HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

NANYANG
TECHNOLOGICAL
UNIVERSITY

Bachelor Thesis

Sky/Cloud Analysis

Design of a Smartphone based Whole Sky Imager

Autumn Term 2016

Supervised by:
Prof. Dr. Heinz Mathis, HSR University of Applied Sciences Rapperswil
Prof. Dr. Lee Yee Hui, Nanyang Technological University

Author:
Julian Müller

Abstract

By monitoring cloud formation during night and day information is gained to help understand the effects of cloud formation on various applications such as satellite-to-ground communication, solar energy generation and weather prediction. In contrast to satellite observation, the images taken by ground-based sensors offer a higher level of detail at a specific are. Four distinct digital and fully automatic Wide Angle High-Resolution Sky Imaging Systems (WAHRSIS) have been developed at the Nanyang Technological University.

The current imagers are expensive and offer a limited portability due to their size and weight. In this thesis a new whole sky manager based on a smartphone was developed and tested. The primary purpose of this approach is to provide a low cost, highly portable, versatile and easy to deploy imager and evaluate the possibilities it offers. The developed imager captures almost the entire hemisphere in three pictures with different exposures that are merged into a tonemapped high dynamic range image on a server. An Android application and a suitable housing were created to protect the device from the hot and humid climate in Singapore.

It was concluded that modern smartphones provide a powerful embedded system that offer great potential in the field of whole sky imaging but the quality of the created images is lower than that of photographs taken by high quality cameras. The developed imager has great potential to extend the current fleet of imagers used by providing more and new data such as GPS and compass information.

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Design of a Smartphone based Whole Sky Imager

is original work which I alone have authored and which is written in my own words.

Author

Julian	Müller

Student supervisor(s)

Prof. Dr. Lee	Yee Hui	Nanyang Technological University, Singapore
Dr. Stefan	Winkler	University of Illinois Research Center in Singapore

Supervising lecturer(s)

Prof. Dr. Heinz	Mathis	HSR Hochschule für Technik Rapperswil, Switzerland

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' The citation conventions usual to the discipline in question here have been respected.

Place and date	Signature

Contents

List of Figures

List of Tables

List of Acronyms

Acronyms and Abbreviations

ADSC	Advanced Digital Sciences Center
AF	Autofocus
API	Application programming interface
Av	Aperture value
CA	Certification Authority
DR	Dynamic range
DSLR	digital single-lens reflex camera
EV	exposure value
Exif	Exchangeable image file format
f	f-number
HDR	High dynamic range
HTTP	Hypertext Transfer Protocol
HTTPS	High dynamic range
IDE	Integrated development environment
IMU	Inertial measurement unit
LDR	Low dynamic range
NTU	Nanyang Technological University
SDK	Software development kit
SSL	Secure Sockets Layer
TLS	Transport Layer Security
URL	Uniform Resource Locator
WB	White balance

Chapter 1

Introduction

1.1 Background and Motivation

Although whole sky imager systems (WSI) are commercially available[1][2], four different whole sky imagers have been developed at the Nanyang Technological University (NTU) in Singapore. This research was produced due to to comercally available imagers not being able to offer the desired flexibility and functionality, whilst additionally being expensive. The imagers at the NTU were named Wide Angle High Resolution Sky Imager System (WAHRSIS) and they all use high quality digital single lens reflex cameras (DSLRs) with ultra wide-angle lenses.
In order to provide a cheaper, low maintenance, small and easy to install imager a different approach was taken. The main focus of this thesis is to explore the potential of a smartphone based whole sky imager.
Such an imager should be easy to rebuild and set up in order create an array of imagers that could be placed at different locations. This would result in a large amount of data that could be merged to obtain 3D images for a better and finer understanding of cloud formations.

1.2 Literature Review

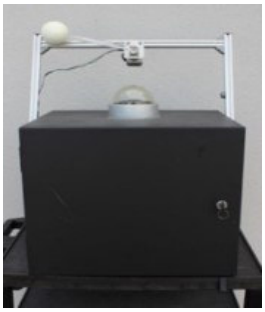
The following section briefly discusses the process and the state of the Sky/-Cloud Analysis research at the Nanyang Technological University. It also explains why Android is used as base in this thesis.

1.2.1 WAHRSIS 1 to 4

Figure 1.1 shows images of the four WAHRSISs that were built at the Nanyang Technological University. Imager 1 uses a mechanical sun blocker to eliminate the over-exposed part in the image that is caused by the bright sunlight. WAHRSIS 2, 3 and 4 use high dynamic range algorithms to create high dynamic images where the over-exposed sun spot can be automatically reduced

[1] `http://www.estrato.hu/cloud-detectors`
[2] `http://www.yesinc.com/products/cloud.html`

to a negligible level (see fig 1.2d) [3]. In figure 1.2b the sun has been replaced automatically by a black spot. By using HDR imaging technique solving cloud segmentation problems around the sun can be bypassed. Next to cloud detection other applications are being researched including analysis of cloud coverage, cloud formation and cloud type as well as cloud prediction. The cost of these imaging systems was about 4000 USD each.



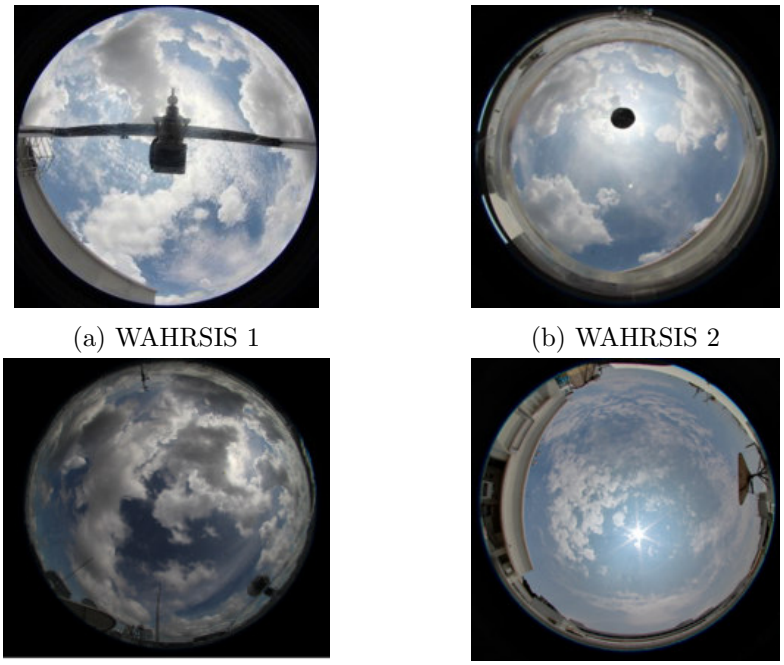
(a) WAHRSIS 1

(b) WAHRSIS 2

(c) WAHRSIS 3

(d) WAHRSIS 4

Figure 1.1: Pictures of existing WAHRSISs.



(a) WAHRSIS 1 (b) WAHRSIS 2

(c) WAHRSIS 3 (d) WAHRSIS 4

Figure 1.2: Tonemapped HDR photos of existing WAHRSISs.

1.2.2 Current research

One part of the research that is currently being done is to analyse and predict signal attenuation due to clouds using the high resolution images of the sky and other sensors. The objectives can be summarized as follows:

- Detection and classification of clouds.
- Correlation of images with other ground-based sensor data.
- Tracking of cloud small scale movements.
- 3D cloud reconstruction to estimate cloud base height and thickness.
- Detecting rain events from image luminance.

Figure 1.3 shows an example of how clouds are being detected and figure 1.4 illustrates how two imagers that are situated differently can be combined to create a 3D point cloud. In addition to the imagers weather stations are used to provide temperature, humidity, pressure, solar irradiance and other data. One goal is to derive a relationship between whole sky images and total solar irradiance. 3D reconstruction is currently done with a set of two HDR images (WAHRSIS 3 and 4) that are taken at a distance of approximately 100m apart from each other. The stereo reconstruction is based on the disparity between images and uses triangulation of the corresponding light rays.
One issue to overcome in the process of stereo calibrating two images is that the two imagers may not face precisely north and may not be completely level. This

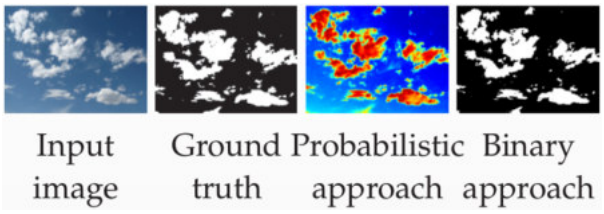


Figure 1.3: Cloud detection [1].

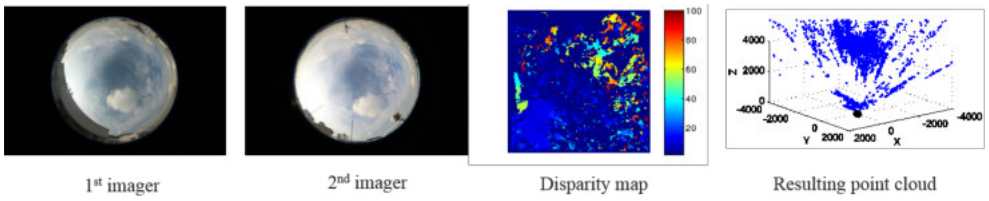


Figure 1.4: Cloud base height estimation [1].

issue can be bypassed by providing azimuth, roll and pitch data as proposed in this thesis.

1.2.3 Android

Android was introduced by Google and Open Handset Alliance in July, 2005. Ever since, it has only gained popularity and market shares which have made it the most popular operating system in the world. Android is based on Linux, which includes operating system, middle ware, libraries and APIs written in C. Android continues to change rapidly. Code snippets and information found online can be of varying quality and applicability. This also affects the code created in this thesis. Although the core is written in C applications for Android are written in Java.
For this thesis Android was chosen as a platform. The alternative would have been Apple's iOS. Apple uses the programming languages Objective C or Swift that are not as common amongst other systems as Java and require a Apple Mac to develop applications. The approaches for this Android application would be similar for iOS devices would but no further investigation has been made on this topic.
For this application the Integrated Development Environment (IDE) Google Android Studio has been used. It is based on InelliJ IDEA and is the most common IDE to develop Android applications next to Eclipse and most tutorials are based on Android Studio.

1.3 Objective and Approach

The objective of this project was to build a new whole sky imager based on a smartphone. The whole sky imager was to be portable consisting of multiple hardware parts such as an attachable fish-eye lens and a waterproof case suit-

able for the climate in Singapore. These parts would have to be selected. At the core of this project was a smartphone application that could autonomously capture images at a regular interval and upload them on a server.

The application will run on any Android smartphone with a minimum API level of 23 (Android Lollipop). The images captured by the smartphone will be stored on the existing server that is developed and maintained at the Nanyang Technological University and the ADSC research center of Illinois in Singapore.

The application will be used in parallel with other DSLR based whole sky imagers that are deployed on roofs at the NTU. In addition to the basic functions the imaging possibilities of a smartphone will be evaluated and tested.

In a first step, the various functions and options in the Android Environment were examined. Manuals and tutorials were used to gain familiarity with the Android Studio which was used throughout the whole project to manage the code.
A basic framework was created that allowed the application to grow in functionality by adding code fragments.
In a first step, the image processing capabilities were tested, then the camera functionality was examined. Further, the use of several sensors was evaluated. In a next step, the internet connectivity was developed and tested. Finally the different components were put together and added with more functionality. Several different approaches were evaluated on how to perform parallel tasks and background processing several different approaches were evaluated.
Next the image processing capabilities were tested. Additionally, different Android parts were evaluated including cameras, sensors, location services, handlers and other classes.
To create images with a large field of view a fish-eye lens was needed. Several such attachable lenses were tested. To protect the new imager from the environment in Singapore, a robust case was created that houses the Android device and other electronics that provide power for the Android device.

1.4 Outline of Thesis

The thesis consists of six chapters that are briefly described here.
Most code fragments shown in this thesis are small snippets serving the purpose of clarifying the way different functions were implemented and helping others to reproduce similar functionalities. The complete project including all resources can be obtained from GitHub under the following link: `https://github.com/JulMueller/WholeSkyImager`. Certain information such as credentials, authorisation tokens, access passwords etc. could not be published in this thesis.

Chapter 1: Introduction In this chapter the background of this thesis is described. Also Android as the underlying platform is introduced.

Chapter 2: High Dynamic Range Imaging In this chapter the image pro-

cessing library OpenCV is introduced. Further HDR is discussed and examples are given.

Chapter 3: Ulra wide-angle lens This chapter discusses different fish-eye lenses that can be used with smartphones. A calibration resulting in a mapping function is explained.

Chapter 4: Android Application This chapter contains the coding phase. It deals with the stand alone parts (camera, server connection, sensor integration etc.) and the completed application including different classes and user interfaces.

Chapter 4: Housing This chapter describes the specifications of a casing to protect the smartphone from the environment and how it was built.

Chapter 5: Conclusion and Future Work Finally a conclusion of this thesis and an outlook are proposed.

Chapter 2

High Dynamic Range Imaging

In this chapter the use of the OpenCV library in combination with Android is evaluated. OpenCV is a library of programming functions aimed at real time computer vision [4]. Further the creation of high dynamic range (HDR) images is discussed.

2.1 Using OpenCV on Android

2.1.1 Installation

The OpenCV library is written in C/C++ but Java wrappers are available[1] that make it possible to use OpenCV function directly in Java code. OpenCv4Android is available as a SDK with samples and Javadoc documentation for OpenCV Java API. NVIDIA released a Tegra Android Development Pack (TADP)[2] that includes Android development tools and the OpenCV4Android SDK. It also includes the Android Native Development Kit (NDK) that is needed to run native C++ code on Android. The set up of OpenCV for Android is well documented and can be verified by adding this snippet to the code:

Listing 2.1: OpenCV verification

```
1 //Check if OpenCV works properly
  if (!OpenCVLoader.initDebug()) {
  Log.e(this.getClass().getSimpleName(), "
      OpenCVLoader.initDebug(), not working.");
  } else {
  Log.d(this.getClass().getSimpleName(), "
      OpenCVLoader.initDebug(), working.");
6 }
```

2.1.2 Example

The following code 2.2 and figure 2.1 show an example implementation to test the OpenCV library in form of a canny edge detector.

[1] used to run C/C++ code in Java

[2] https://developer.nvidia.com/codeworks-android

Listing 2.2: Canny edge detector.

```
// File organization
String inputFileName = "NTU_1_GS";
String inputExtension = "png";
String inputDir = getCacheDir().getAbsolutePath();  // use the
    cache directory for i/o
String outputDir = getCacheDir().getAbsolutePath();
String outputExtension = "png";
String inputFilePath = inputDir + File.separator + inputFileName
    + "." + inputExtension;

// Load image file.
Mat image = Imgcodecs.imread(inputFilePath);
Log.d(this.getClass().getSimpleName(), "loading " + inputFilePath
    + "...");

// Set canny edge detector thresholds.
int threshold1 = 70;
int threshold2 = 100;

// Execute canny edge detector.
Mat im_canny = new Mat();
Imgproc.Canny(image, im_canny, threshold1, threshold2);
String cannyFilename = outputDir + File.separator + inputFileName
    + "_canny-" + threshold1 + "-" + threshold2 + "." +
    outputExtension;
Log.d(this.getClass().getSimpleName(), "Writing " +
    cannyFilename);

// Save and display processed image.
Imgcodecs.imwrite(cannyFilename, im_canny);
Bitmap bitmapToDisplay = null;
bitmapToDisplay = BitmapFactory.decodeFile(cannyFilename);
outputImage.setImageBitmap(BitmapFactory.decodeFile(cannyFilename));
Log.d(this.getClass().getSimpleName(), "Edge detection finished");
```

OpenCV is a powerful image processing platform that has proven to run on Android. Nonetheless heavy processing should be done on the server to relieve the Android device's CPU.

2.2 Exposure Fusion

Exposure fusion is a concept for creating a low dynamic range (LDR) image from a series of images of different exposure times. This process assigns weights to each pixel of each image in the sequence according to luminosity, saturation and contrast. Depending on these weights the specific pixels are then included or excluded from the final image. This does not create a real HDR image and is thus not further discussed in this thesis. Exposure fusion has certain advantage over classic HDR algorithms. It does not require Exif data and no intermediate HDR image is created therefore no tone mapping is needed which makes this process faster. Also it does not require a gamma correction value. When it comes to displaying high detailed images exposure fusion should be used. In this case the HDR image is needed in order to do data processing.

Figure 2.1: Canny edge detector user interface.

2.3 HDR

There are several different ways of obtaining a HDR image. In this thesis, a row of images with different exposure values are being used to obtain a high dynamic range image with up to 32-bit floating values per color channel. This results in large files that are of use when it comes to cloud estimation. To make the image displayable on common screens, it is necessary to lower the dynamic range to usually 8 bit per color channel. The process of lowering the dynamic range is called *tonemapping*.
In order to obtain satisfying HDR images it is necessary for the images of the exposure sequence to be aligned properly and that no moving objects are captured within the sequence. In this project, the camera was completely still so it could be assumed that the images the images are aligned correctly. The speed of moving clouds is of no concern since taking the image sequence takes less than a second. Fast moving objects or a moving lens can cause errors in the resulting image that is known as "Ghosting".

2.3.1 HDR Algorithm

The creation of a HDR and LDR image follows in general the following steps:

1. Load images and corresponding exposure times.
2. Estimate camera response function (CRF).
3. Make HDR image.
4. Tonemap HDR image.
5. Write results.

The camera response function is not linear, but has the property of compressing signal at the high irradiance region and expanding signal at the low irradiance region [5]. OpenCV estimates this function by using the known exposure times and the according images.
The debevec HDR algorithm uses the differently exposed images to recreate the response function of the camera [6]. The algorithm can then fuse the images into a single HDR radiance map. The pixel values are proportional to the true radiance values in the scene. In the OpenCV the values range from 0 to 1.
Tonemapping is needed to decreases the dynamic range to a displayable level. This is done while retaining the localized contrast. The Durand algorithm is based on a bilateral filter but only one of many (Drago et al., Reinhard et al.,Ashikhmin spatially-variant operator, Fattal et al. etc.). Different algorithms are available for the purpose of creating HDR and tonemapped LDR images. The existent WAHRSIS all use a Debevec/Durand combination.

2.3.2 OpenCV HDR Algorithm

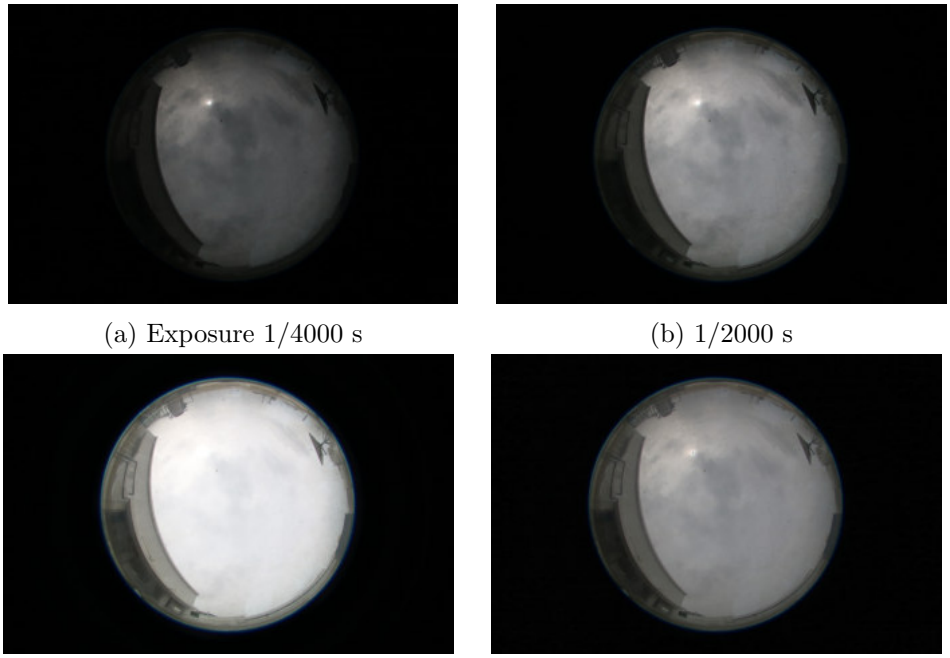
The following code 2.3 shows how a HDR algorithm can be implemented using C++ and openCV [4]. This code runs on computers with C++ compilers. Android normally uses Java code, but allows the use of implementing C++ code by using Android's NDK.
The two most HDR related functions in this example are *createMergDebevec()* and *createTonemapDurand()*. The first function can be seen on line XX. It uses Debevec's weighting scheme to generate a HDR image. The second function maps the HDR image to a 8-bit displayable image. This tonemapper uses bilateral filtering and the gamma correction value is 2.2.

Listing 2.3: C++ HDR generation using openCV

```
#include <opencv2/photo.hpp>
#include "opencv2/imgcodecs.hpp"
#include <opencv2/highgui.hpp>
#include <vector>
#include <iostream>
#include <fstream>
using namespace cv;
using namespace std;
void loadExposureSeq(String, vector<Mat>&, vector<float>&);
```

```
int main(int, char**argv)
{
  // load input images and exposure times
  vector<Mat> images;
  vector<float> times;
  loadExposureSeq(argv[1], images, times);
  // estimate camera response
  Mat response;
  Ptr<CalibrateDebevec> calibrate = createCalibrateDebevec();
  calibrate->process(images, response, times);
  // create HDR image
  Mat hdr;
  Ptr<MergeDebevec> merge_debevec = createMergeDebevec();
  merge_debevec->process(images, hdr, times, response);
  // tonemap HDR image
  Mat ldr;
  Ptr<TonemapDurand> tonemap = createTonemapDurand(2.2f);
  tonemap->process(hdr, ldr);
  // write results
  imwrite("ldr.png", ldr * 255);
  imwrite("hdr.hdr", hdr);
  return 0;
}
void loadExposureSeq(String path, vector<Mat>& images,
    vector<float>& times)
{
  path = path + std::string("/");
  ifstream list_file((path + "list.txt").c_str());
  string name;
  float val;
  while(list_file >> name >> val) {
    Mat img = imread(path + name);
    images.push_back(img);
    times.push_back(1 / val);
  }
  list_file.close();
}
```

The following images show the result of this algorithm running in a similar configuration but with adapted parameters.



(a) Exposure 1/4000 s (b) 1/2000 s

(c) 1/500 s (d) Tonemapped image.

Figure 2.2: WAHRSIS 4 image sequence and tone mapping.

Chapter 3

Ultra wide-angle lens

WAHRSIS 1 to 4 all use high quality wide angle lenses manufactured by SIGMA combined with a Canon EOS 600D. The actual angle of view of this lens is close to 180°. Fisheye lenses are ultra wide-angle lenses with an angle of view between 100 and 180 degrees. This wide angle is caused by an extremely short focal length compared to normal lenses. The focal length of small fish-eye lenses such as in smartphones is only around 1 to 2mm.
Interestingly, the first practical use was in meteorology to study cloud formation, hence the name "whole-sky lenses".
Distortion is inevitable when using wide angle lenses. Two types of distortion are prevalent in this case. The *barrel distortion*, which causes otherwise straight lines to appear bent if they do not pass through the centre of the image, and the *edge distortion*, which causes stretching of objects at the extreme edges of the frame [7]. Knowing the effects of distortion plays an important role in image processing for cloud analysis that is being researched by a team at the NTU. One way to cope with these distortions is to use a special function that allows mapping each pixel of the two dimensional image into a three dimensional hemisphere. This is explained in the section 3.2.

3.1 Evaluation

Fish-eye lenses for smartphones are generally cheap and of low quality. This can be explained by their main purpose which does not lie in professional application. In order to find a suitable lens for the mobile imager different models were ordered from lazada.sg and evaluated ranging from 3 SGD to 21 SGD (see tab 3.1). It was not possible to find any useful information on the different models thus testing was the only way to find out whether the lenses could be used in this project or not. The following requirements needed to be fulfilled in order to build a prototype:

- True wide angle close to 180°.
- No vignetting effect[1].

[1]Reduction of an image's brightness or saturation at the edges compared to the image centre.

(a) (b) (c) (d)

Figure 3.1: Different smartphone fish-eye lenses.

Figure 3.2: Lens (d) HDR and tonemapped image.

- High penetrability.
- Low colour aberration.

Of the four lenses depicted in figure 3.1, only 3.1d features a truly wide-angle, clear and non vignetting image. The cost of this model is SGD 10.20 and comes with a clip for easy installation. Figure 3.2 shows a HDR image taken with a Xiaomi Mi 4c Smartphone in combination with wide-angle Lens D.

3.2 Mapping function

Normal, ideal lenses do not introduce distortion into images. This means that straight lines in the real environment remain straight in the photograph. For fish-eye lenses this is strictly not the case as explained previously. The mapping functions vary depending on the model of the fish-eye lens and the smartphone camera.
Since fisheye lenses for smartphones are mostly manufactured for non-professional use almost no parameters are documented. This is aggravated by the fact that these lenses are placed onto the existing lens in a smartphone thus making it

Table 3.1: Tested fish-eye lenses.

	Lens (a)	*Lens (b)*	*Lens (c)*	*Lens (d)*
Name	AUKEY 180° Fisheye Lens	HKS Mini Clip Fish Eye Detachable Lens	Fancyqube Wide-angle Fish Eye Lens	OEM Universal More Advanced 185 Degree
Price (SGD)	22	14	2.55	10.20
Effective wide-angle	no	no	no	yes

impossible to calculate the mapping function without the use of calibration images.

3.3 OCamCalib Toolbox

The OcamCalib Toolbox for Matlab offers a way to calibrate any panoramic camera. It does not require any knowledge about intrinsic parameters of the camera and the lens. The implementation is based on the procedures presented in the papers [8], [9] and [10]. The calibration process needs to be done once for each smartphone-lens combination. It should also be redone if the lens has been moved.
After a successful calibration the toolbox provides the two functions CAM2WORLD and WORLD2CAM. These functions express the relation between a given pixel point and its projection onto the unit sphere. Both functions are implemented in the smartphone application but have to be fed with the function parameters resulting from the calibration first. The Matlab toolbox does not run on Android, therefore a computer running Matlab is needed to do the calibration. It could be possible to remotely send the calibrations pictures of the smartphone/fish-eye lens combination to a server running the Matlab OcamCalib Toolbox and then retrieve the output parameters. This was not implemented in this thesis.
As a calibration image the toolbox uses a chequerboard pattern. In a first step, several images are taken through the wide-angle lens that should be calibrated, this can be seen in figure 3.3. In a next step, the toolbox attempts to extract all grid corners. Missing corners have to be manually placed. Once all grid corners are extracted the calibration can be started. In this case, a polynomial order of 4 is used. At the end of the calibration the following graphs (figure 3.4) are shown.
In the upper plot the function F is shown. The lower graph shows the angle THETA of the corresponding 3D vector with respect to the horizon.
Further, the centre of the image can be automatically be found by using an iterative method. This results in an output as shown in listing 3.1.

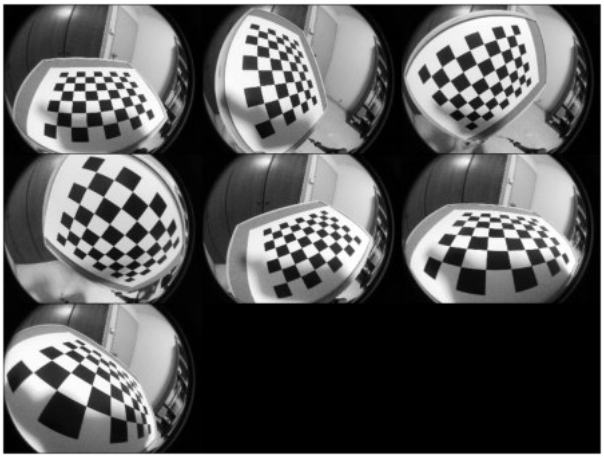


Figure 3.3: Calibration photographs.

Listing 3.1: OCamToolbox Output

```
Computing center coordinates.
Iteration 1...2...3...4...5...6...7...8...9...
Average reprojection error computed for each chessboard [pixels]:

1.91 +/- 1.42
3.57 +/- 2.58
3.24 +/- 2.80
2.09 +/- 1.71
2.39 +/- 2.06
2.34 +/- 1.38
4.07 +/- 2.41

Average error [pixels]
2.801620

Sum of squared errors
3767.455575
```

The calibration parameters are stored in a different file under the variable name *SS*. In this case four polynomial coefficients of function F are calculated.

$$F = a_0 + a_1\rho + a_2\rho^2 + a_3\rho^3 + a_4\rho^4$$

ρ is the distance from the centre of the omnidirectional image measured in pixels. For this camera and lens combination the resulting parameters are given in following form where each line represents one parameter:

$$\begin{aligned} ss = 1e2 \cdot & \\ & 6.800825080096965 \\ & 0.000004183600322 \\ & 0.000000000027440 \\ & 0.000000000002139 \end{aligned}$$

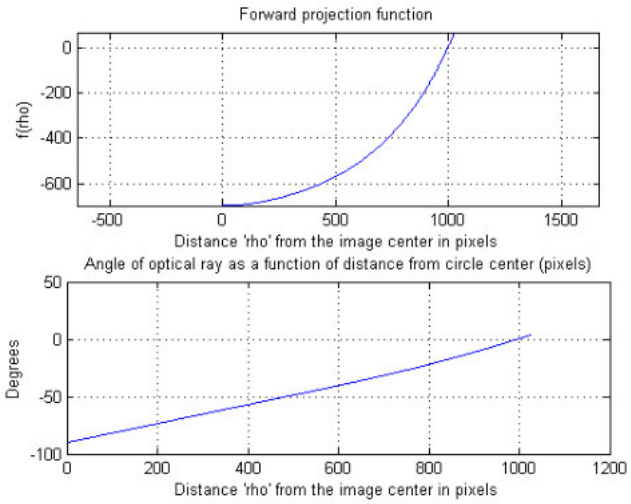


Figure 3.4: Projection function.

This data is useful for instance for the 3D reconstruction of images from multiple imagers which is part of the sky/cloud analysis research at the NTU [11]. It could be shown that calibration is possible with a smartphone camera / fish-eye lens combination but the average errors introduced are relatively high. This could be circumvented by using more calibration images and calibration images of better quality (different angles, better lighting etc).

Chapter 4

Android Application

With their increasing computational power and functionality, smartphones have become a versatile platform for standalone solutions.
The large variety of on board sensors and their powerful and mostly well documented APIs make smartphones convenient to use in many different fields. The main programming language used to develop Android application is Java. All libraries and application programming interfaces (API) provided by Google Android are written in Java, unlike the underlying core that is written in C and C++. These libraries make it convenient to use certain technologies such as the inertial measurement unit or the phone's camera.

4.1 Overview

Figure 4.1 shows how the imager's application can be split up into different parts. The parts resemble the steps of development from left to right. The combination of all these parts resulted in the final application. The following sections discuss these parts individually and the different approaches that were necessary to create the desired functionality. Figure A.1 on page 55 in the appendix also shows these software elements together with other all other parts to create this imager.

4.2 Testing

Testing was done with the help of the Android Studio Debugger and the logging function. Since the new whole sky imager application is a prototype it may still contain bugs. To prove the reliability of the application long term tests would be necessary. For further information on this topic chapter 6.2 should be considered.

4.3 Camera

Android provides two distinct classes to operate the camera. Developers can choose between the older *Camera* class or the *new android.hardware.camera2* API.

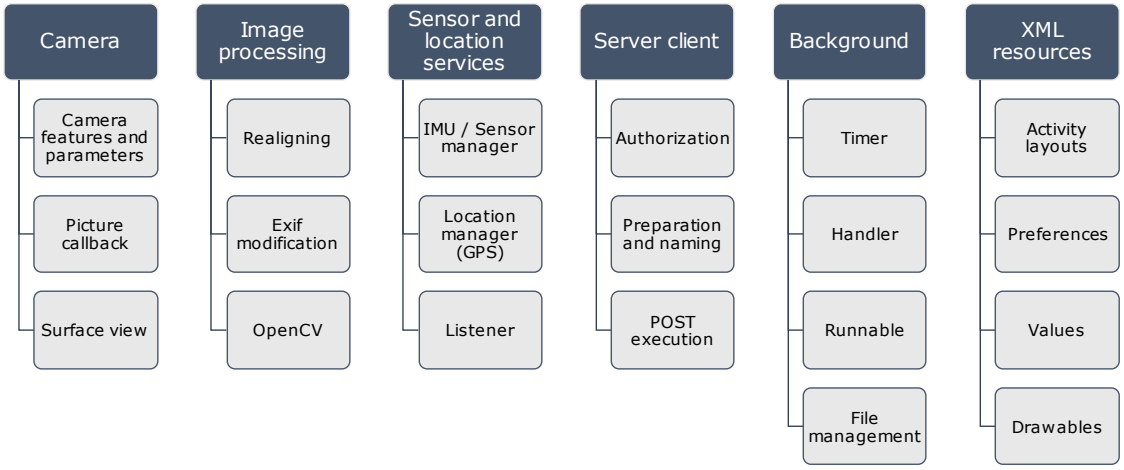


Figure 4.1: Application overview.

For classic camera applications it is recommended to use the newest API, especially because the older Camera class was deprecated in API level 21.
In this thesis the older Camera class [12] is used for the following reasons:

- It is still supported, runs stable and is simpler to use than camera2.
- Many problems are solved and their solutions are publicly available on the large software development forum Stackoverflow[1].
- Books, tutorials and examples are often based on the older Camera class.
- The necessary functionality is provided in the Camera class. The newer Camera2 API does not provide any important new methods.

4.3.1 Initialisation

The initial set up of the camera follows the flow chart shown in figure 4.2. To provide a live preview of the camera, a *SurfaceHolder* can be passed to the *setPreviewDisplay*() method. *SurfaceHolder* is a abstract interface that allows control of the surface size, format and the pixels shown. It has been used in this application to show the camera preview. This live preview is not necessary but was helpful in the process of debugging and evaluating the camera settings. Pictures can be taken by calling *takePicture(Camera.ShutterCallback...)* on the camera object. A callback will then provide the actual image data, either in RAW or JPEG format. The long-running operations like preview and photo capture run asynchronously and invoke callbacks. Since this app does not need to share the camera with other applications this is of no concern, otherwise the camera needs to be released and unlocked accordingly.
Different Android devices may have different camera specifications, therefore it

[1] http://stackoverflow.com/

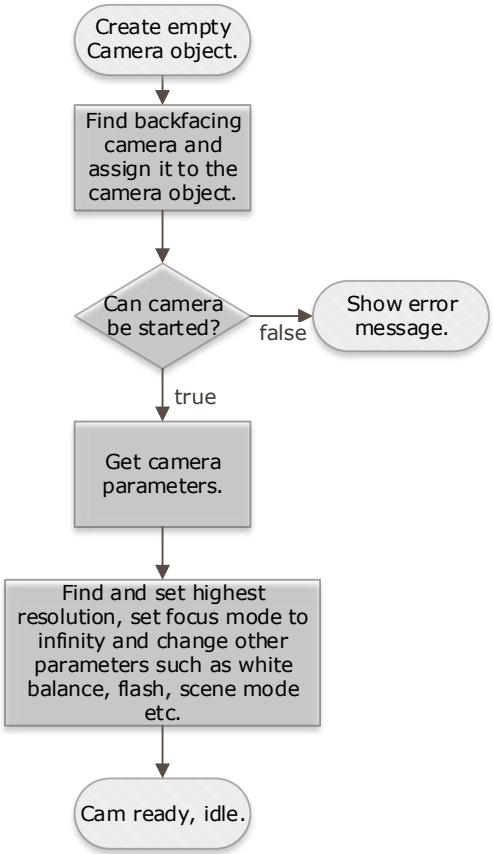


Figure 4.2: Flowchart camera initialisation.

is necessary to get all available parameters first before changing them. Calling *getParameters()* on the camera object returns all available default parameters. For instance the minimum and maximum exposure values, the available resolutions and ISO values may differ.

Listing 4.1 shows the implementation of the camera initialisation. This code is part of the final application. Lines 59 to 74 show the part where the back facing camera is identified. Once the camera ID is returned an attempt is executed to open the camera (Line 21). Only in case of success will the lines 22 to 45 then cause the camera to change its settings.

Listing 4.1: Camera initialisation

```
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;

public class MainActivity  {
  private Camera camera;
  Camera.Parameters params;
  camera = checkDeviceCamera();

  protected void onCreate(Bundle savedInstanceState) {
    camera = checkDeviceCamera();
```

```
}

/**
* Find camera and set it up.
* @return camera
*/
private Camera checkDeviceCamera() {
  int cameraId = findBackFacingCamera();
  Camera mCamera = null;
  try {
    mCamera = Camera.open(cameraId);  //try to open camera
    Camera.Parameters params = mCamera.getParameters();

        // find largest resolution (indicated by size)
    List<Camera.Size> sizes = params.getSupportedPictureSizes();
    Camera.Size size = sizes.get(0);
    for (int i = 0; i < sizes.size(); i++) {
      if (sizes.get(i).width > size.width)
      size = sizes.get(i);
    }
    params.setPictureSize(size.width, size.height);

    // check if focus mode infinity is available and set it
    List<String> focusModes = params.getSupportedFocusModes();
    if
        (focusModes.contains(Camera.Parameters.FOCUS_MODE_INFINITY))
        {
      params.setFocusMode(Camera.Parameters.FOCUS_MODE_INFINITY);
    }

    // set white balance to auto and turn off flash
    params.setJpegQuality(100);
    params.setWhiteBalance(Camera.Parameters.WHITE_BALANCE_AUTO);
    params.setFlashMode(Camera.Parameters.FLASH_MODE_OFF);

    // save settings
    mCamera.setParameters(params);
    d(this.getClass().getSimpleName(), "Camera started
        successfully.");
  } catch (Exception e) {
    // show error message if camera can not be accessed
    e.printStackTrace();
    Log.e(this.getClass().getSimpleName(), "Could not start
        camera.");
  }
  return mCamera;
}

/**
* Find back facing camera
* @return cameraId
*/
private int findBackFacingCamera() {
  int cameraId = -1;
  // get number of cameras
  int numberOfCameras = Camera.getNumberOfCameras();
  // check every camera
```

```
      for (int i = 0; i < numberOfCameras; i++) {
        Camera.CameraInfo info = new Camera.CameraInfo();
        Camera.getCameraInfo(i, info);
        if (info.facing == Camera.CameraInfo.CAMERA_FACING_BACK) {
68        cameraId = i;
          break;
        }
      }
          // return ID of back facing camera
73    return cameraId;
    }
  }
```

4.3.2 Usage

The following code 4.2 reflects the basic implementation of a picture callback method which is called whenever the *takePicture()* method is executed. For the final application this code is extended to allow the taking of multiple pictures with different exposure values. Line 13 indicates how the file name will be created dynamically. *evState* is an array that contains the strings "low", "med" and "high". Line 31 causes the bitmap to be shown scaled down to 400*300 pixels in the layout of the main activity. This is not done in the final application but in this case it was helpful for debugging.
The method *getOutputMediaFile(fileName)* checks if a subfolder "WSI" exists and creates one if this is not the case. Furthermore, it generates an empty file with the name given as a parameter and returns the full fil epath plus file name string eg. */storage/emulated/0/WSI/2016-12-19-12-16-00-wahrsis5-low.jpg.*

Listing 4.2: picture Callback

```
//Picture Callback Method
PictureCallback pictureCallback = new PictureCallback() {
  @Override
  public void onPictureTaken(byte[] data, Camera camera) {
5   Bitmap bitmap = BitmapFactory.decodeByteArray(data, 0,
        data.length);
    Bitmap bitmapRotated = null;
    if (bitmap == null) {
      Toast.makeText(MainActivity.this, "Captured image is
          empty", Toast.LENGTH_LONG).show();
      return;
10  }

    //prepare filename according to naming convention eg.
        2016-11-22-14-20-01-wahrsis5.jpg
    String fileName = timeStampNew + "-wahrsis" + wahrsisModelNr
        + "-" + evState[pictureCounter] + ".jpg";
    File pictureFile = getOutputMediaFile(fileName);
15
    if (pictureFile == null) {
      return;
    }
    try {
20    FileOutputStream fos = new FileOutputStream(pictureFile);
```

```
        fos.write(data);
        fos.close();
        Log.d(TAG, "Save successful: " + pictureFile.getName());
      } catch (FileNotFoundException e) {
        Log.e(TAG, "Save falied.");
      } catch (IOException e) {
        Log.e(TAG, "Save failed.");
      }

      // show picture in UI
      outputImage.setImageBitmap(scaleDownBitmapImage(bitmap, 400,
          300));
      if (mImageSurfaceView.getPreviewState()) {
        mImageSurfaceView.refreshCamera();
      }
    }
};
```

The minimum and maximum exposure values can be found by calling the following methods. These parameters can then be set as shown in listing 4.1. The step size can be obtained in a similar manner. For the Xiaomi Mi 4c device the step size is 0.166667 with a maximum value of +12. By multiplying these two values this translates into a minimum and maximum EV of +/-2 respectively.

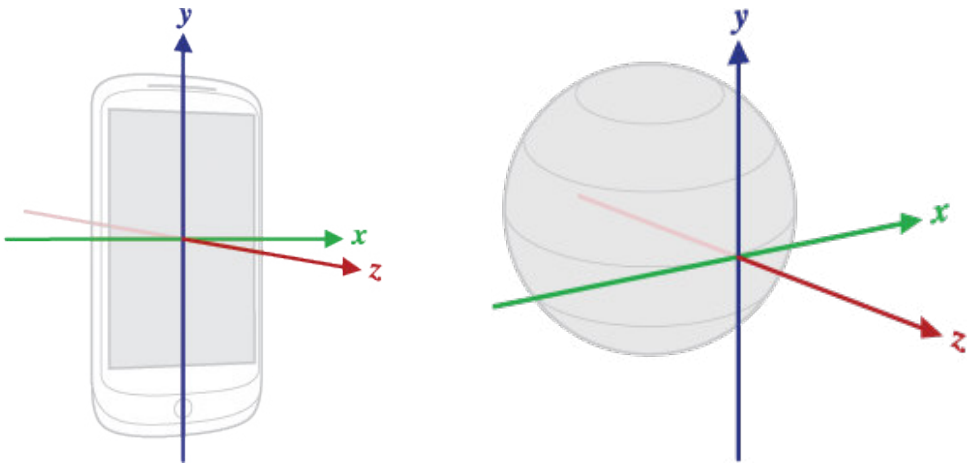
```
maxExposureComp = params.getMaxExposureCompensation();
minExposureComp = params.getMinExposureCompensation();
```

4.4 Sensor and location integration

WAHRSIS 1 to 4 do not use additional sensors for image processing. Instead, the trajectory of the sun is used to align the images of different imagers [11]. For example, the realignment is needed to create 3D images which are used to estimate the cloud base height.
By providing the absolute position in terms of GPS coordinates, compass heading and orientation of the camera the research team at the NTU had more options to create merged images or process the data in different ways. Android provides classes for these tasks that are discussed in the following subsections. It distinguishes between three different categories of sensors:

- Motion sensors: Accelerometers, gravity sensors, gyroscopes and rotational vector sensors.
- Environmental sensors: Including ambient air temperature, pressure, humidity etc. These are not available on the used devices but could be of great use (see chapter 6.2).
- Positions sensors: Orientation sensors (GPS) and magnetometers.

To work with these sensors, the Android sensor framework is used. This frameworks provides useful functions such as registering and unregistering sensor event listeners, that in turn monitor sensor changes or determine which sensors are available on a device.



(a) Coordinate frame relative to the device.

(b) Coordinate frame relative to the earth.

Figure 4.3: Different frames of reference used from [2]

.

Android offers a great variety of virtual sensors that combine physical sensors (using Kalman and other filters) and are based either on the coordinate system relative to the device or relative to the earth's frame as depicted in figure 4.3 [2]. In the world frame X is tangential to the ground at the device's location and roughly points east, Y is tangential to the ground and points towards the magnetic North Pole and Z points towards the sky and is perpendicular to the ground [2].
When taking images, the device's camera should ideally point upwards in the direction of the Z axis.

4.4.1 Azimuth, Pitch and Roll

In this application the two accelerometer and magnetometer (hardware sensors) are used to obtain the inclination and rotation matrix by using the two sensors in conjunction with the *getRotationMatrix()* method. This method transforms a vector from the device's coordinate system to the world's coordinate system. Listing 4.3 shows how the sensors are initialized. Line 8 and 16 respectively check if the specific sensors are available. If both sensors are available the boolean variable *success* is returned as true. Code 4.4 contains the functions to retrieve azimuth (the heading), pitch and roll information. These values are radians, therefore lines 19 to 21 transpose the value to degrees. Essential is the method *getRotationMatrix(R, I, mGravity, mGeomagnetic)* on line 14 that transforms vectors from the device coordinate system to the world's coordinate system as explained before.
Method *getOrientation(R, orientation)* on line 17 computes the device's orientation based on the rotation matrix. The sensor listener runs automatically in the background and is called at a specific interval given in microseconds. In this case *SENSOR_DELAY_LONG* is defined as 1000000 (us).

```
mSensorManager.registerListener(this, accelerometer,
    SENSOR_DELAY_LONG);
mSensorManager.registerListener(this, magnetometer,
    SENSOR_DELAY_LONG);
```

Listing 4.3: Sensor initialisation

```
public boolean instantiateSensors() {
  boolean success = false;

  // initialise device sensor
  mSensorManager = (SensorManager)
      getSystemService(SENSOR_SERVICE);

  // accelerometer sensor (for device orientation)
  if( mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
      != null ) {
    Log.d(TAG, "Found accelerometer.");
    accelerometer =
        mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
  } else {
    Log.e(TAG, "No support for accelerometer.");
  }

  // magnetic sensor (for compass direction)
  if( mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
      != null ) {
    Log.d(TAG, "Found magnetic sensor.");
    magnetometer =
        mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
    success = true;
  } else {
    Log.e(TAG, "No support for magnetic sensor.");
  }
  return success;
}
```

Listing 4.4: Sensor listener

```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
  Sensor mySensor = sensorEvent.sensor;

  if (mySensor.getType() == Sensor.TYPE_ACCELEROMETER)
    mGravity = sensorEvent.values;
  if (mySensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
    mGeomagnetic = sensorEvent.values;

  // calculate heading
  if (mGravity != null && mGeomagnetic != null) {
    float R[] = new float[9];
    float I[] = new float[9];
    boolean success = SensorManager.getRotationMatrix(R, I,
        mGravity, mGeomagnetic);
    if (success) {
      float orientation[] = new float[3];
      SensorManager.getOrientation(R, orientation);
```

```
18        // orientation contains: azimuth, pitch and roll
          float thisAzimuth = (float) Math.toDegrees(orientation[0]);
          float pitch = (float) Math.toDegrees(orientation[1]);
          float roll = (float) Math.toDegrees(orientation[2]);
          // round azimuth to one decimal
23        azimuth = Math.round(thisAzimuth * 10f) / 10f;
          pitch =  Math.round(thisPitch * 10f) / 10f;
          roll =  Math.round(thisRoll * 10f) / 10f;
          Log.d(TAG, "Azimuth: " + azimuth + " deg.");
        }
28    }
  }
```

The resulting data (azimuth, roll, pitch) can be stored and added to each uploaded image in the Exif format (see section 4.5.2).

4.4.2 Location

Obtaining the absolute location of the Android device works in a similar manner as the sensor initialisation. One difference is that the location service must be enabled by a user since the Android permission policy does not allow applications to automatically turn on GPS. The location listener is initialised with following code:

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
    300000, 5, locationListener);
```

The third parameter (300000) defines the update interval in miliseconds (here 5 minutes), the fourth parameter (5) defines the minimum distance difference (in metres) before the location listener method will be called.
Listing 4.5 shows the code to initialise the location service. Lines 5 to10 handle the situation if GPS is disabled. On line 9 an intent is called that shows the GPS settings. An intent is an abstract class that can be used to start an activity (user interface, notification etc.). A user has to then manually activate the GPS before the application can proceed. Line 12 to 25 attempt to find the GPS as a location provider. In this application GPS is the desired provider. The provider criteria can be for instance GPS hardware and visibility to a number of satellites. Other providers require the use of the cellular radio, access to a specific carrier's network or to the internet [2].

Listing 4.5: Position initialisation

```
public boolean instantiateGPS() {
  boolean success = false;
  LocationManager locationManager = (LocationManager)
      getSystemService(LOCATION_SERVICE);
4
  boolean enabled =
      locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
  if (!enabled) {
    Log.d(TAG, "GPS is not activated.");
    Intent gpsSettingsIntent = new
        Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
9   startActivity(gpsSettingsIntent);
```

```
  }
  // Define the criteria how to select the locatioin provider ->
      use default
  Criteria criteria = new Criteria();
  provider = locationManager.getBestProvider(criteria, false);
  // Register the listener with the Location Manager to receive
      location updates
  try {
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
        300000, 5, locationListener); //minimum interval 5
        minutes, minimum distance 5 meter
  } catch (SecurityException e) {
    Log.d(TAG, "GPS access has not been granted.");
  }

  // Initialise the location fields
  if (location != null) {
    Log.d(TAG, "Provider " + provider + " has been selected.");
    success = true;
  } else {
    Log.d(TAG, "Could not find location.");
  }
  return success;
}
```

Listing 4.6: Position listener

```
LocationListener locationListener = new LocationListener() {
  // if location update occurs
  public void onLocationChanged(Location location) {
    // get coordinates in floating point format
    longitude = location.getLongitude();
    latitude = location.getLatitude();
    Log.d(TAG, "Long: " + longitude + " Lat: " + latitude);
  }

  // if GPS is turned off
  public void onProviderDisabled(String provider) {
    if (provider.equals("gps")) {
      Log.d(TAG, "GPS Provider disabled.");
    }
  }
};

// transforms floating point long/lat into DMS String
public String coordinatesToDMS(double position) {
  if (location == null) {
    return "0/1,0/1,0/1000";
  }
  String[] degMinSec = Location.convert(position,
      Location.FORMAT_SECONDS).split(":");
  return degMinSec[0] + "/1," + degMinSec[1] + "/1," +
      degMinSec[2] + "/1000";
}
```

Code snippet 4.6 shows how the GPS listener is implemented and how the longitude and latitude can be transformed to a degree minute second (DMS) string

that is needed for the Exif tags (see section 4.5.2). The latitude and longitude in the Exif format are expressed as three rational values giving the degrees, minutes, and seconds respectively. The DMS format is dd/1,mm/1,ss/1.
For instance, the latitude of the Nanyang Technological University 1.3483099 becomes 1/1, 20/1, 53916/1000.

4.5 Image processing

With the data obtained from the sensor, limited image processing can be done on the Android device. In the previous chapter (see 2.1), it was shown that OpenCV function can be executed on Android devices. For many applications using the OpenCV is not necessary because Google provides its own bitmap manipulation classes. Finally it is recommended to do image processing on the server because it is computationally expensive and drains the battery of the Android device. The following section will show an example of what is possible in the Android environment.

4.5.1 Rotation

Figure 4.4 illustrates the original image and the automatically rotated version according to azimuth information. On the left image the top of the watch points towards West (271°) which indicates that the images needs to be turned at an angle of 360-271 = 89° in an anti-clockwise direction. The resulting image is larger than the original file, but the size will vary depending on the angle that is needed to rotate the image.

(a) Original image.

(b) Rotated image.

Figure 4.4: Rotation according to azimuth.

The following code 4.7 shows the implementation of the rotation sequence using the original image and the angle given by the orientation sensor listener. The accuracy of this rotation depends on the error of the compass. To reduce this error, the compass should be calibrated before setting up the imager. The

matrix used is a standard 3 by 3 transformation matrix. The resulting image should be cropped to reduce the file size. This is especially useful when using a fish-eye lens where the information is only distributed around the centre. The rotated image only contains information in a frame of the size of the *width of original images * width of original image*. In this case, the original image has a resolution of 4160 x 3120. To create a cropped version of the rotated images *createBitmap(source, offsetX, offsetY, widthOriginal, widthOriginal)* has to be called where:

$$offsetX = \frac{widthNew - widthOriginal}{2}$$

$$offsetY = \frac{heightNew - widthOriginal}{2}$$

widthNew and *heightNew* are width and height of the rotated but uncropped image. This cropping process is illustrated in figure 4.5 where the red and white coloured parts are excluded. The process of rotating and cropping high resolution images is very CPU-intensive and is not part of the prototype application. Instead this could be implemented on the server.

Listing 4.7: Rotation method.

```
public static Bitmap RotateBitmap(Bitmap source, float angle)
{
  Matrix matrix = new Matrix();
  matrix.postRotate(angle);
  return Bitmap.createBitmap(source, 0, 0, source.getWidth(),
      source.getHeight(), matrix, true);
}
```

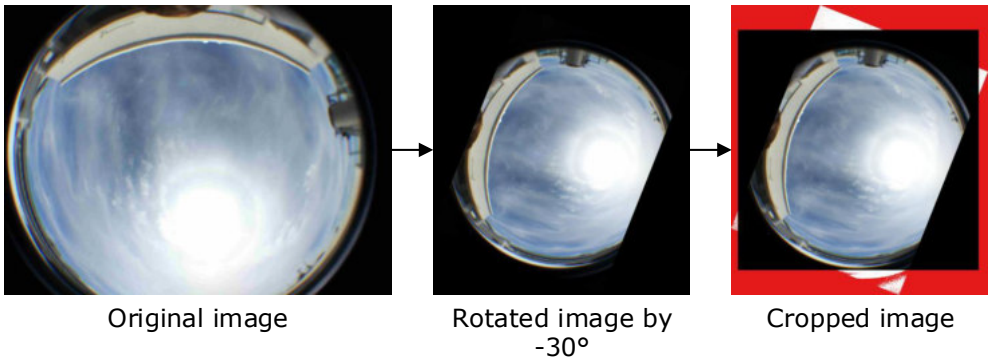


Figure 4.5: Image rotation and cropping.

4.5.2 EXIF tags

Exif tags are used to store metadata such as camera make and model, parameters, date and time etc. within a picture [13]. The available tags are limited and defined by a specific ID. For instance, the ID "33434" belongs to tag *ExposureTime* [13]. It is not possible to add new custom tags. Alternatively one

can use the tag *UserComment* to store specific information. In this case it can be used to store roll and pitch data.
Android provides a *ExifInterface* class to read and write Exif tags in a JPEG file or a RAW image file. The following list shows the important tags that are used in this application. Most are self-explanatory, but unfortunately all tags marked with an asterix (*) are only supported as of API level 24 (Android 7). According to the Android dashboard [2] by December 5, 2016 only 0.4% of all Android devices use Android 7. This was the main reason why an alternative class was used instead of the *ExifInterface*.

The Android-Exif-Extended library that was used was created by Alessandro Crugnola[2]. It provides various functions such as copying all available tags from one file to another. The main reason to use this library was that it includes all tag definitions that are not available when using the Android *ExifInterface* class.

- GPSImgDirection*
- GPSImgDirectionRef*
- GPSLatitude
- GPSLatitudeRef
- GPSLongitude
- GPSLongitudeRef
- GPSAltitude
- GPSAltitudeRef
- ExposureTime
- FNumber
- ExposureProgram
- ISOSpeedRatings
- DateTimeOriginal*
- DateTimeDigitized*
- ShutterSpeedValue
- ApertureValue*
- ExposureIndex*
- ExposureMode*
- WhiteBalance
- Make
- Model
- UserComment

The listing 4.8 represents the implementation of the Exif data copy function. It takes all available tags from the original image and copies them onto the rotated bitmap file. A bitmap object/file only contains the pixel data and no meta data. In the prototype application it is only necessary to add the location and custom tags (UserComment) in addition to the common tags (iso, exposure etc.). This can be done by simply setting the same file as source and target parameter.

Listing 4.8: Exif copy and write.

```
// copy exif tags from one file to another
private void copyExif(String source, String target) {
  ExifInterface oldExif = new ExifInterface();
```

[2]https://github.com/sephiroth74/Android-Exif-Extended

```
4   ExifInterface newExif = new ExifInterface();
    try {
      oldExif.readExif(source, ExifInterface.Options.OPTION_ALL);
      List<ExifTag> all_tags = oldExif.getAllTags();
      newExif.writeExif(source, target);
9     newExif.setExif(all_tags);
      // add position tags + user comment (absolute roll, pitch
          angles)
      if (flagWriteExtendedExif) {
        newExif.setTagValue(ExifInterface.TAG_GPS_IMG_DIRECTION,
            Math.round(azimuth));
        newExif.setTagValue(ExifInterface.TAG_GPS_IMG_DIRECTION_REF,
            'M');
14      newExif.setTagValue(ExifInterface.TAG_GPS_LATITUDE,
            coordinatesToDMS(latitude));
        newExif.setTagValue(ExifInterface.TAG_GPS_LATITUDE_REF,
            latitude < 0 ? "S" : "N");
        newExif.setTagValue(ExifInterface.TAG_GPS_LONGITUDE,
            coordinatesToDMS(longitude));
        newExif.setTagValue(ExifInterface.TAG_GPS_LONGITUDE_REF,
            longitude < 0 ? "W" : "E");TAG_USER_COMMENT
        newExif.setTagValue(ExifInterface.TAG_GPS_LONGITUDE,
            coordinatesToDMS(longitude));
19      newExif.setTagValue(ExifInterface.TAG_USER_COMMENT,
            Float.toString(roll)+","+Float.toString(pitch));
      }
      newExif.writeExif(target);
      Log.d(TAG, "Writing Exif completed.");
    } catch (IOException e) {
24    Log.d(TAG, "Writing Exif unsuccessful.");
      e.printStackTrace();
    }
  }
```

Certain tags (exposure time, iso, date, time etc.) are extracted by the server in order to make it possible to look for specific images.

4.6 Background processing

Several tasks run in the background and do not block the main thread that is used by the main activity as illustrated in figure 4.6. Callback functions such as the picture callback or the HTTP POST execution run in the background until they are finished or aborted. This is managed internally by Android handlers and other structures. This sections discuss how this can be controlled.

4.6.1 Handler

To take pictures at a regular interval, a reliable timer needs to be created that is not affected by other tasks. Android offers several ways of creating time based, repeating functions. The way that this best in this application was by using the Handler's *postDelayed()* method.
A Handler object can be used to post data to the main thread. This is partic-

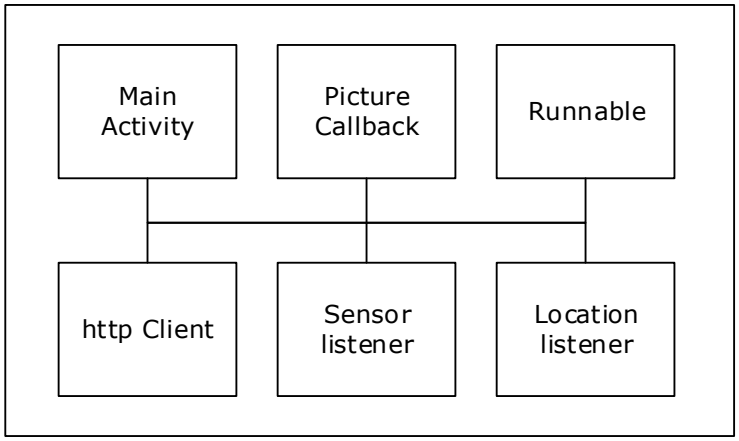


Figure 4.6: Independent application tasks.

ularly useful in this application as data has to be posted to the main thread multiple times. In this application, the Handler class is an instance of the *Runnable* class. A runnable is a type of class (an interface) that can be placed into a thread, describing the actions that the thread is supposed to execute. This Runnable Interface requires the class to implement the method *run*().

Listing 4.9: Handler and runnable.

```
final Handler handler = new Handler();
final Runnable runnable = new Runnable() {
  @Override
  public void run() {
    try{
      Date d = new Date();
      CharSequence dateTime = DateFormat.format("yyyy-MM-dd
          hh:mm:ss", d.getTime());
      d(TAG, "Runnable execution started. Time: " + dateTime + ".
          Interval: " + pictureInterval + " min.");
      //start imaging process
      runImagingTask();
    }
    catch (Exception e) {
      d(TAG, "Error: Runnable exception.");
    }
    finally{
      //no action needed.
    }
    //post handler message after specific time given by
        "pictureInterval" in minutes.
    handler.postDelayed(this, pictureInterval * 60 * 1000);
  }
};
handler.post(runnable);
```

The implementation of the runnable in the prototype application is extended in such a way that it starts the fist runnable only at a full minute (eg. 16:02:00).

4.6.2 AsyncTask

The AsyncTask is used in the process of uploading images to the Server. It can report the progress of the running tasks and synchronize with the main thread. For the prototype application no custom AsyncTask was created but it was important to understand the way it works in order to find the best suiting way of running the imaging process at a regular interval.

4.7 Server connection

The following sections discusses the implementation of the server connection. The server connection turned out to be the most difficult part because it was not well documented and security issues had to be bypassed. The server URL is: `https://www.visuo.adsc.com.sg` (Login credentials required).

4.7.1 HTTP

The Hyper-Text Transfer Protocol (HTTP) is used to deliver files and data such as HTML files, images, query results etc. It functions as the spine of the World Wide Web [14]. To create good web applications it is important to understand HTTP.
A simple HTTP command and probably the most common, is fetching a web page through a user driven browser by providing a specific URL.
This protocol allows automation of the communication between a Web-Client and Server.

In this thesis HTTP is used to transfer files from the web-client to the server. Another protocol suitable for this application is the File Transfer Protocol (FTP). The main reason why HTTP is used in this case is because servers listen on default port 80. FTP uses different ports that are usually blocked by the provider or require a manual configuration to allow using custom ports. HTTP does not need to communicate through port 80 but can use any other port.
HTTP is a stateless request-/response-protocol. This means that neither client nor server remember any communication status. The two most important commands are GET and POST. In this case GET is used to retrieve information from the server and POST to upload images. Picture 4.7 shows the http communication path between client and server. A request consists of three parts:

- Status line: Contains http-version, status code of the request and a text.
- List of response headers: Different headers are possible. In this case the authentication is placed here..
- Entity body: This part contains data of the request. For instance a jpeg file.

In this application the process works as follows:

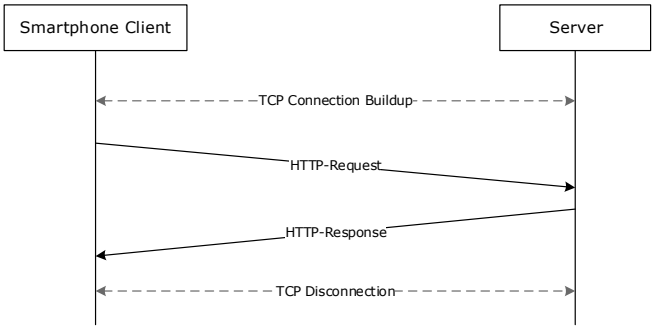


Figure 4.7: HTTP communication.

1. Client gets hostname of the server from the URL and obtains its IP via DNS.
2. Client builds a TCP connection to the socket of the webserver with port number 80.
3. Server accepts the connection.
4. Client sends a HTTP command (POST)
5. Client sends data files as JSON object.
6. Client waits for response.
7. Server sends status line (response header) with three fields (version, statuscode, readable text) (eg. HTTP/1.1 200 OK)
8. Server sends HTTP-Header (eg. contenttype:text/html)
9. TCP-connection disconnects if no "keep alive" header has been placed.

The statuscode 200 indicates that the client request has been successfully finished. 400 means that there is a syntax error and 500 signalises an error on the server.

4.7.2 HTTPS, SSL and TLS

The server used in this application has a security layer that causes an encrypted communication between client and server. HTTPS is just an extension of HTTP that uses port 443 instead of 80. The Transport Layer Security (TLS) is an encryption protocol for data transmission that has become a standard amongst web-browsers and web-servers [14].
Normally, in the connection build up phase a key distribution happens. This happens on a handshake basis where secret keys are exchanged. After that step the according public keys in form of digital certificates (X.509-Certificate[3]) are

[3]X.509 is a ITU-T-Standard for digital certificates.

securely exchanged. These public certificates are then used by both communication parties (client and server) to provide a secure, tap-proof communication. This is important when sending login details but causes a large protocol-overhead.
Since HTTPS relies on signed certificates, the security is not guaranteed if no such certificate exists. In the case of the server at the NTU/ADSC no such certificate was initially installed. This can lead to man-in-the-middle attacks.
In the past, signed certificates were very expensive because only certification authorities (CA) can issue digital certificates. In the process of writing this thesis a public CA was being used to provide a secure SSL connection.
Clients like Android smartphones contain a list of trusted CA's. When building up a HTTPS connection the certificate needs to be validated against its list of trusted CA's. This was not possible in this case, thus this validation needed to be bypassed by simply trusting any CA or completely ignoring the validation.

After bypassing the CA validation, the security issue still existed therefore a new Certificate Authority had to be found. *Let's Encrypt* is such an authority that only launched in April 2016. It provides free X.509 certificates for Transport Layer Security (TLS)[4]. It uses an automated process designed to eliminate the complex process of manual cretaion, validation, signing, installation and renewal of certificates for secure websites. More details about internet security goes beyond the scope of this thesis.
Each WAHRSIS has its unique username password combination that is needed to access the server. Instead of having to authenticate each protected resource with a username and password, the user (here the imager) authenticates itself with a given authorisation token. The token has the form of a base64 encoded hexadecimal string of varying length, depending on username and password [15].

4.7.3 Implementation

The following subsections show the current implementation of the communication process. It follows these steps:

1. Create client object with specific URL and authorisation token.
2. Add authorisation header.
3. Check internet connectivity.
4. Load three images into *params* object.
5. Execute POST request with params object.
6. Evaluate response code.
7. Idle.

[4] letsencrypt.org

Table 4.1: HTTP classes and libraries comparison.

Library	*Pro*	*Contra*	*Comment*
Volley	Official Android library	bad documentation, few examples	no success, no longer supported
android-async-http (Loopj) [16]	callback based, supports streamed JSON uploads, supports multipart file uploads, includes examples	only works on API 23 or higher	based on Apache HttpClient 4.3.3, best and simplest solution
Apache's HttpClient [17]	good documentation, light	only low level functions available	no success

Before implementing the POST command, a simple GET command was sent to the server in order to test the connection. Since the POST command is more complex, different libraries and classes were being evaluated in terms of functionality and simplicity. After various attempts and research on Stackoverflow [5], the android-async-http/loopj library was found which is based on Apache's HttpClient but offers more functions such as GET/POST params builder (RequestParams) or multipart file uploads. The main advantage of this client is that it allows making asynchronous HTTP requests. All HTTP requests happen outside the UI thread and are independent of the main activity. The responses are handled in anonymous callbacks. This is similar to the case of the camera picture callback which also happens in parallel to the main thread.

Authentication

Listing 2.1 shows the constructor code for the client class constructor. The necessary parameters are the context[6], the URL and the authorisation token. The URL does not change in this application but could be altered in the future, hence it appears in the list of parameters.

① Here the client is set up in such a way that ignores the certificate validation. If this line is deactivated no SSL/TLS validation bypass happens in the application.

② The authentication header is being created. Now a POST request to the visio server url would require that an authorisation header is present containing the correct token. Here it would look like:

[5] stackoverflow.com

[6] The underlaying object to access Android system services and resources such as location, notification, messages etc.

”Authorization: Token ”2c4c10a44874e3d7ee6656cc3c548febbc38f8” (randomly generated token).

Listing 4.10: Client Constructor.

```
//WSIServerClient constructor
public WSIServerClient(Context mContext, String url, String
    token) {
  this.mContext = mContext;
  clientUrl = url;
  //this enables to bypass the certificate validation
  permission issue
 (1) client.setSSLSocketFactory(MySSLSocketFactory.
    getFixedSocketFactory());

  //authentication header
 (2) client.addHeader("Authorization", "Token " + token);
  Log.d(TAG, "AsyncHttpClient succesfully created.");
 }
```

Preparing of data for request

Now that the connection is build up, the HTTP request needs to be prepared. Listing 2.2 shows the code snippet for the file preparation. The method *http-POST* requires the parameters *timeStamp* and the WAHRSIS Model Nr. The time stamp is needed so that only the most recent row of images is uploaded. The model number is used on the server to distinguish the different imagers. Since this application could run on different models simultaneously, the model number needs to be variable.
The http post command only allows to send ASCII plain text files without any compression. In the first approach, the image files were stored as a bytestream which would then be placed into the post request. Later, a Multipart Entity was used to attach images to the POST request. All these approaches did not lead to success. A different approach was taken through using the LoopJ library that allows adding images directly. The underlying structure of the post format is in JSON (JavaScript Object Notation). JSON is completely language independent and creates a standard of building data structures. A JSON Object begins with a left bracket and ends with a right bracket, containing an unordered set of name/value pairs [18]. Names are followed by a colon and the name/value pairs are separated by a comma. The following listing shows an example of JSON object that is used on the server. In this case it is the low EV image.

Listing 4.11: JSON Object

```
"imageLow": {
    "type": "image upload",
    "required": false,
    "read_only": false,
    "label": "Imagelow"
}
```

The time stamp follows the nomenclature of the existing WAHRSIS. This naming convention demands the format YYYY-MM-DD-HH-MM-SS-wahrsisN.jpg where N stands for the number of the specific WAHRSIS eg. 2016-11-22-14-20-01-wahrsis5.jpg. The time stamp is created in the main activity when the image row is about to be taken. The file path that leads to the WSI subdirectory (Line 3) is created in the main activity and should always exist. If this folder or the image files can not be found an exception is triggered (Line 16).

Listing 4.12: http POST part 1

```
public int httpPOST(String timeStamp, int wahrsisModelNr) {
  //file management
  String filePath =
      Environment.getExternalStorageDirectory().getPath() +
      "/WSI/";

  //prepare files to upload (normal image, low, med, high ev
      photo)
  File imageFileLow = new File(filePath+timeStamp+"-wahrsis" +
      wahrsisModelNr + "-low" + ".jpg");
  File imageFileMed = new File(filePath+timeStamp+"-wahrsis" +
      wahrsisModelNr + "-med" + ".jpg");
  File imageFileHigh = new File(filePath+timeStamp+"-wahrsis" +
      wahrsisModelNr + "-high" + ".jpg");

  //create object that contains the images
  RequestParams params = new RequestParams();
  try {
    params.put("imageLow", imageFileLow);
    params.put("imageMed", imageFileMed);
    params.put("imageHigh", imageFileHigh);
  } catch(FileNotFoundException e) {
    Log.d(TAG, "Could not find file " + imageFileLow + " and
        others (med, high).");
  }
  ...
```

Execution

The execution of the HTTP POST command only requires the specific URL, the params (containing the images) and a JSON response handler. The response handler is required to evaluate the response from the server. In this case, this can contain error messages or the URLs of the uploaded images (See Listing 2.5). Depending on the response of the server, the *client* method either enters the *onSuccess* or the *onFailure*. This is automatically managed by the LoopJ library. In the end, the http status code is returned to the main class where a validation of the upload can be done.

Listing 4.13: http POST part 2

```
  client.post(clientUrl, params, new JsonHttpResponseHandler() {
    @Override
    public void onSuccess(int statusCode, Header[] headers,
        JSONObject response) {
```

```
        Log.d(TAG, "AsyncHttpClient onSuccess, received JSON
            object: " +     response.toString());
        Log.d(TAG, "Http Status Code: " + statusCode);
        httpStatusCode = statusCode;
      }

      @Override
      public void onFailure(int statusCode, Header[] headers,
          Throwable e, JSONObject response) {
        Log.d(TAG, "AsyncHttpClient onFailure. Status code: " +
            statusCode);
        Log.d(TAG, "Response JSON: " + response.toString());
        httpStatusCode = statusCode;
      }
    });

    return httpStatusCode;
  }
```

If the authentication fails the response is logged as follows:

Listing 4.14: Unsuccessful authorisation or other errors.

```
D/ContentValues: AsyncHttpClient Failure. Status Code: 403
D/ContentValues: Response JSON: {"detail":"Invalid token header.
    No credentials provided."}
....
D/ContentValues: AsyncHttpClient Failure. Status Code: 500
D/ContentValues: Response: <h1>Server Error (500)</h1>
```

The status code 403 stands for "Forbidden". This means that the server understands the request but is refusing to execute it [15]. This can be caused if the authorisation token is void. Status code 500 means "Internal Server Error". The reason for this can have different origins. One possible cause is the complete lack of an authorisation header or issues on the server.

A successful upload results in the output shown under listing 4.15. Code 201 stands for "created" which means that the newly created resource can be referenced by the URI returned in the entity of the response. "image" is the HDR image that is generated on the server, hence the name "TODO.jpg".

Listing 4.15: Successful upload log.

```
D/ContentValues: AsyncHttpClient onSuccess, received JSON
  Object:
  {"image":
    "https://www.visuo.adsc.com.sg/media/sky_images/
    TODO.jpg",
  "imageLow":
    "https://www.visuo.adsc.com.sg/media/
    sky_images/2016/12/19/2016-12-19-12-25-04-wahrsis5-low.jpg",
  "imageMed":
    "https://www.visuo.adsc.com.sg/media/sky_images/
    2016/12/19/2016-12-19-12-25-04-wahrsis5-med.jpg",
  "imageHigh":
    "https://www.visuo.adsc.com.sg/media/sky_images/2016/12/19/
```

```
     2016-12-19-12-25-04-wahrsis5-high.jpg"}
D/ContentValues: Http status code: 201
```

4.8 Graphical User Interface

Android layouts are based on XML files. Although Android Studio offers a way to graphically creating and edit layouts, a basic understanding of the xml interface is necessary to create the desired look of layouts. The xml files can be found in the apendix under A.2. The basic element of creating user interfaces (UI) for Android is the activity class. Each activity is a single, focussed view that a user can interact with. The activity class manages the creation of windows that can be filled with any UI. This application consists of the main activity that is shown after the start up, the preference screen and an about activity that displays some brief information about the application. This section focusses on the layout and the user interface and not on the activity life cycles or their configurations.

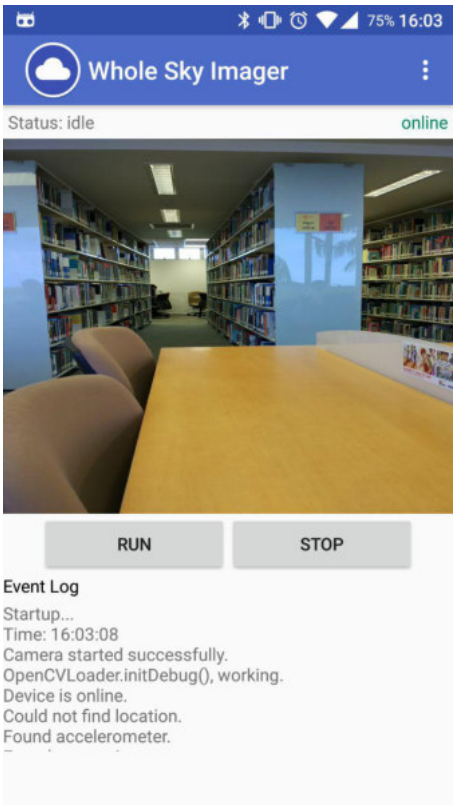
4.8.1 Main Activity

The main activity depicted in figure 4.8a consists of only a few elements (from top to bottom). The xml code to create such a UI can be seen in the appendix under A.2. It is meant as an example of how the UI creation works. The preference and about screen are built in a similar way (Code can be accessed via GitHub).

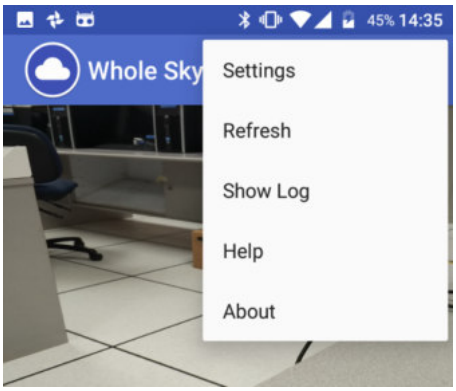
- Actionbar with the logo, title and menu of this application.
- A simple status bar that shows basic information.
- SurfaceView that shows the camera preview.
- Buttons to start and stop the capture mode.
- A scrollable TextView that contains logging information.

4.8.2 Preferences and About

The preference screen as shown in figure 4.9a is scrollable and can be easily extended or changed. All parameters can be accessed by all classes throughout this application. At the start of the application all parameters are read. The parameters are stored in a xml file on the device. If the option "Create HDR Image" is set, the device uses the standard HDR function provided by Android to create a tonemapped HDR image.
The option "Keep Files" is left for debugging purposes, however it can cause the application to abort if no storage space is available.
It should be possible to remotely change these parameters by changing specific values in the preferences xml file. A better solution would be to fetch all values from a server, although this would make the settings activity redundant (see chapter "Future Work" 6.2). Figure 4.9b shows a basic "about" screen.



(a) Main activity layout.



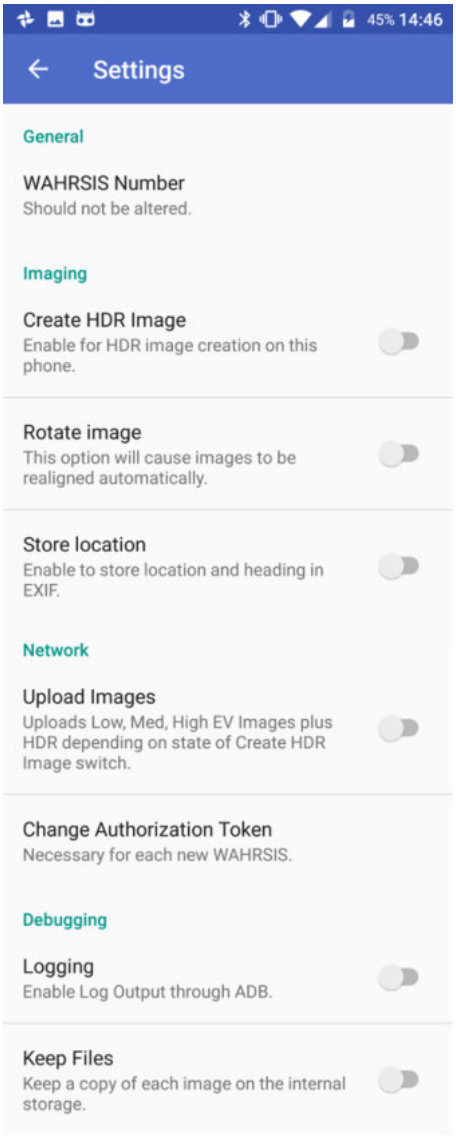
(b) Menu layout.

4.8.3 Prototype application

All parts were merged to create the application for the new WAHRSIS 5. A flowchart of the application is depicted in figure A.2 on page 56. For specific services such as accessing the internet or using the camera Android requires an application to ask for permissions. The following permissions are requested by this application:

- WAKE_LOCK: This is needed to keep the phone awake and not step out of the running application.
- CAMERA: To use all camera related features.
- WRITE_EXTERNAL_STORAGE: To write files on the storage.
- ACCES_NETWORK_STATE: To test the network connection.
- INTERNET: To access web servers, URLs etc.
- ACCESS_COARSE_LOCATION: This provides a coarse location by using wi-fis, gsm stations etc.
- ACCESS_FINE_LOCATION: This is requested in order to use GPS as a location provider.

The permissions are requested in the Android manifest as shown in listing A.1 on page 57. The whole project can be accessed via GitHub under the following url: `https://github.com/JulMueller/WholeSkyImager`. The Android device used for testing was a Xiaomi Mi 4c with an API level 23. Revelant specifications for this device are the screen resolution (1080x1920 pixels) and the camera (13 MP, f/2.0, 24mm).



(a) Preferences Activity.

(b) About screen.

Figure 4.9: Secondary user interfaces.

Chapter 5

Housing

To protect the smartphone and other electrical components from the hot and humid climate in Singapore a housing was necessary. The prototype would have to be portable, self-sufficient and easy to install. This section explains the requirements for such a casing, the different possibilities and the steps needed to build one.

5.1 Environment

On average in Singapore it rains on 178 days of the year (0.2mm of total rainfall or more). From November to January the highest rainfall occurs. This is caused by the major tropical rainbelt which is positioned near Singapore during that period. Based on long-term records from 1869 to 2015, the mean annual rainfall total is 2331.2 mm[1].
Temperatures in Singapore vary little from month to month and also from day to day. The minimum daily temperature is not usually below 23-25°C during the night and does not rise above 31-33°C during the day. Relative humidity also shows a fairly uniform distribution throughout the year. The daily variation in humidity shows a clear pattern with usually more than 90% in the morning before sunrise. Until mid-afternoon the humidity falls to around 60% if no rain occurs. After sunset the humidity rises again above 80%.
Understanding the climate in Singapore is important in order to build a suitable housing. The high humidity in the morning causes the lens dome to fog. Some sort of ventilation is needed to prevent this. By ventilating the housing the temperature difference between the inside and outside should be close to zero. The large number of rainy days indicate that the housing needs to be perfectly waterproof. Another factor to keep in mind is the high temperature and strong irradiance causing stress on the material and a further increase in temperature in and outside the housing.

The finished housing will be placed on a rooftop on the Block 2 at the Nanyang Technological University. Due to the proximty to a large area of woodlands the humidity and rainfall rate is even higher than the mean rainfall in Singapore.

[1]Source: http://www.weather.gov.sg/climate-climate-of-singapore/

5.2 Functional Specifications

The following list shows the required specifications for a prototype housing. Most of these requirements follow the needs caused by the local climate explained in section 5.1.

- Rainproof
- Heat-resistant up to 45° C
- Lightweight and portable.
- Ventilated
- Low-priced
- Self-sustaining
- Easy to install and maintain.

Previous WAHRSIS models built at the Nanyang Technological University consist of large metal or plastic housings and acrylic domes to cover the DSLR and the wide angle lens as shown in figure 1.1. One problem that became apparent was that the lens cover would fog in the morning, thus making it impossible to capture detailed images. Different approaches were taken to circumvent this problem by either using simple fans to ventilate the casing or by using active cooling elements.
Another issue that developed was corrosion. The first WAHRSIS had a mechanical sun blocker that was driven by multiple stepper motors. These motors and other mechanical elements malfunctioned after a while due to the high humidity and the frequent occurrence of rain.
WAHRSIS 2, 3 and 4 do not contain a mechanical sun blocker but use HDR technology to create highly detailed images. Unfortunately, the high humidity and temperatures can still cause electronics to fail. Additionally, all these WAHRSISs need to be powered externally and connected to the Internet via an Ethernet cable (or in range of a Wi-Fi network) to provide remote connection for maintenance and access to the server.

5.3 Options

Previous research [19] at the NTU has shown that in a ventilated casing the temperature and humidity follow the trend of the environment whereas in a closed box the temperature increases steadily during the day under sunshine and the humidity does not vary much. Temperatures up to 45 °C have been measured in a closed box.

5.3.1 Possible housings

Custom 3D Print A custom 3D print would help the imager to be compact and perfectly adapted to the internal electronics. However creating a

custom housing that is rugged and waterproof is time consuming and expensive.

Metal housing A metal box would diminish the wireless connectivity of the smartphone and is prone to corrosion if steel is used. A waterproof aluminium case would be a long lasting solution but is much higher in price than a PVC case.

Plastic case A PVC case can deform under high temperatures and can be destabilized by UV radiation. Thus the life time is limited. The readily available waterproof cases offer an easy to manipulate and low-priced base for a whole sky imager.

A polyprolyen (PP) case was chosen as the base of the new imager. PP is resistant to fatigue, is tough and has a good heat resistance. The maximum continuous use temperature is 80° C[2].

5.3.2 Lens cover

To protect the wide angle lens from the elements, dirt and dust, a clear cover was needed.
In terms of translucency and ease of installation a glass pane would be the best solution. A flat surface has the disadvantage of accumulating dust over time thus making it impossible for the imager to operate for a long period of time. Another disadvantage of a flat surface cover is the property of breaking light hence decreasing the actual usable field of view of the lens.
Because of the disadvantages of a flat pane a hemispheric dome needed to be found. Glass was the preferable material but hard to get and high in price. Glass has an ideal translucency, does not age through UV irradiance (yellowing) and is heat resistant. For this prototype an acrylic dome was chosen because of the low price and the availability. In a previous thesis [19] it has been shown that custom glass domes are costly and take time to manufacture. During testing the acrylic, dome caused little distortions in the image. Therefore it is recommended to install a glass dome or a better quality acrylic dome in the future that does not cause any distortion, does not yellow and is scratch resistant.

5.4 Prototype

Figure 5.1 shows a model of the mobile WAHRSIS prototype. This imager will be called WAHRSIS 5 or mobile WAHRSIS. The smartphone and its detachable lens are protected by an acrylic dome. Next to the dome a detachable 10W solar module is held in place by Velcro fastener. Because of the lack of space between the module and the case, the case suffered a drastic increase in temperature. To allow air circulation between the solar module and the case, two 2cm thick aluminium profiles can be placed in between.

[2]British Plastics Federation

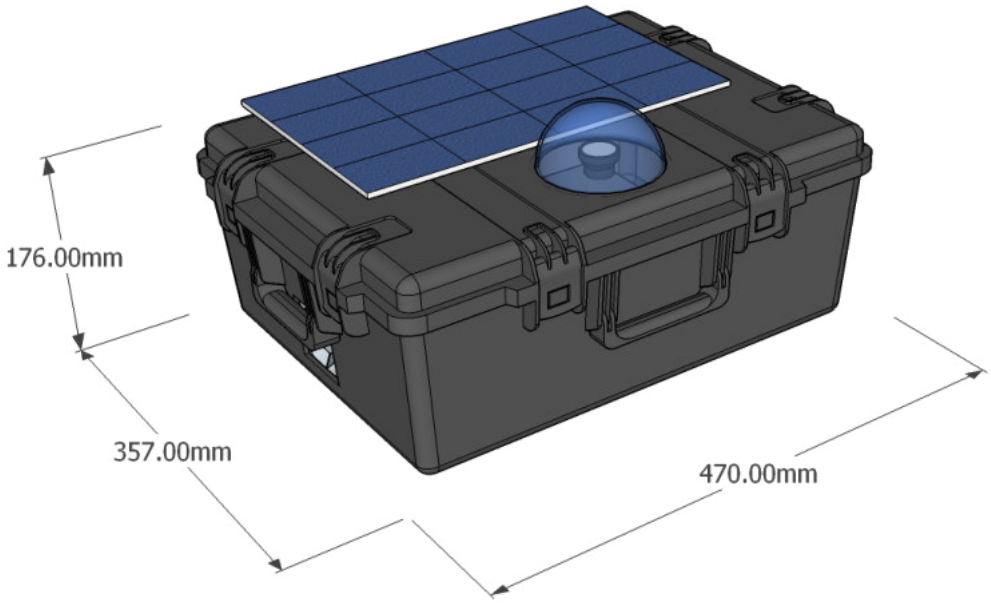


Figure 5.1: Model of first prototype.

5.4.1 Components

The solar panel powers a 80 mm fan to maintain a steady stream of air along the acrylic dome. A DC Step Down Converter provides a 5V output for any Android smartphone. Figure 5.2 shows an overview of the electrical components used and their connection. The following table 5.1 lists all components and their prices. The price of the Smartphone may vary, but using a modern device that supports the newest APIs and exhibits high quality camera is recommended. Additionally, never devices will provide better image quality and faster processing speeds while being more energy efficient. The holes for the

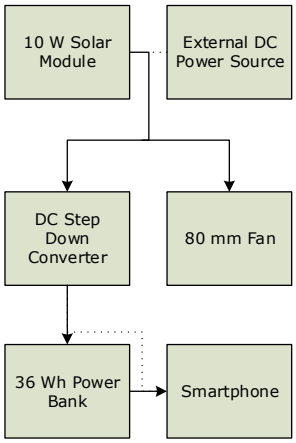


Figure 5.2: Electrical components.

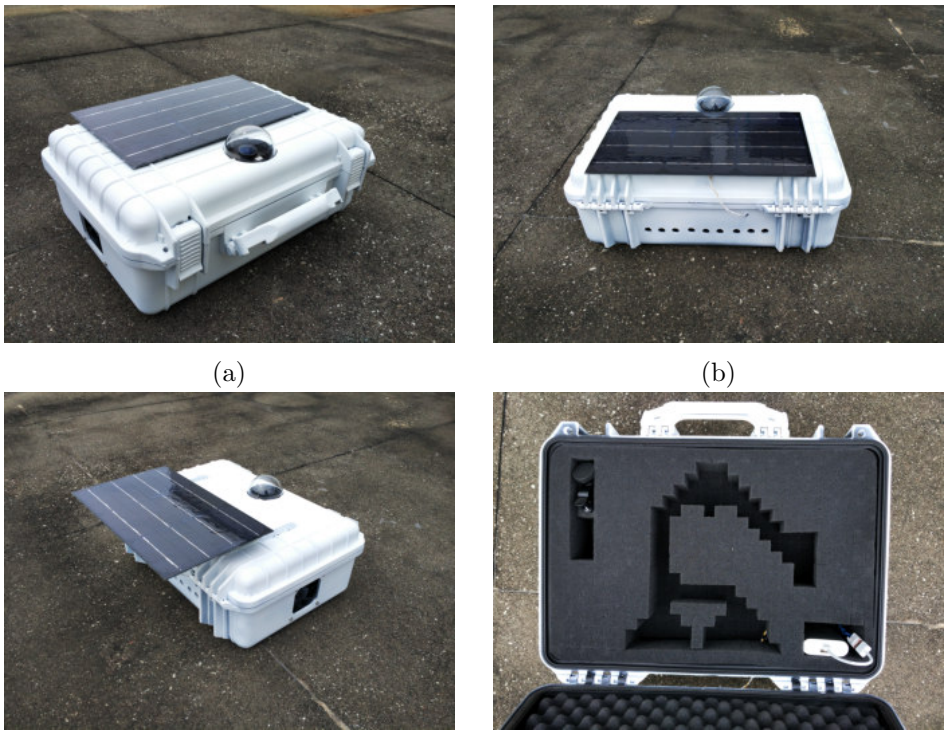
Table 5.1: Prototype components.

Part Name	*Description*	*Distributor*	*Price (SGD)*
Case	Waterproof case 18", 470x357x176 mm, PP, BLK	Element14	35.93
Lens Dome	Dummy 2.5" Security Camera	San-Ace Electronics	4
Fish-eye lens		approx. 20	Lazada
Solar Module	WiseBuy Monocrystalline Silicon Hobby Solar Panel 18V 10W 555mA, 344x208 mm	lazada.sg	35.03
Fan	80mm 12V 0.07 ± 0.02 Amp 24CFM Cooling Fan	lazada.sg	5.10
Step Down Converter	6-24V to 5V 3A DC Buck Step Down Converter	lazada.sg	3
Power bank	Xiaomi 10000 mAh	22	Challenger Singapore
Miscellaneous	Velcro, Screws, Nuts etc.	Selffix	ca. 20

fan, ventilation and the dome were drilled at the Robotic workshop at the NTU. Since the case is black it gets very hot. The small gap between the solar panel and the case allows for ventilation and the box to stay shadowed. This way overheating can be avoided.
The box should withstand rain and storms, whilst remaining portable and light. Thus it needs to be kept in place on a flat roof by a rope or similar concept. The case could also be weighed down by placing heavy objects inside such as a large power bank.

5.4.2 Completion

Picture 5.3 depicts the finished mobile whole sky imager. The black case was painted white by using spray paint. This results in a higher reflection of sunlight thus minimising the heating of the inside. The row of holes on the rear side of the case provide ventilation. The fan sucks air in on the left side and directs it towards the lens dome before it leaves the case through the holes on the rear side. To form an air flow simple pluck foam was used, this can be seen on picture 5.3d. The detachable wide angle lens and the power bank can also be seen on this picture. Image 5.3c shows how the distance between the solar module and the case can be increased through two aluminium profiles. Figure 5.4 is a HDR tonemapped image taken from inside the case.
The total cost of this prototype without the smartphone is about 125 SGD.



(a) (b)

(c) (d)

Figure 5.3: WAHRSIS5 prototype

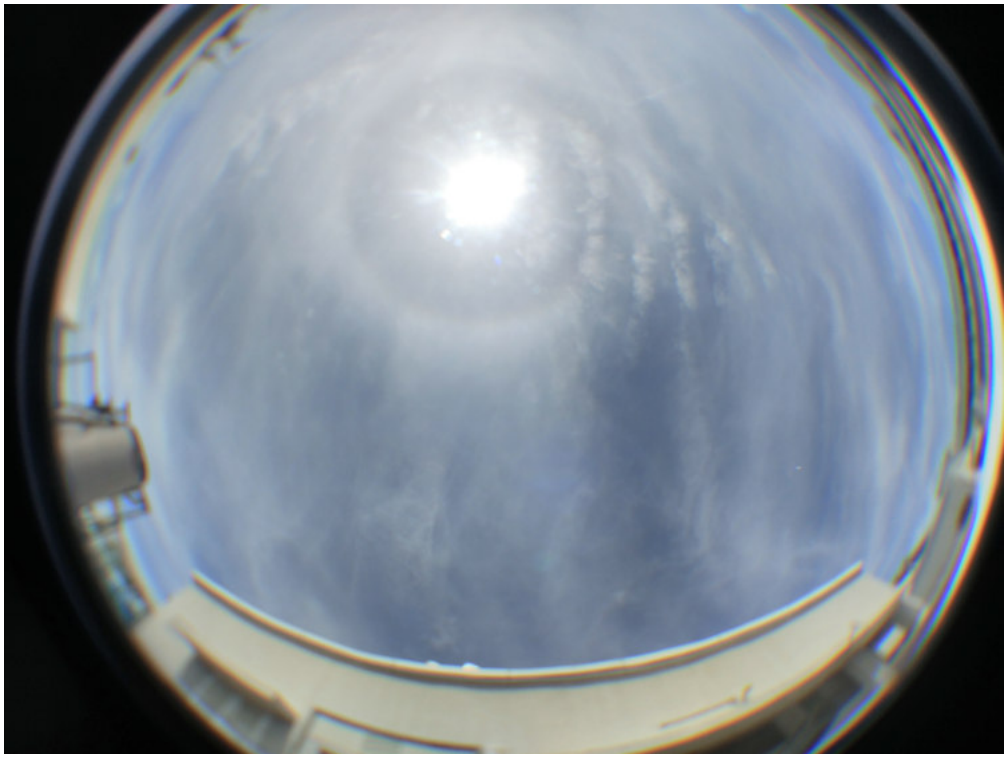


Figure 5.4: HDR image from inside the case.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In the beginning the primary objective was to create a basic smartphone application capable of capturing images at regular intervals and uploading these to the already operating server. This task could be solved by using multiple libraries and their classes in the Android environment. In the process of implementing parts of the whole application, it was discovered that Android often offers multiple ways of implementing a specific task. It turned out to be difficult to find working solutions and time consuming to test various attempts before finding a reliable solution.
In addition, a portable low maintenance casing had to be made capable of protecting the smartphone and other electronics from the humid and hot climate in Singapore. This newly created case is based on a rugged, waterproof case. Even though holes were drilled, it is still rainproof and can operate without any external power source for a limited time. It is recommended to power the case externally to ensure its functionality throughout cloudy days. Corrosion is of no concern as no metal parts are exposed to the outside. A fan prevents the lens dome from fogging up. In this prototype, a low price acrylic dome serves the purpose of protecting the lens. Although the distortion in the lower area of the dome is prominent, the images taken through the dome are of good quality.
The created application takes pictures on a desired, regular interval, is capable of preparing images in different ways if this is desired and uploads them onto a server. The smartphone can be controlled remotely by using third party applications, assuming it has access to the Internet. The created Whole Sky Imager is low cost, easy to maintain, light and thus portable and can operate without additional electricity. With the increasing capabilities of smartphone cameras this approach can be a versatile low cost and portable alternative to common Sky Imagers. The functioning WAHRSIS5 can be of great use for the Sky/Cloud Analysis research team as it provides more and new data.

6.2 Recommendation in Future Work

The application can be further enhanced and several other functionalities can be added. The application can be made available for other platforms like Windows, Apple and Blackberry. The present system is only applicable to Android devices.
The housing can be improved to provide a more reliable power source for instance by adding a larger solar panel. As mentioned previously, the lens dome could be replaced with a high quality glass dome to decrease the image distortion.
In the future, the smartphone's orientation data (azimuth, pitch, roll) could be used to create fused 3D images by using multiple smartphone based imagers. The image quality can be enhanced by fine tuning the various camera parameters. A two way communication with the server would be useful to remotely start and stop the capturing mode and to fetch or store preferences. The lens calibration parameters could also be stored in the preferences and shared with the server.
Adding temperature and humidity sensors or using a smartphone with these sensors in built could provide more useful data. Further, the power consumption of this application could be optimized. Finally, long term testing should be done in order to proof the reliability of this approach.

Bibliography

[1] Y. S. Florian M. Savoy, J. C. Lemaitre, "Cloud base height estimation using high-resolution whole sky imagers," *International Geoscience and Remote Sensing Symposium (IGARSS)*, 2015.

[2] Google, "Android developers," https://tools.ietf.org/html/rfc7235, 2016, [Online; accessed December 2016].

[3] Y. S. Florian M. Savoy, Soumyabrata Dev, "Design of low-cost, compact and weather-proof whole sky imagers for high-dynamic-range captures," *International Geoscience and Remote Sensing Symposium (IGARSS)*, 2015.

[4] OpenCV, "High dynamic range imaging," http://docs.opencv.org/master/d3/db7/tutorial_hdr_imaging.html, 2016, [Online; accessed December 2016].

[5] M.-P. T. Tian-Tsong Ng, Shih-Fu Chang, "Using geometry invariants for camera response function estimation," *Ecma International*, 2007.

[6] J. M. Paul E. Debevec, "Recovering high dynamic range radiance maps from photographs," *Ecma International*, 1997.

[7] C. in Colour, "Using wide angle lenses," http://www.cambridgeincolour.com/tutorials/wide-angle-lenses.htm, 2011, [Online; accessed 13 November 2016].

[8] M. A. Scaramuzza, D. and R. Siegwart, "A Flexible Technique for Accurate Omnidirectional Camera Calibration and Structure from Motion," *Proceedings of IEEE International Conference of Vision Systems (ICVS'06)*, January 5-7 2006.

[9] ——, "A Toolbox for Easy Calibrating Omnidirectional Cameras," *Proceedings to IEEE International Conference on Intelligent Robots and Systems (IROS 2006)*, October 7-15 2006.

[10] S. D. Rufli, M. and R. Siegwart, "Automatic Detection of Checkerboards on Blurred and Distorted Images," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2008)*, September 2008.

[11] Y. S. Florian M. Savoy, Soumyabrata Dev, "Geo-referencing and stereo calibration of ground-based whole sky imagers using the sun trajectory," 2016.

[12] Google, "Android developers," https://developer.android.com/reference/android/hardware/Camera.html, 2016, [Online; accessed December 2016].

[13] S. of Japan Electronics and I. T. I. Association, "Exchangeable image file format for digital still cameras: Exif Version 2.2 ," April 2002.

[14] J. Peter Mandl, Andreas Bakomenko, *Grundkurs Datenkommunikation : TCP/IP-basierte Kommunikation: Grundlagen, Konzepte und Standards*. Wiesbaden: Vieweg + Teubner in GWV Fachverlage, 2008.

[15] N. W. Group, "Hypertext transfer protocol – http/1.1," https://www.w3.org/Protocols/rfc2616/rfc2616.html, 1999, [Online; accessed December 2016].

[16] J. Smith, "Android asynchronous http client - a callback-based http client library for android," http://loopj.com/android-async-http/, 2016, [Online; accessed December 2016].

[17] T. A. S. Foundation, "Apache httpcomponents - httpcomponents http-client overview," https://hc.apache.org/httpcomponents-client-ga/, 2016, [Online; accessed December 2016].

[18] "The json data interchange format," *Ecma International*, 2013.

[19] D. Jokic, "Design and implementation of the whole sky imager," bachelor's thesis, Nanyang Technological University, HSR University of Applied Sciences Rapperswil, Singapore, December 2014.

[20] G. A. Dave MacLean, Satya Komatineni, *Pro Android 5*. Apress, 2015.

[21] W. Jackson, *Android Apps for absolute Beginner*. Apress, 2014.

[22] G. Allen, *Beginning Android*. Apress, 2015.

[23] J. Friesen, *Learn Java for Android Development*. Apress, 2014.

[24] C. C. Adam Gerber, *Learn Android Studio*. Apress, 2016.

[25] L. Gimena, "Exposure value in photography. a graphics concept map proposal," *Proc. of the First Int.Conference on Concept Mapping*, 2004.

[26] vogella GmbH, "Android development," http://www.vogella.com/tutorials/android.html, 2016, [Online; accessed December 2016].

Appendix A

Appendices

A.1 Figures

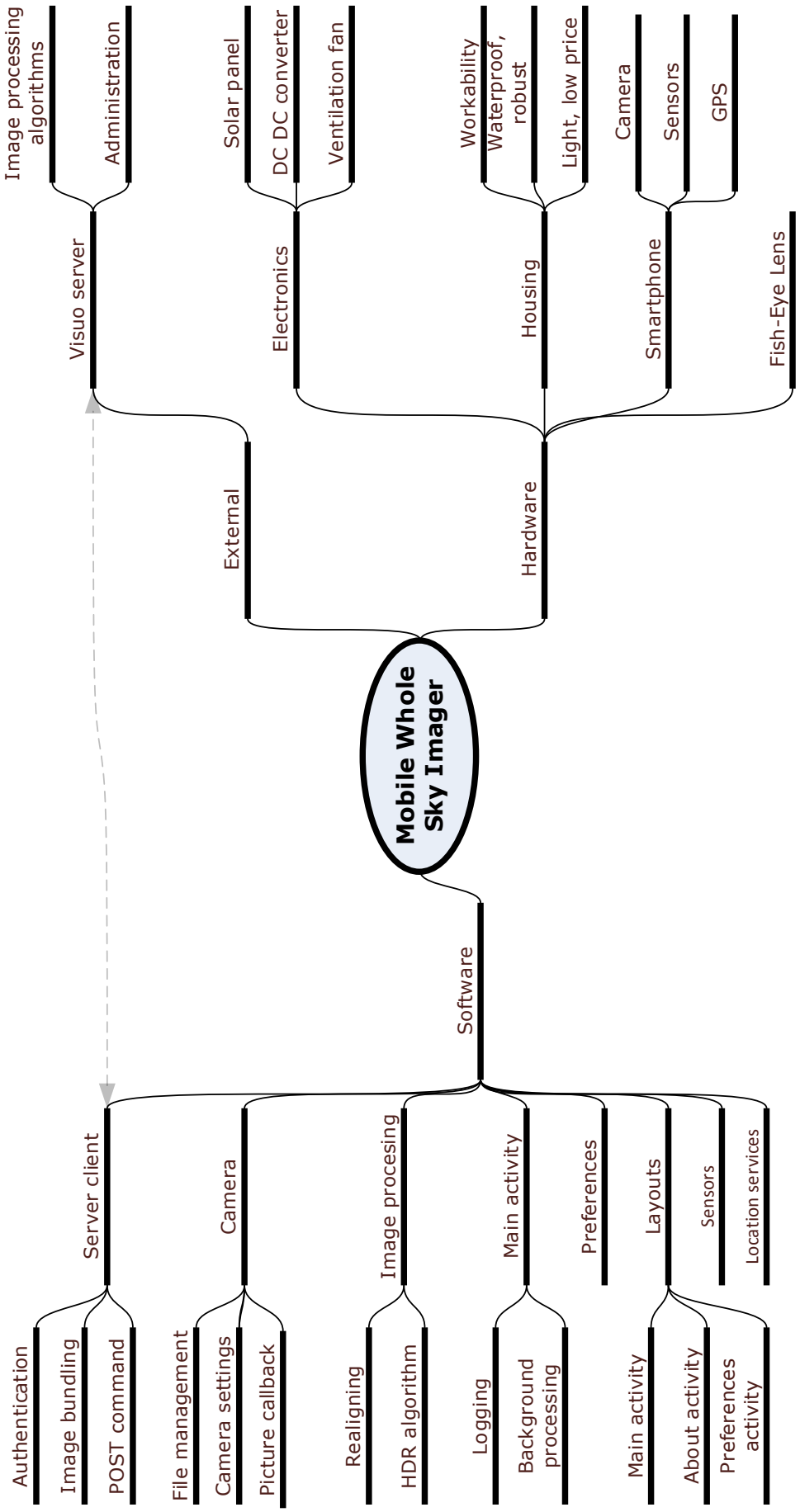


Figure A.1: Mobile WAHRSIS overview.

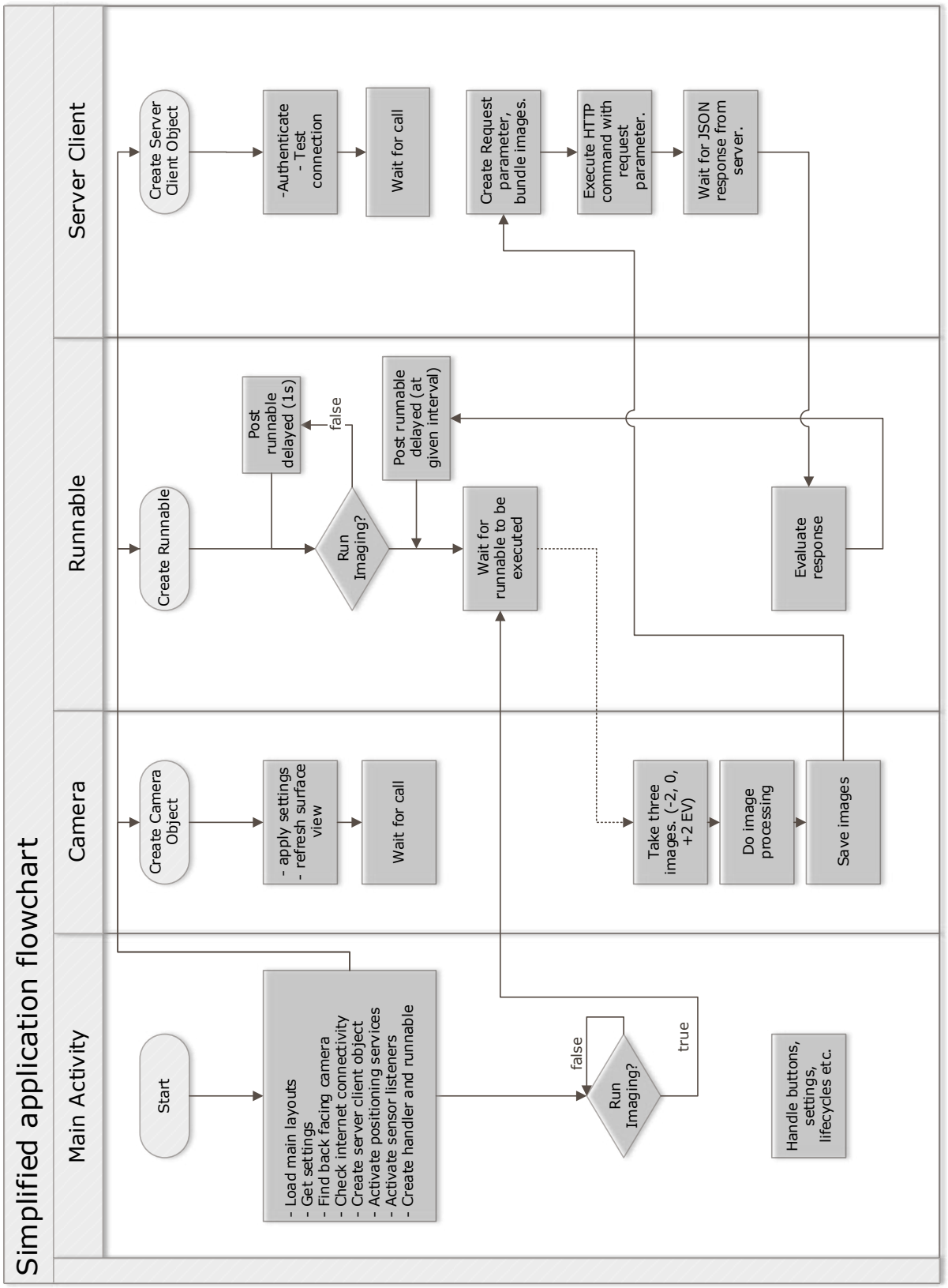


Figure A.2: Simplified application flowchart.

A.2 Listings

Listing A.1: Manifest permissions.

```
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION" />

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

Listing A.2: Main Activity Layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="ntu.com.wholeskyimager.MainActivity">

  <ImageView
    android:id="@+id/imageOutput"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="center"
    android:adjustViewBounds="true"
    android:background="#cccccc"
    android:maxHeight="150dp"
    android:maxWidth="400dp"
    android:minHeight="100dp"
    android:minWidth="400dp"
    android:nestedScrollingEnabled="false"
    android:visibility="gone"
    android:layout_weight="1.94"/>

  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:weightSum="1"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true">

    <android.support.v7.widget.Toolbar
```

```
    android:id="@+id/my_toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:elevation="4dp"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
    app:titleMarginStart="20dp"/>

  <GridLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginBottom="4dp"
    android:layout_marginLeft="4dp"
    android:layout_marginRight="4dp"
    android:layout_marginTop="4dp">
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:text="Status: "/>
    <TextView
      android:id="@+id/tvStatusInfo"
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:text="NULL"/>
    <TextView
      android:id="@+id/tvConnectionStatus"
      android:layout_width="wrap_content"
      android:layout_height="match_parent"
      android:layout_gravity="right"
      android:text="offline"/>
  </GridLayout>

  <FrameLayout
    android:id="@+id/camera_preview"
    android:layout_width="match_parent"
    android:layout_height="300dp"/>

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:orientation="horizontal">

    <Button
      android:id="@+id/buttonImport"
      android:layout_width="150dp"
      android:layout_height="wrap_content"
      android:onClick="startImaging"
      android:text="@string/button1_text"/>
    <Button
      android:id="@+id/buttonEdgeDetect"
      android:layout_width="150dp"
      android:layout_height="wrap_content"
      android:onClick="stopImaging"
      android:text="@string/button2_text"/>
  </LinearLayout>
```

```
    <TextView
      android:text="Event Log"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/textTitleEventLog"
      android:textColor="#000000"
      android:layout_marginBottom="3dp"/>
    <TextView
      android:text="Startup..."
      android:layout_width="fill_parent"
      android:layout_height="120dp"
      android:id="@+id/tvEventLog"
      android:maxLines = "50"
      android:scrollbars = "vertical"
      android:layout_weight="7.36"/>
  </LinearLayout>
</RelativeLayout>
```