, Google announced the availability of App Inventor for Android, a Web-based visual development environment for novice programmers, based on MIT's Open Blocks Java library and providing access to Android devices' GPS, accelerometer and orientation data, phone functions, text messaging, speech-to-text conversion, contact data, persistent storage, and Web services, initially including Amazon and Twitter. "We could only have done this because Android's architecture is so open," said the project director, MIT's Hal Abelson. Under development for over a year, the block-editing tool has been taught to non-majors in computer science at Harvard, MIT, Wellesley, Trinity College (Hartford,) and the University of San Francisco, where Professor David Wolber developed an introductory computer science course and tutorial book for non-computer science students based on App Inventor for Android.

2.15.5 Hyper Next Android Creator

Hyper Next Android Creator (HAC) is a software development system aimed at beginner programmers that can help them create their own Android apps without knowing Java and the Android SDK. It is based on HyperCard that treated software as a stack of cards with only one card being visible at any one time and so is well suited to mobile phone applications that have only one window visible at a time. Hyper Next Android Creator's main programming language is simply called Hyper Next and is loosely based on HyperCard's Hyper Talk language. Hypertext is an interpreted English-like language and has many features that allow creation of Android applications. It supports a growing subset of the Android SDK including its own versions of the GUI control types and automatically runs its own background service so apps can continue to run and process information while in the background.

2.16 The Simple Project

The goal of Simple is to bring an easy-to-learn-and-use language to the Android platform. Simple is a BASIC dialect for developing Android applications. It targets professional and non-professional programmers alike in that it allows programmers to quickly write Android applications that use the Android runtime components.

Similar to Microsoft Visual Basic 6, Simple programs are form definitions (which contain components) and code (which contains the program logic). The interaction between the components and the program logic happens through events triggered by the components. The program logic consists of event handlers which contain code reacting to the events. The Simple project is not very active, the last source code update being in August 2009. [13]

2.17. App Components

Like other application android application has its own components, below I will describe these components.

2.18. Application Fundamentals

Android applications are written in the Java programming language. The Android SDK tools compile the code-along with any data and resource files-into an Android package, an archive file with an .apk suffix.

All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application. Once installed on a device, each Android application lives in its own security sandbox:

The Android operating system is a multi-user Linux system in which each application is a different user.

By default, the system assigns each application a unique Linux user ID (the ID is used only by the system and is unknown to the application). The system sets permissions for all the files in an application so that only the user ID assigned to that application can access them.

Each process has its own virtual machine (VM), so an application's code runs in isolation from other applications.

By default, every application runs in its own Linux process. Android starts the process when any of the application's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other applications.

In this way, the Android system implements the principle of least privilege. That is, each application, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an application cannot access parts of the system for which it is not given permission. However, there are ways for an application to share data with other applications and for an application to access system services:

It's possible to arrange for two applications to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, applications with the same user ID can also arrange to run in the same Linux process and share the same VM (the applications must also be signed with the same certificate). An application can request permission to access device data such as the user's contacts, SMS messages, the mountable

storage (SD card), camera, Bluetooth, and more. All application permissions must be granted by the user at install time.

That covers the basics regarding how an Android application exists within the system. The rest of this document introduces you to:

The core framework components that define your application.

The manifest file in which you declare components and required device features for your application.

Resources that are separate from the application code and allow your application to gracefully optimize its behavior for a variety of device configurations. [11]

2.19. Application Components

Application components are the essential building blocks of an Android application. Each component is a different point through which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role-each one is a unique building block that helps define your application's overall behavior.

There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

Here are the four types of application components:

- **Activities**

An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As such, a different application can start any one of these activities (if the email application allows it). For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture. An activity is implemented as a subclass of Activity and you can learn more about it in the Activities developer guide.

- **Services**

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. A service is implemented as a subclass of Service and you can learn more about it in the Services developer guide.

- **Content providers**

A content provider manages a shared set of application data. You can store the data in the file system, a SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information.

As such, any application with the proper permissions can query part of the content provider (such as ContactsContract.Data) to read and write information about a particular person. Content providers are also useful for reading and writing data that is private to your application and not shared. For example, the Note Pad sample application uses a content provider to save notes. A content provider is implemented as a subclass of Content Provider and must implement a standard set of APIs that enable other applications to perform transactions. For more information, see the Content Providers developer guide.

- **Broadcast receivers**

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system-for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts-for example, to let other applications know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event. A broadcast receiver is implemented as a subclass of

Broadcast Receiver and each broadcast is delivered as an Intent object. For more information, see the Broadcast Receiver class.

A unique aspect of the Android system design is that any application can start another application's component. For example, if you want the user to capture a photo with the device camera, there's probably another application that does that and your application can use it, instead of developing an activity to capture a photo yourself.

You don't need to incorporate or even link to the code from the camera application. Instead, you can simply start the activity in the camera application that captures a photo. When complete, the photo is even returned to your application so you can use it. To the user, it seems as if the camera is actually a part of your application. When the system starts a component, it starts the process for that application (if it's not already running) and instantiates the classes needed for the component. For example, if your application starts the activity in the camera application that captures a photo, that activity runs in the process that belongs to the camera application, not in your application's process.

Therefore, unlike applications on most other systems, Android applications don't have a single entry point (there's no main() function, for example).

Because the system runs each application in a separate process with file permissions that restrict access to other applications, your application cannot directly activate a component from another application. The Android system, however, can. So, to activate a component in another application, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

2.20. Activating Components

Three of the four component types-activities, services, and broadcast receivers-are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your application or another.

Intent is created with an Intent object, which defines a message to activate either a specific component or a specific type of component-an intent can be either explicit or implicit, respectively.

For activities and services, intent defines the action to perform (for example, to "view" or "send" something) and may specify the URI of the data to act on (among other things that the component being started might need to know). For example, intent might convey a request for an activity to show an image or to open a web page. In some cases, you can start an activity to receive a result, in which case, the activity also returns the result in an Intent (for example, you

can issue an intent to let the user pick a personal contact and have it returned to you-the return intent includes a URI pointing to the chosen contact).

For broadcast receivers, the intent simply defines the announcement being broadcast (for example, a broadcast to indicate the device battery is low includes only a known action string that indicates "battery is low").

The other component type, content provider, is not activated by intents. Rather, it is activated when targeted by a request from a Content Resolver. The content resolver handles all direct transactions with the content provider so that the component that's performing transactions with the provider doesn't need to and instead calls methods on the Content Resolver object. This leaves a layer of abstraction between the content provider and the component requesting information (for security).

There are separate methods for activating each type of component:

You can start an activity (or give it something new to do) by passing an Intent to **startActivity()**or **startActivityForResult()** (when you want the activity to return a result).

You can start a service (or give new instructions to an ongoing service) by passing an Intent to **startService()**. Or you can bind to the service by passing an Intent to **bindService()**.

You can initiate a broadcast by passing an Intent to methods like **sendBroadcast()**, **sendOrderedBroadcast()**, or **sendStickyBroadcast()**.

You can perform a query to a content provider by calling **query()** on a Content Resolver.

For more information about using intents, see the Intents and Intent Filters document. More information about activating specific components is also provided in the following documents: Activities, Services, Broadcast Receiver and Content Providers.

2.21. The Manifest File

Before the Android system can start an application component, the system must know that the component exists by reading the application's AndroidManifest.xml file (the "manifest" file). Your application must declare all its components in this file, which must be at the root of the application project directory.

The manifest does a number of things in addition to declaring the application's components, such as:

Identify any user permissions the application requires, such as Internet access or read-access to the user's contacts.

Declare the minimum API Level required by the application, based on which APIs the application uses.

Declare hardware and software features used or required by the application, such as a camera, Bluetooth services, or a multi touch screen.

API libraries the application needs to be linked against (other than the Android framework APIs), such as the Google Maps library.

2.22. Declaring components

The primary task of the manifest is to inform the system about the application's components. For example, a manifest file can declare an activity as follows:

<?xml version="1.0" encoding="utf-8"?>

<manifest ... >

<applicationandroid:icon="@drawable/app_icon.png" ... >

<activityandroid:name="com.example.project.ExampleActivity"

android:label="@string/example_label" ... >

</activity>

...

</application>

</manifest>

In the <application> element, the android: icon attribute points to resources for an icon that identifies the application.

In the <activity> element, the android: name attribute specifies the fully qualified class name of the Activity subclass and the android: label attributes specifies a string to use as the user-visible label for the activity.

You must declare all application components this way:

<activity> elements for activities

<service> elements for services

<receiver> elements for broadcast receivers

<provider> elements for content providers

Activities, services, and content providers that you include in your source but do not declare in the manifest are not visible to the system and, consequently, can never run. However, broadcast receivers can be either declared in the manifest or created dynamically in code (as BroadcastReceiver objects) and registered with the system by calling **registerReceiver()**.

2.23. Declaring components capabilities

As discussed above, in Activating Components, you can use an Intent to start activities, services, and broadcast receivers. You can do so by explicitly naming the target component (using the component class name) in the intent. However, the real power of intents lies in the concept of intent actions. With intent actions, you simply describe the type of action you want to perform (and optionally, the data upon which you'd like to perform the action) and allow the system to find a component on the device that can perform the action and start it. If there are multiple components that can perform the action described by the intent, then the user selects which one to use. The way the system identifies the components that can respond to an intent is by comparing the intent received to the intent filters provided in the manifest file of other applications on the device.

When you declare a component in your application's manifest, you can optionally include intent filters that declare the capabilities of the component so it can respond to intents from other applications. You can declare an intent filter for your component by adding an <intent-filter> element as a child of the component's declaration element.

For example, an email application with an activity for composing a new email might declare an intent filter in its manifest entry to respond to "send" intents (in order to send email). An activity in your application can then create an intent with the ―send‖ action (ACTION_SEND), which the system matches to the email application's ―send‖ activity and launches it when you invoke the intent with **startActivity()**.

2.24 Declaring application requirements

There are a variety of devices powered by Android and not all of them provide the same features and capabilities. In order to prevent your application from being installed on devices that lack

features needed by your application, it's important that you clearly define a profile for the types of devices your application supports by declaring device and software requirements in your manifest file. Most of these declarations are informational only and the system does not read them, but external services such as Google Play do read them in order to provide filtering for users when they search for applications from their device.

For example, if your application requires a camera and uses APIs introduced in Android 2.1 (API Level 7), you should declare these as requirements in your manifest file. That way, devices that do not have a camera and have an Android version lower than 2.1 cannot install your application from Google Play.

However, you can also declare that your application uses the camera, but does not require it. In that case, your application must perform a check at runtime to determine if the device has a camera and disable any features that use the camera if one is not available.

Here are some of the important device characteristics that you should consider as you design and develop your application:

- **Screen size and density**

In order to categorize devices by their screen type, Android defines two characteristics for each device: screen size (the physical dimensions of the screen) and screen density (the physical density of the pixels on the screen, or dpi—dots per inch). To simplify all the different types of screen configurations, the Android system generalizes them into select groups that make them easier to target. The screen sizes are: small, normal, large, and extra-large. The screen densities are: low density, medium density, high density, and extra high density. By default, your application is compatible with all screen sizes and densities, because the Android system makes the appropriate adjustments to your UI layout and image resources. However, you should create specialized layouts for certain screen sizes and provide specialized images for certain densities, using alternative layout resources, and by declaring in your manifest exactly which screen sizes your application supports with the <supports-screens> element. For more information, see the Supporting Multiple Screens document.

- **Input configurations**

Many devices provide a different type of user input mechanism, such as a hardware keyboard, a trackball, or a five-way navigation pad. If your application requires a particular kind of input hardware, then you should declare it in your manifest with the <uses-configuration> element. However, it is rare that an application should require a certain input configuration.

- **Device features**

There are many hardware and software features that may or may not exist on a given Android-powered device, such as a camera, a light sensor, Bluetooth, a certain version of OpenGL, or the fidelity of the touch screen. You should never assume that a certain feature is available on all Android-powered devices (other than the availability of the standard Android library), so you should declare any features used by your application with the <uses-feature> element.

- **Platform Version**

Different Android-powered devices often run different versions of the Android platform, such as Android 1.6 or Android 2.3. Each successive version often includes additional APIs not available in the previous version. In order to indicate which set of APIs are available, each platform version specifies an API Level (for example, Android 1.0 is API Level 1 and Android 2.3 is API Level 9). If you use any APIs that were added to the platform after version 1.0, you should declare the minimum API Level in which those APIs were introduced using the <uses-sdk> element. It's important that you declare all such requirements for your application, because, when you distribute your application on Google Play, the store uses these declarations to filter which applications are available on each device. As such, your application should be available only to devices that meet all your application requirements.

2.25. Application Resources

An Android application is composed of more than just code—it requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the application. For example, you should define animations, menus, styles, colors, and the layout of activity user interfaces with XML files. Using application resources makes it easy to update various characteristics of your application without modifying code and—by providing sets of alternative resources—enables you to optimize your application for a variety of device configurations (such as different languages and screen sizes). For every resource that you include in your Android project, the SDK build tools define a unique integer ID, which you can use to reference the resource from your application code or from other resources defined in XML. For example, if your application contains an image file named logo.png (saved in the res/drawable/ directory), the SDK tools generate a resource ID named R.drawable.logo, which you can use to reference the image and insert it in your user interface. [10]

One of the most important aspects of providing resources separate from your source code is the ability for you to provide alternative resources for different device configurations. For example, by defining UI strings in XML, you can translate the strings into other languages and save those

strings in separate files. Then, based on a language qualifier that you append to the resource directory's name (such as res/values-fr/ for French string values) and the user's language setting, the Android system applies the appropriate language strings to your UI.

Android supports many different qualifiers for your alternative resources. The qualifier is a short string that you include in the name of your resource directories in order to define the device configuration for which those resources should be used. As another example, you should often create different layouts for your activities, depending on the device's screen orientation and size. For example, when the device screen is in portrait orientation (tall), you might want a layout with buttons to be vertical, but when the screen is in landscape orientation (wide), the buttons should be aligned horizontally. To change the layout depending on the orientation, you can define two different layouts and apply the appropriate qualifier to each layout's directory name. Then, the system automatically applies the appropriate layout depending on the current device orientation. [10]

2.26 Tesseract

Tesseract is an open-source OCR engine that was developed at HP between 1984 and 1994. Like a supernova, it appeared from nowhere for the 1995 UNLV Annual Test of OCR Accuracy [1], shone brightly with its results, and then vanished back under the same cloak of secrecy under which it had been developed. Now for the first time, details of the architecture and algorithms can be revealed. Tesseract began as a PhD research project [2] in HP Labs, Bristol, and gained momentum as a possible software and/or hardware add-on for HP's line of flatbed scanners. Motivation was provided by the fact that the commercial OCR engines of the day were in their infancy, and failed miserably on anything but the best quality print. After a joint project between HP Labs Bristol, and HP's scanner division in Colorado, Tesseract had a significant lead in accuracy over the commercial engines, but did not become a product. The next stage of its development was back in HP Labs Bristol as an investigation of OCR for compression. Work concentrated more on improving rejection efficiency than on base-level accuracy. At the end of this project, at the end of 1994, development ceased entirely. The engine was sent to UNLV for the 1995 Annual Test of OCR Accuracy [3], where it proved its worth against the comer engines of the time. In late 2005, HP released Tesseract for open source. It is now available at http://code.google.com/p/tesseract-ocr. Tesseract is a free software optical character recognition engine for various operating systems. Tesseract is considered one of the most accurate free software OCR engines currently available. [4] Tesseract has been selected because it more adaptable than Ocrad engine, as seen in the paper [5] and in the application in [6]. The version used is 3.02.

2.27 Optical Character Recognition

Optical character recognition, usually abbreviated to OCR, is the mechanical or electronic conversion of scanned images of handwritten, typewritten or printed text into machine-encoded text. It is widely used as a form of data entry from some sort of original paper data source, whether documents, sales receipts, mail, or any number of printed records. It is a common method of digitizing printed texts so that they can be electronically searched, stored more compactly, displayed on-line, and used in machine processes such as machine translation, text-to-speech and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision. [7]

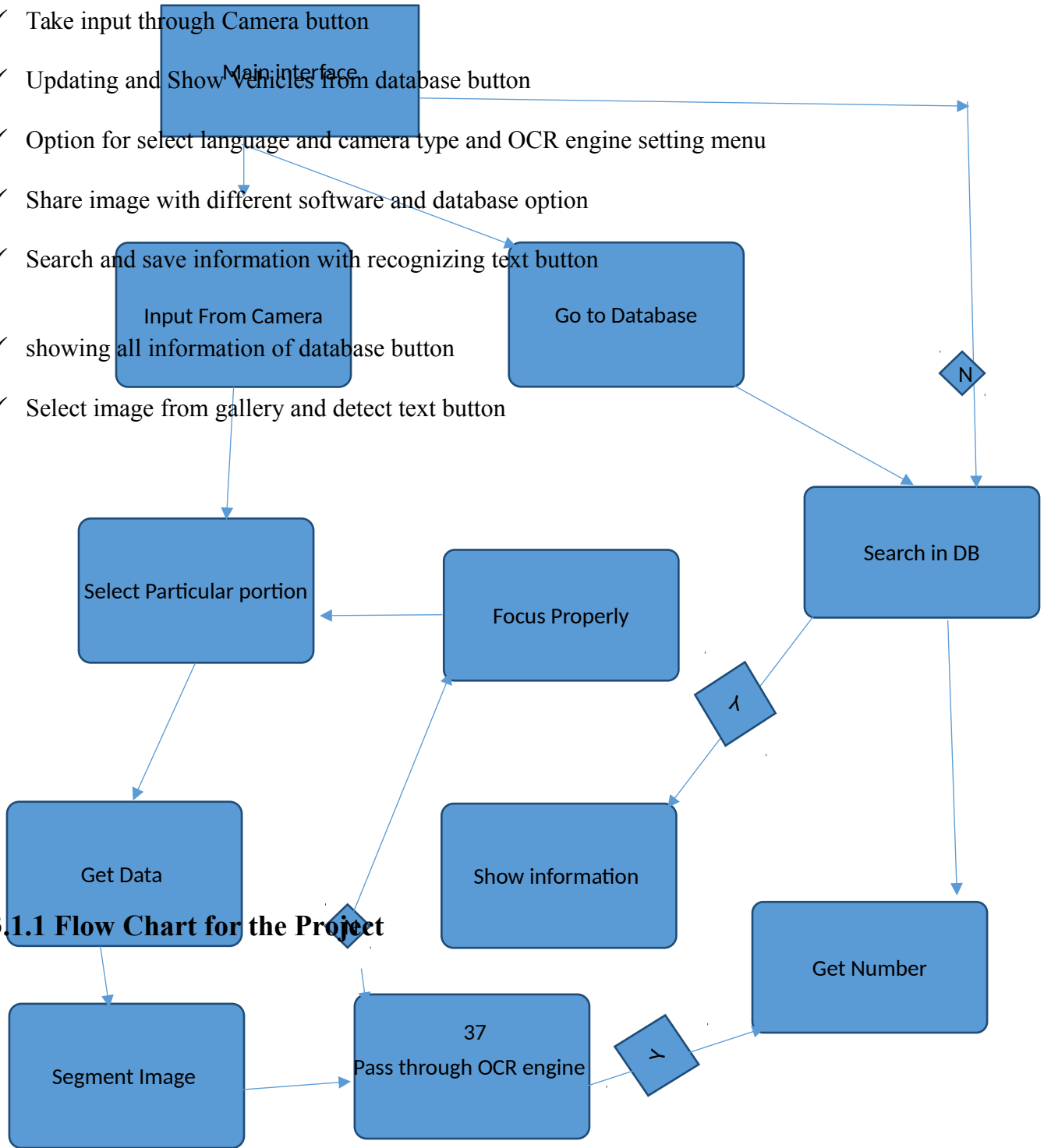
Chapter 3

Proposed Models

3.1 Design

My application is android utility application based software. My project contains...

- ✓ Main Activity Interface
- ✓ Take input through Camera button
- ✓ Updating and Show Vehicles from database button
- ✓ Option for select language and camera type and OCR engine setting menu
- ✓ Share image with different software and database option
- ✓ Search and save information with recognizing text button
- ✓ showing all information of database button
- ✓ Select image from gallery and detect text button



3.1.1 Flow Chart for the Project

Figure: Flow chart for Nameplate Text Detector using Android Application.

3.2 Implementation Procedure

Creating a fair environment for android development:

Here are some simple pre-requisites one must have to develop an android app.

Developer Requirement:

- Advanced knowledge of java
- Basic knowledge of XML

Hardware Requirement:

Development PC must be a fast one. I used a quad core machine clocked at @ 3.2 GHz with 4GB ram clocked @3.07Ghz. A big monitor or two is also helpful. During debugging it really eases the pain.

3.2.1 Android Development Environment

For developing application I had to create android development environment. Google basically supports the "Eclipse Classic" version. But there are also other IDEs. I have used Eclipse for my development. There are other IDEs like Intelligent Idea. But I choose Eclipse because I wanted such IDE in which I could write java code and at the same time using the same IDE I could work on GUI for android. Eclipse provides both of these features. I work in eclipse for a long time and also use to with eclipse. So I used Eclipse for my project development.

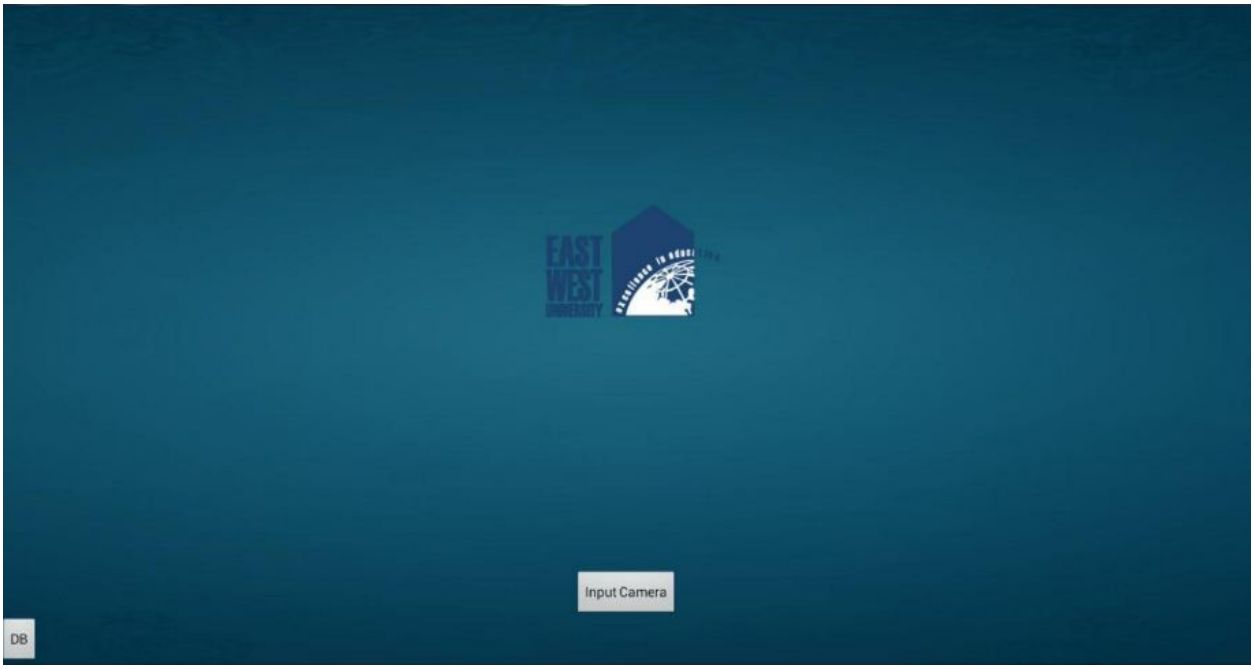
3.2.2 Project Setup:

For setup an android project I have to go to “file” then “new” then “android project” After completing these requirement you can get “hello word” simple android application. Now if you want to run the project then you need to click “run” button then you get window to setup adb emulator. Then you have to select your project as target, after completing these work your project application will be run.

3.2.3 Library Insertion:

For completing my project I need to add two library with my project. Tess-two and eye-two are the library. I get the library from https://github.com/rmtheis/tess-two website. After downloading the project, I need to import these into my eclipse project library. For this you should go File-> import->Android-> Existing Android Code into Workspace ->next-> Browse-> next. Then you should import downloaded project. After successfully import you must use these project as you project library. For this click right button of mouse on your project then Properties-> Android-> add. Now import library is done.

3.3 Main Activity:



Main Activity is connected with all others class so that easily I can access all the fetchers. Here you see three buttons one image view. By clicking them you can access gallery, input image from camera and also can handle database.

3.3.1 Image View:

This is a view of an image where I can set image. I use this image view for set image of logo from gallery. It is increasing my application fairness.

3.3.2 Select Gallery logo image:

I set the logo image from gallery. As result you can chose an image of a logo and crop a certain portion. Here I use my University logo.

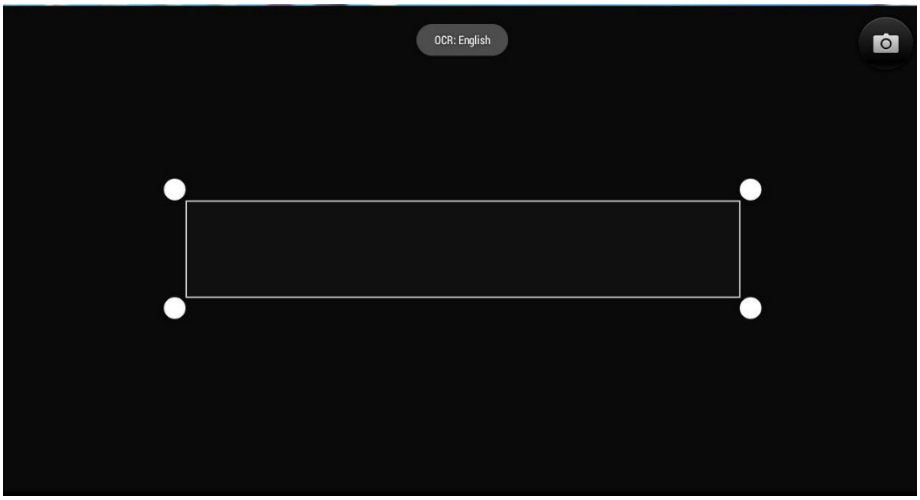
3.3.3 Button for select Camera input image:

Actually this project helps you to take an instant picture through android camera and analysis this picture and recognize text. When you press the button it open a camera surface and you will able to take short.

3.3.4 Button for work with database:

This button takes you to the database handler interface. Where you have option or button to save and show information of vehicles.

3.3.5 Number Plate Recognition Activity:



This class plays an important role for my project. Recognize text is my main contribute from nameplate. I use Google tesseract OCR technology for extracting text from the image.

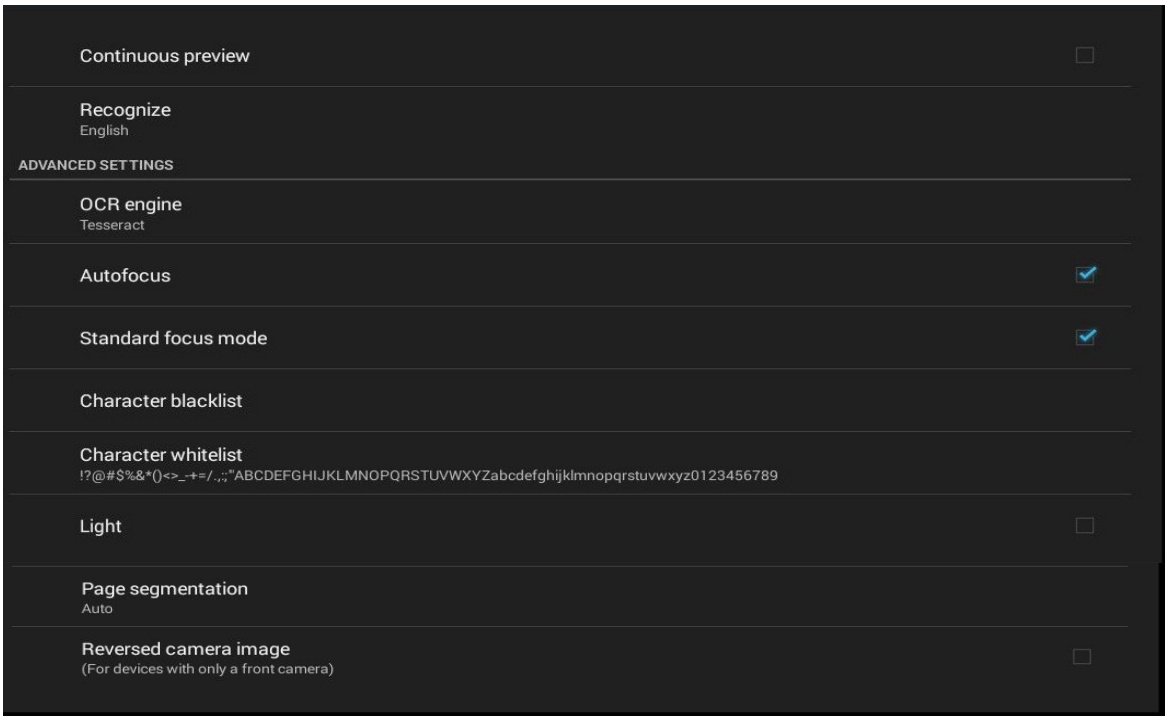
3.3.6 Select View:

Select View is that option by using this you can select the nameplates text portion using rectangle dragging. When you drag touching rectangle edge, it changes its size automatically so that you can adjust you text portion of nameplate and distance.

3.3.7 Capture Button:

This button is special button, when you click on it, it takes image and if fail then reset object focused.

3.3.8 Select option:



To do these work done, you can chose some option. These option are describe in below.

3.3.9.1 Continuous Preview:

Continuous Preview will show you instant streaming nameplate image text. Means when you point any nameplate text then it automatically extract text from the image before capture an image, so that before capture image you ensure about number of nameplate. It increases accuracy by resizing distance and rectangle size and autofocus.

3.3.9.2 Recognize Language:

You can recognize text from Bangla or English nameplate in my project. Here you can get two option Bangla or English. If you select Bangla then you can recognize Bangla text otherwise English text.

3.3.9.3 OCR Engine:

I use Google tesseract OCR technology for my project to recognize text. It is actually take an input image then analyses it and give text as output.

3.3.9.4 Auto and Standard Focus Mode:

When you select autofocus mode then it focus automatically when you taking image from camera. Otherwise you need to focus manually by clicking on shutter button.

3.3.9.5 Character Whitelist and Blacklist:

This option is for select which type of letter you don't want to recognize. If you specify any character into blacklist then it will ignore these character and recognize remaining. In my project default blacklist is empty.

3.3.9.6 Light:

If you select this option then when you taking an image it on the flash light. Otherwise it remaining off.

3.3.9.7 Page Segmentation:

This is very important for my project. Because you will find many type of segmentation mode for example: Auto, Auto (no OSD)[Orientation and Script Detection], Single block, Single character, Single column, Single line, Single word, Vertical block, sparse text. When you select auto then it segment as input image. But when you select mode then it will recognize as you select. It is also from Google tesseract OCR technology.

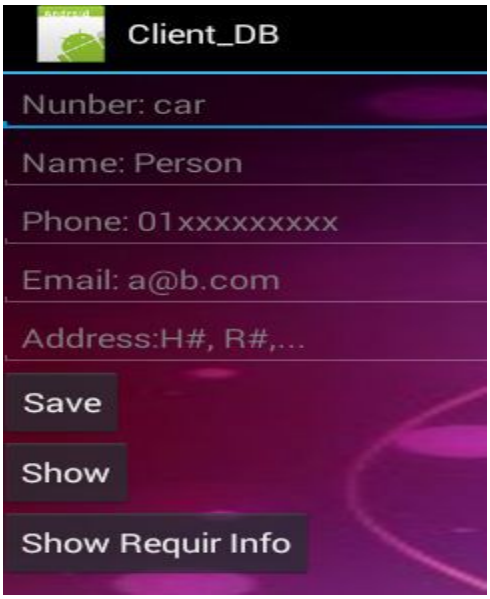
3.3.9.8 Reverse Camera Image:

For selecting this option your camera will take image but it reverse its data when previewing and store. Otherwise it remaining normal camera.

3.3.10 Option to share copy and send data:

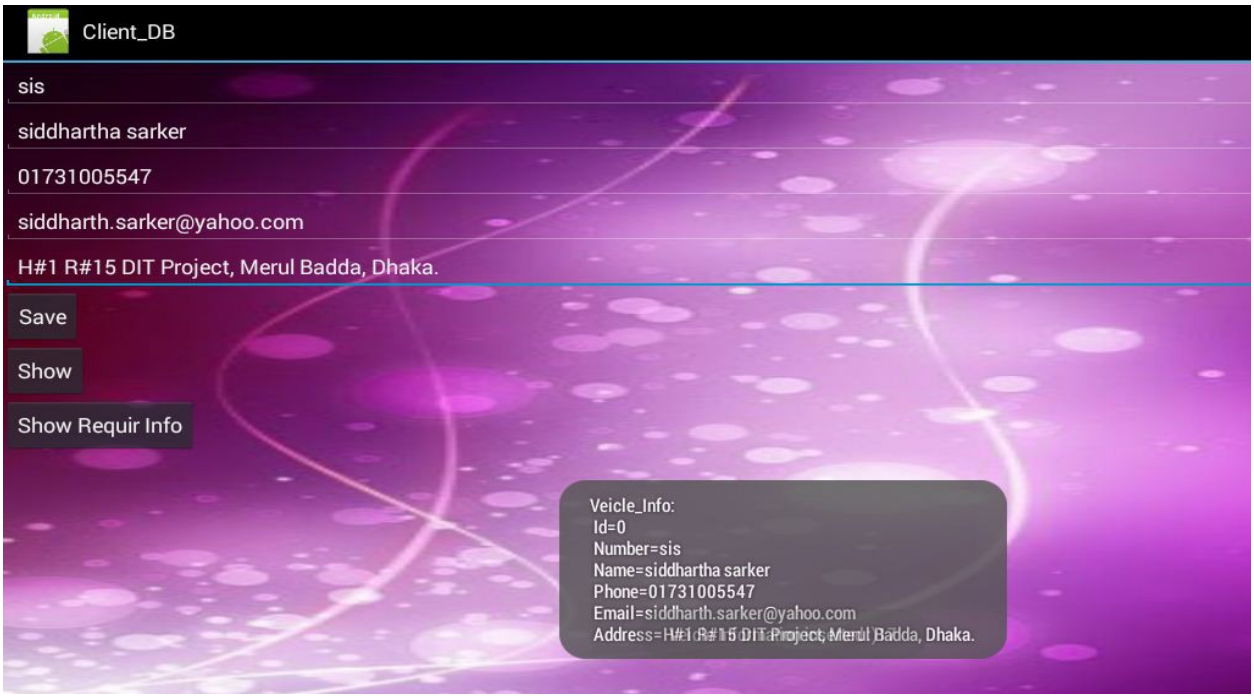
After find the number from nameplate you will able to share copy to clipboard or send data to database class for search or save vehicle information by long clicking on recognize text.

3.3.11 Database Activity:



Actually my target is to access government database where all vehicle dada are stored. But for experiment I use my own database which is created by using SQLite Database.

3.3.12 Insert Vehicle information:



This work is done by save button. There are some customize view including vehicle number, owner name, phone number, email number and address.

3.3.13 Show all information:

Above information can be shown using show button. For watching all information of full database you can use it.

3.3.14 Show Particular Vehicle information:

Show particular button will show information of a particular vehicle. You can search information about any vehicle using vehicle number.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

Android was my choice because it is the most popular, user-friendly mobile operating system of the world. It is also most selling smart phone in the world .I think the application will be able to give the outcome that I wanted from the very beginning of my development process. In the process of developing the application I learned many things about the android operating system and the development related tricks.

I also adopted easier and fresh ways, tricks and techniques that would definitely help in future development. On the other hand it will help other android developers to develop it and modify it according to their way to make this application more fruitful and global. I will not say that I was perfect to make the application. I have also made a lot of mistakes. But I think this will help a lot to develop android application. Nameplate Text Detector using Android Application Very soon takes a place in the android market. And for sure some modification will come to make it useable globally. It is really very hard task to fulfill all the requirements with an application of smart phone. And I also try to contract a traffic polish agency or troll collector for taking my apps. So there are a lot of chances of further development of the application in future. I have made my application keeping some space for further development.

4.2 Future Work:

I will improve my project work in future where you will able to check information through gallery image of nameplate.

- **Button for Detecting Text from Gallery input image:**

In current time I develop the button for accessing gallery image. You can select an image and crop its particular text portion and set as an image view.

- **Future Project Work:**

On the other hand using this project I want to detect National ID card and after accessing all the information it will also help to detect vote center for particular persons ID number. It will help all over the people and will make easy our life.

References:

[1] S.V. Rice, F.R. Jenkins, T.A. Nartker, *The Fourth Annual Test of OCR Accuracy, Technical Report 95-03*, Information Science Research Institute, University of Nevada, Las Vegas, July 1995.
[2] R.W. Smith, *The Extraction and Recognition of Text from Multimedia Document Images,* PhD Thesis, University of Bristol, November 1987.
[3] R. Smith, "A Simple and Efficient Skew Detection Algorithm via Text Row Accumulation", *Proc. of the 3rd Int. Conf. on Document Analysis and Recognition* (Vol. 2), IEEE 1995, pp. 1145-1148.
[4] Wikipedia. Tesseract. http://en.wikipedia.org/wiki/Tesseract_(software)[15-4-2015].
[5] A. B. Cambra, A. C. Murillo: Towards robust and efficient text sign reading from a mobile phone. IEEE International Conference on Computer Vision Workshops, ICCV 2011 Workshops, Barcelona, Spain, November 6-13,2011, pp.64-71, ISBN 978-1-4673-0062-9 http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6130223 [4-5-2015]
[6] Gautam Gupta . Making a Simple OCR Android App using Tesseract. http://gaut.am/making-an-ocr-android-app-using-tesseract/ [10-2-15]
[7] Wikipedia. Optical Character Recognition. http://en.wikipedia.org/wiki/Optical_character_recognition [24-4-2015]
[8].Liaqat, Ahmad Gull. "Mobile Real-Time License Plate Recognition." (2011).
[9]http://venturebeat.com/2013/01/28/android-captured-almost-70-global-smartphone-market-share-in-2012-apple-just-under-20/ [08-04-2015]
[10]http://en.wikipedia.org/wiki/Google_Play [21-04-2015]
[11] http://en.wikipedia.org/wiki/Android_(operating_system) [5-05-2015]
[12] http://forum.xda-developers.com/showthread.php?t=1595487 [723-05-2015]
[13] http://developer.android.com/guide/components/fundamentals.html [03-05-2015]