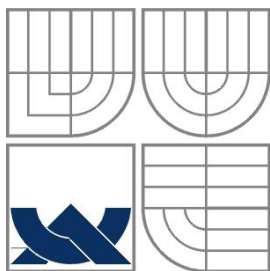


TDK DOLGOZAT

-

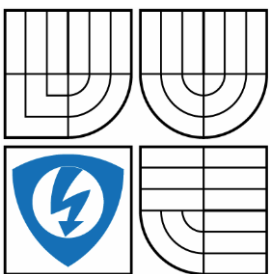
HORTAI FERENC



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

BRÜNNI MŰSZAKI EGYETEM



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH
TECHNOLÓGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

KÉPFELDOLGOZÁS ANDROID-KÉSZÜLÉKKEL - RENDSZÁMTÁBLA-FELISMERÉS KIÉRTÉKELÉSE

IMAGE PROCESSING USING ANDROID DEVICE - LICENSE PLATE DETECTION AND RECOGNITION

SZERZŐ: FRANTIŠEK HORTAI

AUTHOR

TÉMAVEZETŐ: Ing. PETER HONEC, Ph.D. (EGYETEMI ADJUNKTUS)

SUPERVISOR

BRNO 2014

Abstract

This thesis describes the design and workflow of creating an image processing application in Android system. First the possibilities are explained in choosing the development environment and their implementation. There is a description what the author needed to study and obtain to get enough information for the solutions and the possibilities of creating an Android application to evaluate images for license plate detection and evaluation. It is shown how the selected development environment was configured.

The thesis continues about the author's workflow of creating the application named HFImage. In the practical part is written about how they progressed in the solution. The description of designing the graphical user interface and the functionality of image processing for Android. Then the communication with the camera is explained also how the saving and loading data from storage is accomplished. This served as the basis for the application where the algorithms were implemented for image processing and image evaluation. The used algorithm is explained step by step and shown with pictures as the evaluation proceeds. There is flowchart too to get the whole picture how the used algorithm works.

Product of this thesis is a functioning application that allows to its user to capture images and analyze video stream. The algorithm evaluates the entering data - detects where the license plate is located. The application also allows recognizing texts and numbers from images. The included Tesseract OCR algorithm supports almost every image formats for optical character recognition. The application has other various practical features and options implemented within the application. For example: GPS location can show the geographic location of the taken picture and indicate some other data too of the taken picture. It also supports using camera lights as a flashlight and related data security issues.

The final product can be modified for further purposes. The appendices includes hyperlinks to online demonstration videos how HFImage works in real time.

Keywords

Android, image processing, machine vision, eclipse, Java, OpenCV, OCR, license plate recognition.

Content

1	Android.....	6
1.1	System architecture.....	7
1.1.1	Linux kernel.....	7
1.1.2	Libraries	8
1.1.3	Android Runtime	8
1.1.4	Application Framework.....	8
1.1.5	Applications.....	8
1.2	Android Application Components.....	9
1.2.1	Activities	9
1.2.2	Services	9
1.2.3	Broadcast Receivers.....	10
1.2.4	Content Providers	10
1.2.5	Additional Components.....	10
1.3	Native code	11
1.4	Connecting Android devices with external peripherals	11
1.4.1	Debugging considerations	13
1.5	Managing the application.....	14
1.5.1	Android activity lifecycle	14
1.6	Installing and starting an application	16
2	The required Software to get Started	17
2.1	Java	17
2.2	Eclipse.....	18
3	possibilities of image pressing in Adroid System	19
3.1	OpenCV on Android.....	20
3.2	Optical Character Recognition	21
3.2.1	ABBYY-OCR.....	21
3.2.2	Aspire-OCR.....	21
3.2.3	Online OCR API.....	21
3.3	Tesseract engine for OCR	22
4	Vehicle registration plate Recognition	24
4.1	Algorithm flowchart	28
5	Creating the basis of the application.....	29
5.1	Naming and creating an icon for the application.....	31

5.2	Using the camera	31
5.3	Load and save.....	32
5.4	Get actual location using the location manager.....	32
5.5	Binding pieces together.....	33
6	Hfimage Application	34
6.1	Main activity	35
6.1.1	Find licence plate on image	38
6.2	OCR simple.....	40
6.3	Get location	41
6.4	About activity	42
6.5	Camera activity.....	43
6.5.1	Camera activity's find plate mode	43
7	Data security	46
7.1	Confidentiality.....	46
7.2	Authenticity of messages	47
7.3	Authenticity of users.....	47
7.4	Security summary.....	47
	Conclusion.....	48

List of figures

Figure 1.1: Description of the Android platform architecture [9].....	7
Figure 1.2: USB Host and Accessory Modes [13].....	12
Figure 1.3: How debugging tools work [14]	13
Figure 1.4: The activity lifecycle [16].....	15
Figure 3.1: OpenCV logo [21].....	20
Figure 3.2: tesseract OCR [25].....	23
Figure 4.1: Coloured complex scene into greyscale	25
Figure 4.2: Applying Gaussian filter.....	25
Figure 4.3: Greyscale into B&W	26
Figure 4.4: Applying Canny algorithm	26
Figure 4.5: Detected rectangles	27
Figure 4.6: Detected licence plate.....	27
Figure 4.7: Algorithm Flowchart	28
Figure 5.1: Demonstration diagram to compare taken time	29
Figure 5.2: Binding parts together	30
Figure 5.3: Android emulator in Eclipse platform	30
Figure 5.4: HFimage application icon.....	31
Figure 5.5: HFimage application installation	33
Figure 6.1: HFimage application's hierarchy	34
Figure 6.2: HFimage application's main window.....	35
Figure 6.3: HFimage photo taking progress	36
Figure 6.4: HFimage image loading progress.....	37
Figure 6.5: HFimage two step method	38
Figure 6.6: HFimage three step method	39
Figure 6.7: HFimage four step method	39
Figure 6.8: HFimage OCR simple activity	40
Figure 6.9: HFimage OCR simple activity	41
Figure 6.10: About HFimage activity	42
Figure 6.11: Camera activity default preview	43
Figure 6.12: Find plate mode from greater distance	44
Figure 6.13: Find plate mode from closer	44
Figure 6.14: Find plate mode set to greater tolerance	45
Figure 6.15: Camera activity has found more plates.....	45
Figure 7.1: Block diagram of the cryptosystem for message confidentiality	46
Figure 7.2: Block diagram of the cryptosystem for message authentication.....	47

List of tables

Table 1.1: Additional Android Application components [11].....	10
--	----

Introduction

Computer vision is a field of science that includes methods for acquiring, processing, analyzing, and understanding images. The reason for the development in this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image [1]. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory [2]. One field of research in pattern recognition, artificial intelligence and computer vision is the optical character recognition, usually abbreviated as OCR¹. OCR is the mechanical or electronic conversion of scanned or photographed images of typewritten or printed text into machine-encoded/computer-readable text. This topic belongs to the field of image recognition and it is a very interesting area, which has great prospects.

For the most of us our electronic devices (phones, tablets) have become a part of our live. Most people are near to their phones all day. They would no more leave the house without taking a phone or some electronic device that allows them to be electronically accessible all the time. Android in 2013 ruled more than 80% of smartphone market share with a rising tendency. Of course there are a lot of other systems to use, but creation an application for an Android system is the closest way to reach most of the users.

My thesis describes and deals with the creation and design of an application for image processing in Android system. The first part of my thesis is about the recognizing of the solutions and the possibilities of creating applications and graphical user interfaces for Android system. An individual chapter is devoted to the selection of development environments. What are the possibilities and applications and where to begin. What is required and where to begin with this type of project.

In the practical part I have written about my workflow, as I progressed in my solution, the choice of the development environment and its implementation. Then I have described my approach in design of GUI and the basic functionality of image processing for Android, further the description of communication with camera, saving and loading data from storage. This work serves basis, where the algorithms are implemented for image processing and image evaluation.

¹ OCR - Optical Character Recognition



1 **ANDROID**

Android is an operating system based on the Linux kernel. Initially developed by Android Inc., which Google backed up financially in the beginning and bought later in 2005 [3]. Android was unveiled in 2007 along with the founding of the Open Handset Alliance: a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. [4] The first publicly available smartphone running Android was the HTC Dream and it was released on October 22, 2008. Now Android is the world's most widely used smartphone platform, overtaking Symbian in 2010. [5] Android in the end of 2013 ruled more than 80% of smartphone market share. [6]

The main motivation for Android was to build software that enables developers to create compelling mobile applications that take full advantage of everything that handset has to offer. It was built to be truly open. The applications can call upon any of the phone's core functionality such as making calls, sending text messages, using the camera, or to use any other sensors which the device offers, allowing developers to create richer and more cohesive experiences for users. Android utilizes a custom virtual machine that was designed to optimize memory and hardware resources in mobile environment. Android does not differentiate between the phone's core applications and third-part applications. They can all be built to have equal access to a phone's capabilities providing users with a broad spectrum of applications and services. Android is an open source, it can be liberally extended to incorporate new cutting edge technologies as they emerge. The platform will continue to evolve as the community of developer works together to build innovative mobile applications. [4]

As it was mentioned Android is an open-source software and Google releases the source code under the Apache License 2.0. This open-source code and permissive licensing allows the software to be freely modified and distributed by device manufacturers, wireless carriers and enthusiast developers. Precise definition of this licence can be read from the source. [7] In practice, Android devices ship with a combination of open-source and proprietary software. Android has a large community of developers, writing applications that extend the functionality of devices, written primarily in the Java programming language. [8]

To be accessible for development they created the Android SDK. Android can be easily found on the internet and it provides lots of support, documentations and example codes. Thanks to these features now we can find thousands and thousands of different applications. They can be downloaded simply form Google Play.

1.1 System architecture

The official name is Android; it is a pack of software components that includes the operating system, key applications and software for communication between them. The Android operating system is roughly divided into five sections and four main layers as shown below in the architecture diagram to make it more clear and understandable.

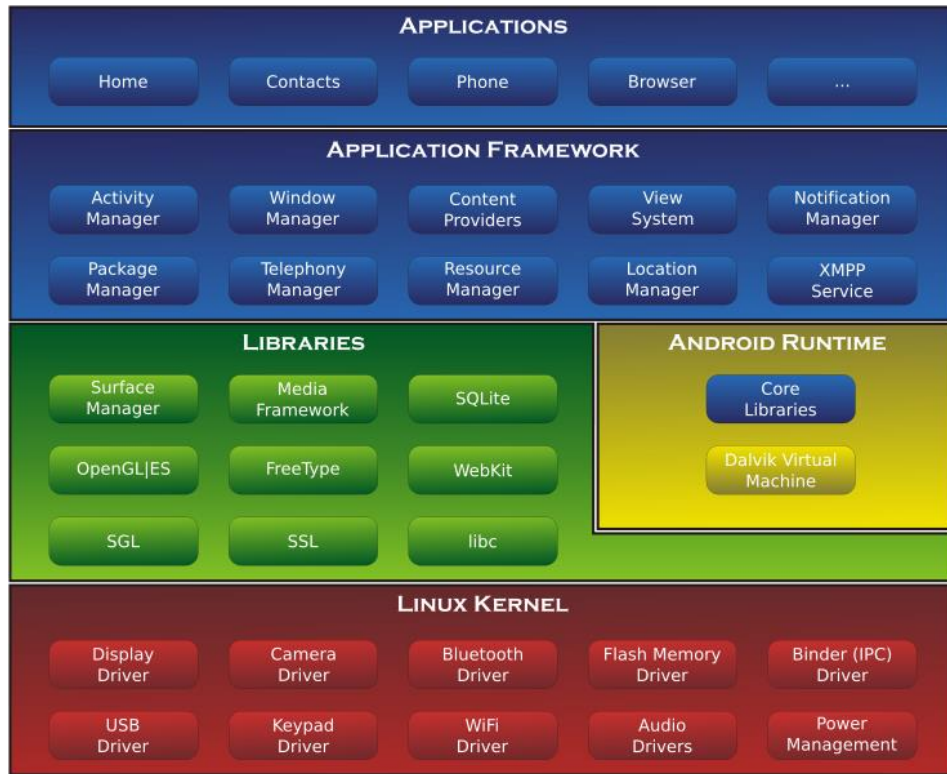


Figure 1.1: Description of the Android platform architecture [9]

1.1.1 Linux kernel

Android is based on the **Linux kernel** - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. The kernel also handles all the things that in Linux is really good at, such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware. This means that drivers and hardware specific code are all bound to the monolithic approach of this well-known operation system. Manufacturers can branch the source tree and write their own drivers for their devices. [10]

The next layers of this model use especially its stability, regardless of the energy performance and the advanced approach for multi-threaded processing. The core, however had to be cut because some of the features would be difficult to use in mobile devices.

1.1.2 Libraries

On the top of Linux kernel there is a set of libraries including open-source web browser engine WebKit, well known library *libc*, SQLite database which is a useful repository for storage and sharing the application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

1.1.3 Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. The Android runtime provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

There is an important key component that has to be introduced to the reader to understand the problem behind real-time applications on Android, this is the **Dalvik Virtual Machine**. DVM² is a kind of Java Virtual Machine which is specially designed, re-implemented and optimized for Android. The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the DVM. It has many advantages in resolving hardware incompatibilities for wide variety of Android devices. Of course it has the side-effect of dropdown in performance. The overhead is caused by Dalvik VM, which significantly decreases the ability of Android to host real-time applications.

1.1.4 Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. The application developers are allowed to make use of these services in their applications.

1.1.5 Applications

All the Android applications are at the top layer and are written to be installed to this layer only. Each application in the Android environment runs in its own process inside its own Dalvik VM instance. The virtual machine has been designed to driving efficiently with more of its instances. His whole concept is based on the Java VM.

² DVM - Dalvik VM - Dalvik Virtual Machine

1.2 Android Application Components

The application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file *AndroidManifest.xml* that describes each component of the application and how they interact. [11]

There are four main components that can be used within an Android application:

- Activities
- Services
- Broadcast Receivers
- Content Providers

1.2.1 Activities

They dictate the user interface and handle the user interaction to the smartphone screen. 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 to read emails. If an application has more than one activity then one of them should be marked as the activity that is presented when the application is launched. Each activity has its default window in which it can plot. Commonly the window fills the entire screen. However the size and the position of windows can be changed. On one screen there can also be more windows at the same time, which can be variously overlapping each other.

Applications may contain one or more activities. How many there are, how they will behave and refer to each depend on specific applications. One activity that is designed as the first will be launch with the start of the current application. Switching from one activity to another is accomplished by the current activity, which starting the following activity. An activity is implemented as a subclass of *Activity* class as follows:

```
public class MainActivity extends Activity { ... }
```

1.2.2 Services

They handle background processing which is associated an application. It is a component that runs in the background to perform long-running operations. 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's interaction with an activity.

A service is implemented as a subclass of *Service* class as follows:

```
public class MyService extends Service { ... }
```

1.2.3 Broadcast Receivers

Broadcast Receivers handle communication among applications or with the system, they respond to broadcast messages. For example: applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver which will intercept this communication and will initiate appropriate action. [11]

A broadcast receiver is implemented as a subclass of *BroadcastReceiver* class and each message is broadcasted as an Intent object:

```
public class MyReceiver extends BroadcastReceiver { ... }
```

1.2.4 Content Providers

A content provider component supplies data from one application to others on request. To handle data and database management issues, such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere entirely else. [11]

A content provider is implemented as a subclass of *ContentProvider* class and must implement a standard set of APIs that enable other applications to perform transactions:

```
public class MyContentProvider extends ContentProvider { ... }
```

1.2.5 Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic and wiring between them. These components are

Components	Description
Fragments	Represents behaviour or a portion of user interface in an Activity.
Views	UI elements that are drawn onscreen including buttons, lists forms etc.
Layouts	View hierarchies that control screen format and appearance of the views.
Intents	Messages wiring components together.
Resources	External elements, such as strings, constants and drawable pictures.
Manifest	Configuration files for the application.

Table 1.1: Additional Android Application components [11]

One of the most used things for designing GUI is the component View. The contents of the windows are available through hierarchical views. These are the basic functions of objects “View”. Each activity controls its own space inside the box. The basic View object contains the union of these spaces. Each space also contains the functions for user interaction. For example: by clicking inside a text input field the cursor begins to blink. Android has a lot of these preconfigured extension classes of View including

buttons, text fields, sliders, menu items, etc. The entire content of the window is then integrated into an activity by the method:

```
setContentView(R.layout.layout_name)
```

The sets which will appear in the window and how the different spaces will be arranged in the window. It is determined by an XML template (XML - eXtensible Markup Language). It is possible to create and define a number of these templates so it is easy to change the display of individual activities. Chyba! Záložka není definována.

1.3 Native code

The SDK is based on Java language. The official documentation states that the primary way of building applications is through this framework. It provides a standard set of classes that are present in the Java API and additional classes associated with the Android OS. A native tool-chain called NDK is provided as an optional part of the SDK. It enables to write part of the application in a native language such as C or C++. Some functions or libraries can be compiled to native libraries or shared libraries. A specific interface, the Java Native Interface (JNI), is responsible for connecting these native functions with the Java world. JNI is also responsible to handle the code when calling C/C++ functions from Java or accessing Java objects from C++. [12]

The manufacturer says that the NDK will not benefit most apps. “As a developer, you need to balance its benefits against its drawbacks. Notably, using native code on Android generally does not result in a noticeable performance improvement but it always increases your app complexity. In general, you should only use the NDK if it is essential to your app—never because you simply prefer to programme in C/C++.” [12]

Typical good candidates for the NDK are self-contained, CPU-intensive operations that don't allocate much memory, such as signal processing, physics simulation and so on. NDK module is supported from Android version 1.5 and the most supported processors are from the ARM family.

1.4 Connecting Android devices with external peripherals

In order to connect anything with a mobile device it has to be equipped with a set of ports which allows interaction with other devices. A wireless solution advantage is to avoid the use of cables and to maintain the “mobility” of mobile devices. The Android OS has support for peripherals like a keyboard or a mouse thanks to the mentioned Linux kernel which has a decent collection of drivers for external devices. Beside the wireless methods a connection can also be made via the USB port.

Android supports a variety of USB peripherals and Android USB accessories through two modes:

- USB accessory
- USB host.

In USB accessory mode, the external USB hardware acts as the USB host. When the Android-powered device is in USB accessory mode, the connected USB hardware (an Android USB accessory in this case) acts as the host and powers the bus. Examples of accessories might include robotics controllers, docking stations, diagnostic and musical equipment, kiosks, card readers and much more. This gives Android-powered devices that do not have host capabilities the ability to interact with USB hardware. Android USB accessories must be designed to work with Android-powered devices and must adhere to the Android accessory communication protocol. [13]

In USB host mode, the Android-powered device acts as the host and it powers the bus. Examples of devices include digital cameras, keyboards, mice, and game controllers. USB devices that are designed for a wide range of applications and environments can still interact with Android applications that can correctly communicate with the device. [13]

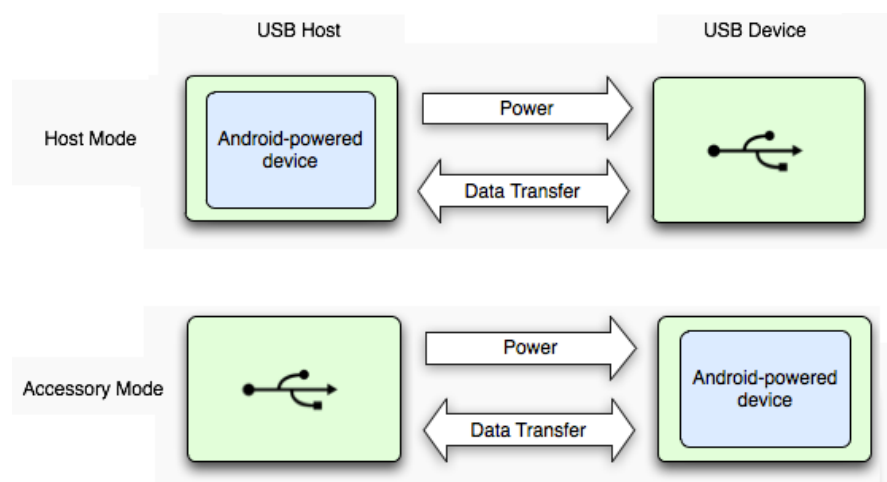


Figure 1.2: USB Host and Accessory Modes [13]

The primary purpose of a USB connector on a mobile device is to make connection with a PC and for charging a battery. In these situations the PC has the role of a USB host and the mobile or tablet acts as a USB device.

For image processing purposes it can be used to connect an external camera to the device to get images or video stream. Most of the phones have already a built in camera, but these cameras have small lens so captured picture has lower quality than pictures from professional cameras. To get a better picture through higher resolution or greater focused lens can be achieved with an external camera.

1.4.1 Debugging considerations

The Android SDK provides most of the tools that are needed to debug the applications. It needs a JDWP-compliant debugger if it is wanted to be able to do things such as step through code, view variable values, and pause execution of an application. [14]

When debugging applications that use USB accessory or host features, it is necessary to have USB hardware connected to the Android-powered device. The Android device has to allow the USB debugging. It is easy to allow in: Setting > System > {} Developer Options > USB debugging.

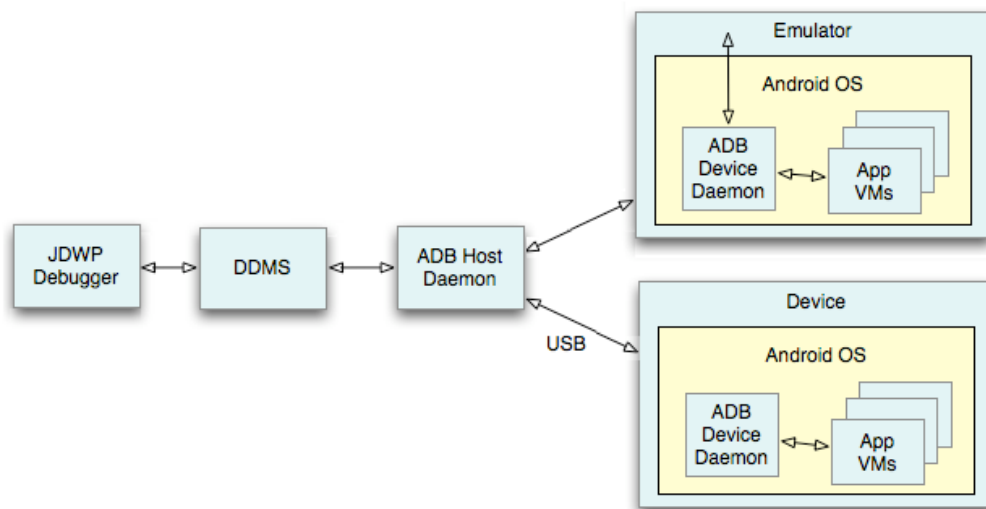


Figure 1.3: How debugging tools work [14]

This figure shows how the various debugging tools work together in a typical debugging environment.

1.5 Managing the application

This part describes the theory how to manage an application. It was necessary to understand for me, because I am using more activities in my application which share and transfer data among themselves. My application communicates with other application (OpenCV manager) and handles they status. It also communicates and handles system recourses, accessing camera and GPS locator, read and write files to available storages. To put it in a nutshell, I had to put all this together and prevent my application from any crash downs and errors.

Whenever an Android components starts an application object is created. It is started in a new process with a unique ID under a unique user. Even if you do not specify one, the Android system creates a default object for you. This object provides the main lifecycle methods.

The Android platform supports lifecycle event which are called in the case of process or component termination as well as in case of a configuration change. The developer is responsible for maintaining the state of the application. He is also responsible for restore the activity instance state. The instance state of an activity is the non-persistent data that needs to be passed between activities restarts during a configuration change to restore user selections. [15]

Android system also allows recycling Android components to free up resources. The activity state can be running, paused, stopped, killed (terminated by calling *finish()* method). The user should not notice if an activity which is still part of an activity stack has been terminate or not. For this the developer needs to store the state of the activity at the right point in time and restore it. He also should stop any unnecessary actions if the activity is not visible anymore to save system resources.

1.5.1 Android activity lifecycle

When an activity changes different states described above, it is notified through various callback methods. All of the callback methods are hooks that can be overridden to do appropriate work when the state of the activity changes. These methods define the entire lifecycle of an activity.

The entire lifetime of an activity happens between the call to *onCreate()* and the call to *onDestroy()*. The user can see the activity on-screen and interact with it within the visible lifetime of an activity which happens between the call to *onStart()* and the call to *onStop()*, which is called when a new activity starts and this one is no longer visible.

Between these two methods, you can maintain resources that are needed to show the activity to the user. For example, you can register a *BroadcastReceiver* in *onStart()* to monitor changes that impact your UI, and unregister it in *onStop()* when the user can no longer see what you are displaying. The system might call *onStart()* and *onStop()* multiple times during the entire lifetime of the activity, as the activity alternates between being visible and hidden to the user. [16]

The foreground lifetime of an activity happens between the call to *onResume()* and the call to *onPause()*. During this time, the activity is in front of all other activities on screen and has user input focus. For example *onPause()* is called when the device goes to sleep or when a dialog appears. Because this state can transition often, the code in these two methods should be fairly lightweight in order to avoid slow transitions that make the user wait. Chyba! Záložka není definována.

For better understanding a figure is shown below. This flowchart shows when and what method is called, illustrates loops and the paths an activity might take between states. The rectangles represent the callback methods for implementation to perform operations when the activity transitions between states

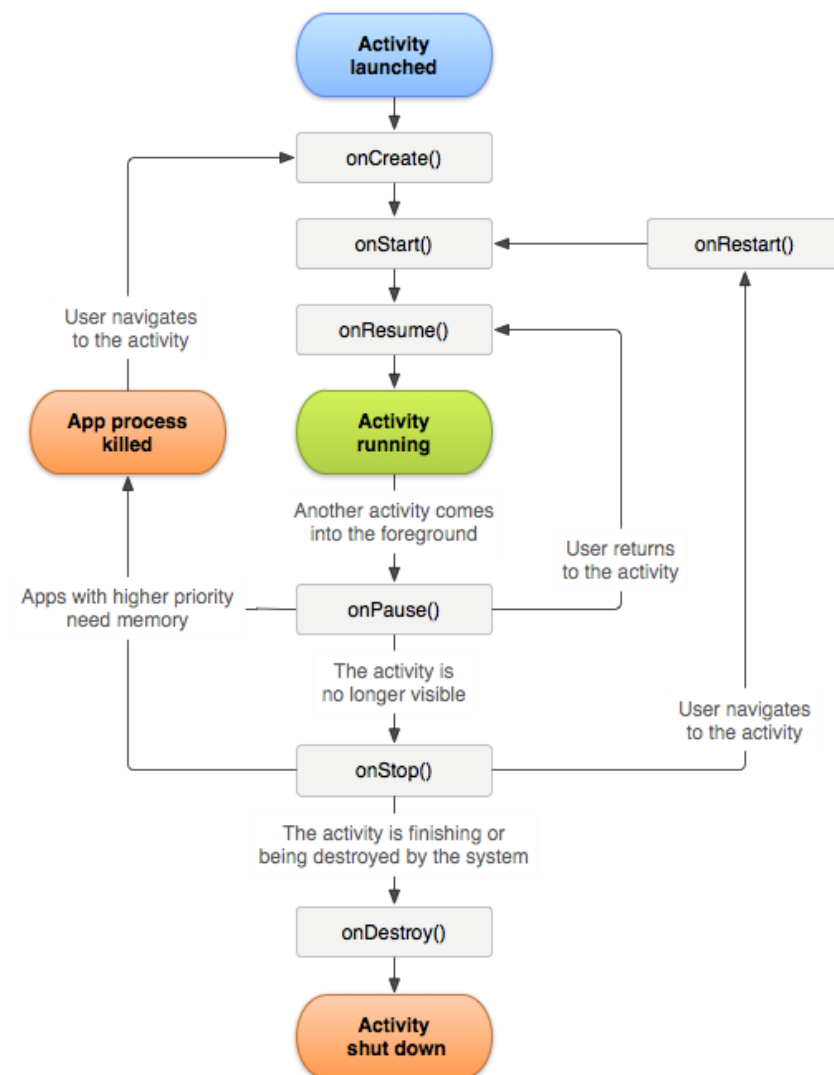


Figure 1.4: The activity lifecycle [16]

1.6 Installing and starting an application

The compiled code, along with the other data is packed in package with an extension **.apk*. This is then called an application that can be saved to the user's device and can be installed. For completing the final application the *aapt* tool is needed. The *aapt* (Android Asset Packaging Tool) is the tool for creating applications from files with extension **.apt* on the Android platform. It is included in a wide range of tools which are freely available to developers under one name Android SDK.

When an application start occurs, in a certain level of abstraction a separated world is created. It is due to a separation of memory and creation of another instance of the Dalvik VM. A Linux process with a unique identification of the user is formed on the kernel layer. This ensures that the individual applications cannot access the memory and files of other applications. Each user, in this case application has implicitly access only to its "property". To share data among applications special methods are created. There may also be applications with the same user identification. It is one of several options for sharing memory resources. This way Android ensures strength security, and yet not loses its functionality.

There is a whole range of layers for starting an application, from running the process in the background to the display methods. These processes will run regard to their priority. The entire system ensures maximum utilization of system resources.

In case there is insufficient memory the low priority processes will be ended as first. However, it is possible to disable individual parts of the applications. Applications in Android are mainly built from four basic components. Those combined components create the final application. The system is able to separate the components according to their priority and then effectively manage the computing power and memory. However, the priority of each component is not controlled only by the system. The priority of the application's components can be set during the development.

2 THE REQUIRED SOFTWARE TO GET STARTED

This chapter provides a generic introduction of the main notions used in this thesis: to summarize the necessary information from the initial research gravitating towards the Android OS and tries to filter the results through the viewpoint of this platform.

This chapter also includes my workflow at the beginning, how I install the necessary software to create an application on my android device, how I install the tools on my Microsoft OS the Java JDK, Eclipse and Android SDK. These software will be explained in the subchapters. Connected to the chapter's name it can also be used as a handbook: how to begin to programme an application for an Android device.

The internet is full of explanations what is what and why it is used, as to my thesis I try to use a shorter form of explanation to clear why I decided to use these SW.

2.1 Java



It is developed by the company Oracle. Java is the foundation of virtually every type of networked application and is the global standard for developing and delivering mobile applications, games, Web-based content, and enterprise software. There are lots of developers worldwide, because Java enables its users to efficiently develop and deploy exciting applications and services. With comprehensive tooling, a mature ecosystem, and robust performance, Java delivers applications portability across even the most disparate computing environments [18]. Java is an object oriented programming language. Source programmes will not compile into machine code but in the intermediate stage called "Byte-code", which is independent from the used platform.

Writing Java applets and applications needs development tools like JDK. The Java Development Kit (JDK) is an implementation of either one of the Java SE, Java EE or Java ME platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, Mac OS X or Windows. The JDK includes the Java Runtime Environment, the Java compiler and the Java APIs. [19]

To use eclipse it was necessary to install Java (I choose Version 7 Update 45), it can be simply downloaded from URL: www.java.com. To download it and use it you have to accept their terms of the end user license agreement.

2.2 Eclipse



Eclipse is an open source integrated development environment. It is mostly used to develop applications in Java. Eclipse contains a base workspace and an extensible plugin system for customizing the environment. So the current version supports various programming languages, and it is easy to extend with further plugins. So by simply adding a new module Eclipse is able to learn a completely new feature, from languages like Java, C, C++ or XML and so on. There are some well-prepared packages already with plugins in it and they can be easily downloaded from URL:

<http://www.eclipse.org/>

I downloaded: Eclipse Standard 4.3.1. For creating Android application I had to install the Eclipse plugin for Android Development Tools called “ADT”. Android offers a custom plugin for the Eclipse IDE. This plugin provides a powerful, integrated environment in which to develop Android apps. It extends the capabilities of Eclipse to let its user quickly set up new Android projects, build an app UI, debug app, and export signed (or unsigned) app packages (APKs) for distribution. There are step by step instructions how to configure and install it. It can be downloaded from URL: <http://developer.android.com/sdk/installing/installing-adt.html>

The next step was to install the Android Standard Development Kit called “SDK”. The Android SDK provides the API libraries and developer tools necessary to build, test, and debug apps for Android. To find it on the internet it is just to look it up by Google “Android SDK” and it will throw out the appropriate link on “developer.android.com”.

There is a possibility to download the full package with a single download. But the step by step version helps to understand what it contains. The ADT Bundle includes everything that is necessary to begin developing apps. ADT bundle already contains the Eclipse + ADT plugin, Android SDK Tools, Platform-Tools and Android system image for the emulator.

Now Eclipse with Android SDK is functioning on a MS Windows OS. The last step is to download and install the existing versions of Android. It is the Platform-Tool called Android SDK manager (it can be run from an icon in Eclipse). It was necessary to install those versions on which I wanted to run my applications on my Android devices. My device runs the 4.0.3 version so I installed all the versions from 2.3 to the latest version to secure the compatibility (Android 4.4 KitKat - 2014).

One part of this whole package is the emulator. It has similar options like a real device. Of course some of the features are missing such as accelerometers or vibrations. This tool can help while developing the application but the final application must be tested directly on a specific device by the manufacturer.

3 POSSIBILITIES OF IMAGE PRESSING IN ADROID SYSTEM

This part is small research about the possibilities for image processing and evaluation in Android system. Of course there are lots of other possibilities where to begin, but in this part I would like to introduce shortly what I found. In the future I would like to create some own libraries for image processing, to create own methods and classes specified for my image processing evaluation.

JJIL is a Java image processing library. It includes an image processing architecture and over 60 routines for various image processing tasks. JJIL is particularly targeted towards mobile applications. It includes interfaces so images can be converted to and from native formats for J2ME, Android, and J2SE. (In this library is a good Face-recognition example).

It can be found at URL: <https://code.google.com/p/jjil/>

The Catalano Framework is a framework for scientific computing for Java and Android. The project started as an initial port of the many features of the AForge.NET and Accord.NET frameworks for .NET, but is steadily growing with more advanced features which are now being shared between those projects. The project is organized under the same architecture found in the mentioned frameworks. Experienced users who would wish to leverage their expertise on Accord and AForge to work on Android can rely on the same namespaces for: Image Processing, Fuzzy Logic, Math, Statistics, Machine Learning, and Neural Network.

It can be found at URL: <https://code.google.com/p/catalano-framework/>

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS.

It can be found at URL: <http://opencv.org/platforms/android.html>

3.1 OpenCV on Android

I asked around some experienced developers, looked up some forums and read some Google answers to get an opinion. Because image evaluation and character recognition needs much computing I had to use some other tools and not just Java, what is not the best choice for such types of work. Most of the responses recommended using OpenCV. So to move to a real time environment I choose OpenCV.

I downloaded **OpenCV for Android** (version 2.4.9). Because OpenCV uses C and C++ codes first I had to implement the latest NDK toolset that allows implementing parts of application to use native-code languages. OpenCv4Android is available as a SDK with a set of samples and Javadoc documentation for OpenCV Java API. It also contains prebuilt apk-files, which can be run on an Android device “instantly“. The second step was to implement the correct libraries to my application, which I did first wrong. It made lots of trouble to find out, that I had implemented the wrong libraries for ARM hardware (library armeabi). I had to use (library armeabi-v7a) which finally worked. To ensure full compatibility is good to implement all libraries (x86, mips, armeabi-v7a, armeabi) to the application. OpenCV Manager shows the hardware information about the current device, as an example my device’s hardware information are:

ARM v7 with Neon and VFPv4.

OS version: REL (4.0.4), API level 15.

For the proper functionality is necessary to install the latest **OpenCV manager** which can be found and downloaded from Google play. OpenCV manager is an application that provides the best version of the OpenCV for hardware. OpenCV library can be used by other application for image enhancement, object detection, recognition and tracking and so on. To prevent my application from crashes I had to handle OpenCV library statuses, initializations and OpenCV Manager version compatibility. This all can be easy looking, but I had lot of trouble to implement it.



Figure 3.1: OpenCV logo [21]

3.2 Optical Character Recognition

Optical Character Recognition is used to extract text from an image or a scanned document. This text is used for further processing such as it can be edited, formatted, searched, indexed and automatically translated or converted to speech.

There are already solutions for OCR. It is better to look around on the internet first than to write an own code for OCR. This will spare a lot of time and effort for the developer. That is why this thesis also involves a small research of some existing OCR libraries which are:

- Tesseract-OCR
- ABBYY-OCR
- Aspire-OCR
- Online OCR API

3.2.1 ABBYY-OCR

ABBYY is a company that provides software for document recognition, data capture and linguistic software. They also provide OCR SDK for mobile platforms. Using their SDK, developers can develop OCR based applications for mobiles. They provide SDKs for following mobile platforms.

Mobile OCR Engine is a powerful software development kit which allows mobile developers and small footprint applications to integrate highly accurate optical character recognition (OCR) technologies that convert images and photographs into manageable and searchable text. Toolkit supports the most popular mobile platforms and devices - iOS (iPhone) and Android. [21]

ABBYY OCR Engine is not for free. It is surely very useful and powerful engine, but I decided to search further. Unfortunately ABBY is not the type of an open-source company.

3.2.2 Aspire-OCR

Aspire-OCR is an optical recognition engine that provides APIs for Java, C, C++, Delphi, CSharp.Net, Visual Basic.Net and Visual C++.Net. It also includes functionality for Barcode reading for Barcode reading [23]. The SDK is used to develop Aspire-OCR oriented applications and it has support for platform Windows, Mac, Linux and Solaris. It takes almost every format of images like .png, .jpg, .bmp, etc. as input. They support Java platform but it is hardly to find any existing application that uses this engine in Android.

3.2.3 Online OCR API

As the name indicates, it is an online service that provides APIs for OCR. By using these APIs, it is possible to develop different kinds of mobile and web based

applications. It supports various types of images to evaluate and can read text from every well-known language. The handicap is that to use it you have to be online to use this feature and I wanted to work my application in offline mode too. So I decided to choose a built-in OCR engine instead of an online OCR.

3.3 Tesseract engine for OCR

It is rumoured that Tesseract is the best open source OCR machine available. Some time ago I had tried some other open source OCR programs without much success. So I decided to try something new, to try Tesseract.

Tesseract is a well-known OCR machine. It works better than any other open source OCR system I have tried so far. It is free to use under the Apache License 2.0. Combined with the Leptonica Image Processing Library it can read a wide variety of image formats and convert them to text in over 60 languages. It was one of the top 3 engines in the 1995 UNLV Accuracy test. Between 1995 and 2006 it had little work done on it, but since then it has been improved extensively by Google. First it was released as open source in 2005 by Hewlett Packard, now Tesseract development has been sponsored by Google since 2006. [24]

My expectations were that if Android was backed up and bought by Google so Tesseract was, it is surely possible to make then compatible. Some Google answers also recommended using Tesseract OCR on Android. There are some applications that already use it and are functioning. So that task was set: implement Tesseract in my application.

Tesseract works on Linux, Windows (with VC++ Express or CygWin) and Mac OSX, so how to implement it on android the question that remains. To use for other platforms it has to be compiled. So that is the NDK tool needs to work again. To spare time I tried it on other platform. When it finally worked to spare time in repeated compiling I decided to compile the whole library and use it as a final form that works. The second reason is that the automatic memory management in Java is carried by a daemon thread called Garbage Collection. C++ doesn't have (built-in) garbage collection, but that doesn't mean that the developer should let dynamically allocated memory stay allocated forever. The developer can and should free it. The most basic way to do that is to free your object reference (or pointer, in case of C++) manually when you don't need it anymore. When Tesseract finally worked I did not want to touch it anymore, it made lots of trouble to debug C/C++ codes through Java.

The conclusion is that Tesseract is very usable OCR tool, especially for people who can fix minor problems in the source and enable to use it on android. I implemented these parts to my application:

Tesseract OCR v3.03.

- Available at: (<https://code.google.com/p/tesseract-ocr/>) [cit. 2014.05.03]

Leptonica 1.70 image processing libraries.

- Available at: (<http://www.leptonica.com/>) [cit. 2014.05.03]

A v3.02 trained data file for a language. The language data files must be extracted to a subdirectory named “tessdata”.

- Available at: (<https://code.google.com/p/tesseract-ocr/downloads/list>) [cit. 2014.05.03].

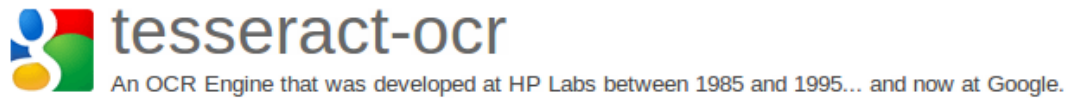


Figure 3.2: tesseract OCR [25]

4 VEHICLE REGISTRATION PLATE RECOGNITION

A vehicle registration plate is a metal or plastic plate attached to a motor vehicle or trailer for official identification purposes. The registration identifier is a numeric or alphanumeric code that uniquely identifies the vehicle within the issuing region's database. In some countries, the identifier is unique within the entire country, while in others it is unique within a state or province. Whether the identifier is associated with a vehicle or a person also varies by issuing agency. Depending on the country, the vehicle registration plate may be called a license plate or tag (United States), licence plate (Canada), or number plate (United Kingdom, Australia et al.), or rego plate (Australia). [26]

License plate recognition is a technology combining the methods of computer vision, image processing and pattern recognition. The licence plate detection has to comfort some difficulties which result from uncontrolled imaging, conditions such as complex scene, weather conditions, low contrast, blurring viewpoint changes.

License plate detection through Haar-like features

OpenCV provides us tools and functions to implement Haar-like features. In OpenCV, Haar-like features are implemented through Haar-Training. Haar-like feature is a training process from lots of license plate samples. It can learn where the license plate is located. From the trained data it can then predict on a new picture where the licence plate will be. To execute this on Android would be quite complicated for me. First I did not have a large amount of samples (thousands of samples) to make the trained data. Second, to learn the algorithm is needed a computer with high computing power where another platform would be used. Then the compatibility issues are present to implement it on Android. So I choose another way to achieve my goal.

License plate detection through Edge Detection

To find license plate for humans is easy, because we can separate forms and we can read symbols. To explain this to a computer it is necessary to prepare an algorithm that explains the computer what to do. A license plate has a form of a rectangle. So the task is to find a rectangle objects within an image which includes numbers or characters. The first step is to find edges on the image for creating the rectangles. There are more possibilities how to achieve this I will explain my approach. At the end I expected to return the license plate in coloured image if it is possible to achieve it from the analysed picture. First step is to load the image or the acquired camera frame into the memory (in case of coloured image it is a 3D matrix).

Licence plates have reflective surfaces with high contrast so they do not need to be searched in coloured picture. The algorithm first step is to make a greyscale copy of the original input. So from 3D matrix we retrieve 2D matrix which numbers represents the input's pixels in 1 byte (value from 0 to 255). If the camera gives us already a grayscale picture this step is skipped.



Figure 4.1: Coloured complex scene into greyscale

The built-in cameras which mostly have small lens accumulates noise on the acquired picture. To reduce this unwanted noise a Gaussian filter is used as effective noise suppression. With naked eye we cannot see the difference but after evaluating the edges there will be a difference. The smoothing reduces the amount of edges.

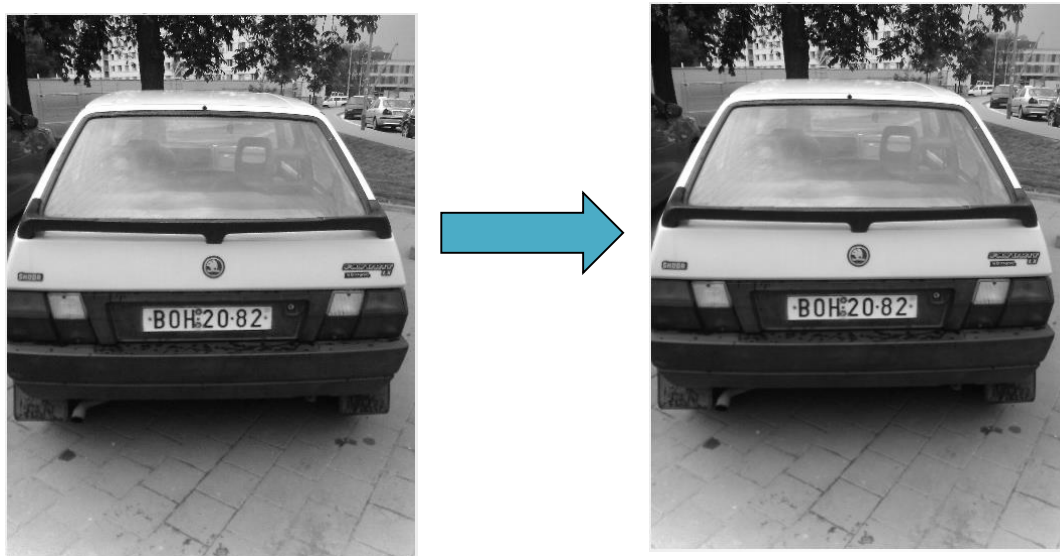


Figure 4.2: Applying Gaussian filter

The next step is to make a binary value matrix from this greyscale image. Simply said: grayscale picture into a black-and-white picture. This can be achieved by a threshold. This binary matrix is further used to find edges. The pixels are represented by 1 and 0 values.

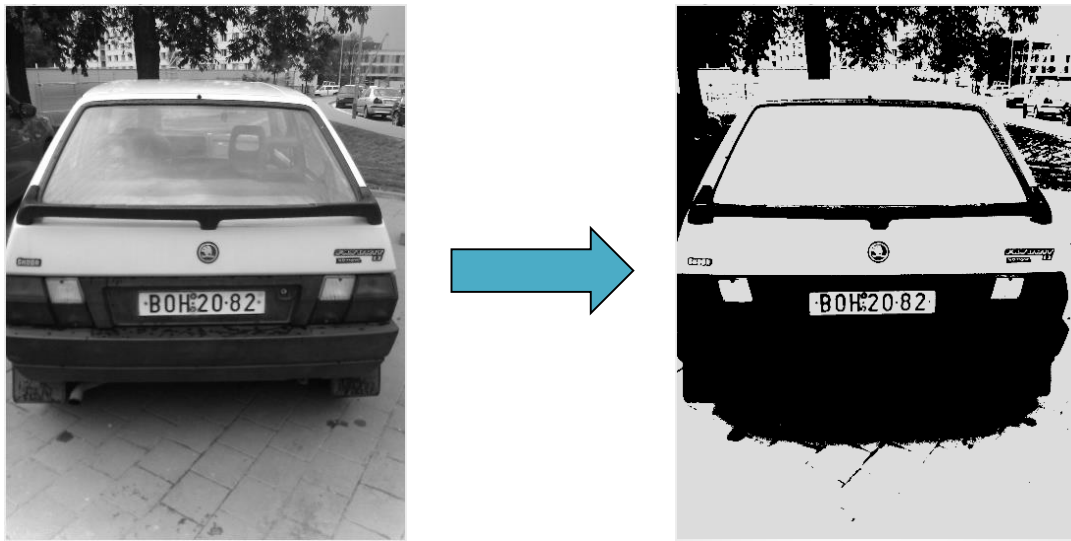


Figure 4.3: Greyscale into B&W

Finds edges in an image the Canny86 algorithm is used. The function finds edges in the input image and marks them in the output map edges using the Canny algorithm. The smallest value between threshold1 and threshold2 is used for edge linking. The largest value is used to find initial segments of strong edges. There is lots of literature which explains in details how this algorithm works.



Figure 4.4: Applying Canny algorithm

Next step: find from these edges rectangles. There are several morphological operations to achieve this. Finds contours in a binary image (using the algorithm [Suzuki85] for contour extraction). The contours are a useful tool for shape analysis and object detection and recognition. Then Approximate contours into curves. Approximate polygonal curve(s) with a specified precision. These functions approximate a curve or a polygon with another curve/polygon with fewer vertexes so that the distance between them is less or equal to the specified precision. The approximation accuracy parameter needs to be specified. This is the maximum distance between the original curve and its

approximation. From these curves rectangles are bounded. For example I put picture below. On the left side is a picture with Gaussian filter applied on the right side no filter was used. This is a good demonstration why is necessary to use filtration on the input.



Figure 4.5: Detected rectangles

From these rectangles are sorted out the ones which do not meet the condition of license plate ratio. The ratio is within a tolerance range.

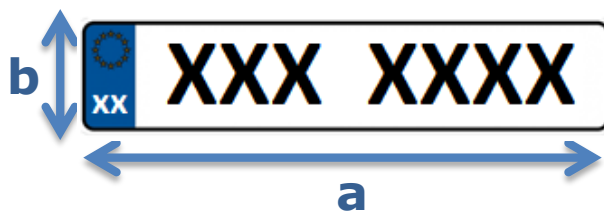


Figure 4.6: Detected licence plate



The remaining rectangle(s) should show the coordinates of the license plate(s). This can be cut out from the original picture for further evaluation. In my case it is character recognition.

To summarize the character recognition I used three steps:

- Pre-processing: `bitmap.Config.ARGB_8888` - to put the input to the correct form
- OCR (`bitmap`, `tessdata`) – use it on the picture with the learned language data.
- Post processing: filter ['a-z' || 'A-Z' || '0-9'] characters and return as data.

4.1 Algorithm flowchart

The flowchart represents the simplified algorithm to see the whole process at once.

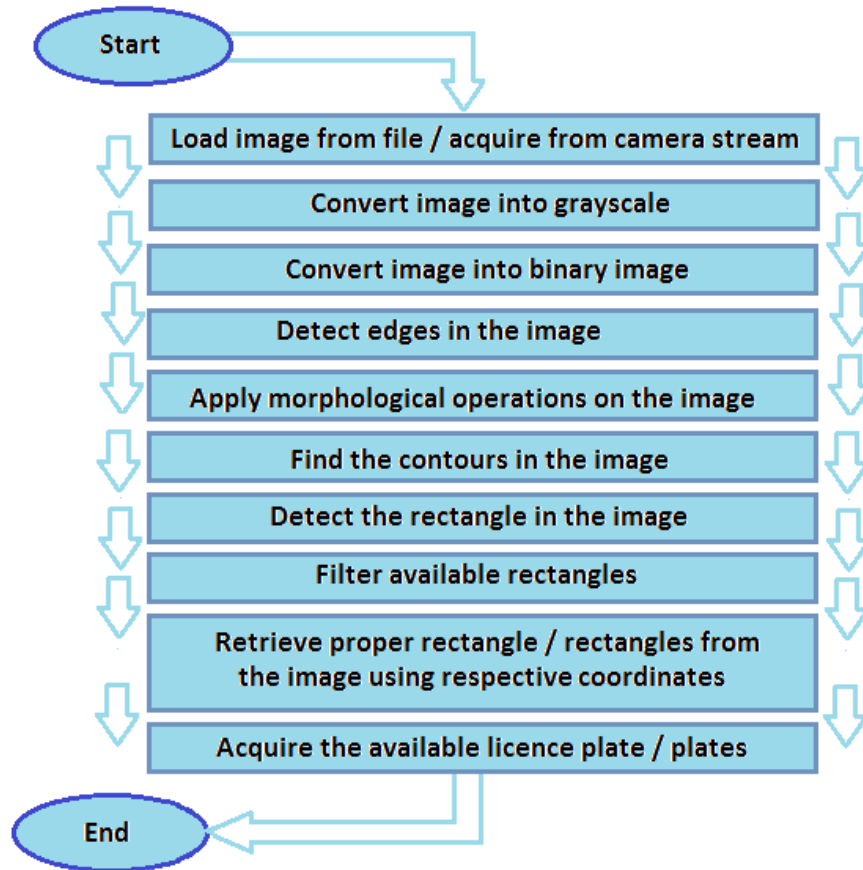


Figure 4.7: Algorithm Flowchart

5 CREATING THE BASIS OF THE APPLICATION

As it was mentioned Android applications are programmed in java. The nature of Java is: “write once run anywhere” in contrast to C or C++ languages which are “write once compile anywhere”. With Java the developer can expect to run they code on various platform on which the JVM is installed. In C or C++ it is necessary to recompile the code to all the different hardware platforms which wants the developer support. When Java code is built, it is “compiled” into a so called Byte-code instead of the native machine language of the targeted hardware. In the case of Android the Byte-code is compiled by the Dalvik virtual machine at Android Runtime to execute the instructions on the CPU. This is what gives Java its portability to run “anywhere”. This Java’s “gift” needs computing time for compiling the Byte-code into the hardware’s machine language. This advantage to run “anywhere” has the disadvantage in speed reducing. For better understanding, look up the figure in 1.1 System architecture subchapters.

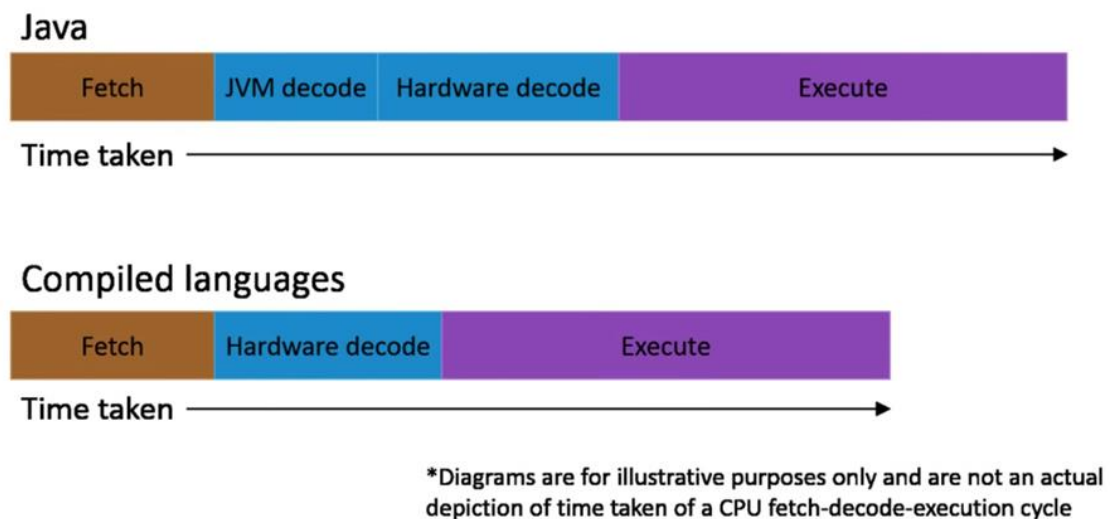


Figure 5.1: Demonstration diagram to compare taken time

In case of computer vision when we are doing masses of computing every bit of the computer performance is needed. So it was necessary to bypass the Dalvik VM layer and run the program directly on the kernel. Google supports this with the Android NDK to compile C and C++ code on the Android kernel. OpenCV comes with Java wrappers so it can be used as a Java code, but to run it the source code which is programmed in C and C++ needs to be compiled first.

For demonstration how I put all parts together I made a figure shown below.

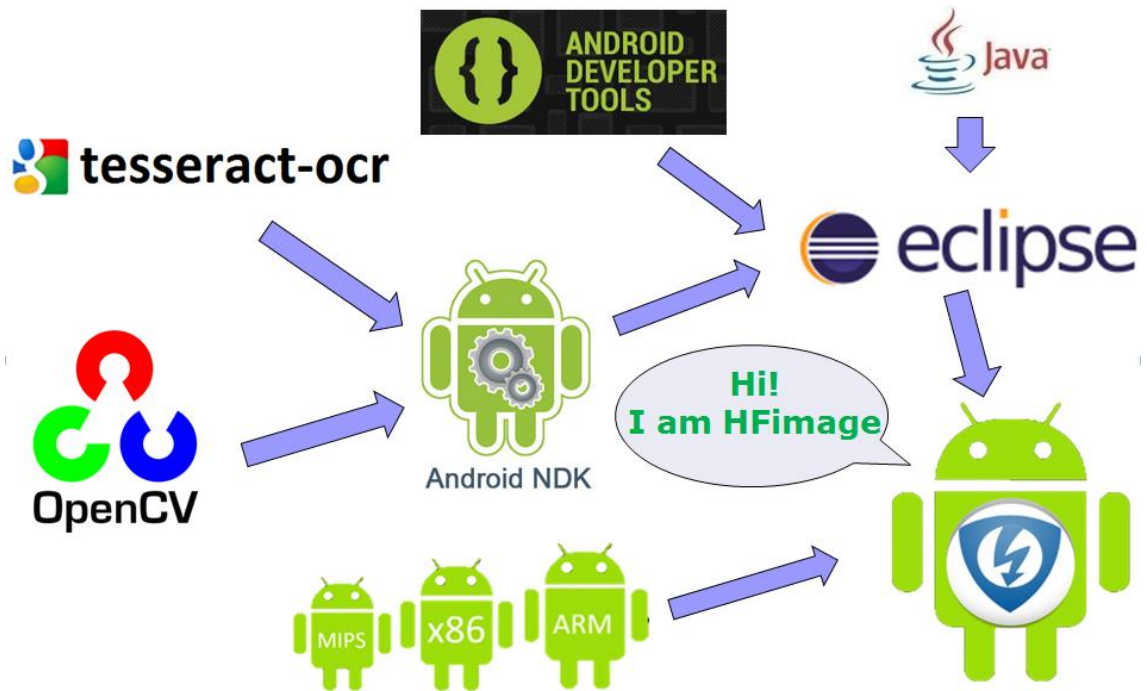


Figure 5.2: Binding parts together

After the development environment was installed and all the tools, plugins and libraries were implemented within it and was this all was tested the application's code writing part began.

My workflow began by creating a new android application project in Eclipse (File > New > Android Application Project). I chose to support systems versions from Android 2.3 to the latest version 4.4 (2014). It included approximately 95% of the Android system market. To test my first steps I also created a similar virtual android device as my smartphone by clicking on the icon *Android Virtual Device Manager* . As I progressed in my project I tried the application on some real Adnroid devices too. It was nessessary when the real time camera scene analizing was tested.



Figure 5.3: Android emulator in Eclipse platform

5.1 Naming and creating an icon for the application

To make it easy to find my application among the applications I choose a well-known icon among our students. The application is called “HFimage”. So if you installed it look for it under this information.



Figure 5.4: HFimage application icon

5.2 Using the camera

In my application I use the built-in camera in the Android device. To ensure that devices with no camera cannot find the application I added to the *Manifest.xml* file:

```
<uses-feature android:name="android.hardware.CAMERA"/>
```

Added permission:

```
<uses-permission android:name="android.permission.CAMERA"/>
```

At the beginning the application’s icon will be shown in the main activity’s image view. The camera can be loaded by pushing the Photo taking menu item. It will call the camera provider and show it if it is accessible. When its camera is accessible the screen shown us what is acquired from the camera, after pushing the photo item image capturing action will be executed. After the acceptance of the taken picture it will be streamed and stored in a bitmap and shown in main activity’s image view part. The built-in gyroscope should send information how the picture was taken and the picture should be automatically rotated and shown in the correct form.

The other using for the camera is while the Camera activity is on. It shows on the screen the information from the camera’s stream. In the menu can be set one of the various acquiring effects or choose the evaluating method.

The third camera using is to access camera’s flash light and use it as a flashlight. The user can choose whether to turn it on or off.

5.3 Load and save

The main task is to create a folder for the application and for that it is searched in the device's external storage. It creates a directory and saves its data files. For other use of these functions can be used by pushing the menu items *Save picture/text* and *Load picture*. This serves as a request for the application to do the correct task. While saving and loading is necessary took care of proper memory accessing and storage accessing. I solved it with a buffered stream. Buffered input and output stream ensured prevention from memory collapses, without it sometimes unfortunately crashed the application when it was working with large sized images.

To save files it is needed to add permission in the *Manifest.xml* file:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

To load files it is needed to add:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

5.4 Get actual location using the location manager

To enable to the application location services it was necessary to add permission in the *Manifest.xml* file:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

The activity for location implements *LocationListener* and uses *LocationManager* methods. *LocationListener* is used for receiving notifications from the *LocationManager* when the location has changed. These methods are called if the *LocationListener* has been registered with the location manager service using the request update methods.

LocationManager class provides access to the system location services. These services allow applications to obtain periodic updates of the device's geographical location, or to fire an application-specified Intent when the device enters the proximity of a given geographical location. To request for location updates is necessary to register for location updates using the named provider, and a pending intent. Parameters:

- Provider: the name of the provider with which to register.
- MinTime: minimum time interval between location updates.
- MinDistance: minimum distance between location updates.
- LocationListener: whose *LocationListener.onLocationChanged* method will be called for each location update

The user can choose from which provider they want to acquire updates.

5.5 Binding pieces together

In the end there were 7 activities and with different layouts, 3 different menus, various Java, Android and 2 external imported libraries. All seven activities has they own classes, private and public methods, icons and string resources. The layouts and menus are written in Extensible Markup Language (XML). Every activity calls the proper items and set their content view and if it is necessary it creates an options menu so I had to override their onCreate methods. Of course I had to override plenty of other methods to ensure fluent data sharing among the activities, to create references for Java builder, add permissions and so on. Data transfer and data sharing is ensured with intents and public static variables. To enable methods for other activities some methods are public too, it was necessary while developing and testing functionality.

At the end an *.apk file is crated and named *HFimage.apk* what is the install pack for my application for Android system.

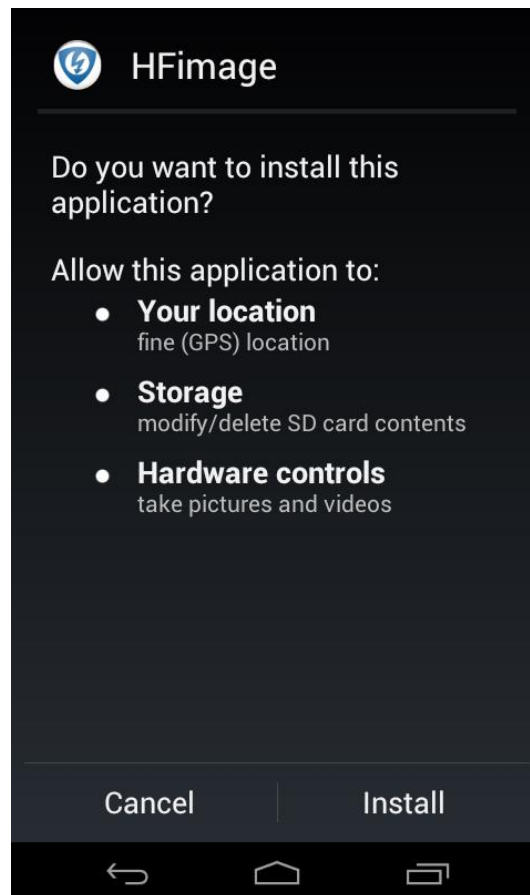


Figure 5.5: HFimage application installation

6 HFIMAGE APPLICATION

This chapter summarize and explains the final application and its functionality. After the “HFimage.apk” is installed it makes a runnable icon among the other apps. On starting the application it first tries to connect the OpenCV Manager application, when it is missing it toasts to install the latest version from Google play. After successfully accessing the OpenCV Manager the HFimage starts, the implemented OpenCV libraries should be compatible with all ARMv7 processors. After running first it searches the accessibility of the device’s external storage and looks for the application’s directory. If the “HFimage” directory cannot be found the application creates a new directory and names it after itself. If HFimage directory exists but if there are missing some subdirectories it creates them. There are two subdirectories “data” and “Mytexts”. In data is located the Tesseract tessdata language trained data pack if its missing the application creates it. Mytexts is created if the application OCR activity is called and here is the location where it saves the evaluated text files if necessary. The saved photos are stored in the main HFimage directory. After all this is done the main activity will be shows.

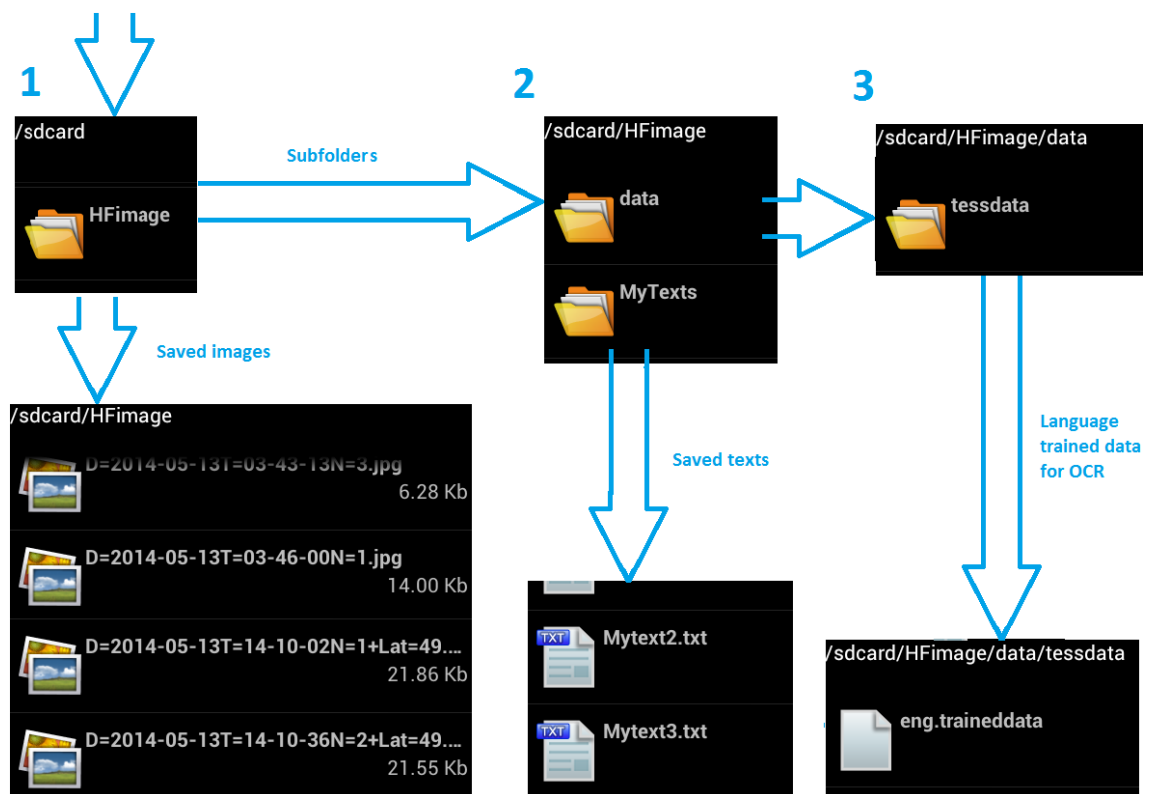


Figure 6.1: HFImage application’s hierarchy

6.1 Main activity

After a successful initialization the Main activity is shown. This activity serves as a global window where all other activities are accessible. The main window serves as the part that takes care for image evaluation from files. On the main screen are found all necessary objects to enable this. All shown items and functions are explained and shown below.

An image-view item shows which picture was loaded to the memory. A text-view item is to show temporary evaluation answers and for acquired text information.

There are two push buttons **Find plate** and **Evaluate**. To use them is necessary to load an image first. If no image is loaded they pop out a toast that there is no image to evaluate. How they work is in detail explained in its subchapter.

The menu can be found it right top corner of the main window. It contains all the items to call the other available activities and functions for image loading and image saving. It also contains the exit function. After pushing menu items they stats the intents they predetermined to do. The **menu** items are:

- Photo taking
- Save picture
- Load picture
- Get location
- Camera
- OCR simple
- About
- Exit & Close

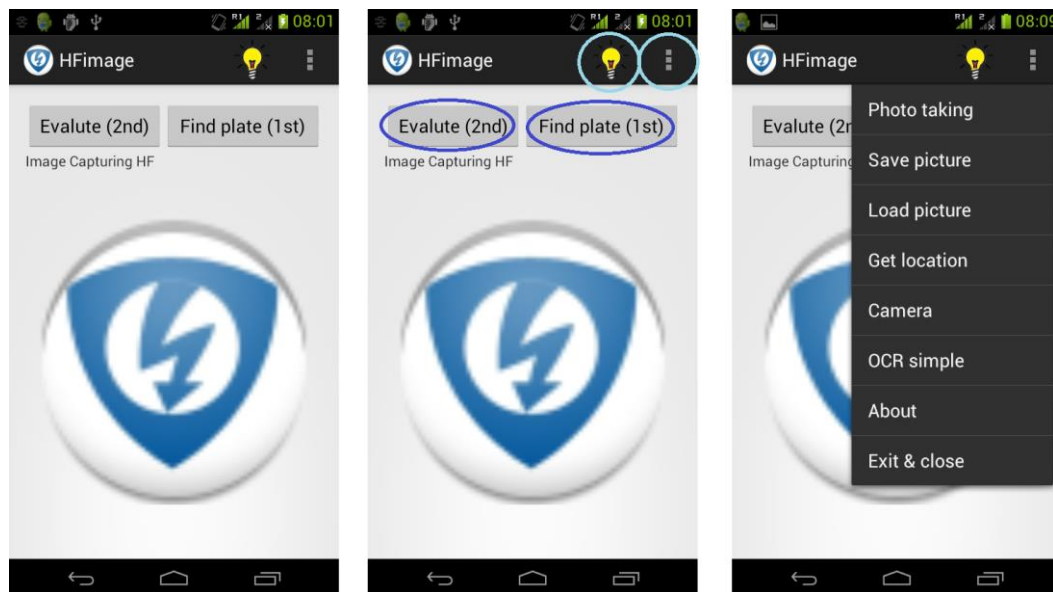


Figure 6.2: HFImage application's main window

Lights

If there is room for menu items the Lights menu item shows itself as a **light bulb**. It serves as intent to **lights activity**. By pushing it automatically calls the lights activity and turn on the camera lights as a torch. The activity itself enables to turn on and off the camera flashlights. By ending the lights activity the lights are turned off and the camera is released to use it by another action. This function is just a practical plug-in in the application. This was my first successfully attempt to commutate with the camera. It can be useful sometimes when you need to see in the dark.

Photo taking

Photo taking menu item calls a standard Intent action that can be sent to have the camera application capture an image and return it. The caller may pass an extra EXTRA_OUTPUT to control where this image will be written. If the EXTRA_OUTPUT is not present, then a small sized image is returned as a Bitmap object in the extra field. This is useful for applications that only need a small image. If the EXTRA_OUTPUT is present, then the full-sized image will be written to the Uri value of EXTRA_OUTPUT. So if no picture was taken and gone back from camera photo taking mode nothing happens, but when photo was taken and the user agreed with it, the image is buffered and shown in the main window's image view.

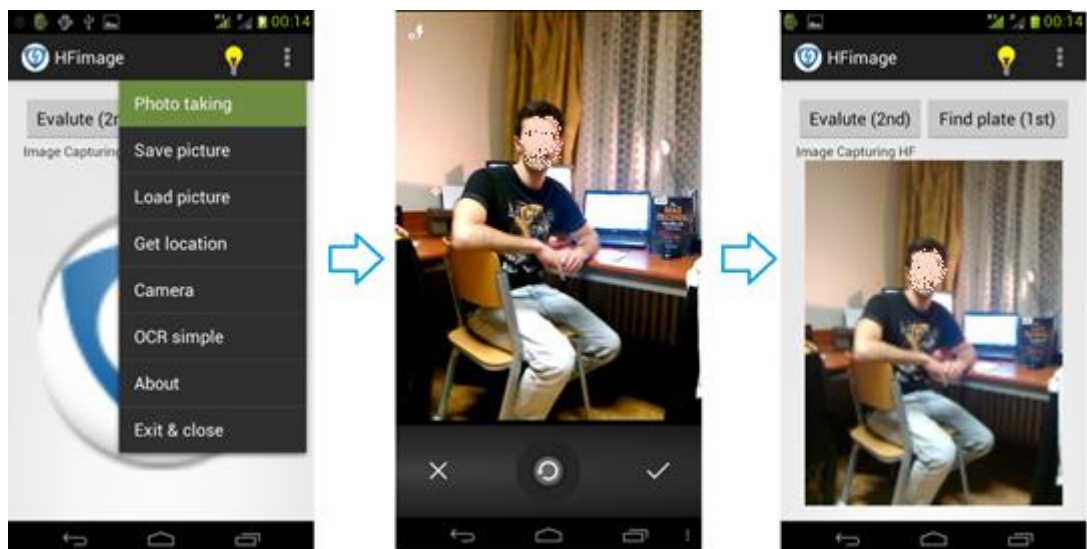


Figure 6.3: HFImage photo taking progress

Load

To filter the images, taken photos and pictures among the tremendous amount of files the Media provider declaration is called. This declaration contains meta data for all available media on both internal and external storage devices, the images part of meta data contains all available images. The user can choose how to complete the action using de pre-set gallery provider or other application. After the uses chooses the image the cursors path resolved and get the content resolved to acquire the file path. The whole process is shown on picture below.

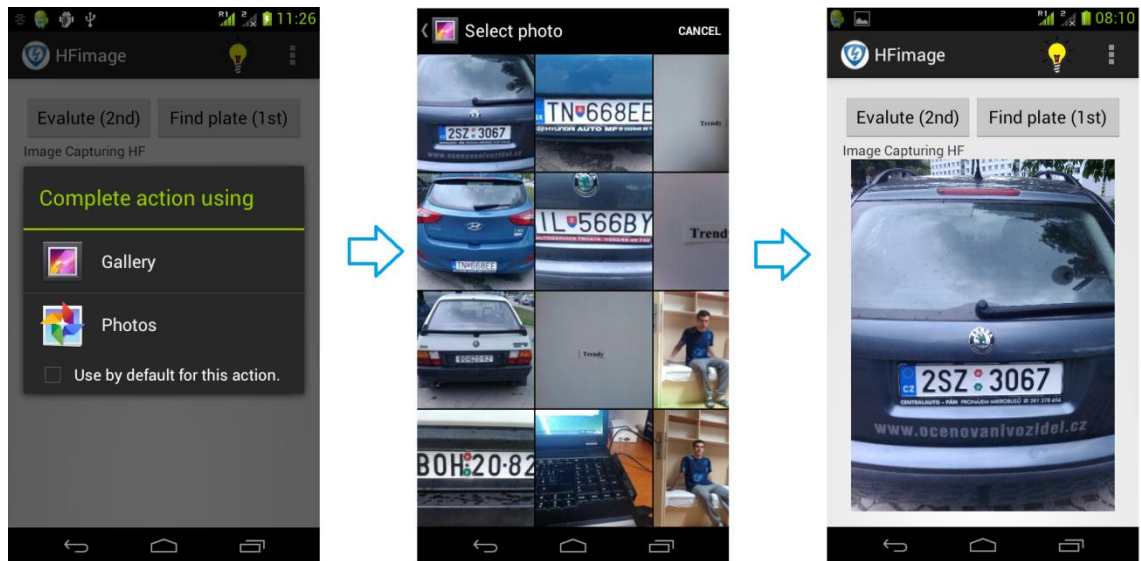


Figure 6.4: HFImage image loading progress

Save

The shown image in the main window's image-view is saved as flushed output stream within an Uri. The image is compressed to .jpg format with max quality available and saved to the HFImage root directory. If it cannot find the directory it creates it.

The created file is named after the calendar's actual date, actual time, an index number and other available data if there are found (location data). The index number represent the number how much saves have been provided from the application start.

"D=yyyy-MM-dd" + "T=HH-mm-ss" + "N=" + Index + "+" + recievedLocationData

For an example a full named file can be:

D=2014-05-13T=14-10-36N=2+Lat=49.23113815Lon=16.57100772.jpg

Get location

Get location menu item calls intent to get to the location activity. This activity is explained in its subchapter.

Camera

Cameramenu item calls intent to get to the Camera activity. This activity is explained in its subchapter.

OCR simple

OCR simple menu item calls intent to get to the OCR simple activity. This activity is explained in its subchapter.

About

About menu item calls an intent to get to the About activity. This activity is explained in its subchapter.

Exit & close

Pushing the Exit & close menu item a pop-up window will appear. This window asks the user if they are sure to exit the application and the choice remains on the user whether or not to exit the application. There is another possibly to exit the application pushing the system back button. Both methods calls the `onDestroy()` method what should take care the memory garbage the application created.

6.1.1 Find licence plate on image

So after the image is loaded it can be evaluated in few steps. Every type is explained with step by step instructions and shown with a figure example.

There is the “two step” method which does not need to find the licence plate, because the licence plate is the picture’s only content. So the user needs just push the evaluate button and the recognized characters will be shown in the text view part.

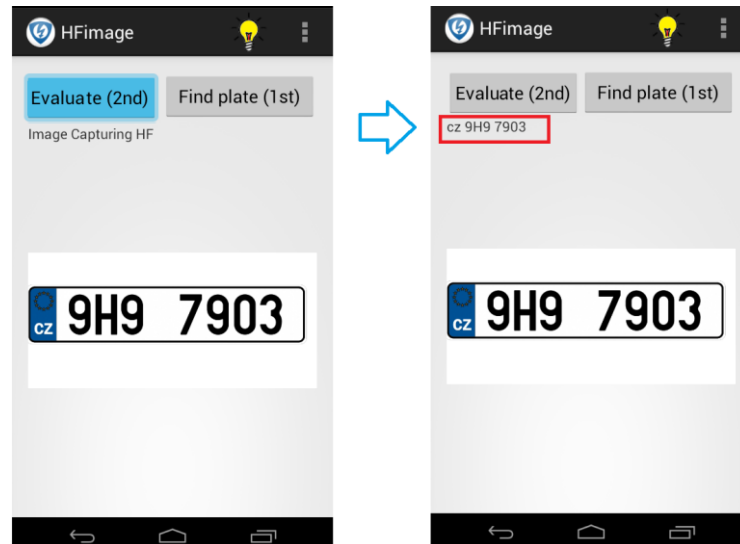


Figure 6.5: HFimage two step method

The “three step” method is that type which needs to find the licence plate first. The user has to push the find plate button. If one piece of licence plate is found during the evaluation it will be cut out from the original image and shown in the image view part. The next step is to push the evaluate button and the recognized characters will be shown in the text view part.

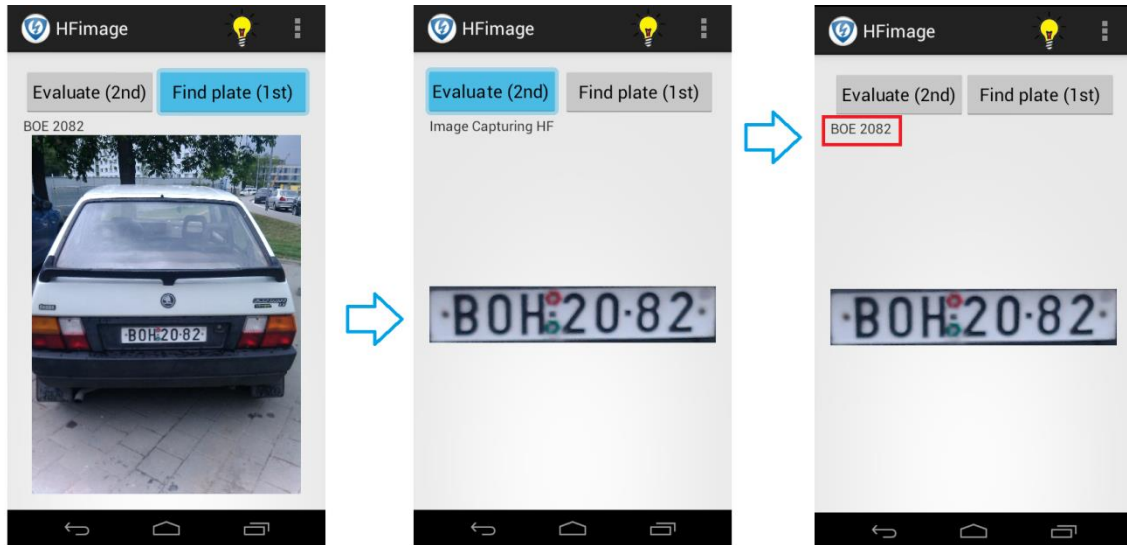


Figure 6.6: HFImage three step method

The “four step” method is that type which needs also find the licence plate first. The user has to push the find plate button. In case when more than one licence plate is found during the evaluation it will call the plate activity to offer the user which one should be cut out from the original image and shown in the image view part. The user can look up for the proper one by using navigation buttons and be clear with the shown information. The next step is to push the evaluate button and the recognized characters will be shown in the text view part.

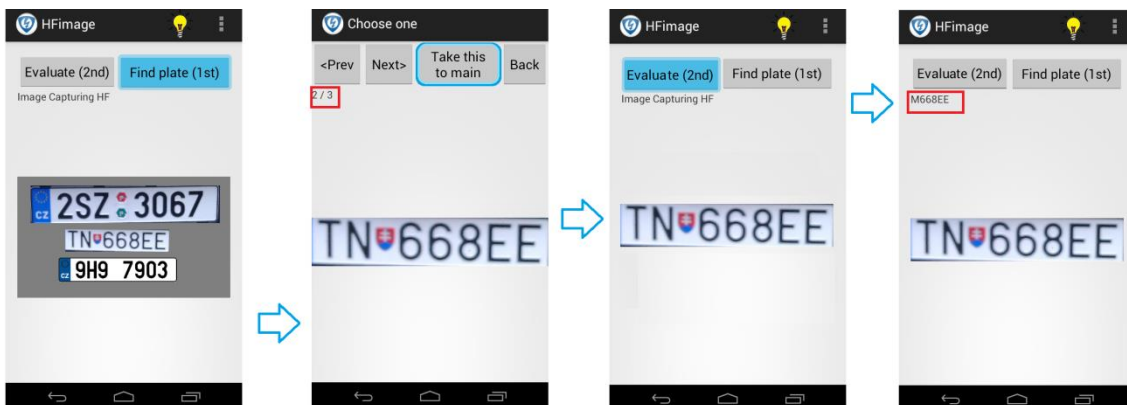


Figure 6.7: HFImage four step method

6.2 OCR simple

OCR simple activity allows loading images, evaluate them, writing texts and saving texts. The loaded images are shown in the image view field. If loaded images contain text the activity can extract them to character field located under the image view. The extraction can be executed by pressing the “Recognize text from image” button. The extraction can be set to continuous mode. For example if the user wants to create continuous text from more images the continuous mode has to be turned on. The next extracted text will be shown under the text that was already shown in the text field. The user can edit the text simply by touching it the keyboard pop out to typewrite or edit. After the user is satisfied with the text they can save it. Saving process is accessed by the menu “Save text” item. The file will be saved in the HFimage “MyTexts” subdirectory. If the activity cannot find the directory it creates it.

So the text is saved, but it still shown in the text field. The user does not have to delete it. Simply by set continuous mode to off the next extraction will clear the text field and show the new text. Settings menu item should serve for choosing various languages, which will be allowed in the next version of HFimage.

Going back to the main window can be done by pushing system back button or accessing it by the menu item “Back to main”. Both way executes the onDestroy() method for the activity.

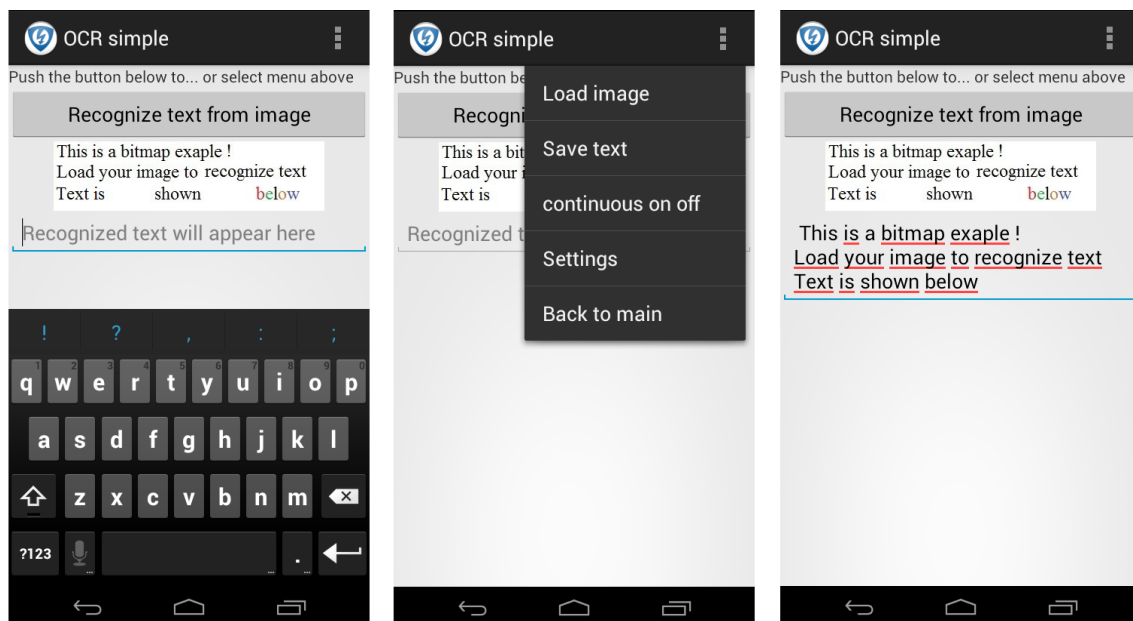


Figure 6.8: HFimage OCR simple activity

6.3 Get location

This part serves for finding the location of the user, to authenticate where was the picture taken. Together with the date and time information should serve as an information of the photo when and where was it taken. So if there are any available location data it can be taken back and set as an add-on in the naming process of the image. For example: if there is a car parking in a wrong place where it is not allowed to parking, the location data together with time plus date data and licence plate data should authenticate that the taken photo describes the reality and the owner of the car should be penalized. First I wanted to create hash function to prevent data faking, but at the end I noticed even this localization activity is something beyond of the goal this thesis wanted to achieve.

The user can choose from which provider wants to acquire location data, there is a possibility to use GPS satellites or online network if it is supported. Passive providers can be other applications which can localize themselves and share location data with other applications. Because it take some time for communication synchronization of the provider after choosing a provider the data acquiring a leave the channel opened. It is interrupted after the application closes. When this activity is never called no channels are opened for providers and no location data are received. When the providers are turned down by the system the activity is just waiting for the right moment if something has changed. I did not wanted to pop up some windows to ask the user if he wants to turn it on, I actually hate those types of notifications. If the users want they should do it manually and the activity will respond to it.

The figure below shows how the activity reacts when there is available information or not. (The second picture was taken while I was going home from school in Brno by foot).

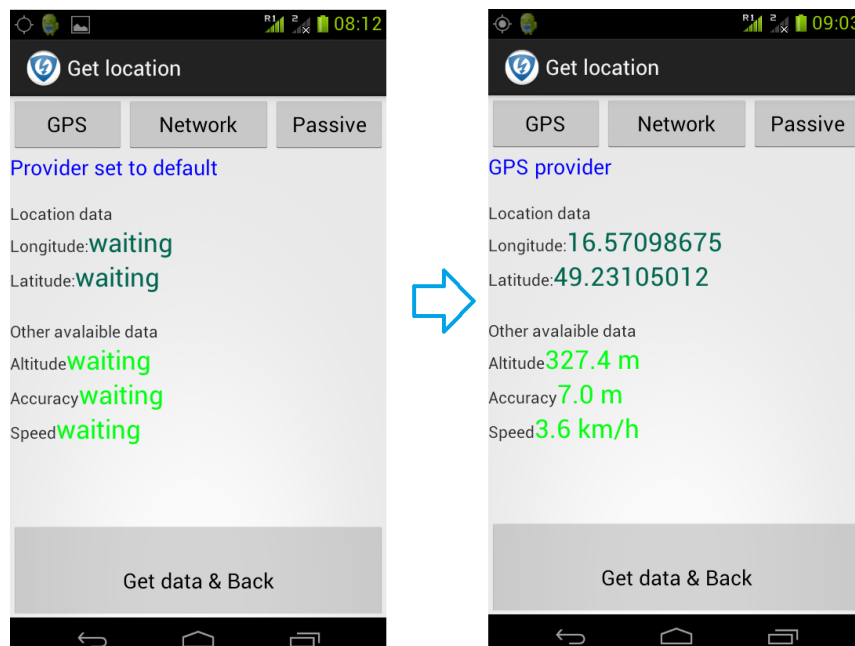


Figure 6.9: HFimage OCR simple activity

6.4 About activity

This activity serves as a reference for contact and added information where was application developed. It shows the name of the work's author and the supervisor. The user can choose for more languages such English, Czech and Slovak. Because long text is shown a vertical slider helps the user to scroll the shown text. At the end is located the back button which takes the user back to the main window. It looks like the picture below:

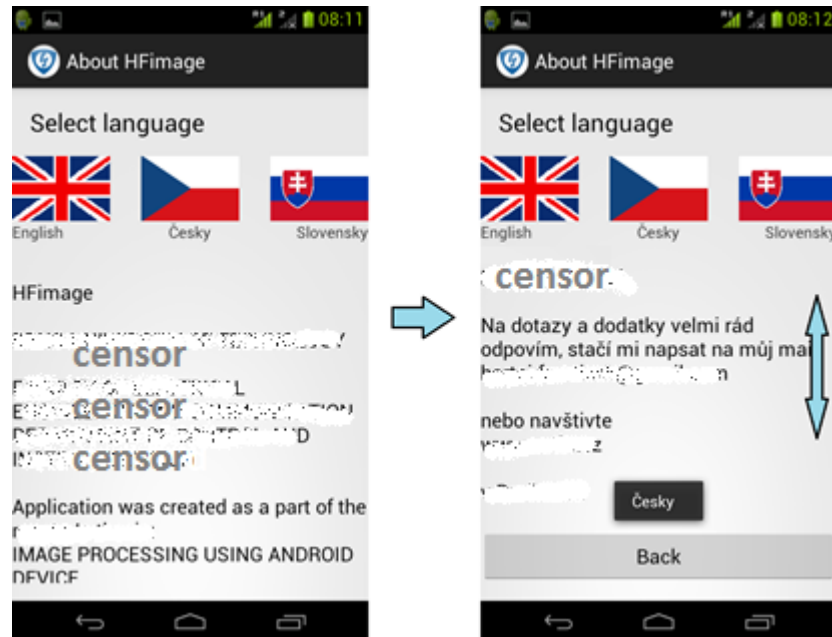


Figure 6.10:About HFImage activity

6.5 Camera activity

The camera activity's layout orientation is set to horizontal. The layout includes an OpenCV Java camera view where the camera stream is shown as frames. It also shows the camera's resolution and frames per second.

To use the prepared functions a menu is created. Two menu items are shown in the activity header other can be accessed by pushing the right top corner. By pushing it a scroll menu will appear. It is up to the user which they want to use. The NORMAL RGB mode is set as a default mode where the coloured camera stream is shown in the camera view. The default and built-in modes are shown in appendix of this thesis on the figures pictures.

The user can go back to the main window by pushing the prebuilt back button or can be choose "Back" menu item.

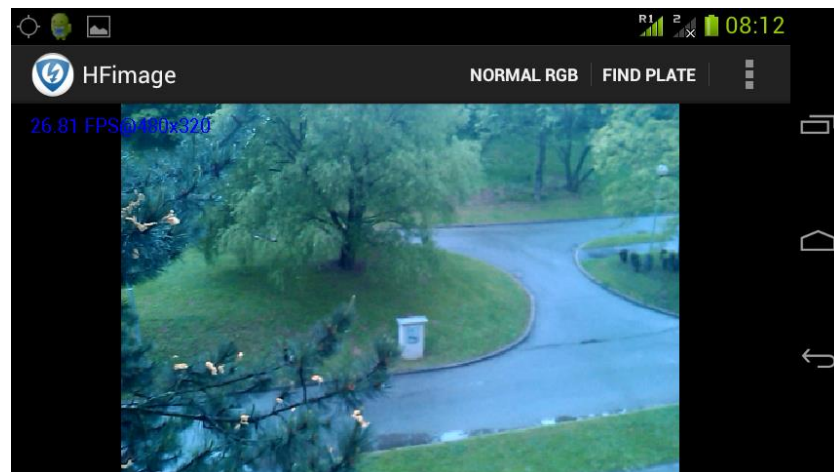


Figure 6.11: Camera activity default preview

6.5.1 Camera activity's find plate mode

The communication with the camera is enabled and it is functioning properly. The next step is to implement the evaluating algorithms to find the correct rectangles on the camera frames. This is included to the activity's find plate mode. The acquired frame is copied into a matrix on which the evaluating algorithm is executed. The suitable rectangles are excepted as the license plate coordinates on the frame and drawn into the original camera frame what is shown to the user by the camera view on the display. If the algorithm found more suitable rectangles all of them are drawn.

It has been tested for various distances; the proper distance when it is functioning correctly is from 0.5 to 5 meters in daylight. When greater tolerance is allowed it can evaluate even licence plates within a rotation tolerance, but the false evaluations number grows. To show which types of license plates can be evaluated a picture is also shown with various license plate in it. To show these operations more examples are shown below:

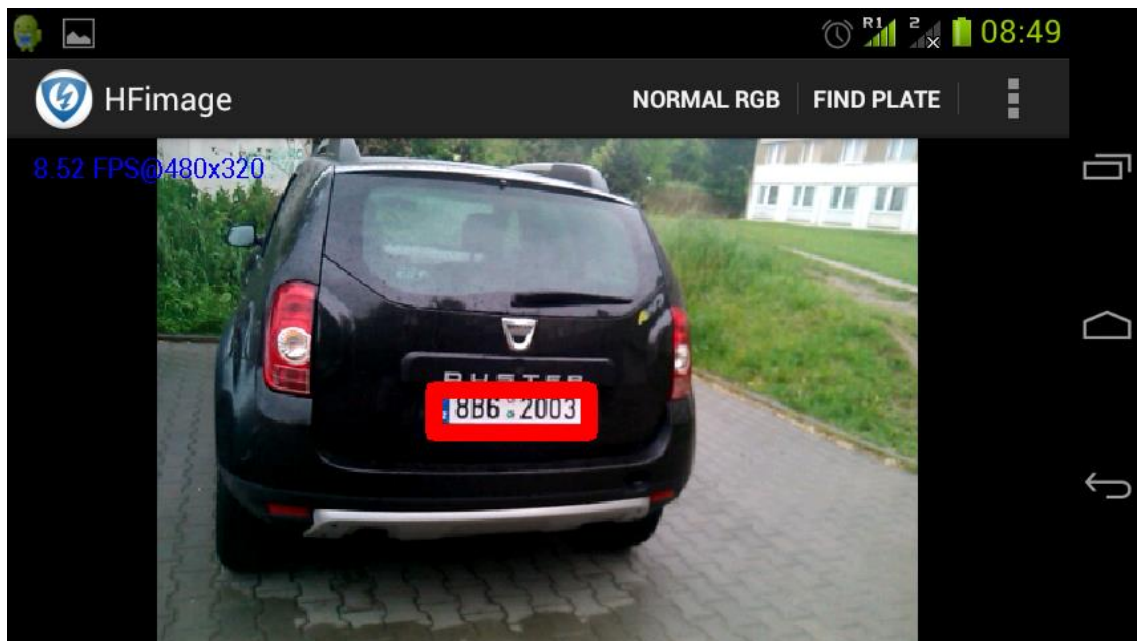


Figure 6.12: Find plate mode from greater distance

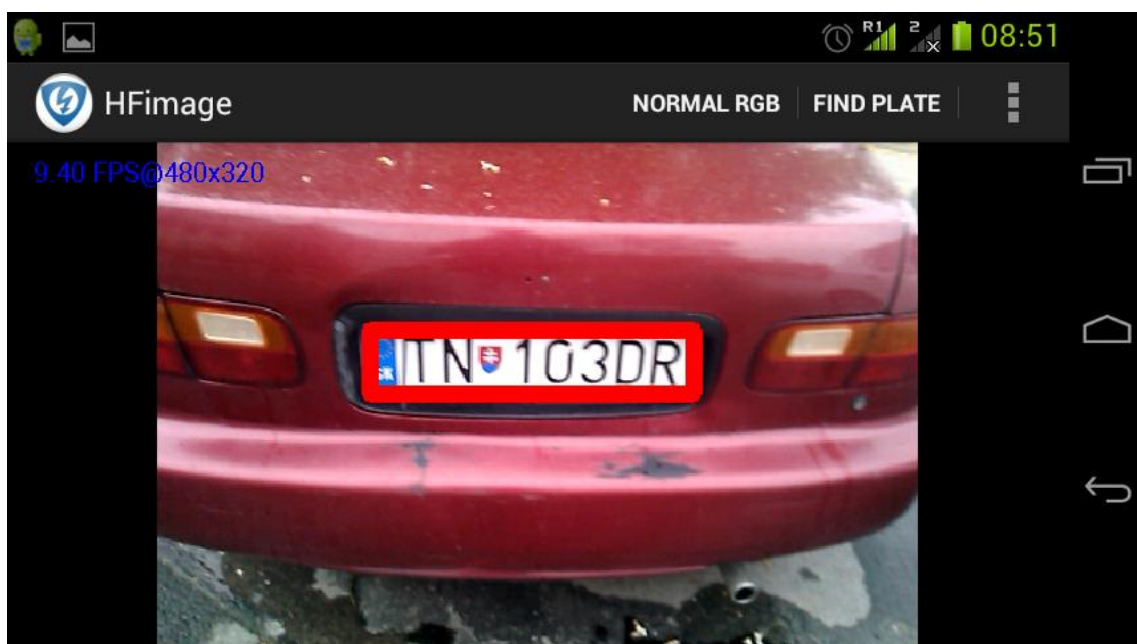


Figure 6.13: Find plate mode from closer

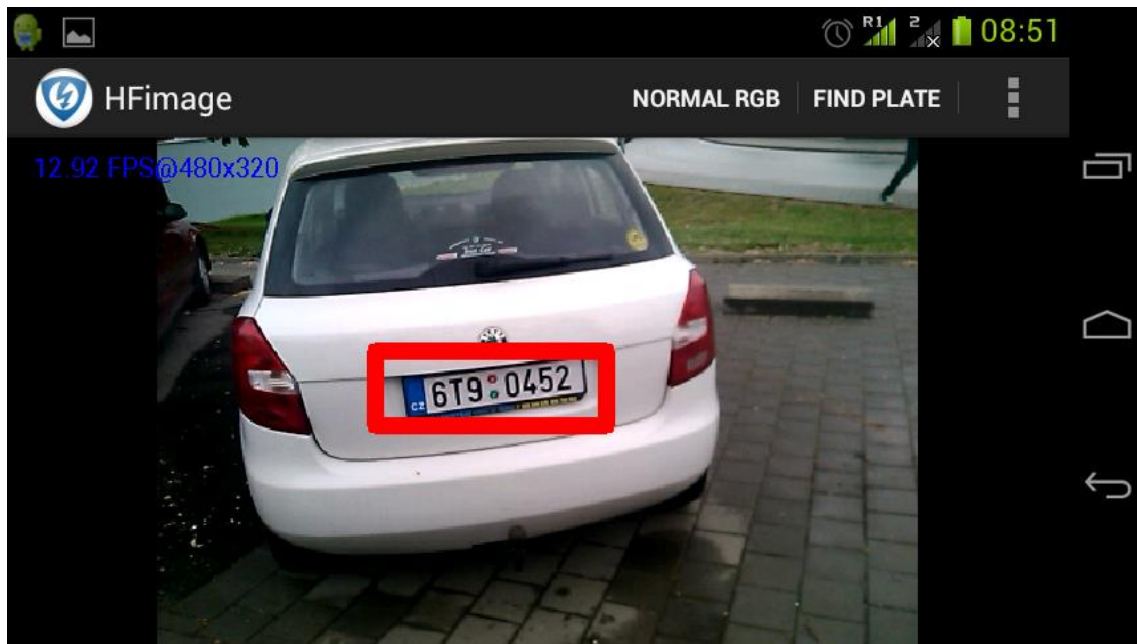


Figure 6.14: Find plate mode set to greater tolerance

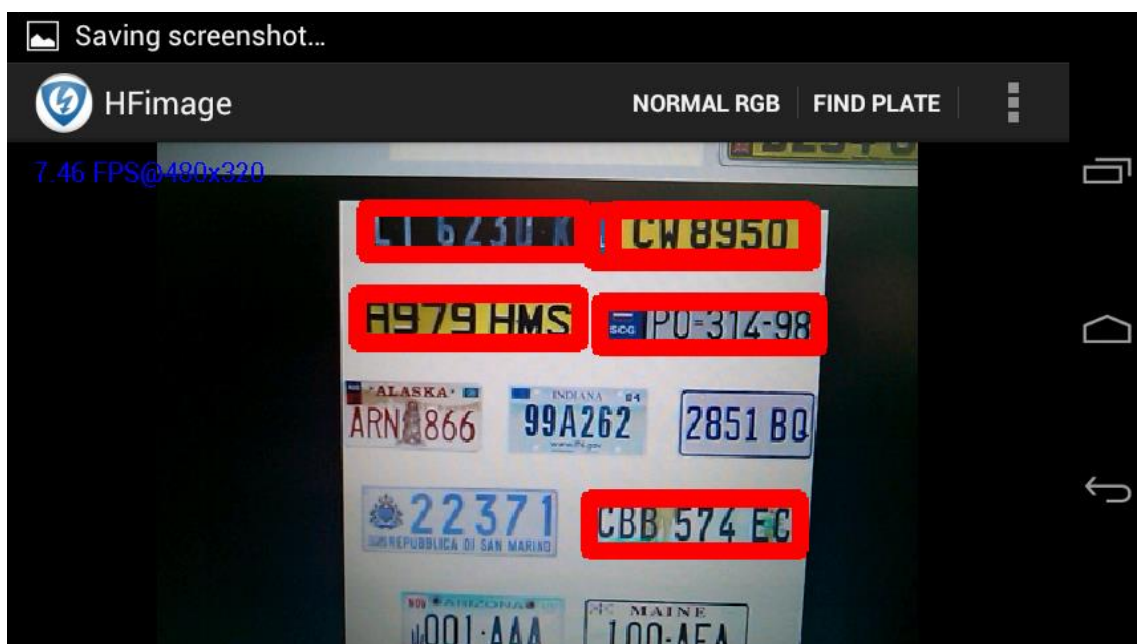


Figure 6.15: Camera activity has found more plates

7 DATA SECURITY

To ensure data security when sending pictures, user authentication and safe data sharing cryptology and cryptography methods are used in order to ensure "safe" communication on a network or to defend software assets against misuse or unauthorized access, if the attacker somehow managed to accede to these assets.

7.1 Confidentiality

Confidentiality of a message generally using cryptography is provided via cryptographic transformation. Transforming the messages (encryption) is performed by a secrecy cryptosystem. Encryptor (E) encrypts the entering message with the encryption key K_E and so the message in which the encoded information is transforms into a cryptogram C via the encryption process. This cryptogram C is transmitted through the transmission channel. Inversion of the encrypting transformation is the decrypting transformation, which is done by the decryptor (D). Decipher using the decryption key K_D the cryptogram transforms into the original message. This process ensures secure transmission of information via the transmission medium.

For better understanding see the image below:

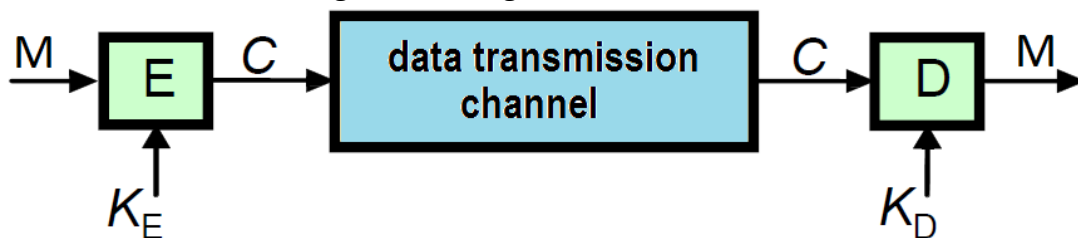


Figure 7.1: Block diagram of the cryptosystem for message confidentiality

In the application there are two possibilities: one is the encryption of the recognized data (licence plate number and characters plus the other possible information). The second possibility is to keep the useful part of the recognized pattern (snipped licence plate picture from the original photo or video stream) and encrypt this information. The encryptor (E) and the decryptor (D) is set in the application. The encryption key (K_E) is stored encrypted in a hidden file among the other data. If it is missing an error report is generated and showed.

An attacker with the proposed option level could not be capable of solving the cryptogram C without knowing the cryptographic key K_D in the specified period of resistance cryptogram.

7.2 Authenticity of messages

Message in which is encoded the information enters into the cryptosystem and via using a secret private key K_{private} creates a signature (S) on the given message. This signature is attached to the message and then is transmitted through the data transmission channel. Recipient of the message can use the attached signature by the originator of the message to verify the origin and authenticity of this message. The process of verifying the authenticity of messages is done using two inputs the message and public key K_{public} .

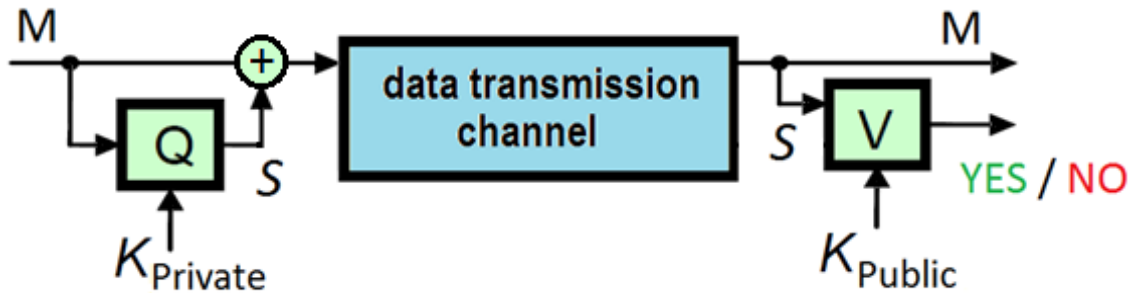


Figure 7.2: Block diagram of the cryptosystem for message authentication

The authenticity is created by the picture data (see similar data analysis in camera mode MODE_HIST among appendices) and the other gathered data (such GPS location, time and date, etc.) this all is put into hash function with the users key and an output is created. This output is the authenticity information which is stored with the other data in the picture metadata in the image file (Exif - Exchangeable image file format for JPEG). The hash function is set in the application. The private key is stored encrypted in a hidden file among the other data of the application. If it is missing an error report is generated and showed.

7.3 Authenticity of users

The users can authenticate themselves with PIN or passwords which are asked when the app starts. This user-authentication is optional for saving metadata to pictures and can be used for user authentication issues. The PIN or password is encrypted in a hidden file among the other data of the application. If it is missing an error report is generated and showed. If the hidden file is missing or "no user" can be authenticated the application runs with the possibility to create a new user but it is not necessary to define one.

7.4 Security summary

All these parts are optional and can be combined to ensure fully data security if necessary. The key and password distribution can be hidden from the user in an encrypted file or allow to set by the user themselves.

CONCLUSION

The smartphone devices sales is growing year after year. Users replace their older types of devices with new versions released by the device manufacturers, so the older devices remain unused and often end up as scrap, although in terms of functionality they have no defect. The main aim of the article is to propose a “technology recycling” possibility, a way for reusing unused smartphone devices. For this issue is created a possible solution, a design and implementation of low-cost data mining application for smartphones. Computing power of smartphones is sufficient for computer vision methods, such as optical pattern recognition and character recognition in images and in camera stream, which briefly explains the theoretical part. This article contains a specific proposal for data mining application for license plate detection and recognition from image and from real time camera input. It can be used for automatic vehicles detection operations as an information mining process (customers origin, the frequency of visits) and also to verify the vehicle (access control, check whether the car is stolen, etc.). To ensure data safety of recognized information and additional data from the device (date, time and GPS position, etc.) the possible software security techniques are explained (encryption methods, the possibility of using steganography in images). To show the whole concept and functioning a smartphone application was created on Android platform, which is the most widespread operating system among smartphone devices. Application and software is based as an open source core project and can be set further for specified commercial purposes used in practice or for academic and teaching activities.

In the practical part I have written my workflow, as I progressed in my solution. I described my approach in design of GUI and the basic functionality of image processing for Android. Then the communication with the camera is explained, how I accomplished saving and loading data from storage. This served as the basis for the application where the algorithms were implemented for image processing and image evaluation. The used algorithm is explained step by step and shown with pictures as the evaluation proceeds. There is flowchart too to get the whole picture.

Product of this thesis is a functioning application that allows to its user to capture images and analyse video stream. The algorithm evaluates the entering data and can show the location of the license plate. The application also allows recognizing texts and numbers from images. There are other various practical features and options implemented within the application. For example GPS location and data of the taken picture or using camera for other options. To see more of its functions please see the appendices, there are hyperlinks to online demonstration videos how HFimage works in real time.

Of course there can be done more. Set more precisely parameters of evaluating, to try other evaluating methods and compare them with themselves. To enable language selections for other counties, to evaluate the characters of cars registration number of other countries too. This work represents mainly a universal application core, what can

be further modified depending to a specific issue or requests. The final product can be modified for further purposes. For example to modify the application for automatic car permission controlling (parking allowance), communicate with databases (stolen car controlling), data mining and so on.

References

- [1] SHAPIRO LINDA, G. Computer vision. Vyd. 1. New Jersey: Prentice-Hall, 2001, 580 s. ISBN 01-303-0796-3.
- [2] FORSYTH, David a Jean PONCE. Computer vision: a modern approach. London: Prentice Hall, c2003, xxv, 693 p. ISBN 01-308-5198-1.
- [3] Ben Elgin: Google Buys Android for Its Mobile Arsenal [online]. Edited 2005-08-17 [cit. 2014.01.02]. Available from URL: <<http://www.webcitation.org/5wk7sIvVb>>
- [4] Open handset alliance: Android Overview [online]. [cit. 2013.12.30]. Available from URL: http://www.openhandsetalliance.com/android_overview.html
- [5] Palo Alto.: Google's Android becomes the world's leading smart phone platform [online]. UK: 2011-01-31 [cit. 2014 01 03]. Available from URL: <<http://www.canalys.com/newsroom/google%E2%80%99s-android-becomes-world%E2%80%99s-leading-smart-phone-platform>>
- [6] Tony Bradley: Android Dominates Market Share, But Apple Makes All The Money [online]. Forbes 2014, aktualizováno 2013-11-15 [cit. 2014.01.02]. Available from URL: <<http://www.forbes.com/sites/tonybradley/2013/11/15/android-dominates-market-share-but-apple-makes-all-the-money/>>
- [7] Apache License, Version 2.0 [online]. 2010 [cit. 2013-12-27]. Available from URL: <<http://www.apache.org/licenses/LICENSE-2.0>>
- [8] ORACLE. Java SE 7 Features and Enhancements [online]. aktualizováno 2012-13-12 [cit. 2013 12 28]. Retrieved from URL: <http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html>
- [9] Tutorialspoint.: Android [online]. 2014 [cit. 2014 01 02]. Retrieved from URL: <http://www.tutorialspoint.com/android/>
- [10] JANOVSKEÝ, Martin. *Návrh a implementace knihovny pro číslicové zpracování signálů na platformě Android*: diploma thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications, 2010. 54 p.
- [11] Tutorialspoint: Android Application Components [online]. 2014 [cit. 2014 01 02]. Retrieved from: http://www.tutorialspoint.com/android/android_application_components.htm
- [12] Android developers: Android NDK [online]. 2014 [cit. 2014.01.03]. Available from URL: <http://developer.android.com/tools/sdk/ndk/index.html>
- [13] Android developers: USB Host and Accessory [online]. 2014 [cit. 2014.01.01]. Available at: <http://developer.android.com/guide/topics/connectivity/usb/index.html>
- [14] Android developers: Debugging [online]. 2014 [cit. 2014.01.03]. Retrieved from <http://developer.android.com/tools/debugging/index.html#tips>

- [15] VOGEL, Lars. Android application and activity lifecycle. [online]. [cit. 2014-05-17]. Available at:
<http://www.vogella.com/tutorials/AndroidLifeCycle/article.html>
- [16] Android developers: Activities [online]. 2014 [cit. 2014.05.13]. Available at:
<http://developer.android.com/guide/components/activities.html>
- [17] Java logo, [online]. 2014 [cit. 2014.01.03] available from official Java site:
<http://www.java.com/en/about/>
- [18] ORACLE. Oracle and Java [online]. 2012-13-13 [cit. 2013.12.28]. Available from URL: <http://www.oracle.com/us/technologies/java/overview/index.html>
- [19] ORACLE. Java SE 7 Features and Enhancements [online]. Updated 2012-13-12 [cit. 2013.12.28]. Available from URL:
<http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html>
- [20] Eclipse logo, [online]. 2014 [cit. 2014.01.03]. Retrieved from official eclipse site: <http://www.eclipse.org/artwork/>
- [21] OpenCV logo, [online]. 2014 [cit. 2014.05.03]) Retrieved from URL:
<http://code.opencv.org/projects/opencv/wiki/OpenCVLogo>.
<http://opencv.willowgarage.com/wiki/OpenCVLogo>
- [22] Abbyy. [online]. 2014. [cit. 2014.05.15]. Retrieved from URL:
<http://www.abbyy.com/mobileocr/>
- [23] Aspire. [online]. 2014. [cit. 2014.05.13].]. Retrieved from URL:
<http://asprise.com/home/>.
- [24] Google, [online]. 2014. [cit. 2014.05.13] available from site
<https://code.google.com/p/tesseract-ocr/>
- [25] Google, Tesseract OCR logo. [online]. 2014 [cit. 2014.05.13]. Retrieved from URL: <http://code.opencv.org/projects/opencv/wiki/OpenCVLogo>.
- [26] Wikipedia. Vehicle registration plate [online]. [cit. 2014.05.18]. Retrieved from URL: http://en.wikipedia.org/wiki/Vehicle_registration_plate

List of appendices

Appendix 1:

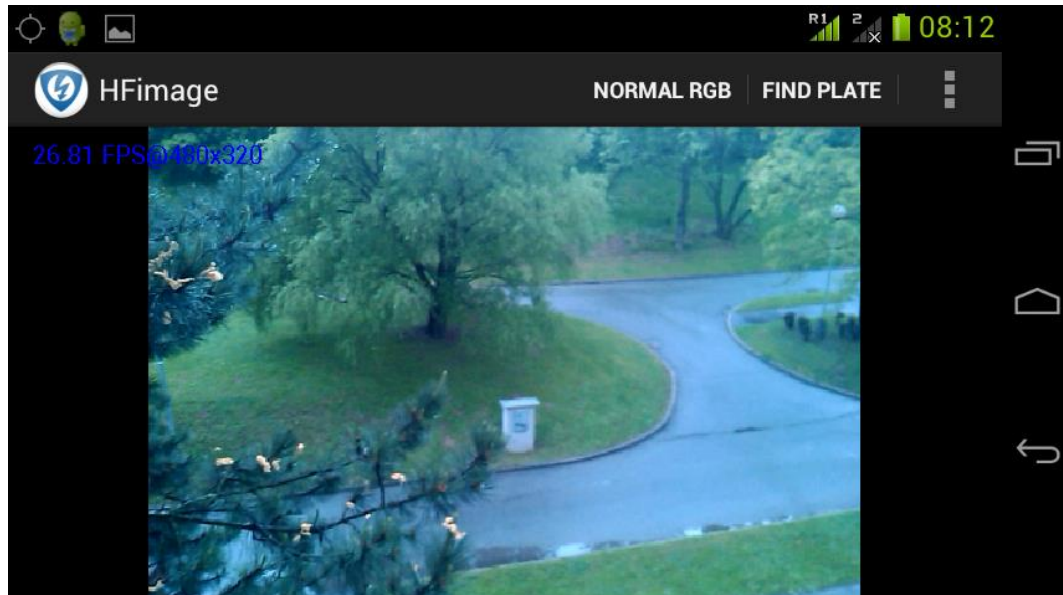
Camera activity's modes as snapshots
(other types of image evaluation with OpenCV).

Appendix 2:

Hyperlinks to videos how HFimage works.

Camera activity's modes as snapshots

NORMAL RGB



MODE_SEPIA



MODE_SOBEL



MODE_PIXELIZE



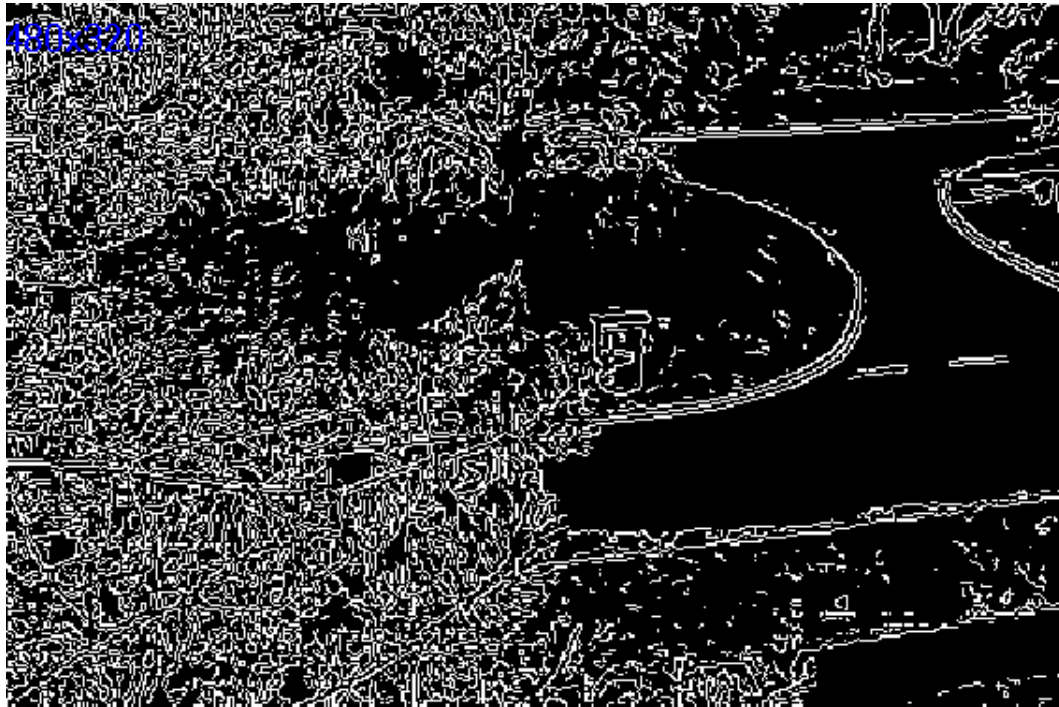
MODE_POSTERIZE



MODE_OBJECT_FIND



MODE_CANNY



MODE_CANNY2 (Blurred Canny)



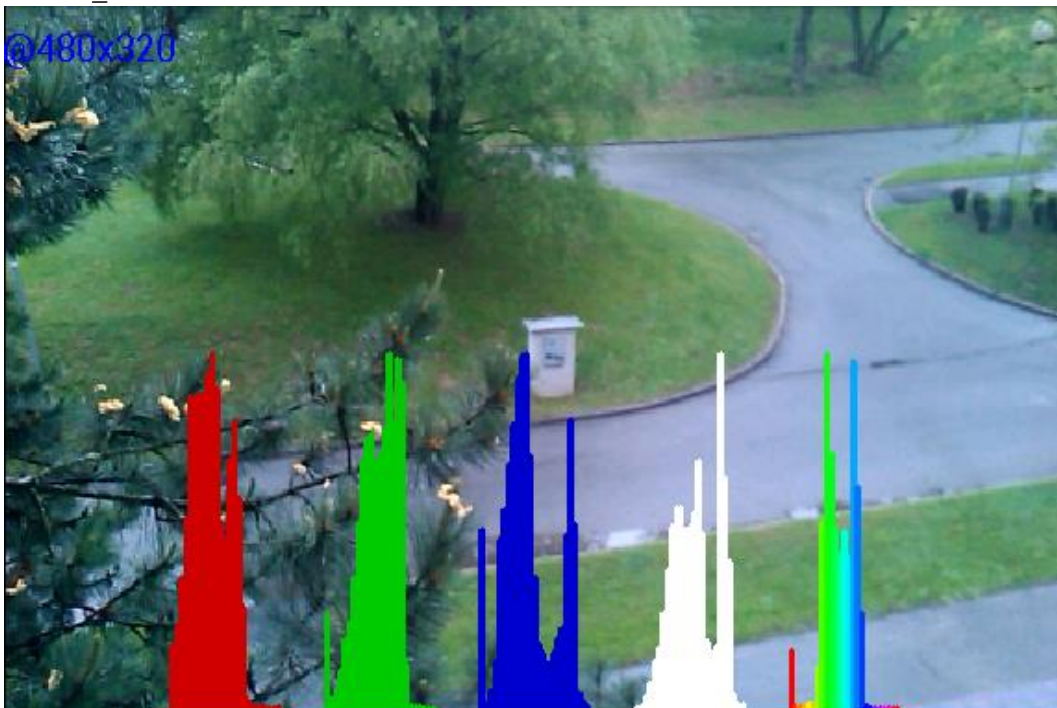
MODE_ZOOM

@480x320



MODE_HIST

@480x320



Hyperlinks to videos how HFimage works

These are hyperlinks to demonstrate the application. There are 4 demonstration videos uploaded to YouTube. The link to the videos is below they names. There is a playlist created to watch all parts together.

HFimage YouTube video description:

HFimage is an android image processing application to detect license plates, recognize characters and numbers from pictures or from camera frames. It also can locate the device by GPS coordinates and use the camera lights as a flashlight. Watch all video parts of HFimage android application too its functions. Thank you for watching!

Hfimage Android application – Part 1 – Using the application for license plate detection

<https://www.youtube.com/watch?v=8e6OCMBf94M>

Hfimage Android application – Part 2 – License plate recognition in a complex scene

<https://www.youtube.com/watch?v=Q2hpBypSCSQ>

Hfimage Android application – Part 3 – Get location data

<https://www.youtube.com/watch?v=22zknft3F80>

Hfimage Android application – Part 4 – Using Tesseract OCR

<https://www.youtube.com/watch?v=X0XpyyRbaBg>

All parts included in a YouTube playlist:

<https://www.youtube.com/watch?v=8e6OCMBf94M&list=PLk8CcQWNWiwH1WfI38ltdjrAOeXiBXCCd>