VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

AUGMENTED REALITY FOR MOBILE DEVICES FOR PRECISE URBAN NAVIGATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MICHAL MURÍN
AUTHOR

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZŠÍŘENÁ REALITA V MOBILECH PRO PŘESNOU NAVIGACI VE MĚSTĚ

AUGMENTED REALITY FOR MOBILE DEVICES FOR PRECISE URBAN NAVIGATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MICHAL MURÍN
AUTHOR

VEDOUCÍ PRÁCE Ing. VÍTĚZSLAV BERAN, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Diplomová práce je zaměřená na rozšířenou realitu a její realizaci pro mobilní platformu Android. V teoretické části je kladen důraz na obeznámení se s principy rozšířené reality, možnostmi její realizace pro mobilní platformu Android, která je také podrobněji představena. V praktické části se dále zaměřuje na návrh a realizaci knihovny a aplikace využívající tuhle knihovnu pro tvorbu rozšířené reality na platformě Android pomocí GPS a polohových senzorů. Aplikace slouží jako navigace a zobrazuje cestu ke konkrétnímu cíly, jak v podobě rozšířené reality, tak i na klasické dvourozměrné mapě. Práce podrobně popisuje navrhované řešení knihovny a aplikace a jejich implementaci. Nakonec popisuje testování, zhodnocuje dosažené výsledky a diskutuje nedostatky výsledného řešení a možnosti jeho vylepšení, respektive rozšíření.

Abstract

The Master's thesis is focused on the augmented reality and its implementation for the Android mobile platform. The theoretical part is aimed at acquaintance with the principles of the augmented reality and the possibilities of its realization for the Android platform, which is described in detail, too. The practical part is focused on the design and implementation of the library, and the application using this library for the development of location–based augmented reality for Android platform, using the GPS and positioning sensors. The application serves as a navigation and displays a route to a specific destination location, both in the form of augmented reality and the standard two–dimensional map. The thesis describes the designed solution of the library and application and their implementation in detail. Finally, the thesis describes the testing of the solution, reviews achieved results and describes limitations and shortcomings of the solution and possibilities of its future updates, extensions respectively.

Klíčová slova

Rozšířená realita, Virtuální realita, Google Android, Navigace, Akcelerometr, Magnetometr, Senzory, OpenGL ES

Keywords

Augmented reality, Virtual reality, Google Android, Navigation, Accelerometer, Magnetometer, Sensors, OpenGL ES

Citace

Michal Murín: Augmented Reality for Mobile Devices for Precise Urban Navigation, diplomová práce, Brno, FIT VUT v Brně, 2013

Augmented Reality for Mobile Devices for Precise Urban Navigation

Declaration

I declare that I have created this thesis myself under the supervision of Ing. Vítězslav Beran, PhD. I have cited all the bibliographic sources and publications used for the creation of this thesis.

. . . .
Michal Murín
May 22, 2013

Acknowledgement

I would like to thank my supervisor who provided me with useful advice, feedback and was willingness to help, thus contributing to the creation of this thesis. I would also like to thank my family and my girlfriend for their support.

Contents

Chapter 1

Introduction

We live in so–called mobile age, in which having a mobile device with a touch screen, either a smartphone or a tablet, is becoming absolutely common. Popularity of this kind of technology has escalated extremely quickly together with capabilities of hardware used in such devices. People use smartphones or tablets as alternatives to computers. Besides messaging and making calls, they check their e–mails, read morning news, watch movies, play games, etc. The hardware equipment offers really good performance and it is still only a fraction of its future possibilities.

A mobile application which could be only hardly replaced by a computer application, whether the computer is a laptop or a desktop, is navigation application. Navigation is available as a device specialized specifically in this problem or as a software application for mobile devices.

Another type of application for mobile devices, which is also becoming very popular, is an application using the augmented reality. The augmented reality represents a combination of reality and virtuality, i.e. the camera view is enhanced with virtual objects which are drawn over this view. This approach enables the user to use the mobile device as a window into real world, mixed with artificial elements which could mark nearby points of interest, represent objects placed in the space (e.g. to show what a living room would look like with a new sofa) or navigate the user to a specific location. The last mentioned case is the primary subject of this thesis.

This thesis is focused on the augmented reality and its possibilities to be realized as a part of an application for Android. It describes an approach suitable for such solution and its realization as a location–based augmented reality library and a demo application for Android, thus using GPS and positioning sensors. The application is meant to serve as a navigation for the user, using not only augmented reality, but a standard map too.

The following chapter discusses the virtual reality (briefly), the augmented reality, ways of displaying the augmented reality and possibilities of its usage in real life. The third chapter describes Google Android mobile operation system, its architecture and principles, and OpenGL ES API. The fourth and the fifth chapters describe the design of the navigation application, its realization in detail, testing and achieved results respectively. In conclusion, there is summarized the content of this thesis and achieved results.

Chapter 2

Virtual Reality and Augmented Reality

The way from real world to virtual reality, and vice versa, can be described with reality–virtuality continuum (Fig. 2.1). Virtual reality, where user is isolated from the real world, can be situated on one end. Real world is situated on the opposite end. In the middle there are augmented reality and augmented virtuality, combinations of real and virtual images. In augmented reality, most of the environment is the real world, and the rest consists of computer–generated virtual objects displayed over the real world layer. Augmented virtuality is an opposite to augmented reality. [12].

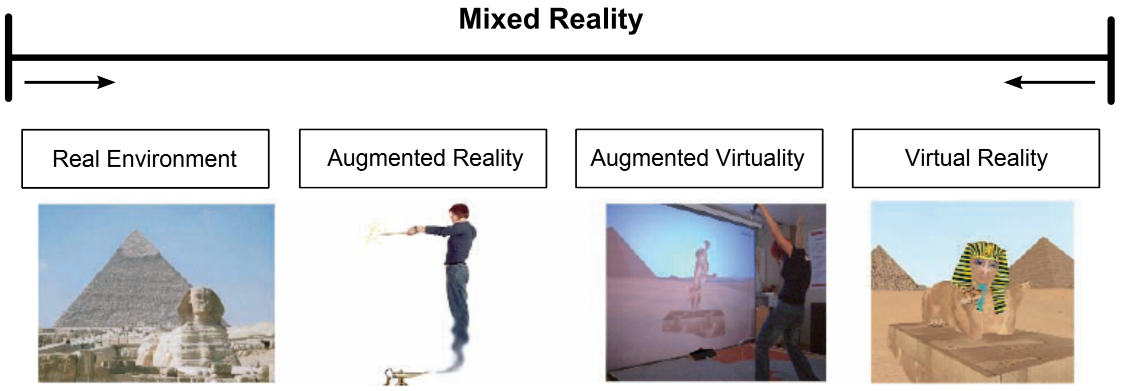


Figure 2.1: Reality–Virtuality Continuum [12].

2.1 Virtual Reality

Virtual reality is a 3D environment created by computers. User can navigate in this environment, i.e user can move around and explore this environment. User is also allowed to interact with virtual objects or manipulate them, e.g. pick up objects.

The main objective of virtual reality is to create such an illusion of being in an environment, that it can be perceived as a believable place with opportunities to interact with its objects. The most important aspect of reality, with respect to virtual reality, is the visual one. It is the dominant perceptual sense and the main provider of information about the world around for the most of the people. Since virtual reality simulates real world, hearing and feeling a touch are very important to support the experience of “reality”. There are two main factors that describe the virtual reality experience from the physical and psychological points of view – immersion and presence.

Virtual reality requires realtime graphics, a stereoscopic display, used to produce the illusion of 3D, and a tracking system to register head and hands motion. Commonly used technology includes HMD[1] (section 2.5.1) and stereoscopic glasses. Interaction can be achieved by using a tracking device, which may be integrated in the HMD itself, for tracking head and body movement and a “data glove” can be used to track hand movements. The glove lets the user point to and manipulate objects in the virtual scene.

Immersion is related to the physical configuration of the user interface of the virtual reality application. Depending on how much user can see, hear, touch the real world during the simulation, virtual reality systems can be classified as:

- **Fully immersive** – the first virtual reality systems using an HMD, the idea was to completely isolate the user from the real world.
- **Semi–immersive** – systems using large projection scenes
- **Nonimmersive** – desktop–based virtual reality systems (e.g. computer games) are very popular because of the good combination of interactivity, ease of use, appealing graphic and sounds. These factors can produce a high level of interest and isolation from the real world.

Presence is a subjective concept, associated with the psychology of the user. Presence is when the multimodal simulations (images, sound, haptic feedback, etc.) are processed by the brain and understood as a coherent environment in which some activities and interactions can be performed. Presence is, therefore, achieved when the user is conscious of being in a virtual environment.

2.2 Augmented reality

Augmented reality can be formally defined and classified in several ways. A question arises, whether augmented reality is a special case of virtual reality or virtual reality is a special case of augmented reality. For purposes of this thesis both the definitions can be accepted. The fact that is important for this thesis is, that augmented reality creates an environment, in which real world is not suppressed at all, instead real world has a dominant role in augmented reality. In comparison to virtual reality, augmented reality does not immerse a user into a completely artificial , computer–generated environment. Augmented reality creates an environment based on real world, or even the whole environment is a real world (e.g. live video stream), that is enhanced with virtual object created with computer. The idea, the augmented reality is based on, represents, in fact, the most difficult problem. While virtual reality creates an artificial world, which is completely controlled by a computer, augmented reality creates only a few virtual objects put into existing real world, which is controlled by no one.

One could say, that a virtual board displaying score of a hockey match on our television screen is augmented reality, what is only partly right. A virtual score board is added to the real environment (a hockey match), indeed, but the augmented information has to have

[1]Head–mounted displays.

a much stronger link to the real environment. This link is mostly a spatial relation between the augmentations and the real environment, i.e. the augmented information is placed into the scene of a real world on specific place, e.g. on the wall the big yellow building, on the specific marker or in geographic location accurately specified by its coordinates (latitude and longitude). This link can be called registration. R2–D2's spatial projection of Princess Leia in Star Wars would be a popular science fiction example for augmented reality [8].

Accurate **registration** between real and virtual objects is essential for augmented reality applications. It means that a virtual object should appear properly placed in the real world view, otherwise the user cannot correctly determine spatial relationships. Dynamic registration is particularly important when the user moves around the environment. The relative position between real and synthetic objects should be constant. The registration can be done in 2 ways:

- Using computer vision to detect markers or other features suitable for tracking.
- Using positioning sensors like accelerometer, magnetometer and GPS.

2.3 Vision–based Augmented Reality

A vision–based augmented reality can be successfully realized either with the help of special markers (2.3.1), or also without them. Markerless tracking is one of the most challenging and promising approaches in the field of augmented reality. Computer vision is very important for both the methods. Computer vision is described in this section, as well as camera calibration and homography.

Computer Vision [9] is the transformation of data from a still or video camera into either a decision based on information included in the visual input or a new representation. All such transformations are done in order to achieve a specific goal. A new representation might mean turning a color image into a grayscale image or creating only black and white output, in which th white lines represent detected edges.

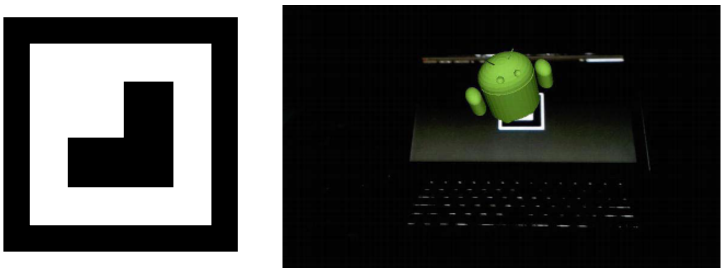
Computer vision tasks are easy to be done. The human brain divides the vision signal into many channels that stream different kinds of information into the brain. The brain works then with this channels in a special way, when it examines only "the important parts" of the received visual information, while the examination of "the less important parts" is suppressed. This process is very difficult and it is not easy to understand it completely.

On the other hand a computer receives only a grid (matrix) of numbers from the camera or from the harddisk and it should be able to do "the brain work". A problem could be, e.g. that the human brain automatically performs extremely fast and accurate pattern recognition, while a computer is not able to do it automatically, so it must be "ordered" to do so with some program. This problem, in fact, is formally impossible to solve, because there is no unique method of 3D signal reconstruction.

Computer vision is very important for augmented reality, especially for the vision–based augmented reality. Computer vision algorithms are used to recognize and track markers or other specific features in order to calculate the position and the orientation of the camera. Location–based augmented reality can also take an advantage of computer vision approaches, that can be used in fusion with positioning sensors. These sensors are used to calculate the position and the orientation of the device and computer vision algorithms can be used to stabilize displayed virtual objects, since sensors produce noise that causes displacement of virtual objects. This effect can be partially suppressed, when data from the sensors are filtered with a suitable filter.

2.3.1 Markers

The markers [14] are small cues used to calculate position of the camera and positions of the virtual objects in augmented reality (virtual objects are placed over the recognized markers). The example of a simple marker is displayed in figure 2.2(a). Markers often contain only two colors – black and white, so there is no need to identify the shades of gray and it is possible to reduce per pixel decision to a threshold decision. Markers can also include additional information, e.g. for error detection. The possibility of recognition of the markers and tracking them even within a large field of view, as well as in distorted images is very important for augmented reality. Because of this, the markers for augmented reality are not too dense and usually do not contain redundant information. Light conditions are another meaningful restriction of this approach. It could be difficult to track markers in dark room with limited visibility. Figure 2.2(b) displays a 3D model rendered over the detected marker.


(a) Example of a simple marker [18]. (b) 3D virtual model over the marker [18].

The simplified process of augmented reality based on marker detection step by step:

1. Obtaining a frame from the camera.
2. Detection of a marker in the frame from the camera.
3. Calculation of the camera position and orientation with respect to detected markers.
4. Drawing a virtual object over the detected marker, based on previously calculated position and orientation of the camera.

2.3.2 Camera calibration

The process of camera calibration is very important for augmented reality. It is a process of determining the two groups of camera parameters [13, 19, 2]:

- intrinsic
- extrinsic

Considering only the geometric aspect of the image, the image obtained with a CCD (Coupled Charged Device) camera is the result of a geometric transformation. The latter takes a 3D description of a scene and changes it into a 2D description.

An image point (pixel), given by its image coordinates, is the result of a transformation of a physical point defined in the world reference frame. The transformation can be expressed as a sequential order:

1. Euclidean transformation from the world reference frame to the camera reference frame (Fig. 2.2). It can be expressed as matrix $\mathbf{M}_{ex}$, in which 6 extrinsic parameters are held.

$$\mathbf{M}_{ex} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & -R_1^T T \\ r_{21} & r_{22} & r_{23} & -R_2^T T \\ r_{31} & r_{32} & r_{33} & -R_3^T T \end{pmatrix} \tag{2.1}$$

where r_{ij}, $i = 1 \ldots 3$, $j = 1 \ldots 3$ are the entries of rotation matrix and the right column represents the entries of translation vector.

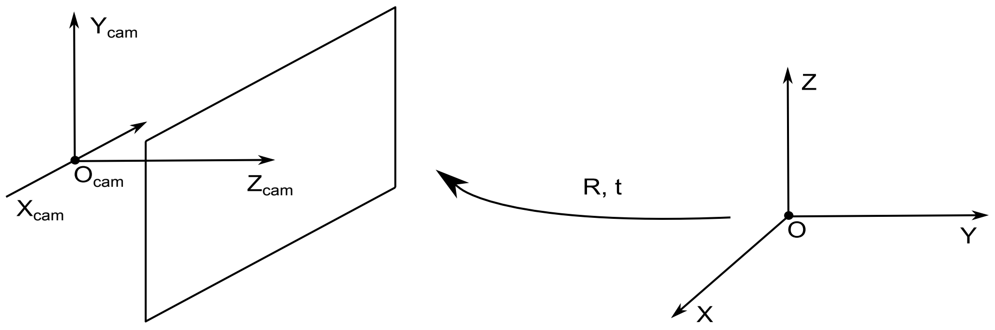


Figure 2.2: Euclidean transformation between world and camera coordinates.

2. A 2D–2D transformation from camera reference frame to image coordinates. It can be expressed as matrix A, in which 5 intrinsic parameters are held.

$$\mathbf{M}_{in} = \begin{pmatrix} \alpha_u & s & x_0 \\ 0 & \alpha_v & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

α_x and α_y represent the focal lengths of the camera in terms of pixel dimensions in the x and y directions respectively (i.e. divided by the pixel width, resp. height), (x_0, y_0) is the principal point on the image plane and s is a skew parameter.

Assuming the camera has square pixels, and thus equal scales in both the x and y directions allows one to set $\alpha_x = \alpha_y = \alpha$. Also in many cases s can be set to 0. Even with making these assumptions, the perspective projection will have 9 degrees of freedom which is one more than a homography which has 8.

The relationship between a space point $P = (X, Y, Z, l)$ and its projection $p = (u, v, 1)$ on the image can be expressed as:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathbf{M}_{in} \cdot \mathbf{M}_{ex} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.3}$$

Equation 2.3 provides two basic projection equations describing the relationship between points defined in space (X, Y, Z) and their corresponding pixels (x, y):

$$x = \frac{m_{11} \cdot X + m_{12} \cdot Y + m_{13} \cdot Z + m_{14}}{m_{31} \cdot X + m_{32} \cdot Y + m_{33} \cdot Z + m_{34}}$$
$$y = \frac{m_{21} \cdot X + m_{22} \cdot Y + m_{23} \cdot Z + m_{24}}{m_{31} \cdot X + m_{32} \cdot Y + m_{33} \cdot Z + m_{34}} \tag{2.4}$$

where m_{ij}, $i = 1 \ldots 3$, $j = 1 \ldots 4$ are the entries of projection matrix. The projection matrix has 9 entries, but only 8 are independent. Therefore, at least 6 points are needed to solve the equation.

2.3.3 Homography

A 2D point (x, y) in an image can be represented as a 3D vector (x_1, y_1, w) where $x = \frac{x_1}{w}$ and $y = \frac{y_1}{w}$. This is called the homogeneous representation of a point and it lies on the projective plane P^2. Homography [11, 13] is such an invertible mapping of points and lines from the projective plane P^2 to itself that three points lie on the same line if and only if their mapped points are also collinear. Another term for this transformation can be *projectivity – a mapping from P2 → P2 is a projectivity if and only if there exists a non-singular* 3×3 *matrix* **H** *such that for any point in* P^2 *represented by vector x it is true that its mapped point equals* $\mathbf{H} \cdot x$.

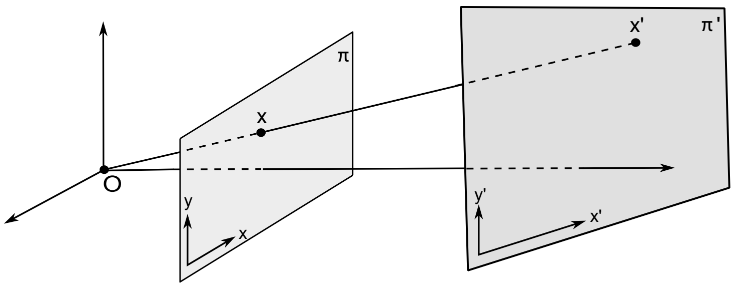


Figure 2.3: Projective transformation [13].

Example of homography (Fig. 2.3) is projection with center O and two planes π and π', which maps points from π to π', $x_i \leftrightarrow x'_i$. For homogeneous representation of points, the mapping can be expressed:

$$x' = \mathbf{H} \cdot x \tag{2.5}$$

where **H** is 3×3 homography transformation matrix [13]

$$\mathbf{H} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \tag{2.6}$$

$$x' = \mathbf{H} \cdot x = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} \tag{2.7}$$

H is considered to be a homogeneous matrix and has only 8 degrees of freedom even though it contains 9 elements. This means there are 8 unknowns that need to be solved for.

Typically, homographies are estimated between images by finding feature correspondences in those images. The commonly used algorithms make use of point feature correspondences, though other features can be used as well, such as lines or conics. Correspondences represent an input of the homography estimation algorithms and for their calculation SURF[2] detector can be used.

RANSAC[3] is the most commonly used robust estimation method for homographies. It is an iterative method – four random correspondences are selected and a homography H is calculated. All other correspondences are then classified as inliers (suitable points) or outliers (not suitable points) depending on their concurrence with H. After all iterations, the one with the largest number of inliers is selected.

2.4 Location–based Augmented Reality

Another way of getting the position and the orientation of the camera is getting these information from the positioning sensors of the device. The 3D orientation (pitch, roll, azimuth, Fig. 2.4) in which the device is being held can be obtained as a combination of the magnetic field sensor data and the accelerometer sensor data[18].

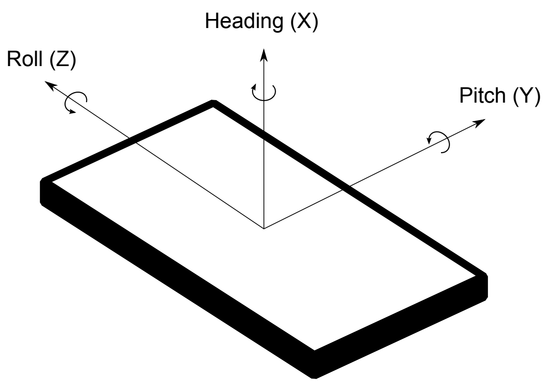


Figure 2.4: Orientation axes of the device. [18].

2.4.1 Magnetic Field Sensor

The earth's magnetic field is a self sustaining magnetic field that resembles a magnetic dipole with one end near the Earth's geographic North Pole and the other one near the earth's geographic South Pole. Although this magnetic field is relatively stable over time,

[2]Speeded Up Robust Features.
[3]RANdom SAmle Consensus.

electric currents in the ionosphere can cause daily alterations which can deflect surface magnetic fields.

A magnetometer (magnetic compass)[4] measures the strength and direction of the local magnetic field. The measured magnetic field is a combination of both the earth's magnetic field and any magnetic field created by nearby objects. This strength is expressed in Tesla [T]. Measurements are also subjected to distortion cause by the magnetic fields in near surroundings, e.g. speakers, piece of magnetized iron, electric currents, etc. There are two types of magnetic compasses [3]:

- **Two–axis compasses** – called also the X and Y axis sensors, each sensor on an electronic compass assembly measures the magnetic field in its sensitive axis and the arc tangent Y/X provides the heading of the compass with respect to the X axis. A two – axis compass can remain accurate as long as the sensors remain horizontal to the gravitational vector. A good example is mechanical compass used for centuries.

- **Three–axis compasses** – contain magnetic sensors in all three orthogonal vectors of an electronic compass assembly. These three magnetic sensors are complemented by a tilt–sensing element to measure the gravitational direction. Three–axis compass is able to measure the direction also in non–horizontal positions.

2.4.2 Accelerometer

An accelerometer[15] is a sensor that measures the dynamic acceleration – the force resulting from velocity change of the device or static acceleration – the force resulting from the earth gravity. An accelerometer can be used to measure:

- Angle of the tilt.
- Interia force.
- Shaking or vibrations.

Accelerometers usually measure in “g” – the unit that measures the earth's gravitational pull ($1g \approx 9,82ms^{-2}$).

The accelerometer in a smartphone measures acceleration along three directional axes: left–right (lateral), forward–backward (longitudinal) and up–down (vertical) (Fig. 2.5) [18].

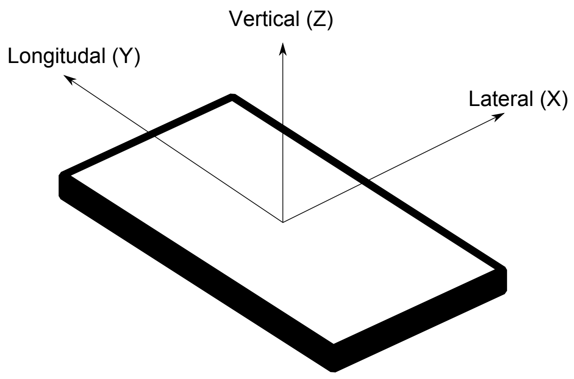


Figure 2.5: Accelerometer axes of the device. [18].

The axes of accelerometer, they measure:

- ***X* axis** – on a normal device with a normal accelerometer, the X measures lateral acceleration, i.e. left to right, right to left respectively. The reading is positive if the device is moving to the user's right side, and it is negative if moved the to user's left.
- ***Y* axis** – the Y functions the same way as the X, except it measures the acceleration longitudinally. Positive reading is registered when the device is moved in the direction of its top, and negative reading is registered if moved in the opposite direction.
- ***Z* axis** – this axis measures the acceleration for upward and downward motion, for which positive readings are upward motions, and negative readings are downward motions. When the device is motionless, reading of approximately $-9.82ms^{-2}$, due to gravity, is obtained. This fact should be considered in the calculations.

2.4.3 Global Positioning System (GPS) and Geography

The global positioning system (GPS) [18] is a location system that can give an accurate location via satellites – it is a space–based satellite navigation system.

To obtain a fix, a receiver must communicate with a minimum of four satellites. The satellites send three pieces of information to the receiver, which are then fed into one of the many algorithms for finding the current location. The three pieces are the time of broadcast, the orbital location of that particular satellite, and the rough locations of all the other satellites. The location is calculated using trigonometry.

Since this work is focused on the navigation, it is explained what latitude and longitude (obtained from GPS sensor of an Android device) are and how to convert these angles to Cartesian coordinates in 3D space.

Combination of the latitude and the longitude determines accurate position on Earth's surface. One minute of latitude equals approximately one nautical mile $(1852m)$[4].

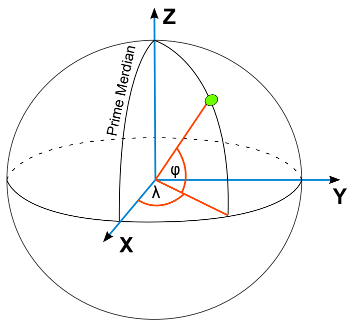


Figure 2.6: Latitude, longitude.

Latitude

Latitude specifies the position on the surface of the Earth towards the north or south from the equator. It is an angle between the plane of the equator and a line passing through the center of the Earth and the appropriate point on the Earth's surface.

[4]See http://www.zemepis.estranky.cz/clanky/zemepisna-delka-a-sirka.html.

Latitude is marked by φ and is measured in degrees. Points lying north of the equator, i.e. in the northern hemisphere, have north latitude, points south of the equator, i.e. the southern hemisphere, have south latitude. Points on the equator have latitude zero. Curves with the same latitude are called parallels.

Longitude

Longitude specifies the position on the surface of the Earth towards the east or west from the Prime (zero) meridian. It is an angle between the plane of the local meridian passing through the point and the plane of the Prime meridian.

Longitude is marked by λ, it is also measured in degrees. Points lying east of the Prime meridian, i.e. in the eastern hemisphere, up to 180° degrees, have east longitude (positive). Points lying west of the Prime meridian, i.e.in the western hemisphere, up to 180° degrees, have west longitude (negative).

2.5 Augmented Reality Displays

Augmented reality can be displayed in various ways nowadays [8]. Displays can be divided into two main categories:

- Head–mounted displays.
- Classic displays, e.g. displays of hand–held devices.

2.5.1 Head–Mounted Displays

This type of display requires to be attached to the user's head[5]. It usually contains a camera and a display. The camera is used to capture the view of the real world, which is then displayed with augmented information on the display, which is situated in front of the user's eyes. There are two different head–mounted display technologies:

- **Video see–through displays** – this is the technology containing a camera and a display situated in front of the user's eyes, so one can see augmented reality.
- **Optical see–through displays** – make use of optical combiners, e.g. half–silvered mirror or transparent LCD, which display only augmented information. The user can see the real world through the augmented information with his/her own eyes. Google Glass[6] is a good example of this technology (Fig. 2.7).

2.5.2 Hand–Held Displays

Hand–held displays for displaying augmented reality are becoming very popular nowadays. Such a display is usually a part of a mobile device, e.g. tablet or smartphone. These examples combine all of the necessary hardware, like processor, memory, display, camera, sensors, into a single wireless device. Video see–through display technology is used within this approach. Integrated camera captures the real world view, the augmented information is added and the augmented reality can be seen on the display of the device. This is the approach used in this thesis and it has its advantages:

[5]For various head–mounted displays see http://www.vrealities.com/hmd.html.

[6]See http://www.google.com/glass/start/what-it-does/.

Figure 2.7: Google Glass.

- Today's devices are small and their weight is low as well.
- All hardware in one device.
- It is easier to develop an augmented reality application.

and disadvantages:

- Hardware performance of mobile devices is lower than hardware performance of laptop or desktop.
- Applications using the hardware needed for augmented reality applications (sensors, camera, GPS, Wi–Fi, permanently displaying something on the display) consume the battery of the device very fast.
- The device must be held in hand, so the user may be limited in interaction with the real environment.
- Quality of the video stream may be sometimes too low for good image processing, e.g. because of fixed focal length.

Hand–held devices represent serious alternative to head–mounted display, since smartphones and tablets are very popular and widely used devices.

2.6 Practical Application of Augmented Reality

There are many possibilities for using augmented reality [10] in an innovative way and four types of applications, in which the augmented reality can be used, are described in this section.

- **Advertising and commercial** – augmented reality is often used to promote new products online. Most techniques use markers that the users present in front of their web cam either on special software or simply on the advertising company's website.
- **Education** – applications, such as cultural applications with sightseeing and museum guidance using augmented reality interfaces, and some smartphone applications that make use of augmented reality for an educational purpose. Among cultural applications, there are a few systems which use augmented reality for virtual reconstruction of ancient ruins. There are also a few systems which exploit augmented reality for museum guidance[7] (Fig. 2.8).

[7]See http://blogs.smithsonianmag.com/ideas/2012/08/augmented-reality-livens-up-museums/.

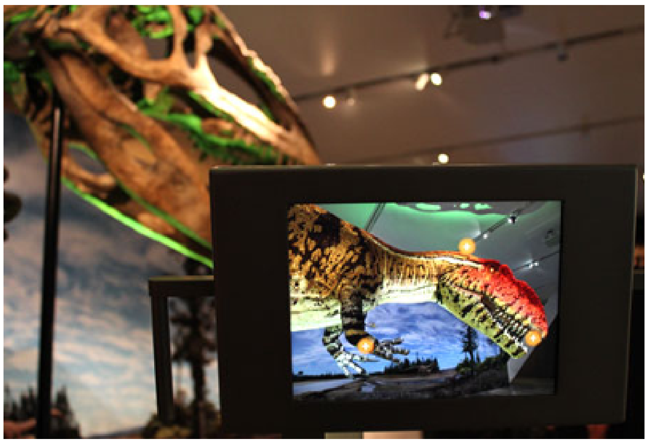


Figure 2.8: Museum guidance – adding flesh and skin on a dinosaur skeleton.

- **Medical** – augmented reality could be used e.g. to display inner human organs during a surgical operation in real time, or to see virtual anatomy of the bones through the "real" skin. It uses polygonal surface models for real time visualization (Fig. 2.9).

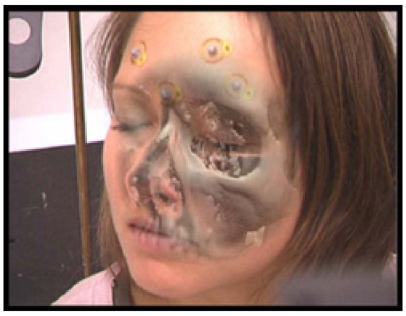


Figure 2.9: System for viewing through the skin [7].

- **Mobile applications** – augmented reality on mobiles is becoming popular. It can be used for displaying various additional information e.g. about the surroundings or navigation. Three examples of location–based augmented reality mobile applications using the sensors of the device are mentioned in the following paragraph[8].

  Layar shows digital information about points of interest in the surroundings of the user directly on the screen of the device. The user can use this application to scan everything around him/her – restaurant menus, grocery products, magazines, posters, billboards, hotels, business establishments, etc.

  3D Compass+ initially works as a compass on a mobile device. The application displays the camera view with an augmented compass over it. The application also uses GPS and displays the map on the screen, thus making the phone a navigation tool.

[8]See http://www.androidauthority.com/best-augmented-reality-android-apps-93950/.

iOnRoad uses the front camera, GPS, and sensors of the device to keep the user safe on the road. Primarily, the application monitors and keeps watch of car's distance from other cars (Fig. 2.10), thus making the device a collision–warning device. While driving, the application shows warning signals in colors.

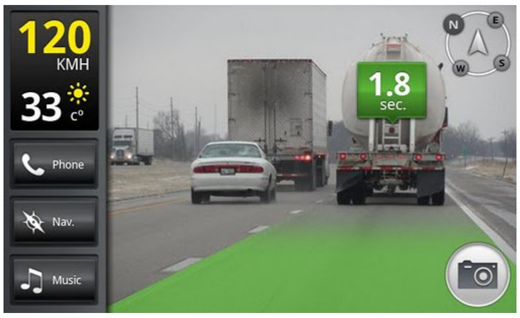


Figure 2.10: iOnRoad application showing the distance from the car in front of the device.

Chapter 3

Google Android Mobile Platform

Android[16, 17] can be defined as a software stack – a set of software subsystems (section 3.2), for mobile devices including:

- **An operating system** – a modified version of the Linux kernel.
- **Middleware** – partly based on Java.
- **Key applications written in Java** – web browser, contact and message manager, etc.

3.1 Android Versions

The latest version of Android is 4.2 Jelly Bean. Table 3.1 displays a brief history of previous versions[1] relevant for purposes of this thesis.

2.3	Gingerbread
3.0/3.1/3.2	Honeycomb
4.0	Ice Cream Sandwich
4.1	Jelly Bean
4.2	Jelly Bean

Table 3.1: Brief history of Android versions [16].

3.2 Architecture of Android

The Android software stack consists of five sections ordered in four layers (Fig. 3.1) [16, 17]:

- **Linux kernel** – Android is based on this kernel, it contains all the low–level device drivers for the hardware components.
- **Libraries** – All the code that provides the main features of Android.

[1]Information missing in [16] were added from `http://visual.ly/brief-history-android`.

- **Android runtime** – shares the layer with **Libraries**. Provides a set of core libraries which enable developers to write applications using the Java language. It also includes the *Dalvik Virtual Machine*[2], which enables every application to run its own process with its own instance of Dalvik Virtual Machine.
- **Application framework** – makes various capabilities of Android available to application developers.
- **Applications** – Applications which are implicitly installed in the device (Contacts, Browser, Messaging, . . .) and applications downloaded from Google Market.

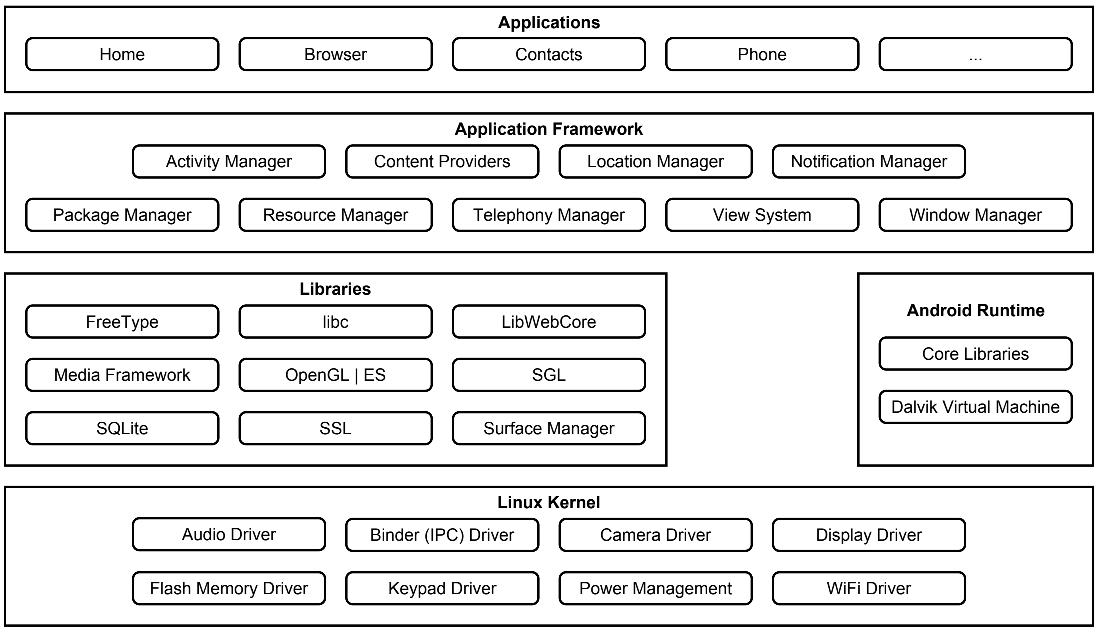


Figure 3.1: Android architecture [17].

3.3 Android Application Structure and Components

An Android application architecture is different from a desktop application architecture. It is a collection of components (activities, services, content providers, and broadcast receivers) which run in a Linux process and which are managed by Android. These components share a set of resources, including databases, preferences, a file system, and the Linux process [17].

3.3.1 Manifest

Manifest is an XML – structured file [16, 17], which contains detailed information (specified for Android) about the application:

[2]Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery–powered mobile devices with limited memory and CPU [16].

- Package name of the application.
- The version and name of the application.
- Minimum version of operation system for which the application is developed.
- Definition of each activity.
- Permissions to access specialized hardware and functionality, e.g. camera, sensors, GPS, internet connection, etc.

3.3.2 Components

An application is a collection of components that run in a Linux process and that are managed by Android [17]. These components share a set of resources, including databases, preferences, a file system, and the Linux process. The components are:

- **Activities** – a component which presents a graphical user interface and code, which handles the user's interaction with the interface. Applications usually contain more than only one activity – the user can switch the activities. Typical life cycle of an activity includes seven inherited methods, which can be overridden (Fig. 3.2).

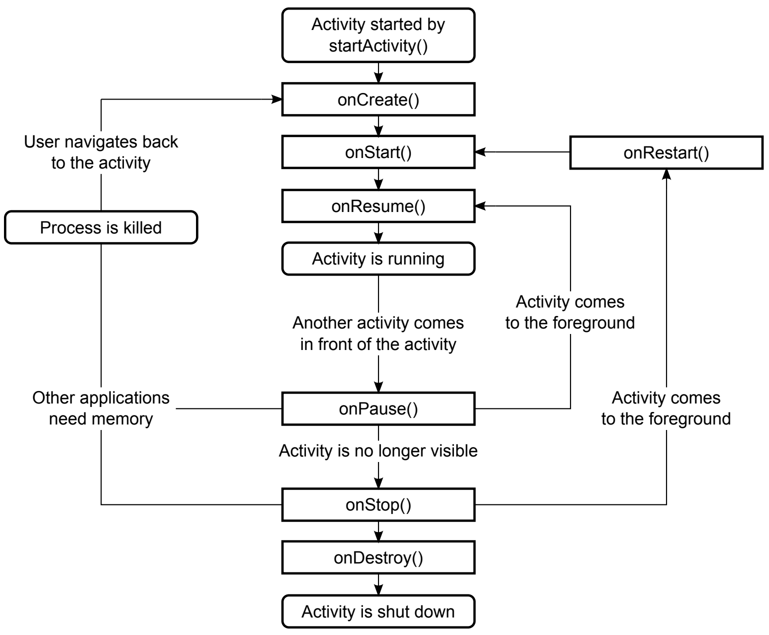

Activity started by startActivity()
onCreate()
User navigates back to the activity
onStart()
onRestart()
onResume()
Process is killed
Activity is running
Activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
Activity comes to the foreground
Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Figure 3.2: The lifecycle of an activity [17].

- **Services** – a background–running component without user interface (code only). The period of time a service is running is not defined. Both service and activity are running on the main thread, so time consuming operations need to run in other threads. Services are classified as **local**, which runs in the same process as the rest of the application (used for background tasks), or **remote**, which runs in a separate process (used for interprocess communications).

- **Broadcast Receivers** – receives and reacts to broadcast. Broadcast can often come from the system itself – notifications like incoming call, received message, low battery indicator, etc. It can also be created by the application, e.g. as notification, that informs the communication with database server is successfully done. The reaction to received broadcast could be notification, starting new activity, specialized operation, etc.

- **Content Providers** – makes specific data available to other applications. The data can be stored e.g. in the Android file system or in an SQLite database.

3.3.3 Intents

Intents [17] represent messages which describe operations or provide descriptions of external events that have occurred (in the case of broadcast). They can be used for communications between the application components or between the components of different applications. Intents can be classified as:

- **Explicit** – the target component is designated by its name. It is typically used for the application–internal messages (activity that launches another activity located within the same application).

- **Implicit** – the target component is not designated by its name. It is often used to start the component in another application (Android searches for the best component or components).

3.4 Development for Android

The applications are developed mostly in Java language. The Android SDK (Software Development Kit) [16, 17] is the most important software needed for the Android development. SDK includes libraries, emulator, documentation, sample code, tutorials and debugger. Android SDK Manager manages (installs new/uninstalls current) the versions of the Android SDK (API levels) installed on the computer. An SDK Platform or a Google APIs are available for each API level. The main difference between Google APIs and SDK Platform is that Google APIs offers also additional libraries for Google Maps.

An alternative is to use C/C++ language. The Android Native Development Kit (NDK) helps to boost (not necessarily) an application's performance by converting C/C++ source code to native code libraries which run on Android devices. The native code is often used with computationally intensive processes, e.g. physics simulations. This thesis does not include any native code.

Using an emulator is a good way how to test the developed application on a virtual device when a real device (smartphone or tablet) is not available. Each virtual device has its own hardware profile, emulated storage and runs a chosen Android version. The developed application can be tested on various Android versions and for various resolutions and densities of displays (important for graphical user interface) using an emulator. It also offers the possibility of emulating an incoming call or message and changes of current geographic location. Disadvantages of an emulator are that it is rather slower than a real device and it is not able to emulate accelerometer, gyroscope, magnetometer or camera.

3.4.1 User Interface of Android Application

Graphical user interface of an Android application is typically defined separately from the source code, in XML files [16]. The interface is loaded in `onCreate()` method handler of an activity or it can also be created dynamically during the runtime.

An activity contains *Views* – widgets, which are displayed on the screen. Android offers a wide variety of pre–defined widgets, like buttons, textboxes or labels. Developers can also create their own widgets or customize the existing ones.

A special type of view is *ViewGroup*, which can contain one or more grouped views and defines a layout of contained views. Android supports following ViewGroups:

- `LinearLayout` – can be oriented vertically or horizontally (one column or one row of views).
- `AbsoluteLayout` – the exact position of inner views can be defined.
- `RelativeLayout` – views are positioned relatively to each other.
- `TableLayout` – arranges the views into rows and columns. One row is designated with `TableRow` element, which can contain one or more views (cells).
- `FrameLayout` – can be used to display a single view, e.g. an image, which is always anchored to the top left of the layout.
- `ScrollView` – a special type of `FrameLayout` which enables the user to scroll through a list of views which occupy more space than the physical screen of the device is able to display.

3.5 OpenGL ES

OpenGL® ES [1, 5] is a cross–platform API for 2D and 3D graphics on embedded systems, including consoles, phones, appliances and vehicles. It consists of well–defined subsets of desktop OpenGL, creating a flexible interface between software and graphics hardware. The interface consists of a set of procedures and functions that allow a developer to specify the objects and operations involved in producing high–quality graphical images.

3.5.1 OpenGL ES 1.X

OpenGL ES 1.0 is defined relative to the OpenGL 1.3 specification and emphasizes enabling software rendering and basic hardware acceleration[3]. OpenGL ES 1.1 is derived from desktop OpenGL 1.5 and is fully backwards compatible with OpenGL ES 1.0. It provides enhanced functionality, improved image quality and optimizations to increase performance and reduced memory bandwidth usage in order to save the power. In these versions, the hardware pipeline (Fig. 3.3) is not programmable, it is fixed.

3.5.2 OpenGL ES 2.X

OpenGL ES 2.0 is defined relative to the OpenGL 2.0 specification and emphasizes a programmable 3D graphics pipeline[4]. It also includes the possibility of creating shader (vertex

[3]See `http://www.khronos.org/opengles/1_X/`.
[4]See `http://www.khronos.org/opengles/2_X/`.

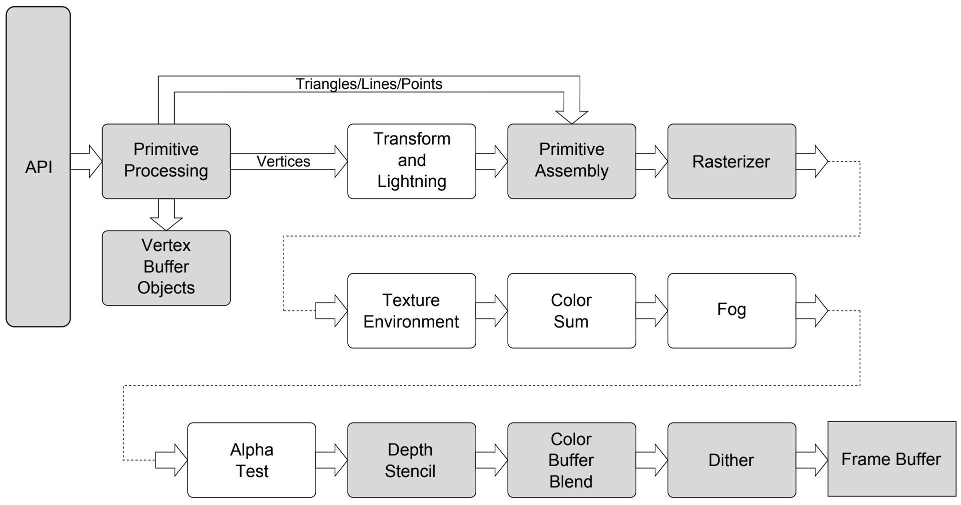


Figure 3.3: OpenGL ES 1.X fixed pipeline [6].

and fragment shaders in the OpenGL ES Shading Language) and program objects. OpenGL ES 2.0 does not support the fixed function transformation and fragment pipeline of OpenGL ES 1.x. The programmable pipeline is similar to the fixed function pipeline. The differences are:

- The transform and lightning block is replaced by the vertex shader block.
- The texture environment, color sum and fog blocks are replaced by the fragment shader block.
- The alpha test block is omitted.

3.5.3 OpenGL ES 3.0

This specification[5] brings significant functionality and portability enhancements to 3D graphics API. It provides access to graphics processing unit functionality with portability across a wide range of mobile devices and embedded systems.

3.5.4 OpenGL ES on Android

The OpenGL ES 1.0 and 1.1 API specifications are supported by Android 1.0 and its later versions. The OpenGL ES 2.0 API specification is supported by Android 2.2 and its later version. It is supported by most Android devices and new applications are recommended to be developed with OpenGL ES [1].

There are two classes in the Android framework API, which are supposed to be used in order to create and manipulate graphics with the OpenGL ES API:

- `GLSurfaceView` – class, where the object can be drawn and manipulated using OpenGL ES API calls.

[5]See `http://www.khronos.org/news/press/khronos-releases-opengl-es-3.0-specification`.

- `GLSurfaceView.Renderer` – interface, which defines methods required for drawing graphics in `GLSurfaceView`. Implementation of this interface must be done in separate class and the renderer must be a property of `GLSurfaceView`. The renderer implicitly runs in its own thread, so calling its methods does not slow down the main process thread. The three important methods are:
  - `onSurfaceCreated` – this method is called only once, while the `GLSurfaceView` is being created, so it is used to perform actions like setting OpenGL ES environment parameters or initializing graphic object (actions that are needed to be performed only once).
  - `onDrawFrame` – this method is called on each redraw of the `GLSurfaceView`, it represents the primary execution point for drawing/re–drawing of graphic objects.
  - `onSurfaceChanged` – this method is called when geometry of the `GLSurfaceView` changes – size or orientation (landscape/portrait mode) is changed.

Not all of the OpenGL ES features which are used in the application can be available on all devices, so it is important to declare the requirements. In the manifest file there is a declaration of required OpenGL ES version. Google Play[6] then restricts this application from being installed on devices which do not support the declared version.

[6]See `https://play.google.com/store`.

Chapter 4

Augmented Reality Library and Application Design

The main goal of the practical part of this thesis was to design and implement a augmented reality system for Android platform. After I had studied the theory of augmented reality, I decided that the location–based approach (section 2.4) of creating augmented reality would be the right choice for this thesis. This system should serve as an navigation application which would draw a route over the camera view.

Location–based augmented reality applications use the sensors of the device to determine its geographic location and the orientation in the real world. The location can be obtained from the GPS or the network provider. Coordinates provided by the network provider are usually not suitable for this kind of applications, because they are often inaccurate. Network provider should be used as a support for GPS, which is the primary provider. Orientation of the device is calculated using data from two positioning sensors – accelerometer and magnetometer. When the orientation is calculated, it can be used to adjust the virtual camera orientation. Together with rendering of the virtual objects (Fig. 4.3) using OpenGL ES, they create an interactive virtual layer. A camera layer is represented by the realtime camera view. The camera layer needs to be "covered" with the virtual layer, which has transparent background, in order to create the augmented reality view. These two layers and orientation calculation represent the core of the augmented reality system. The device running location–based augmented reality application needs to contain previously mentioned hardware components.

The augmented reality view may consist of more than two layers. Another layer, situated on the top of the previous ones, could be e.g. the layer containing a graphical user interface, a compass, a radar, a map fragment, etc.

One part of the designed augmented reality system is a library, which implements the methods for working with data from the sensors and two, augmented reality view–creating, layers. Second part of the system is an application which combines the library functions and its own functions for obtaining navigation information, thus creating the augmented reality view displaying a virtual route on the top of the real world view. The concept of such system is shown in figure 4.1 – bold items were added to the concept during the process of designing the system, and the other items were in the first base concept.

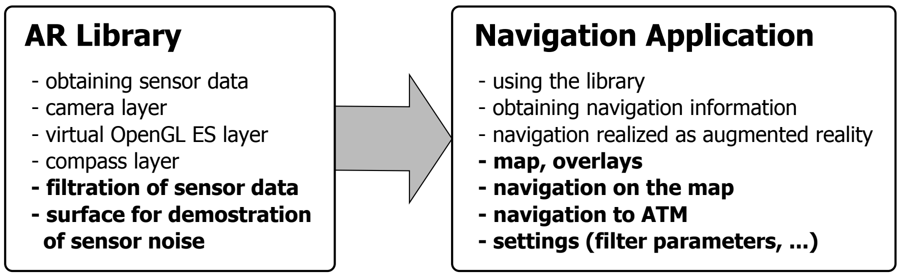

AR Library
- obtaining sensor data
- camera layer
- virtual OpenGL ES layer
- compass layer
- filtration of sensor data
- surface for demostration of sensor noise
Navigation Application
- using the library
- obtaining navigation information
- navigation realized as augmented reality
- map, overlays
- navigation on the map
- navigation to ATM
- settings (filter parameters, ...)

Figure 4.1: Concept of the system.

4.1 Augmented Reality Library

The library was developed as a software framework, with respect to a possibility of being reused in various Android location–based augmented reality applications (e.g. displaying of the points of interest in the surroundings of the user).

The first concept of the library considered implementation of three layers (Fig. 4.2), which display their content in real time (this concept is preserved):

1. **Camera layer** displays the frames captured by the camera. It does not need any additional information or data, only the field of view of the camera needs to be obtained. The real camera field of view and the virtual camera field view should be the same.

2. **Virtual layer** draws a route over the camera view. This layer, unlike the camera layer, needs the device orientation in the real world, its current geographic location converted to 3D coordinates and information about calculated route (especially coordinates of its points).

3. **Compass layer** adds a compass layout on the top of two previous layers. Compass needs only the azimuth value to rotate the layout.

4.1.1 Device Orientation and GPS Position

Orientation and position of the device are the most important data for a location–based augmented reality. Current geographic position is received mainly from GPS. It could be received also from network provider, but it is not so accurate.

Orientation of the device is calculated from the acceleration and magnetic field values. These values are obtained from the positioning sensors of the device – accelerometer and magnetometer. Measured values are often distorted because of the noise caused by sensors, shaking with the device or magnetic fields in the surroundings of the device. These effects need to be suppressed due to stabilization of the position of virtual objects, so the obtained

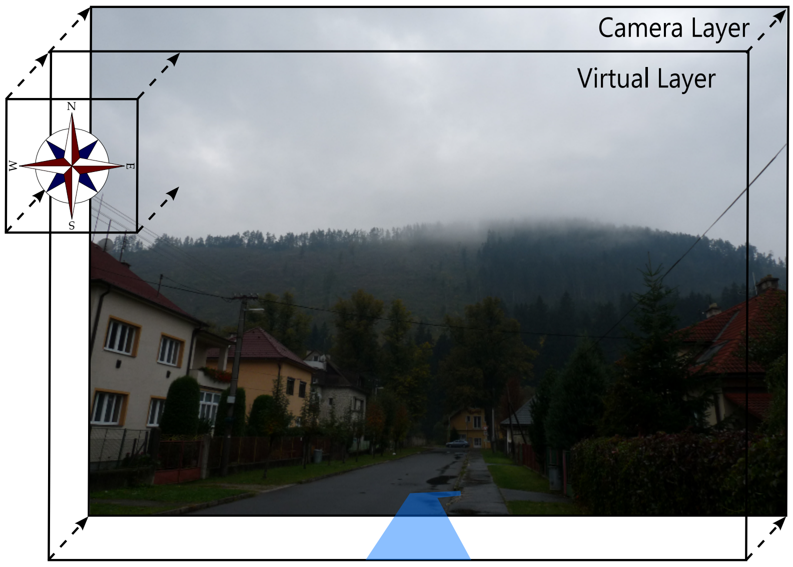


Figure 4.2: Layers of augmented reality view.

values need to be filtered. As a filter could be used simple one–dimensional Kalman filter[1]:

$$p = p + q \tag{4.1}$$

$$\begin{aligned} k &= \frac{p}{(p+r)} \\ x &= x + k \cdot (value - x) \\ p &= (1-k) \cdot p \end{aligned} \tag{4.2}$$

The first formula represents the prediction of the filter and the next three formulas calculate the value update. The variables are:

- x – the filtered value.
- q – the process noise. Adjusting this value is essential to get smoother values. The lower this values is, the smoother the output is, but ale it is slower and delayed.
- r – the sensor noise. Another variable, which value is very important to achieve smoother output. If it is too low (e.g. 1), the sensor give more noisy results. If the value us higher (e.g. 4), the noise is reduced. Too high values may cause significant delay of filtered data in comparison to the original data.
- p – the estimated error. It initial value is not very important, it only needs to be high enough, so it can decrease during the runtime.
- k – the Kalman gain.

A rotation matrix is calculated from filtered values and it is used with OpenGL ES renderer to correctly set the orientation of the camera in virtual layer. This approach

[1]See http://interactive-matter.eu/blog/2009/12/18/filtering-sensor-data-with-a-kalman-filter/.

allows the user to use the device as a "window into virtual world". The figure 4.3 shows only the virtual layer containing a grid situated on the ground. The azimuth value for compass is calculated from the rotation matrix. The virtual layer should be also invariant to the device display mode, whether it is a landscape or a portrait mode.

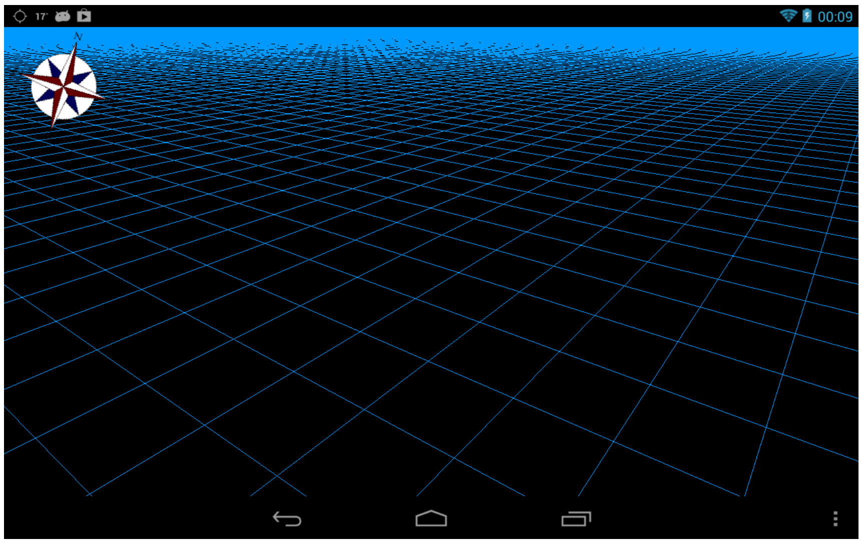


Figure 4.3: Virtual grid situated on the ground.

The coordinate system of the virtual world is adapted to world coordinate system[2] (Fig. 4.4): the positive X axis points towards the east and X is defined as the vector product $Y \times Z$. The positive Y axis points towards the north and positive Z axis points vertically from the ground (an altitude axis). Axes X and Y are tangential to the ground at the current position of the device. This is the coordinate system to which vectors from the coordinate system of the device (Fig. 2.5) are converted. A perspective of the OpenGL layer is set with respect to used units, i.e. $1m$ in the real world equals 1 unit of length in the virtual layer.

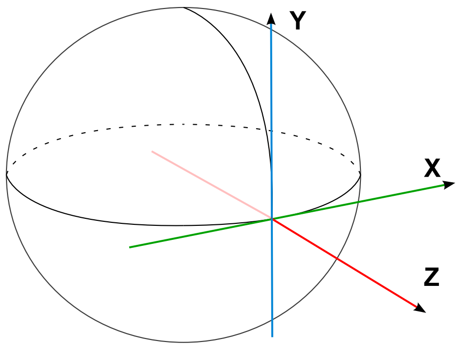


Figure 4.4: World coordinate system.

[2]See http://developer.android.com/reference/android/hardware/SensorEvent.html.

4.2 Augmented Reality Navigation Application

The application was developed using the designed library in order to create an augmented reality view. The application should serve as a navigation, which should correctly draw a route into a camera view, thus creating an augmented reality. First concept of this idea is shown in figure 4.5.

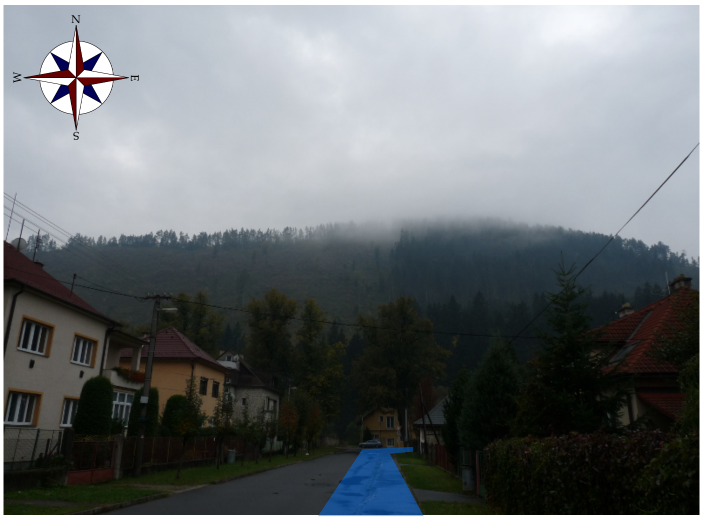


Figure 4.5: Original concept of the camera screen.

4.2.1 Calculation of the Route

The application can navigate to a destination address specified in the text form or to a chosen ATM. If the address in the text form is set as a destination location, it needs to be converted to corresponding geographic coordinates before calculating the route. The list of ATMs is in KML[3] file, so its coordinates are given in suitable form. The calculation itself is done by **Google Directions API**[4], which needs only start and destination coordinates and a flag indicating presence of the location sensor on the device.

4.2.2 Map View

The user can use the augmented reality view for navigation, or the alternative map view as well. I decided to use the **Google Maps Android API version 1**[5], which provides data required to display a map. The map contains overlays to mark current location of the device and to display calculated route and its waypoints. These overlays are touchable, except the lines forming the route, so the application displays additional information about the touched overlay – the current address or a waypoint hint. Zoom controls are included too and displayed route adapts to zoom level or translation changes.

[3]Keyhole Markup Language, based on XML – `http://www.opengeospatial.org/standards/kml/`.
[4]See `https://developers.google.com/maps/documentation/directions/`.
[5]See `https://developers.google.com/maps/documentation/android/v1/hello-mapview`.

4.2.3 Graphical User Interface of Augmented Reality Application

The designed graphical user interface (section 5.7) is very simple and minimalistic, so the user can see almost the whole augmented reality view, the map view respectively. Both the views contains one small button for displaying information about the obtained route. Other controls are designed as an options menu bar (Fig. 4.6), which is implicitly hidden. The map view also displays three different types of overlay, two of which are designed to react to the users touch by displaying additional information.

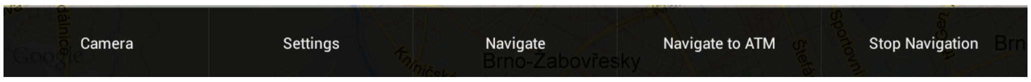


Figure 4.6: Menu bar (map activity).

The settings activity, however, is designed as usual graphical user interface with common controls.

The application also display customized dialog windows in some specific situations, like starting the navigation, touching either *Navigate* or *Navigate to ATM* button.

4.3 Data–flow Scheme of Augmented Reality System

Figure 4.7 is illustrates a simplified scheme, which describes the processing of data and their transformations during the runtime. This scheme is preserved in further work and the designed system is implemented according to it. The gray blocks are processed by the application methods and the white blocks represent the library methods. The *Route rendering* block is, in fact, a combination of both the application and the library methods – the application uses the library OpenGL ES renderer in order to display the obtained route.

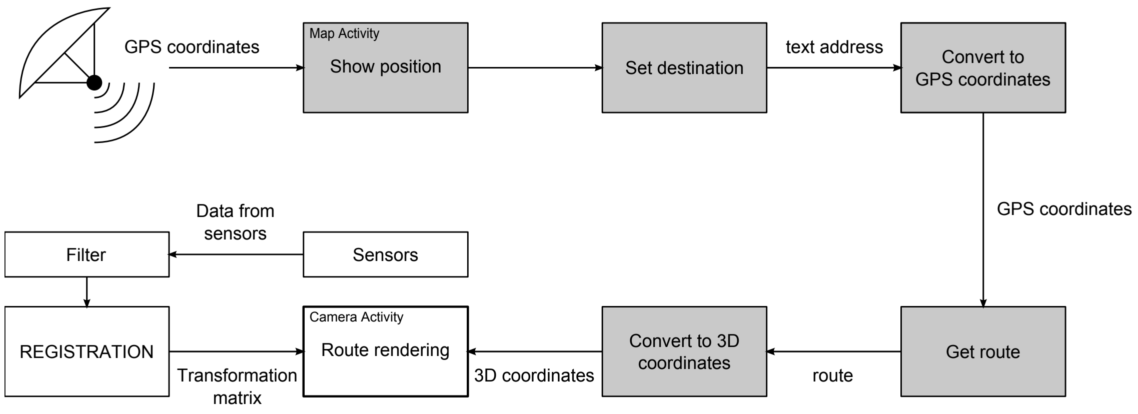


Figure 4.7: Data–flow diagram.

Firstly, the GPS location of the device, i.e. latitude and longitude, are required to show its position on the map. After setting start and destination addresses in the text form, they are converted to corresponding geographic coordinates. The application uses these coordinates to obtain detailed information about the route, using Google Directions API. The obtained route contains an array of geographic coordinates, which are converted to

3D Cartesian coordinates. These are used in OpenGL ES for rendering the route over the camera view. The registration process is essential for rendering the route correctly. This process works with values from the accelerometer and the magnetometer of the device. Values from both sensors are filtered and subsequently used for the registration process, product of which is a transformation matrix for OpenGL ES renderer. Each phase is described in the next chapter in detail.

Chapter 5

Realization of Augmented Reality System for Android

This chapter describes in detail the realization of the designed library and the demo application. First of all, structure of the library and application followed by the class relationships are explained. The order of following sections is based on the scheme 4.7 from previous chapter. Firstly, the way of obtaining GPS position and information needed for navigation is described. Secondly, the most important, calculation of the device orientation and realization of augmented reality is described. Lastly, working with the map and the way of adding overlays on it are explained.

5.1 Structure of Designed Augmented Reality Library and Application

Both the library and the application are divided into packages. Each package represents a group of classes with similar functionality and usage, e.g. data types are grouped together, classes for obtaining an ATM list are in one package, etc.

5.1.1 Library Structure

The library is designed to be as general as possible and it could be used in various applications. It could be used in location–based augmented reality applications or applications focused on working with positioning sensors. The library contains three packages containing eight classes altogether (Fig. 5.1).

Each package has its own function:

- **Main package (mmaureli)** – implements classes used for obtaining data from sensors and filtering them, and two classes, which are used in settings activity for drawing three axes to demonstrate the noise level.
- **Augmented reality surfaces** – classes of this package represent layers of the augmented reality view, i.e. a camera layer, which displays the frames captured with the camera and a virtual layer, which uses OpenGL ES to render a route over the camera layer.
- **Compass** – includes only one class, `Compass`, that represents a visible graphic element and it is displayed over the camera and renderer layer.

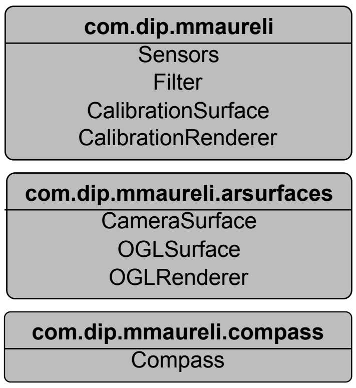


Figure 5.1: Structure of designed library.

5.1.2 Application Structure

The **augmented reality navigation** is developed as a demo application for the designed library. In order to use the application as navigation, the route from one location to another one must be calculated. The navigation can be stopped at any time, thus causing a transition of the application to its initial state.

The main activity of this application is the camera activity – it displays the augmented reality view. This class is that part of the application, in which every class from the designed library is used.

This demo does not contain only augmented reality view, it also contains a map. The map activity is the first activity to be displayed when the application is started (activities are described in more detail in chapter 5). Firstly, only one overlay is displayed on the map – the current geographic location of the device. If navigation is activated, the route and its waypoints are displayed too. Every displayed overlay, except the route itself, is touchable. As a reaction to the touch, a bubble layout is displayed, and its content depends on a type of touched overlay.

The application is designed with respect to good possibilities of adding new, or modifying the existing functionality. The other resources, such as activity layouts, styles or strings are defined in XML files, separately from the source code.

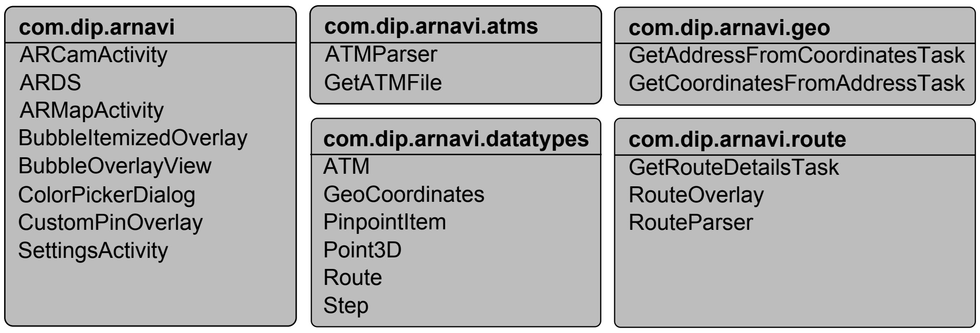


Figure 5.2: Structure of designed application.

The classes are divided into five packages (Fig. 5.2):

- **Main package** – contains three activities and classes used for creation of the customized map overlays, a color picker dialog (taken from an Android sample) and a special class `ARDS` (Augmented Reality Data Storage).
- **Automated teller machines** – the user can be navigated not only to a specific address, but also to an ATM (automated teller machine) of a chosen bank. Classes of this package are used for downloading and parsing a file with ATM.
- **Datatypes** – contains classes that represent custom data types. These classes mostly consist of properties and only a few methods.
- **Geography** – its classes are used for converting geographic coordinates into address in the text form and vice versa.
- **Route** – implements algorithms for getting a route, parsing the answer and initialization of a `Route` object. A custom map overlay, and implementation of the method of drawing it on the map, is included as well.

5.1.3 ARDS Class

All properties and methods of this class are defined as `static`. This approach makes it possible to share important data within the whole application without a need of creating an instance of `ARDS`. Its content can be divided into three following categories.

Values of **variables** are changed during the runtime, e.g. the current or the destination address or geographic coordinates, boolean flags. ARDS also includes a route object and a chosen ATM object as a navigation destination point. The application always stores only one route and one or more ATMs (the list of ATMs situated within a specified radius and a chosen ATM). Figure 5.3 is very simplified and shows only the most important relationships.

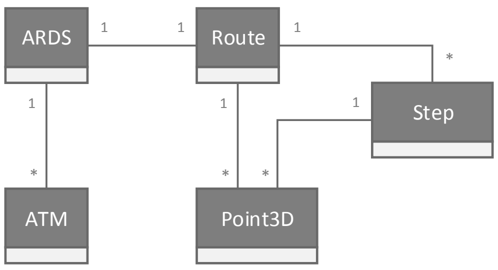


Figure 5.3: Relationship between `ARDS` class and obtained route/ATM.

The **methods** of ARDS are called from various parts of the application. Important methods are:

- `showAlertDialog` – the most widely used method that shows alert dialog with given title and text. An alert is shown mostly in the form of warning, e.g. when the destination address cannot be converted to geographic coordinates.

- `showRouteDetailsDialog` – displays information about route, its start address, destination address, length in meters and warning. If the application is navigating to a specific ATM, its name and its operating ours are added.
- `showNavigateDialog` – opens a dialog window with text boxes for start and destination address. The method is also used for converting the addresses to the corresponding coordinates. If one of them cannot be converted, an alert dialog is displayed. If both addresses are converted successfully, the request for the route is sent to Google Directions API. The sending is implemented as an asynchronous task and its result is sent to the activity, which has invoked this method.
- `showNavigateToATMDialog` – shows a dialog window for setting parameters of the ATMs lookup. The method converts the given address to GPS coordinates in the first place. A list of ATMs of a particular bank is downloaded in asynchronous task after successful conversion. A chosen ATM is stored and the result of the whole process is sent to the activity, which has invoked this method.
- `showATMlistDialog` – displays a list of ATMs found within the given radius, only one ATM can be chosen from the list.
- `stopNavigating` – resets all variables that are used for navigation, including the application state.
- `distance` – calculates direct distance between two geographic locations with *haversine formula*[1] (described in subsection 5.3.2).

The **constants** represent states of the application (not navigating, navigating, navigating to ATM), map overlay types or various numeric constants (e.g. Earth equatorial radius).

5.1.4 Application Activities

`ARMapActivity` – the first activity to be displayed, when the application is started. It contains a map view on which overlays can be added. This activity does not use any of the library classes.

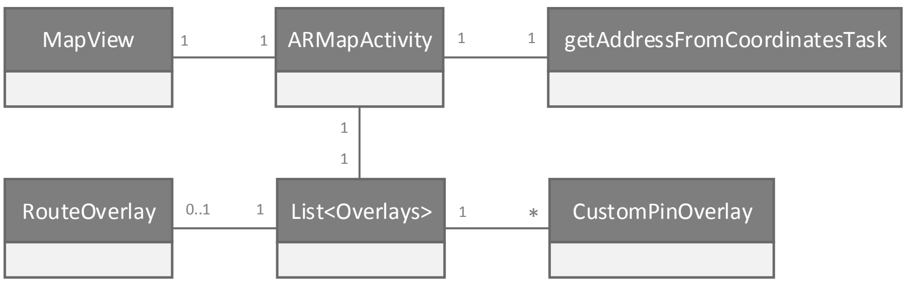


Figure 5.4: Map activity ER diagram.

[1]See http://www.movable-type.co.uk/scripts/latlong.html.

`ARCamActivity` – the most important activity that displays the augmented reality view, thus overlaying the camera view with the OpenGL ES layer. This activity uses most of the classes from the designed library, which are displayed as white in figure 5.5.

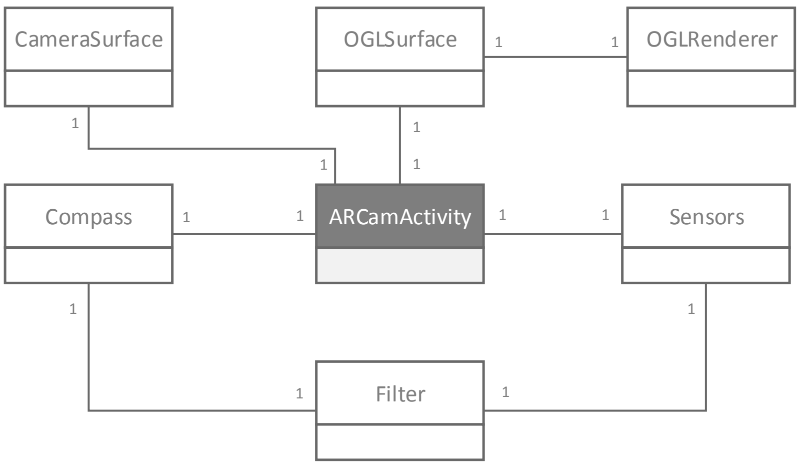


Figure 5.5: Camera activity ER diagram.

`SettingsActivity` – this activity allows the user to set walking/driving mode, color and width of displayed route and change parameters of filters for the data received from the positioning sensors of the device. It also has its own OpenGL ES surface and renderer for demonstration of the sensor noise level.

The OpenGL ES surface is an instance of `CalibrationSurface` class from the designed library. This class extends the Android `GLSurfaceView` class, but does not add any new properties or methods. The OpenGL ES renderer (`CalibrationRenderer`) is one of its properties, which renders three axes of the OpenGL ES Cartesian coordinate system (x is red, y is green, z is blue). This view is used for demonstration of the current noise level of the sensors.

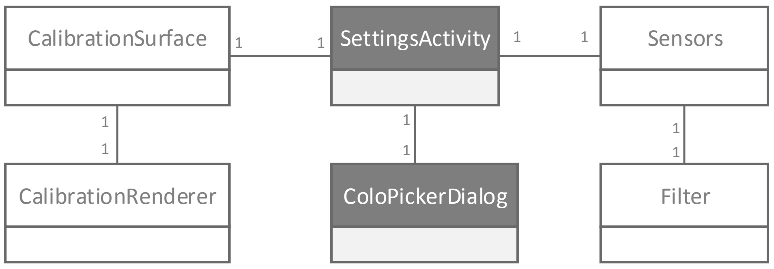


Figure 5.6: Settings activity ER diagram.

5.2 Obtaining Geographic Coordinates

The exact geographic coordinates of the device are very important for location–based augmented reality, as well as for navigation. In the first place, it is necessary to request the user's permission by declaring `ACCESS_FINE_LOCATION` in the manifest file of the applications. These coordinates can be obtained either from GPS provider or Android's Network Location Provider[2], if it is not possible to obtain them from GPS (e.g. in buildings). Although GPS is the most accurate, it works only outdoors, what on the other hand is not a big problem, since this application is used for navigation. Network provider determines the position of the device using cell tower and Wi–Fi signals. This approach works both indoors and outdoors. One of the GPS disadvantages is, that it consumes battery power more quickly in comparison to network provider. GPS also needs longer time period to obtain first fixed position because of connecting to satellites, but once it is connected, the position updates are fast.

The developed demo application uses only GPS at the cost of shorter battery life. The application checks if GPS is turned on during the startup and in case it is not, a dialog window is displayed with a question, if GPS should be turned on. If yes, an activity containing Android location access settings is started. In case of "No" answer, the application is shut down. If the first location is not obtained within 30 seconds, an alert dialog window with a warning is displayed, but listening for location continues.

An instance of `LocationManager` class registers the listener and it requests location updates. The listener is an instance of `LocationListener` class. It receives new location every $5s$ or when the minimum distance between updates (5m) is passed. New location does not have to be necessarily new. It could be still the same, so the listener compares the last location with the new one – if latitudes or longitudes are different, current position in `ARDS` is updated and new coordinates are converted to new address in the text form with `Geocoder` class. This conversion is implemented as an asynchronous task, which sends a message with obtained address to a specified handler, as soon as it is finished. An address in the text form is obtained only for current location, so it can be displayed in a bubble over the current position overlay.

5.3 Obtaining Navigation Information

In order to navigate from one location to another one, a route between these two locations must be calculated. The start address and the destination address have to be set in a dialog window (Fig. 5.10) in the text form. These addresses are converted to corresponding geographic coordinates using the Android `Geocoder` class. As "coordinates to address conversion", this conversion is also implemented as an asynchronous task. When the task is finished, it sends the obtained coordinates to a specific handler. Since there are two addresses for conversion, a route can be calculated after both of the addresses are successfully converted to corresponding geographic coordinates. A service Google Directions API, that calculates directions between locations using an HTTP request, is used for the purpose of calculating a route. This service is not designed to respond in real time, so because of usage limits the application requests for route only once, when the navigation is started. The usage limit is 2500 directions requests per day. Since the request is an HTTP request, it is necessary to have an active network connection. On the startup the application checks,

[2]See `http://developer.android.com/guide/topics/location/strategies.html`.

if the device is connected to network. If it is not, an alert is displayed and the application is closed. The process of downloading and parsing a route also runs separately from the main GUI process as asynchronous task, in order not to overload the main thread.

The URL of the HTTP request has always the following form:

$$http://maps.googleapis.com/maps/api/directions/json?parameters\ ,$$

where *json* indicates output in JSON and the *parameters* are:

- **Starting location** – *origin=latitude,longitude*
- **Destination location** – *destination=latitude,longitude*
- **Sensor** – *sensor=true* (indicates presence of the location sensor on the device)
- **Mode** – *mode=walking* (can be changed to *driving* in settings)
- **Unit system** – *units=metric* (can be changed to *imperial* in settings)

The parameters are separated by & character. Google Directions API offers more optional parameters (e.g. the language of result, alternative routes, . . .) and an alternative output format in XML, but for purposes of this application there is no need to use them.

5.3.1 Route Object

When the URL for Google Directions API is assembled, it is passed as an argument to `RouteParser` class, where the route and its details in JSON are downloaded as a stream. It is then converted to string, which is used as a parameter of `JSONObject` class constructor. Based on this string, the `Route` object is created and filled with data.

Downloaded route consists of a single `leg`, which contains an array of `steps`. One step is the smallest part of the route, containing a specific instruction, distance and points of polyline representing the shape of this step. Figure 5.7 shows the properties of the downloaded route.

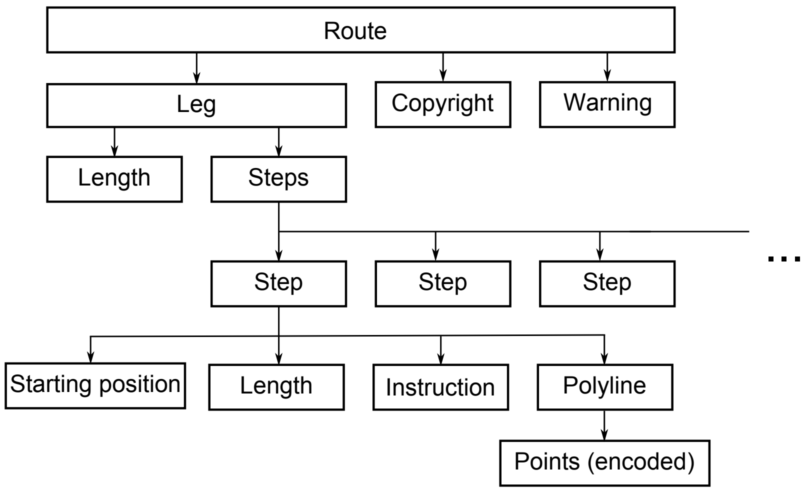


Figure 5.7: The structure of downloaded route (important properties).

The JSON representation of the route is "converted" to the route object. Firstly, route properties copyright and warning (if there is some) from the route and then steps and length from the leg are obtained. The most important property of the leg, especially for visualization of the route, is an array of the steps (`Step` objects). Each step in the array contains start location, its length, an instruction and a polyline. Polyline is an object holding an array of encoded points. These points are decoded[3] for each step of the route in order to obtain them in the form suitable for visualization.

5.3.2 Navigation to Automated Teller Machine

As an extension of the basic navigation functionality I have also implemented a possibility of being navigated to a specific ATM. Instead of setting a specific destination address, it is necessary to choose an ATM as a destination point. The application supports ATMs of only one bank.

It is necessary to set the start address, choose a bank and set radius, in the first place. Given radius determines the circular area of the ATMs lookup. Based on the chosen bank, the application chooses the URL of the KML file. The file contains information about ATMs of the bank.

The file–download task is implemented as an asynchronous task. The file is in separate thread parsed as an XML file using a DOM[4] parser after successful download. Information obtained from the file are the name, the address and the geographic location – properties of one `ATM` object. A product of the parsing is a list of ATMs, that is subsequently reduced to a shorter list, which is displayed in a dialog window. One item of this new list represents one ATM object, distance of which is less than the previously set radius. The distance is calculated with the haversine formula:

$$\begin{aligned} a &= \sin^2(\Delta\varphi/2) + \cos(\varphi_1)\cdot\cos(\varphi_2)\cdot\sin^2(\Delta\lambda/2) \\ c &= 2\cdot\operatorname{arctan2}(\sqrt{a}, \sqrt{(1-a)}) \\ d &= R\cdot c \end{aligned} \tag{5.1}$$

where φ is latitude, λ is longitude (angles are in radians), R is earth mean radius ($6378137m$) and d is the resulting distance. Then geographic coordinates of the chosen ATM are used as a destination location and the route is calculated the same way as it is described in the previous subsection 5.3.1.

5.3.3 Conversion of Geographic Coordinates

When the route is calculated, it contains GPS locations of polyline points representing the shape of the route. The points in this form are suitable only for displaying the route as a map overlay. In order to render the route with OpenGL ES, these points are converted to 3D points.

The position of the device is always considered to be the origin of the OpenGL ES coordinate system. It is necessary to calculate only x and y coordinates, since z coordinate represents the altitude, which can be obtained only from GPS (network provider does not provide information about the altitude). Because of this, the altitude is always set to zero.

[3]The algorithm for decoding the polyline is from http://jeffreysambells.com/2010/05/27/decoding-polylines-from-google-maps-direction-api-with-java.

[4]Document Object Model.

x and y coordinates are calculated with following formulas:

$$\begin{aligned} x &= (pointLon - deviceLon) \cdot (\pi \cdot r/180) \cdot \cos(pointLat) \\ y &= (pointLat - deviceLat) \cdot (\pi \cdot r/180) \end{aligned} \tag{5.2}$$

where latitude and longitude values have to be in degrees and r is the earth equatorial radius in meters. The differences between the point latitude/longitude and the current location latitude/longitude are calculated and converted to meters (1° = half of the equatorial circumference divided by 180). Calculated x coordinate is then multiplied by cosine of the point latitude in radians. This modification is done with respect to the fact, that the Earth is not flat.

5.4 Device Orientation Calculation

The process of calculating a device orientation in real world is essential for the location–based augmented reality. This process can also be called registration. `Sensors` object from designed library uses filtered data from accelerometer and magnetometer to calculate a rotation matrix for OpenGL ES renderer and azimuth for compass. The values are sent to the camera activity every $10ms$.

5.4.1 Filtration of Sensor Data

New acceleration and magnetic field values are filtered in order to reduce the noise produced by the sensors immediately after they are obtained. A simple one dimensional variant of Kalman filter is used for filtration.

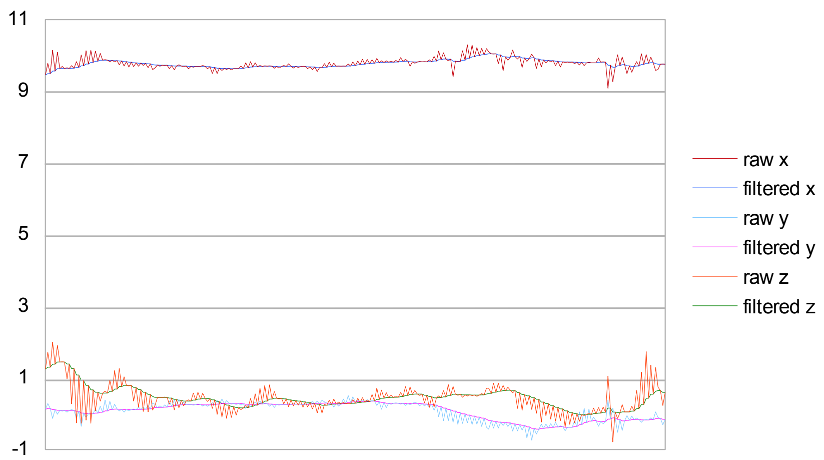
Each sensor has its own instance of `Filter` class from the designed library. The process noise and the sensor noise can be modified during the runtime of the application, because suitable values of these variables can differ for different devices. If the user sets the sensor noise too high and the process noise too low, movement of virtual objects is smooth, but slow and positions of these objects may also be less accurate.

Every change of the variable is sent as a message for a handler property of the `Sensors` object in the settings activity, where filters for both sensors are reset. New values are also stored in `ARDS` class, because when the camera activity is resumed, these values are sent to its own `Sensors` object to reset the filters. The higher these values are, the lower the noise is, but at the cost of slower reactions of rendered objects.

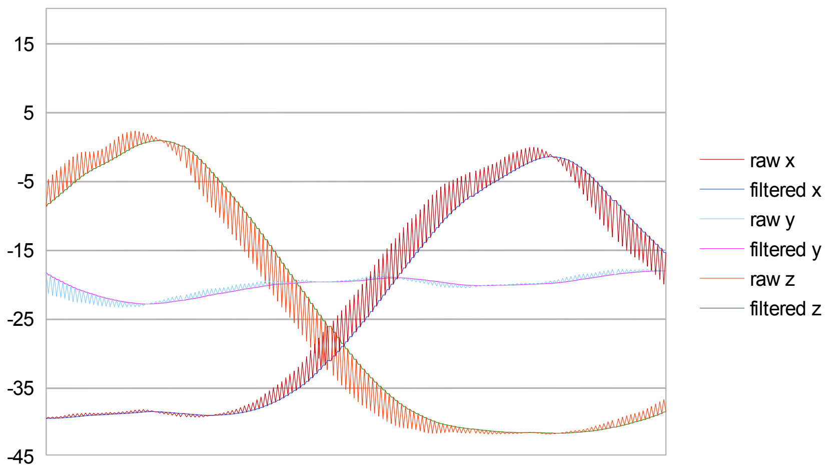
A simple thresholding is applied on the filtered values to eliminate the influence of very small shakes. It means, that if the difference between old and new values is less than a specific threshold value, new orientation is not calculated. Instead of new orientation values, the old ones are used. Comparison of filtered and unfiltered values for accelerometer are displayed in figure 5.8(a) and for magnetometer in figure 5.8(b). The device used for collecting displayed data, Google Nexus 7 tablet, has been slightly shaken and rotated. The process and sensor noise values were default, i.e. the sensor noise was 0.0625 and the process noise was 4.0.

5.4.2 Rotation Matrix and Azimuth Calculations

Filtered acceleration and magnetic field values are used as arguments of `getRotationMatrix` method from Android `SensorManager` class. The first argument of the method is an output



(a) Accelerometer data.



(b) Magnetometer data.

argument `R`, in which a 4×4 rotation matrix is returned. This matrix has the following form:

$$\begin{pmatrix} M_{11} & M_{12} & M_{13} & 0 \\ M_{21} & M_{21} & M_{23} & 0 \\ M_{31} & M_{32} & M_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.3}$$

This matrix should be transposed since the OpenGL ES matrices are column–based, but its transpose is also its inverse. It means that obtained rotation matrix can be used with OpenGL ES without transposing it. The method `remapCoordinateSystem` is used for the recalculation it for the OpenGL ES coordinate system. The X axis of the device is mapped on the world Y axis and the Y axis of the device is mapped on the world $-X$ axis.

Rotation matrix values are used for the calculation of azimuth with following formula[5]:

$$azimuth = 180 + \arctan 2(M_{13},\, M_{23}) \cdot \left(\frac{180}{\pi}\right) \tag{5.4}$$

The calculated azimuth is used for rotation of the compass layout. Android also offers the possibility of getting data from the orientation sensor, which directly returns azimuth, but this value does not rotate the compass layout correctly.

5.5 Realization of Augmented Reality View

The augmented reality view is displayed by the `ARCamActivity`, which does not have its own XML file for the GUI layout definition. The layout is defined in `onCreate` method of the activity. Objects representing three layers – camera surface, OpenGL ES surface and compass are initialized and stacked in the same order as they are listed. The activity needs previously described operations to be done before it is able to display the augmented reality view correctly.

5.5.1 Camera Surface

The first (bottom) layer is the camera surface, which displays a video stream from the camera of the device. Firstly, the presence of the camera is checked and then an instance of Android `Camera` class is created by the static method `getCameraInstance` of `CameraSurface` class from the designed library. The field of view is obtained as a camera parameter and a renderer's property for perspective projection is set. It is also very important to free created camera instance, when the activity is paused or stopped, because the camera hardware is not shareable.

5.5.2 OpenGL ES Surface

When the rotation matrix is calculated, it is sent as a message for a handler in the camera activity. Then the handler checks whether the application is in navigating state. If it is, the rotation matrix property of the OpenGL ES renderer is updated and invalidated (invalidation makes the renderer to redraw).

The renderer runs in separate thread and its background must be transparent in order not to cover the camera surface. It initializes the OpenGL ES, when it is created. On every surface change, e.g. landscape mode $\leftrightarrow$ portrait mode, renderer sets a new viewport and perspective. View size and display orientation are always the same in this application, so the viewport and the perspective need to be set only once. The field of view of the camera (default value is 45°), the ratio of width and height and near and far clipping plane are set with `gluPerspective` method.

In `onDrawFrame` method the frame is cleared and subsequently the rotation matrix is used to adjust the direction of the OpenGL ES camera view by `glMultMatrixf` method. Then the color of the route is set. The route is represented by the line–strip, which is extended to the triangle–strip. Problem of this approach is that the triangle–strip is planar, so not all route steps are always visible. This effect is partly suppressed by translating the virtual layer by $1.5m$ down.

[5]See http://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions.

The end of the route is marked with a green reversed pyramid with a red top, which is rendered as four triangles. Size of the pyramid depends on distance between the user and the end point of the route.

5.5.3 Compass

The compass layout is the top layer and it is not created in the application. It is a bitmap added to applications resources, that is placed in the left upper corner of the augmented reality view.

The `Compass` class extends the Android `View` class and adds a bitmap and azimuth properties. In the overridden `onDraw` method, a bitmap is drawn and rotated by azimuth value.

5.6 Realization of Map

The demo application also includes a map view displaying the current geographic location of the device as a map overlay and it can be used for navigation as well. In order to display a map correctly, the Maps Library must be declared in the manifest file, because it is not a part of the standard Android library.

Firstly, `MapView` need to be defined in the activity XML layout file. The most important attribute of the view is an `apiKey` attribute which holds the Google Maps Android API v1 Key for the application. This is required in order to receive the map data. A reference to this view is obtained in `ARMapActivity`. The map view object is used for adding overlays and the controller instance is used for zooming. Zoom level is calculated from all overlay positions so all of them are visible. Zoom level can also be changed manually with zoom controls, which are added to the map or by a zoom gesture.

5.6.1 Map Overlays

Map overlays are small graphic layouts placed in specific geographic locations on a map. An overlay can be used to highlight some important locations, to add information to a map, etc. An overlay can be represented by a simple image without any functionality or it can respond to a touch by performing a specified action, e.g. displaying a bubble layout with information about the touched overlay. In order to achieve this functionality, new classes extending the existing Android classes (Fig. 5.8) were created to customize them. One overlay can contain more than one item to be displayed, every item of one overlay has the same graphical layout.

Class `PinpointItem` extends Android `OverlayItem` class, but it does not add any new properties or methods.

Properties of `BubbleItemizedOverlay`, which extends the Android `ItemizedOverlay` class, are: an array of `PinpointItem` objects to be displayed and a graphic layout of a bubble for the case of touch events, which are handled by the class itself. A bubble layout is an instance of the `BubbleOverlayView`, which extends the Android `FrameLayout` class.

All of these classes are then wrapped into `CustomPinOverlay` class, which extends the `BubbleItemizedOverlay` class. This wrapper class adds also an overlay type and a method for calculation of the map zoom level. The calculation is based on geographic locations of all overlay items. In the application there are two overlay types with this structure, an overlay for the current position (blue circle) and an overlay for one waypoint of the route (red flag).

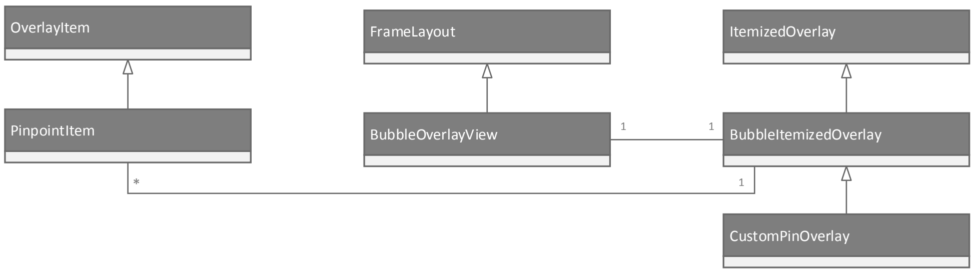
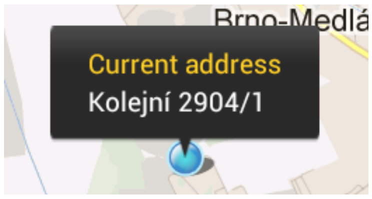


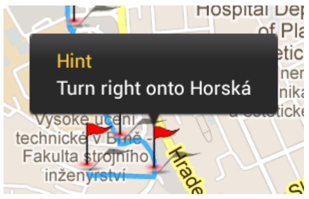
Figure 5.8: Structure of map overlay classes.

Three different types of overlay can be displayed on the map:

- **Current position overlay** – on the map, there is always displayed only one current position overlay and it is always visible, even if the application is not navigating to any destination. It is a small blue circle displayed on the map (Fig. 5.9(a)) and it represents the current geographic location of the device.
- **Waypoint overlay** – unlike current position overlay, waypoint overlay is displayed only if the application is navigating to a specific destination. A waypoint is displayed as a small flag (Fig. 5.9(b)) and it represents one point of the route. Usually it is a point, where the direction should be changed, like a turning. Since the direction of the route usually changes more than once, more than one flag can be displayed on the map.



(a) Current position overlay.



(b) Route waypoint overlays.

- **Route overlay** – Route overlay is different from the current position and waypoint overlays. `RouteOverlay` extends Android `Overlay` class and adds new properties, which contain important information about the graphical representation of the route overlay (points, color, drawing tools). Unlike the other two overlay types, the route overlay does not react to touch events, it is a simple polyline drawn over the map view (Fig. 5.9).

5.7 Graphical User Interface of Designed Application

The graphical user interface of the developed application is very simple and it was designed with respect to minimalistic controls. Every activity is displayed in a landscape mode and it cannot be switched to a portrait mode. The application has three different activities.

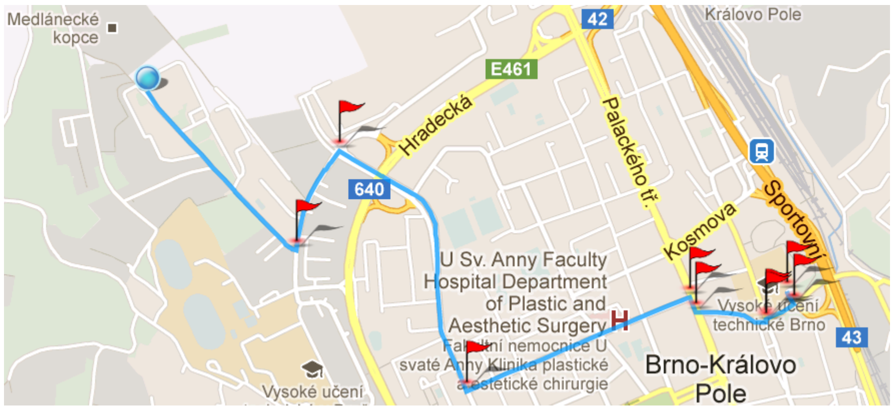
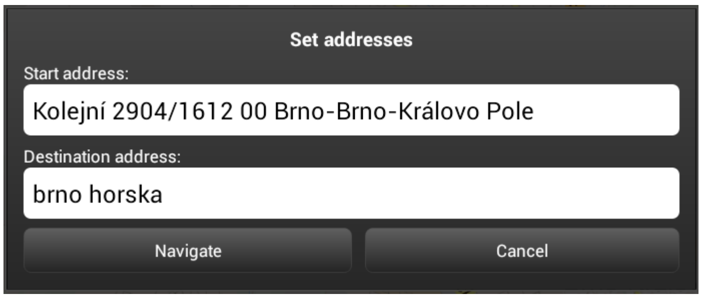

Medlánecké kopce
Hradecká
E461
42
Královo Pole
Palackého tř.
640
Sportovní
Kosmova
U Sv. Anny Faculty Hospital Department of Plastic and Aesthetic Surgery
Fakultní nemocnice U svaté Anny Klinika plastické a estetické chirurgie
Vysoké učení technické Brno
43
Brno-Královo Pole
Vysoké učení

Figure 5.9: Route overlay.

5.7.1 Map Activity

The **map activity** is the first activity to be displayed on startup. The current geographic position overlay is added to the map as soon as the current geographic location is obtained (Fig. 5.12). The controls are implemented as an options menu with five items:

- **Camera**, **Settings** – used for starting camera activity or settings activity respectively.
- **Navigate** – item displays a dialog for setting the start and the destination address (Fig. 5.10) and it is used for starting the navigation.


Set addresses

Start address:

Kolejní 2904/1612 00 Brno-Brno-Královo Pole

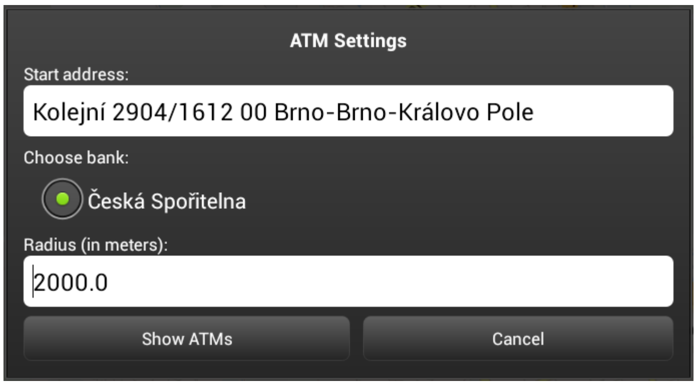
Destination address:

brno horska

Figure 5.10: Dialog for setting route address.

- **Navigate to ATM** – item displays a dialog for setting the parameters of the ATMs lookup (Fig. 5.11) and it is used for starting the navigation to a specific ATM. The start address is automatically filled with the current address of the device in the text form. There is only one bank that can be chosen and the radius is implicitly set to 500m.
- **Stop Navigation** – clears objects used for navigation and stops the navigation.

In the right upper corner (Fig. 5.12) is situated a single button (button is also in camera activity, where it is created dynamically) used for displaying a simple window with information about the route. The information contains the start and the destination


ATM Settings

Start address: Kolejní 2904/1612 00 Brno-Brno-Královo Pole

Choose bank: Česká Spořitelna

Radius (in meters): 2000.0

Figure 5.11: Dialog for setting parameters of ATMs lookup.

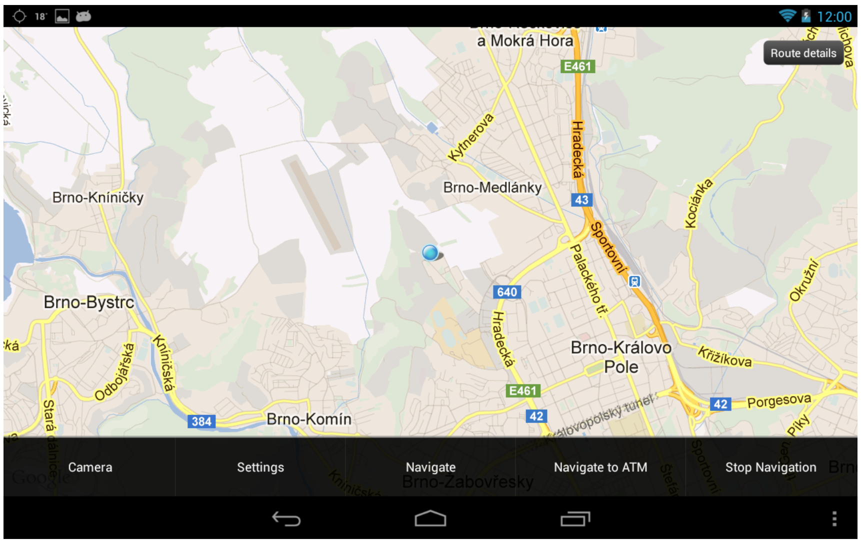


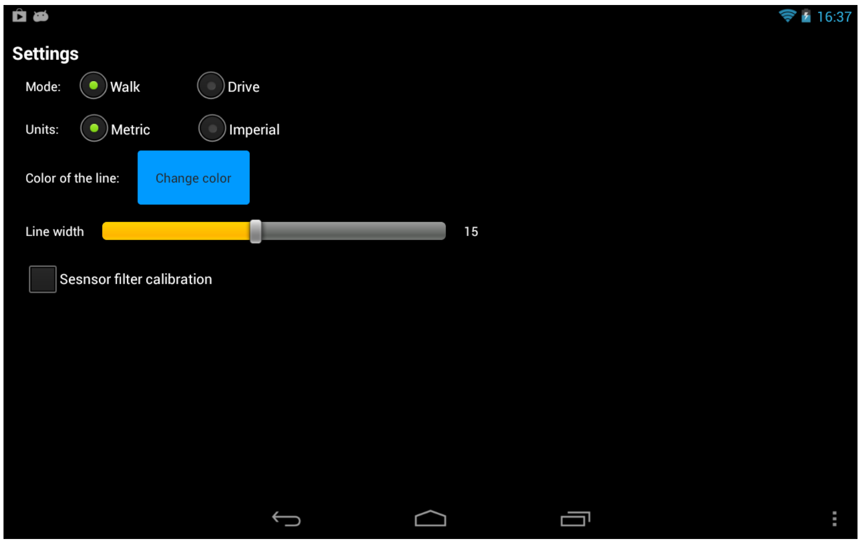
Figure 5.12: Map activity.

address, length of the whole route in meters and a warning, e.g. “Walking directions are in beta.”.

Map activity also contains touchable map overlays. When a circle, representing the current position, is touched, a bubble with the current address is displayed (Fig. 5.9(a)). When a flag is touched, a bubble with the specific hint is displayed (Fig. 5.9(b)). Both the bubbles can be canceled by touching them. Only one bubble can be displayed at a time, so if another overlay is touched, its bubble appears and the previously displayed bubble is hidden. If the map is moved, all the displayed overlays and the visible bubble move with the map, staying in the same geographic locations.

5.7.2 Settings Activity

The **settings activity** has usual user interface (Fig. 5.13) consisting of common controls (radio buttons, sliders and a checkbox) completely defined externally in XML layout file. These controls are used to set the options of the application:


16:37

Settings

Mode: [✓ Walk] Drive

Units: [✓ Metric] Imperial

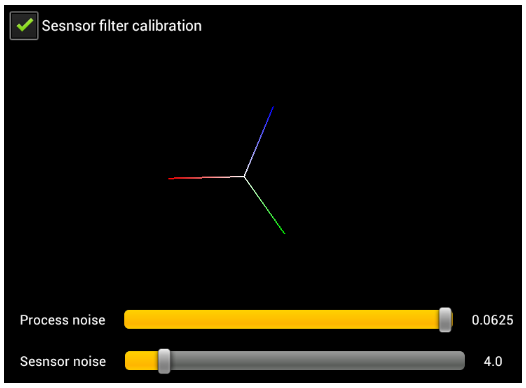
Color of the line: Change color

Line width 15

Sesnsor filter calibration

Figure 5.13: Settings activity.

- **Mode, Units** – parameters of the HTTP request (URL), which is used with Google Directions API to calculate the route.
- **Color and width** – visual parameters of the displayed route in the map activity. When the *Change color* button is touched, a color picker dialog is displayed[6]. The color of the button is then changed to the chosen color. The width affects only the route overlay on the map.
- **Filter calibration** – used for adjusting the parameters of filter (Fig. 5.14). This option is available after checking the "Sensor filter calibration" checkbox. The filter settings consists of an OpenGL ES view and two sliders, one for each modifiable variable. Available process noise values are $\{2^{-x} \mid x = 0..4\}$ and available sensor noise values are $\{1..32\}$. The higher these values are, the lower the noise is, but at the cost of slower reactions of rendered objects (the movement is smoother, but slower).


✓ Sesnsor filter calibration

Process noise 0.0625

Sesnsor noise 4.0

Figure 5.14: Filter settings.

[6]This dialog was taken from an Android sample project.

5.7.3 Camera Activity

The user interface of the **camera activity** is very similar to the interface of the map activity. The controls are implemented as an options menu with five items too, with only one difference – the first item is **Map** instead of **Camera** (Fig. 5.15). The compass layout is situated in the left upper corner of the activity.

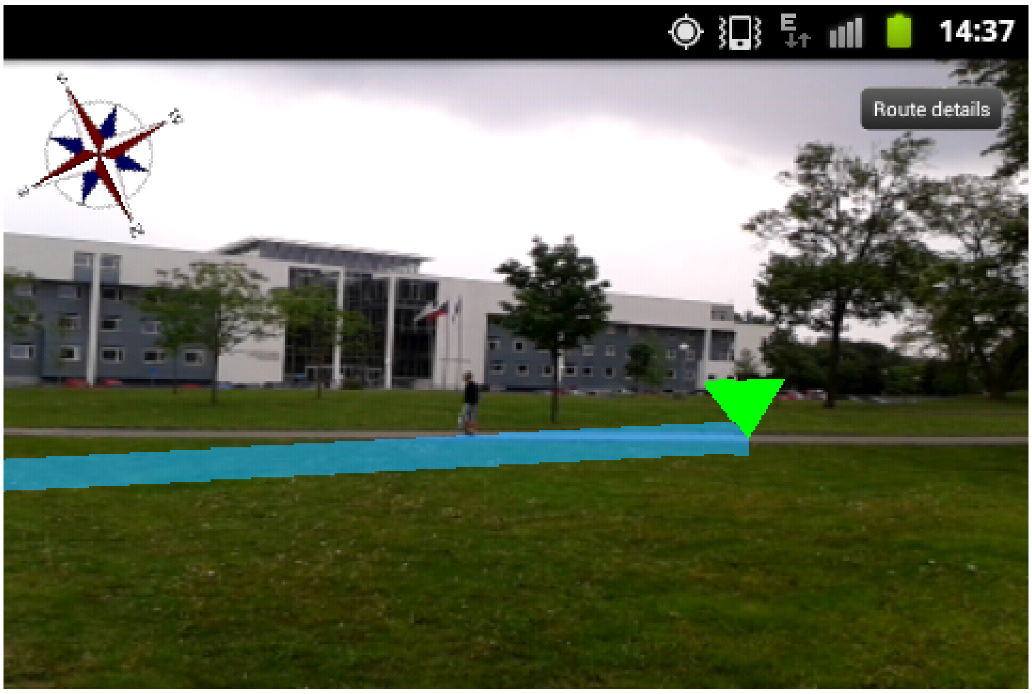
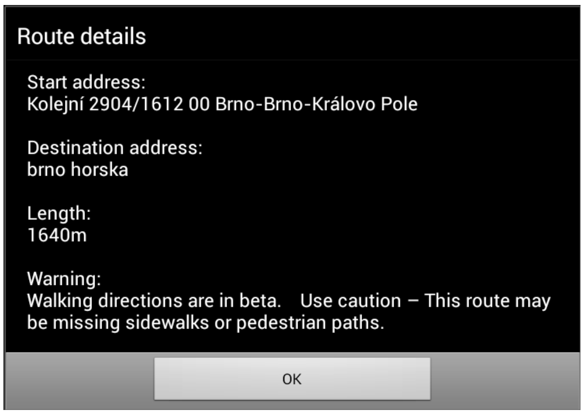


Figure 5.15: Camera activity.

In the right upper corner of the activity there is the *Route details* button. When the application application is navigating and the user touches the button, then a dialog window with information about the current route is displayed (Fig. 5.16). In the map activity there is the same button with the same functionality.



Route details

Start address:
Kolejní 2904/1612 00 Brno-Brno-Královo Pole

Destination address:
brno horska

Length:
1640m

Warning:
Walking directions are in beta.   Use caution – This route may be missing sidewalks or pedestrian paths.

Figure 5.16: Route detail.

5.8 Testing of Implemented Augmented Reality System

In fact, testing was being performed during the whole development of the library and the application. At the beginning of the development, the developed parts were tested on the emulator. I have also used real devices for testing:

- Google Nexus 7 tablet (Android 4.2.2 Jelly Bean) – provides the most noisy data from the positioning sensors.
- Lenovo Thinkpad tablet (Android 4.1 Jelly Bean)
- Samsung Galaxy Ace 2 (Android 2.3.6 Gingerbread) – provides the least noisy data from the positioning sensors.

Testing the application with real device is more efficient than testing it on the emulator. The emulator is significantly slower than a real device and it has enabled me to test only the map activity and only a part of the settings activity, since it does not emulate sensors and camera by the default.

The most important tool used during the development was a debugger in combination with DDMS[7]. DDMS provides thread and heap information of the device and enables to send GPS coordinates to emulator or emulate an incoming call or message. It can also be used for tracking the events (logs) of the emulator/device during the runtime of the application. These events are displayed in the text form and they can have different meanings, e.g. information or exceptions. In case of the exception occurrence, additional information, such as which part of the application threw the exception and what was the reason, are provided too. A developer can create own various types of logs with specific tag and text, used especially for debugging purposes (end user has no access to the logs, unless he uses e.g. the *LogCat* application).

Some parts of the application were tested as separate units (small applications):

- **Positioning sensors** – a separate application for collecting data from sensors was implemented. Purpose of its realization was to get familiarized with accelerometer, magnetometer and orientation sensor of the device. The orientation sensor is not used in the final application. This application was also used for testing filters. A simple low–pass filter and one–dimensional Kalman filter were implemented (Kalman filter is used in the final application). This separate application was also used for drawing a simple plot in order to see results of the filtration process. The original and filtered data were stored in text files. One file contained e.g. a set of values of the accelerometer X axis collected during the runtime. The values from these files were used to create the plots in the subsection 5.4.1.
- **OpenGL ES** – another small application was used for testing simple OpenGL ES rendering. Firstly, only a simple rendering of lines and triangles was implemented (in order to understand the principles). This application was subsequently merged with the previous one in order to use the rotation matrix from sensors with OpenGL method `glMultMatrixf`. After successful merging the Kalman filter was chosen as a filter, because its results were better (visually) – the rendered primitives lagged less. The camera layer and the process of obtaining the geographic location were added in further work.

[7]Dalvik Debug Monitor System, see http://developer.android.com/tools/debugging/ddms.html.

- **Map** – this application was created as a base for the whole augmented reality system. Firstly, displaying of the map and overlays, and then obtaining and displaying a route were implemented and tested. Every other functionality (ATMs, settings and camera activity using the library) was implemented within this application. Active network connection was needed to test maps and obtaining a route, so it could be tested on emulator and, with limitations, outdoors. When testing outdoors, the main subject for testing was accuracy of GPS – it is much more accurate than network provider.

One of the most important parts was **outdoor testing**, since it is a navigation application. The application can be started indoors too, but the possibility of obtaining location data from GPS is very limited or not possible.

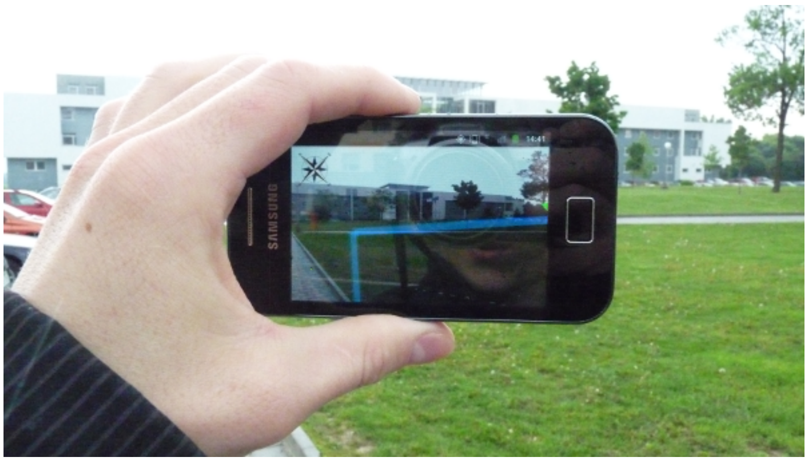


Figure 5.17: Outdoor test of the application.

I tested the application in the streets and the main objectives, on which I focused, were accuracy of the registration, accuracy of the GPS, influence of the noise from sensors on stability of the rendered route, time of obtaining the route and correctness of downloaded data. A lack of registration accuracy is captured in figure (Fig. 5.17), where the position of the right turning is not absolutely correct.

5.9 Summary of Achieved Goals and Future Work

I have designed and implemented the application, which navigates the user to a specified destination. The current position of the device is obtained from GPS and the destination point is specified by the user. The navigation itself is realized as **augmented reality**, which means, that the route is drawn over the camera view. In order to achieve this functionality, the application uses data from the device accelerometer and magnetometer to calculate the orientation of the device in the real world. This process is also called **registration**. These values are filtered with filter, which can be modified during the runtime. The information about the route is received from Google Directions API. As extensions I have added the possibilities of using Google Map for navigation and using a specific ATM as a destination point. Process of obtaining the route is usually very fast, while downloading and parsing the KML file with information about ATMs can take a few seconds.

The current geographic position of the device is usually not very accurate right after startup, if it is from the network provider. When the communication with GPS is stabilized, obtained locations are much more accurate. If the locations were obtained strictly from the network provider, the accuracy would decrease rapidly.

The registration process is accurate enough (not absolutely accurate) to display the route in the right place of the camera view. Waypoints of the route correspond almost correctly to their real world alternatives. However, a displacement of the route occurs during the runtime. This may be caused especially by the deflections in the calculation of the orientation or when this calculation is done with too noisy data from the sensors. Sharp shakes or incorrectly set parameters of filters can cause similar problems too. Another reason could be an influence of large metallic objects and magnetic fields in the surroundings of the device, that may cause a distortion of the magnetic field values measured by the device magnetometer. A little displacement of the points may be caused by the fact, that Google Directions API does not provide any information about the altitude of the route points, so all the 3D points are placed in the ground plane.

The implemented library and application can have much more functionality. Based on testing the application (outdoors as well) and its parts, I have made observations and I have defined the main **limitations and shortcomings** of the realized solution:

- High battery consumption.
- Not fully stabilized and accurate virtual layer.
- The requirement of an active network connection.
- Not all devices have the required hardware.

and possible **future extensions and updates**:

- Possibility to choose from more banks, not only one. ATMs can be stored in database on a server and the application will only send HTTP request to the server, which would do all the computations and send back a result.
- Possibility of navigation to more points of interest (POI), e.g. gas stations or parking lots.
- Displaying of the POIs within the specified radius.
- Improved GUI, e.g. add a map fragment in the camera activity (requires Google Maps API version 2 instead of currently used version 1), displaying the additional navigation information – speed, estimated arrival time, etc.
- Waypoints as "touchable" 3D objects in the camera activity.
- Improved way of recalculating a route.

Chapter 6

Conclusion

The main objectives of this thesis were to acquaint with the principles of augmented reality, especially with its realization for Android mobile operation system, and develop a location–based augmented reality system using GPS and positioning sensors. This system will draw a virtual route over a layer represented by a camera view, thus displaying this route in the right place. The implemented system will consist of a library and a demo application using library functions to create augmented reality view.

I designed the library with respect to its possibilities of being reused in other location–based augmented reality applications. It contains methods for obtaining data from the sensors of the device, creating layers, which represent the augmented reality view. The demo application is, in fact, a navigation, which displays a specific route from one geographic location to another one. I also added a possibility of displaying the route on a standard two–dimensional map. Map data are provided by Google Maps API and information about the route is provided by Google Directions API. I tested the application in the streets and I focused on the accuracy of the augmented reality view, the influence of changing the parameters of the filter and obtaining the route directions. Points of the virtual route correspond to their real world alternatives (geographic coordinates) accurately enough to accomplish the specified objectives.

I have gained valuable experience creating this thesis, especially in the field of augmented reality and Android applications development. I have learned about the possible usage of augmented reality in mobile devices and principles of its realization, and improved my programming abilities. It involves improvement of Java language and OpenGL ES 1.0 programming and techniques of Android development, such as using Google Maps/Directions API. The work on the thesis will continue so I can publish it on Google Play. The possibilities of future improvements are mentioned in the previous chapter.

Bibliography

[1] Displaying Graphics with OpenGL ES. [online], [cit. 2013-1-4].
URL http://developer.android.com/guide/topics/graphics/opengl.html

[2] Geometric Camera Parameters. [online], [cit. 2013-5-20].
URL http://www.cse.unr.edu/~bebis/CS791E/Notes/CameraParameters.pdf

[3] Magnetic Sensors. [online], [cit. 2013-4-26].
URL http://www.magneticsensors.com/compassing-solutions.php

[4] Magnetometer. [online], [cit. 2013-4-26].
URL http://www.vectornav.com/support/library?id=83

[5] OpenGL ES – Common Profile Specification. [online], [cit. 2013-4-26].
URL
http://www.khronos.org/registry/gles/specs/2.0/es_full_spec_2.0.25.pdf

[6] OpenGL ES 2X – The Standard for Embedded Accelerated 3D Graphics. [online], [cit. 2013-4-1].
URL http://www.khronos.org/opengles/2X/

[7] BICHLMEIER, C.; WIMME, F.; HEINING, S.; et al.: Contextual Anatomic Mimesis Hybrid In-Situ Visualization Method for Improving Multi-Sensory Depth Perception in Medical Augmented Reality. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, November 2007, p. 129–138.

[8] BIMBER, O.; RASKAR, R.: *Spatial Augmented Reality*. Indianapolis: John Wiley & Sons, Inc., 2005, ISBN 978-1-118-19954-1, p. 1–8.

[9] BRADSKI, G.; KAEHLER, A.: *Learning OpenCV*. O'Reilly Media, prvé edition, 2008, ISBN 978-0-596-51613-0.

[10] CARMIGNIANI, J.; FURTH, B.; ANISETTI, M.; et al.: *Augmented reality technologies, systems and applications. Multimedia Tools and Applications*, volume 51, no. 11042, December 2010: p. 341–377, ISSN 1573-7721, [online], [cit. 2012-11-24].
URL http:
//link.springer.com/article/10.1007/s11042-010-0660-6/fulltext.html

[11] DUBROFSKY, E.: Homography Estimation. 2009, [online], [cit. 2013-1-1].
URL
https://www.cs.ubc.ca/grads/resources/thesis/May09/Dubrofsky_Elan.pdf

[12] GUTIERRÉZ, M.; VEXO, F.; THALMANN, D.: *Stepping into Virtual Reality*. Springer, 2008, ISBN 978-1-84800-116-9.

[13] HARTLEY, R.; ZISSERMAN, A.: *Multiple View Geomerty in Computer Vision*. Cambridge University Press, second edition, 2003, ISBN 0-521-54051-8.

[14] HIRZER, M.: Marker Detection for Augmented Reality Applications. 2008, [online], [cit. 2013-4-27].
URL `http://studierstube.icg.tugraz.at/thesis/marker_detection.pdf`

[15] HUSÁK, M.: Akcelerometry. [online], [cit. 2013-4-25].
URL `http://www.micro.feld.cvut.cz/home/X34SES/prednasky/08%20Akcelerometry.pdf`

[16] LEE, W.-M.: *Beginning Android 4 Application Development*. Massachusetts: A K Peters, Ltd., 2012, ISBN 1-56881-230-2.

[17] SMITH, D.; FRIESEN, J.: *Android Recipes*. 2011, ISBN 978-1-4302-3414-2.

[18] SOOD, R.: *Pro Android Augmented Reality*. Apress, first edition, 2012, ISBN 978-1-4302-3946-8.

[19] ZONGLEI, H.; BOUFAMA, B.: A semi-automatic camera calibration method for augmented reality. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, volume 4, October 2002, ISSN 1062-922X.

Appendix A

Content of DVD

\ARNavi	source files of the augmented reality application
\bin	a “marnavi.apk” file for installation on Android device
\doc	source files of the technical report in LaTeX
\MMAureli	source files of the augmented reality library
arv.mp4	video of the application
arn-poster.pdf	poster for the application
xmurin03-dp.pdf	text of the report in PDF
readme.txt	hints for the content of the DVD