Android Image stabilization using inertial sensor Data and Feature Tracking

Niall O’ Mahony
B.A.I. Engineering
Final Year Project April 2013
Supervisor: Dr. Fergal Shevlin

School of Computer Science and Statistics
O’Reilly Institute, Trinity College, Dublin 2, Ireland
DECLARATION

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

________________________________________  ______________________________
Name                                      Date
Acknowledgements

I would like to sincerely thank my supervisor Dr. Fergal Shevlin for his continuing support, advice and reassurance all throughout this project. I would not have achieved even as much as I did without your honest practical advice when I needed it.

To Maurice Gavin, who kindly shared his knowledge from time to time and was a huge help for me whenever I really needed an extra little bit of quick know how to solve a problem, I sincerely thank you for your patience and all your advice.

To all my friends for their support, encouragement, kind words and kind messages, thank you.
Abstract

Almost all android devices contain cameras of some sort but do not offer much in the way of image stabilization. Considering that the use of the camera primarily is a person recording a still object whilst usually holding the device in their hand(s), correction of handshake and other small movements would be of great benefit to Android users.

The goal of this project was to create an Android application that would use the device’s internal sensors as well as feature tracking to gain enough information to “stabilize” the image (i.e. cancel out movement from things like handshake) and do it in a format that can be viewed in real time as well as recorded.

This is a project that was attempted a year before using primarily feature tracking. That project however had very poor results in terms of efficiency, frame rate, the ability to capture and record data etc.

My aim was to use sensors as the primary method to stabilize the image and complete the project in a way that was fast, efficient and useful in terms of the ability to capture and record data which is what the user would mainly want.
<table>
<thead>
<tr>
<th>Chapter 1: Introduction</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Layout</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Rationale</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Design Choices</td>
<td>3</td>
</tr>
<tr>
<td>1.4.1 Android</td>
<td>3</td>
</tr>
<tr>
<td>1.4.2 Real Time Recording</td>
<td>3</td>
</tr>
<tr>
<td>1.4.3 Flexibility with Android Devices</td>
<td>4</td>
</tr>
<tr>
<td>1.4.4 Neat Coding and use of Multiple Classes and Threads</td>
<td>4</td>
</tr>
<tr>
<td>1.5 Obstacles</td>
<td>4</td>
</tr>
<tr>
<td>1.5.1 Efficiency</td>
<td>4</td>
</tr>
<tr>
<td>1.5.2 Lack of Functionality in the Android Platform</td>
<td>5</td>
</tr>
<tr>
<td>Chapter 2: Android Platform and Hardware Components</td>
<td>6</td>
</tr>
<tr>
<td>2.1 The Android Platform</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Android Camera</td>
<td>7</td>
</tr>
<tr>
<td>2.2.1 Implementation in Android</td>
<td>7</td>
</tr>
<tr>
<td>2.3.1 Linear Accelerometer</td>
<td>9</td>
</tr>
<tr>
<td>2.3.2 Gyroscope</td>
<td>10</td>
</tr>
<tr>
<td>2.3.3 Implementation in Android</td>
<td>10</td>
</tr>
<tr>
<td>Chapter 3: Additional Libraries and Features</td>
<td>11</td>
</tr>
<tr>
<td>3.1 OpenCV</td>
<td>11</td>
</tr>
<tr>
<td>3.1.1 Feature Tracking</td>
<td>12</td>
</tr>
<tr>
<td>3.1.2 Feature Tracking Implementation in Android</td>
<td>13</td>
</tr>
<tr>
<td>3.2 OpenGL</td>
<td>15</td>
</tr>
<tr>
<td>3.2.1 View</td>
<td>15</td>
</tr>
<tr>
<td>3.2.2 Lighting</td>
<td>16</td>
</tr>
<tr>
<td>3.2.3 Textures</td>
<td>16</td>
</tr>
<tr>
<td>3.3 Java Native Interface</td>
<td>19</td>
</tr>
<tr>
<td>3.3.1 Implementation in Android</td>
<td>19</td>
</tr>
<tr>
<td>Chapter 4: Application Design</td>
<td>21</td>
</tr>
<tr>
<td>4.1 Design Overview</td>
<td>21</td>
</tr>
</tbody>
</table>
4.2 My OpenGL Renderer ................................................................................. 24
  4.2.1 Setup Parameters .............................................................................. 24
  4.2.2 YUV to RGB ...................................................................................... 24
  4.2.3 Textures ............................................................................................ 25
4.3 Sensor Shifting .......................................................................................... 27
  4.3.1 Calibration ......................................................................................... 28
4.4 Feature Shifting ........................................................................................ 29
4.5 Sensor Graphs .......................................................................................... 30
  4.5.1 Readings ............................................................................................ 30
4.6 Save Corrected Video ............................................................................... 31
  4.6.1 Saving Edited Frames ....................................................................... 31
  4.6.2 Saving These Frames to Video ......................................................... 32
4.7 Camera Capture ......................................................................................... 32
Chapter 5: Design Outcomes ......................................................................... 34
  5.1 Rendering Camera Frames in OpenGL ................................................ 34
  5.2 Using the Sensors ................................................................................ 35
  5.3 Feature Tracking ................................................................................... 35
  5.4 Shifting the Image in Real Time ............................................................ 35
  5.5 Saving Video to File ............................................................................. 36
Chapter 6: Conclusion and Personal Reflection .............................................. 37
  6.1 Conclusion ........................................................................................... 37
  6.2 Future Work ........................................................................................ 37
  6.3 Other Uses for Application .................................................................. 38
  6.4 Personal Reflection ............................................................................... 38
Chapter 7: Figures, References & Appendices .............................................. 41
  7.1 Table of Figures ................................................................................... 41
  7.2 References ............................................................................................ 42
  7.3 Appendices ......................................................................................... 44
Chapter 1: Introduction

1.1 Overview
The purpose of the project as previously stated was to create an Android application for real time image stabilization and recording.

For the most part what I was trying to achieve very basically was an application that when the device was moved, the area of the camera you are trying to record would stay still and wouldn’t move with the device. See the figure below. The darkened background is just to give an idea of the movement of the device. The lighter central image is the corrected one.

![Image Stabilization Example](image.png)

Figure 1 Image Stabilization Example
As can be seen from the figure on the previous page, the goal was that however the user moves the device, the centre image will stay still. I attempted to implement this using a number of features that I will go through throughout this report.

In this section I will go into details of how the report is laid out, the rationale for this app, what decisions I made regarding how to approach the problem and list some of the major obstacles that had to be overcome in order for the project to be successful.

1.2 Layout
In chapter 1 as previously stated I will give an overview of the project, obstacles, decisions made etc. to give an overview of what I was trying to achieve.

In chapter 2 I will describe in detail the Android platform as well as the different hardware components required for this project as well as describing the features I used relating to these.

Chapter 3 will deal with describing any libraries or other software I used to get the project working (OpenGL, OpenCV etc.). I will describe them in general but go into detail the features I used for my project and how they are implemented in Android.

Chapter 4 will describe basically my design “Schematic”, detailing exactly how the project was to work with full functionality. This will go into detail on the design algorithm, how all the pieces work together etc.

Chapter 5 will detail how successful each area of my project was.

Chapter 6 will contain the conclusion and what I learnt from the project as well as detailing exactly what work is needed to bring the project to full functionality, other uses for the project and my own personal reflections.

Chapter 7 will simply be my references and appendices.

1.3 Rationale
Android is the world’s most popular operating system for mobile applications such as smartphones and tablets [1].

Nearly all of these devices contain cameras of some sort for use in capturing images and video. As they are mobile devices, they inevitably are primarily used by individuals holding them in their hand(s) which means that the majority of videos taken with these devices suffer from distortion due to small movements like body movements and hands shaking. As such, an application designed to reduce the effects of handshake would be hugely beneficial to a huge number of people. That is where this project comes in.

As well as a camera, nearly all of these devices feature internal sensors, and since they run on a common operating systems, they have all have similar capabilities in terms of
running features like feature tracking (although for CPU intensive applications performance varies from device to device).

As such it seems logical to use these features built into Android to gather as much information from the device as possible to improve how the camera operates in order to make it easier for the user to record smooth video without having to worry about holding the device completely still which can be impossible at times.

1.4 Design Choices

There are a number of choices I made regarding the design of this project. I will go into more detail about all of these later on in the project but for now I wanted to explain the reasoning behind some of my decisions regarding how the project is designed and why certain things are done certain ways when there may or may not have been easier solutions.

1.4.1 Android

I decided before I even knew I was working with image stabilization that I wanted to work with Android. I had some very minor experience with Android from an earlier project and I enjoyed working with it and was eager to learn more about it.

Android is the world’s most popular mobile operating system and as such if I were to make something useful for Android it would have the largest reach on this operating system as opposed to other operating systems.

Also, Android is open source, meaning that in order to develop for it, it is a simple matter to simply go onto their website and download the developer tools. Unlike iOS (Apple’s operating system for mobile devices like iPhone & iPad), there is no need to purchase a developer licence and no prerequisite hardware is required to be allowed to develop for it other than some computer capable of running Eclipse (the compiler I used to code the app) or some other similar compiler capable of working with Java and the Android SDK (Software Development Kit).

1.4.2 Real Time Recording

I decided from the beginning really that this project was going to work in real time. That is, the user would see on screen the video stabilized as they were looking at it/ recording.

The reason for this is that real time correction would make it much easier for the user to see exactly what they are recording as they are recording it which is far more beneficial than the video being altered after the user is finished recording.

Also since the sensors in the device (which will be explained in detail later on) are capable of working so quickly without much overhead it seemed like it should be possible to do this in real time and as such I made the goal of this project to be stabilization of the image in real time.
1.4.3 Flexibility with Android Devices

Android is an open source common environment, which means that in theory, an App that works on one device should work on them all. In practice this is not always the case and careful and considerate programming is often required to ensure that an App is flexible enough to work on all devices.

For example, new versions of Android are released every few months. These usually contain new features which is good, however, only the newest devices will be running this new version so these new features may not work on older devices. Therefore, care would have to be taken so as to ensure that the use of these new features does not prevent the application from working on most devices.

One of the main reasons I decided this is that when I first began, I started by looking at the project I mentioned in my abstract that attempted it before. However, it would not even run on my device due to the fact that it was coded for an exact screen size. This meant that although my device was a much newer and more advanced model than the project was coded for it would not work without modification due to something as simple as the size of the screen. I really wanted to avoid this happening in the future so I did my best throughout to be careful so as to make the project as flexible as possible so in the future it would work on a wide variety of devices.

1.4.4 Neat Coding and use of Multiple Classes and Threads

Another problem with working with the previous project as mentioned was the method that was used to code it. The project consisted of one main thread that did all of the feature tracking, sensor tracking and screen drawing and shifting, all in one class.

This made it incredibly difficult at first to understand how the code worked and what parts were for what purpose. It also made it next to impossible to alter one part of the code without breaking the whole thing. I immediately decided upon beginning creating my own project to make separate classes and threads for all of the different functions. This increases efficiency but just as importantly, it keeps the code relatively neat and makes it easier to debug.

1.5 Obstacles

There were a number of major obstacles that had to be overcome for this project to work. These are simply some of the ones that were a constant issue from the very beginning, there were other major issues that came up during development that will be mentioned later on.

1.5.1 Efficiency

This project involves quite a large number of different features all running at the same time, some of them fairly CPU intensive. As such it was important to make everything as efficient as possible to ensure a smooth running, after all there wouldn’t be much point to this app if it ran so slowly that no one would want to use it.
Also as mentioned before, one of the major problems with the project that attempted this before was that it ran far too slowly whilst using less features than I intended to implement, as such, efficiency was definitely on my mind from the beginning.

1.5.2 Lack of Functionality in the Android Platform

Android is still a relatively new platform, as such it still lacks some functionality necessary for developers to do certain applications.

The main function missing that was a serious obstacle to my project is the ability to edit and record video. Android has functions built in to record video straight from the camera using the MediaRecorder Class [2]. However, this simply records the video straight from the camera with no methods whatsoever to access the frames before they are saved and edit them. There is also no real method in Android for accessing and modifying frames before recording. As such it was necessary for the sake of my project to create some sort of recording method myself.
Chapter 2: Android Platform and Hardware Components

2.1 The Android Platform

The Android Operating system was first introduced in 2007 and the first phone to run on Android was released in 2008. Since then Android has quickly risen to prominence to become what is now the most popular mobile operating system in the world.

What sets it apart from its main rival iOS (And to a lesser extent Windows) is its' open source nature. Apple strictly controls and regulates iOS meaning that only their own devices can run it and only developers who have special licences can develop for it.

This makes Android a very tempting operating system for both phone manufacturers and developers. Its open source nature means that companies can use it without paying massive royalties to Google (Who own Android Inc. the company who developed Android) and also anyone with the necessary expertise can develop applications for it.

From a developer perspective (including my own), coding for Android is done in Java, and there is a specialised Android SDK available for download off their website. The SDK works perfectly with open source compilers such as Eclipse so setting up the development environment is relatively easy. All functions in Android are also well documented on their website [3]. This makes Android relatively easy to get started with and there is a lot of aid in developing for it so long as you are using Android’s provided built in functionality. This is good since I didn’t have a huge amount of experience developing for Android before this project. It becomes very difficult though if you need to do something that isn’t directly provided by Android.

Android programs are designed as individual Applications (Apps) and each operates for the most part as its’ own separate entity much more so than programs on a pc. App projects have a relatively set structure as defined by the Android SDK. Again more information can be found on the Android Developer site [3].
2.2 Android Camera

As stated before, the vast majority of Android devices contain a camera of some sort. Most of these cameras are relatively decent quality devices (usually around 2-7 Megapixels but this can vary hugely) capable of playback and recording at fairly high quality and frame rates. It is also worth mentioning that as standard they all record in YUV colour format which can cause complications as will be mentioned later on in this report.

2.2.1 Implementation in Android

From a developer perspective, the typical Android camera application [4] is basically designed as follows:

1. An instance of the camera is “opened” and its parameters are accessed and modified if necessary (e.g. change the image size).
2. This camera instance is passed to a “SurfaceHolder” [5] and it is told to start the preview. This basically displays a preview of the camera on an assigned surface.
3. From this point there are mainly three things that can be done:
   a. Call “Takepicture” to take a standard image photo straight from the camera and save to file.
   b. The camera is “Unlocked” and the MediaRecorder class is used to take a video straight from the camera and record to a file.
   c. A “PreviewCallback” is used to send the data from the camera to another part of the application (In the case of this project the OpenGL renderer).
4. After this is done the camera needs to be released (and locked in the case of the MediaRecorder). Failing to do this can result in the camera becoming unusable to other applications or even this application until it is somehow reset. The reason for this is the camera may only be accessed by one application at a time, so the application that opens it is the only one that can use it until it releases it.

(Please note that in the interest of simplicity, this code description and all descriptions of how code is carried out for the rest of the report are simplified)

It is also worth noting that for picture taking and preview callbacks, the displayed camera preview can be bypassed in a number of ways, for example, by defining the main activity as implementing a surface holder and calling a camera preview callback in it. This means that the camera preview frames can be accessed and used without necessarily displaying them on screen. This is not the tactic I used for my project as I am using a preview. But in hindsight it would have been a more efficient way to do it.
Basic Android Camera Implementation Overview

- Open The camera, Set and modify any parameters
- Surface Holder displays a preview of the camera (on screen if wished) and holds the camera data

Three Main things that can be done next:

- Call Picture Callback
- Call Preview Callback
- Unlock Camera

- Take Picture
  - Save image to file
  - Send camera frames wherever one wants (In the case of my project the OpenGL renderer)

- Take Video
  - Save video to file
  - Use MediaRecorder Class to take video

Figure 2 Android Camera Overview
2.3 Sensors
Android devices contain all sorts of sensors for all types of purposes. There are three main types of sensors in Android [6]:

1. Position sensors measure the physical position of the device. For example, orientation sensors and Magnetometers.
2. Environmental sensors measure different environmental parameters such as temperature, light (Illumination), pressure etc. These can include temperature sensors, light sensors and pressure sensors.
3. Motion sensors measure movement in terms of acceleration and rotational force along the axis of the device. These can include linear accelerometers, gyroscopes etc. These are the sensors that I chose to use in my application.

![Figure 3 Android Sensor Axis Diagram. Source: Android website](image)

### 2.3.1 Linear Accelerometer
The linear accelerometer (similar to a regular accelerometer) returns a vector value for acceleration of the device in the x, y and z axis.

The difference between the linear accelerometer and the accelerometer is that the linear accelerometer has the force of gravity removed from its value so we can simply ignore gravity and focus only on the movement of the device.

**Linear Acceleration = Acceleration - Acceleration due to gravity**

This sensor was very useful for my project and is the primary sensor for use in moving the screen based on movement of the device. When the app is using sensor tracking, when it detects a movement upwards in the y direction, the app tries to shift the image up in the y direction to match the movement of the device based on the value output by the sensor. Same for the negative y direction and the positive and negative x direction.
As we are working with a 2d image, this project ignores the z direction as it is not needed. Although if the project was completely functional and we wanted extremely accurate sensor data, movement in the z direction could theoretically be useful in detection changes in distance between the device and the target the app is tracking.

2.3.2 Gyroscope
The gyroscope measures rotation around the x, y and z axis of the device. Counterclockwise is taken as positive rotation and clockwise is taken as negative rotation. Similar to the linear accelerometer, the gyroscope returns a vector value for acceleration of the device around the x, y and z axis.

This sensor was used in my project for detecting rotation. Detecting rotation of the device is very obviously useful in rotating the image to match this, however it is also useful for use alongside the feature tracking as will be mentioned later on in this report.

2.3.3 Implementation in Android
In order to receive the data from the sensor in Android, a “SensorManager” is required [7].

1. Set up the sensormanager class and call it as a new SensorManager.
2. Define the sensor you are interested in by calling
   sensormanager.getDefaultSensor(sensor.Type_.....).
3. Set up the “onSensorChanged” function to receive values for how much the sensor has changed, these values are received as float values to do with as one will.

In my application, this sensor data is then processed in a number of different ways before being used to shift the image and display the sensor output as graphs on screen.
Chapter 3: Additional Libraries and Features

3.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source library used for computer vision applications [8]. It is widely used all across the world in all sorts of applications such as facial recognition, motion tracking, machine learning, augmented reality, object recognition, feature detection and countless more. It is used extensively all across the world by everyone from research groups and companies to government bodies.

It is an incredibly extensive library with over 2500 different optimized algorithms for all of its different applications and it is being constantly updated. Its library is written natively in C++.

There is a version of OpenCV known as OpenCV4Android and that is the version of OpenCV that I used for this project. It is a version of the library designed specifically for the purpose of development of Android applications. It, like regular OpenCV, is open source and free to download.

In order to use OpenCV4Android:

1. Download the Android project off of their website [8].
2. Import it into the same eclipse (or similar compiler) workspace as the project.
3. Assign it to be used as a library.
4. Add it to the Android properties of the main application project as a library.

This is fairly simple to implement, although it is worth noting that I did have some difficulty implementing it at first due to my initial lack of experience and knowledge of the Android environment and how external libraries worked with applications.

This version of OpenCV4Android is distributed as a precompiled library. This makes it relatively easy to implement. It is still coded in C but uses the Java Native Interface (JNI) to run in Android Java (this is described later). There are some downsides to this method. Applications running it sometimes require the user to download an app called OpenCV Manager in order to run which can be annoying. It also effectively makes it impossible to modify OpenCV in any real way.

Another possible method is to statically link OpenCV to the application. This involves compiling OpenCV from binaries straight into the application project. This can be awkward to implement however and it makes the file size of the application hugely larger, increasing compile time as well as taking up more space on the device using it. The plus side however is the app will just function on its own without needing to download another application. Also it is theoretically possible to modify OpenCV if necessary.
OpenCV4Android is not the same as regular OpenCV. As Android is still relatively new and being constantly updated and developed, OpenCV4Android is even newer and again is constantly being updated. As such it is missing some of the functionality that OpenCV for windows has, including many of the functions that work with cameras that if they did function on Android may have made many aspects of my project much easier such as its windows functions for accessing data from cameras and writing to files.

It is also worth noting that there is some support available to help developers get started with using OpenCV for Android, although not nearly as much support as there is with working with regular Android code. This is likely down to the fact that OpenCV is much more of an Open Source project than Android itself which is fully backed by a massive company (Google).

It is also worth mentioning that the latest version of OpenCV is 2.4.4. However, my project uses 2.3.1. I was working with the latest version for much of my project, however like many things, it is not without its bugs. A bug in OpenCV to do with how OpenCV creates new “Mats” (Matrices) caused unhandled exceptions (crashes) in my project that I was unable to solve as it was to do with the OpenCV library itself. As such I scaled back and started using an older version that was not subject to these bugs.

OpenCV is such a vast library that I couldn’t possibly even mention all its different aspects in this report so I am going to focus in on the main application I had for OpenCV for this project.

3.1.1 Feature Tracking
Feature tracking is one of the key features of my project.

In computer vision, a feature is generally defined as a characteristic in a scene that we can recognise easily. Often these features are edges, corners or blobs in the image. For my project I used corner detection.

Corner detection was the best type to use for my project because it very easily gave an exact point which is necessary for this project. Edge detection will return a line and blob detection an area, but a corner is the intersection of two edges and can only be a point.

There are many types of corner detection in OpenCV, some of which are listed below: (Note, these are very simplified summaries and also, the code in the references given is C code and does not necessarily apply to how the feature detectors work in Android)

- Harris [9] is one commonly used method of corner detection and is one of the oldest. At locations where a corner is present there is a large variation in the “gradient” of the image (gradient is defined as a quantity of directional change in intensity of colour in an image). Harris creates “windows” (areas in the x and y axis) to basically scan the image and look for large variations in gradient inside the window. If a large enough variation is found, a corner is located.
• FAST (Features from Accelerated Segment Test) is a very commonly used corner detector, particularly in OpenCV for real time applications. FAST works by selecting “candidate” key points in the image and drawing a circle of radius 3 pixels around the candidate. This creates 16 “parameter pixels”. The candidate is a corner point if 12 consecutive pixels out of the 16 parameter pixels are brighter than the centre pixel by a given threshold.

For my project FAST is the feature detection used, primarily due to the fact that it operates significantly faster than other feature detection types.

It is worth noting that there are other types of non-corner detection that may have worked, for example SURF which uses Blob detection to detect areas and then uses an integral over the area to find a key point. Although this is fairly optimized, it still cannot compare to FAST in terms of speed.

3.1.2 Feature Tracking Implementation in Android

On a very basic level, feature tracking in Android works as follows:

1. Define the feature detector wanted (In this case FAST).
2. Set a “Mat” Image (see below) containing the image data over the area we want the features tracked.
3. Create an empty vector list of “keypoints” (see below).
4. Use the “detect” function: defineddetector.detect(Mat, keypointlist). This tells the detector to basically scan the image for keypoints, and save them to the keypointlist.
**KeyPoint**

Important to note. A keypoint from an intuitive point of view is a feature point as defined earlier. However when spoken about in the context of code I am referring to the OpenCV class “KeyPoint” [10].

KeyPoint in OpenCV is a structure class. It is used to represent important points in an image and includes:

- The x and y (point) coordinates of the keypoint
- The size of the keypoint
- The angle of the keypoint
- The response of the keypoint. Response is a way of basically saying how good a keypoint it is and is very useful in comparing keypoints or finding the most prominent keypoints in an image. For example, with FAST, a perfect right angle corner that divides completely black and completely white areas should give a very strong response. But a less right angled corner where there may not be much of a colour difference on each side of the corner edges should give a fairly weak response.
- The Octave (Unused in my project). This represents the “scale-space” in which the feature is found.
- The Class_id (Also unused in my project).

**Mat**

A Mat (usually referred to as a Matrix) in OpenCV is basically a 2d numerical array that can act as a number of different things, including an image.

OpenCV for Android, particularly the newest versions, make vast use of Mats as its image format. As such they are crucial to OpenCV. OpenCV used to use IplImages but is increasingly using Mat’s for a number of reasons. For example it is object oriented and it can be easier to access and modify individual pieces of the data stored in Mats. It also handles all its own memory management and deallocates memory itself making it more efficient at times.
3.2 OpenGL

OpenGL (Open Graphics Library) is a cross platform, multi-language library used for 2D and 3D graphics applications [11]. It was initially created in 1992 by Silicon Graphics and since then it has become an industry standard for computer graphics applications.

It is used for countless applications including computer graphics, gaming, simulation, 2d animation, image processing, augmented reality etc. Like OpenCV it is used by all sorts of groups of people, from individual programmers to research groups to companies.

OpenGL is designed for 2D and 3D rendering (rendering is the generation of an image from a given model e.g. generating a 2D image given the image data and certain variables such as lighting and colour overlay). Although its functions can be implemented in software, primarily, OpenGL functions are designed to be implemented in hardware, usually the GPU (Graphics Processing Unit). This makes processing in OpenGL incredibly fast when compared to many other methods of rendering.

Unlike OpenCV, there is a version of OpenGL built straight into the Android API. As such there are no external libraries to download and link or anything of the sorts, so there is no work needed in order to implement OpenGL in Android. This version of OpenGL is known specifically as OpenGL ES and is a royalty free version designed for a number of platforms [12].

Like OpenCV, OpenGL is incredibly complex and will take a huge amount to describe all of its different functions so I am going to hone in on the area I am working on. For my application, only 2D rendering is used and all 3D features are ignored as they are unnecessary.

3.2.1 View

Before we can start drawing the images and objects we want to display, there are a few things in OpenGL that have to be gotten right. First off is the view. Think of rendering as looking at a digital world through a window. The position of the window is entirely responsible for what part of the world we are seeing. The camera in OpenGL is basically where the viewer’s eyes would be relative to this window. In “Camera” we are referring to the OpenGL Camera, not to be confused with the actual device’s camera. The View is the setup of this window and camera which define what the viewer sees.
3.2.2 Lighting

Lighting is normally very important in OpenGL for a number of reasons. It can add amazing effects to 3D worlds, it can change the depth of colour that is in the scene and it adds a huge array of effects to any OpenGL scene. It is also necessary for anything to be visible. However, it is also extremely complex, and since I am using a camera image I do not want the OpenGL lighting to have any noticeable effect on my frame so all my lighting is very simple standard lighting that can effectively be ignored but is necessary for the frame to be seen.

For my project, the most important aspect of setting up the rendering for a number of reasons is the Texturing.

3.2.3 Textures

Textures are the really important area of my OpenGL renderer. After all, in my project, the camera frames that are visible on screen are in fact OpenGL textures. Textures are also responsible for how the frames move on screen due to the sensors and feature tracking.
A Texture is an OpenGL object that contains one or more images that all have the same format. A texture can be used in two ways. It can be the source of a texture accessed from a “Shader”, or it can be used as a render target [13]. In my project I exclusively use textures as target for rendering so I will focus on that.

Textures are created as follows:

1. Assign the texture a name “glGenTextures”.
2. Select the texture you want and the type “glBindTexture”.
3. Set Texture parameters “glTexParameteri”. For example, add filters, set if the texture repeats if it doesn’t fit where it is mapped etc.
4. Load the image into the texture. There are quite a few ways to do this. For example, loading the image data into a buffer and calling “glTexImage2D” with the buffer data.
5. Enable 2D texturing “glEnable (GL_TEXTURE_2D).
6. Map the texture (See next section).

(Note, there are quite a few ways to use textures. This is a fairly standard way to generate textures although it is very possible to deviate from this. My own method has extra steps and is quite a bit more complex)

It is also worth noting that in order for textures to work efficiently, the size of the image being bound to them must be a power of 2 (e.g. 2*2, 4*4, 8*8 etc.). The reason for this is that GPU hardware processes power of two (POT) textures much faster than non-POT textures. It is possible with some versions of OpenGL to use non-POT textures but in CPU intensive processes it can noticeably reduce performance.
Texture Mapping

Textures, although made up of triangles (polygons), use a rectangular coordinate system.

The coordinates of the four corners of a texture are shown above. These are the texture coordinates. A texture with the coordinates \((0, 0), (1, 0), (0, 1), (1, 1)\) would completely fill the vertex it is mapped on to. If the coordinates meant the texture was larger than this, only part of it would be mapped on. If the coordinates meant the texture was smaller than this, then depending on the coordinates, the texture would for example repeat and appear as a sort of tiled surface with each tile containing the texture image.

Texture coordinates define how the texture is mapped onto the surface but Vertex coordinates define the surface the texture is mapped on to. It uses a rectangular system the same as the texture coordinates but where texture coordinates imply a size relative to the vertex coordinates, vertex coordinates imply an absolute size. For example, for my project, the projection was set such that the screen was 200.0f high and 200.0f\(^*(\text{devicewidth/} \text{deviceheight})\) wide. Thus the vertex coordinates of the full screen texture were:
Figure 7: Code Snippet of Full Screen Texture Vertex Coordinates

If the projection defined the screen as half the current width, this vertex would be twice the width of the screen and only half of it could appear on screen at a time.

If the texture coordinates \([0, 1), (1, 1), (0, 0), (1, 0)]\) were mapped onto those vertex coordinates, that texture would appear as a single image full screen.

3.3 Java Native Interface

The JNI (Java Native Interface) defines a way for managed code (written in Java) to interact with native code (written in C, C++ or Assembly) [14]. Now it is very possible to write an application entirely in Java but there are several reasons why one might use the JNI for Android development:

1. Android does not support a particular feature that is required (Although, sometimes if Android does not support it, it might not work with the JNI anyway).
2. Sometimes one might already have code/ a library written in another language and it might be easier to import it in the JNI rather than rewriting it in Java.
3. Sometimes Java isn’t efficient enough for some particular application and that application will run faster in another language.

3.3.1 Implementation in Android

The basic steps to implementing very simple custom JNI code are as follows:

1. Make sure you have the Android NDK (Native Development Kit) successfully downloaded on to your computer then create an empty folder called jni in the project.
2. Place the .c and .h files (for this example we are using C as the native code) into the jni folder.
3. Write the Android.mk file and Application.mk (If needed) files for this project. The content of these files varies completely depending on what code you are trying to implement. These are makefiles (makefiles are files written in a make programming language. It is designed to automatically build executable programs and libraries from source code) that contain lists of modules needed amongst other things.
4. Compile the library using the ndk-build script contained within the Android NDK. This is done a number of ways depending on the environment you are working in. In Linux it can be done using Terminal. Set Terminal to be focused on the jni folder in the project folder where the .c, .h and .mk files are located. Then run the ndk-build script with this location in focus.
5. Assuming all the files have been written correctly, the ndk-build script should handle the rest and build the JNI libraries into the libs folder of the project.

The library can then be called a number of ways, usually it is statically called using System.LoadLibrary and then defined as a function similar to defining a function in C.

A few steps have been rather greatly simplified there and in reality it can be a lot more complicated to implement JNI libraries but that is the basic method that I used in my project.

Now it can be incredibly difficult at times to get certain applications working with the JNI for quite a number of reasons:

1. Android may not support the features that the native code you are trying to get working is trying to carry out.
2. Some code compiled through the JNI will work on some Android devices and not others.
3. It can be difficult to debug as, if it is not working for any reason, usually you only get a link error saying the library cannot be found and it can be difficult to pin down why. This just means any one of the steps to implementing the JNI didn’t work right.
4. It is simply difficult. It is awkward to define at times and requires Android.mk and Application.mk files have to be made perfectly.
Chapter 4: Application Design

The last two chapters have covered all of the background information for all the different aspects of the project. I will now go over my full design schematic and exactly how my project would work at full functionality.

4.1 Design Overview

Let’s go over step by step how the application works (or in some cases would work) from a very top level view with full functionality:

1. The Camera takes in frames.
2. These frames are sent to the preview.
3. The preview sets camera properties and uses a “PreviewCallback” to send the frames to the OpenGL Renderer.
4. The OpenGL Renderer uses “OnPreviewFrame” to receive the frames from the preview.
5. It then converts these frames from YUV colour format to RGB using the C library YUV420s2rgb and the JNI.
6. It then creates a texture out of this image data and maps it on to 2 different textures, the background and the foreground textures.
7. The background texture is made darker than the foreground so they can be distinguished from each other. They are aligned up perfectly using texture mapping such that if the background wasn’t darker and no image shifting was taking place it would be impossible to notice they were two different textures. The purpose of the two different textures is purely so the user can distinguish between image shifting and normal camera input. The foreground texture gets shifted by the sensors and feature tracking and the background does not so it creates a nice contrast.
8. Meanwhile the Renderer is taking in data from both the sensors and the feature tracking (assuming they are turned on):
   a. Firstly the sensors. There are two sensors in operation, the Linear Accelerometer and the Gyroscope. The values for each of these are being used for several different applications:
      i. Firstly the sensor graphs. The values for both of these sensors are being stored in arrays. The values for these arrays are being sent to the “SensorLayer” view to be used to make the graphs of the sensor data over time.
      ii. Secondly, the values are being passed to the Renderer which is using them to shift/rotate the image every frame based on the magnitude of the motion.
      iii. Lastly, the values of the Gyroscope are being passed to the feature tracking for use by the feature tracking to correct for changes in the orientation of the device.
   b. Next the Feature tracking. The Feature Tracker class scans the image for keypoints.
i. While it is getting key points, it finds the ten best key points (assuming there are 10) and displays them on screen as red circles.

ii. If the user touches a keypoint, it is set as the “anchor point” and the feature tracking fixes on that point. It creates a green circle at this point and stops scanning for other points. The feature tracker then tries to keep it at the same point in the image by passing the appropriate values for how much to move the image by to the renderer.

9. With the information from the feature tracking and sensors the renderer can remap the foreground texture using texture mapping in order to shift the image.

10. The remapping is done in a clock cycle.

11. Firstly the sensors are calibrated using the feature tracking.

12. The sensors then correct the image every frame with the feature tracking correcting every say 50 frames or so. Thus the frames are corrected accurately and efficiently.

13. The Renderer can also record the frames into a video if the record button is pressed.

14. This is done by using “glReadPixels” to get the values of each pixel, save these to a bitmap, compress this into an image and use a C library like FFMPEG to create a video out of all these images.

Thus this is step by step how the application would work with full functionality. As I said at the very beginning I did not achieve full functionality but for now I will describe the application in terms of full functionality and address the issues I had later in the next chapter.

![Screenshot of App](image-url)
Camera

Preview

Takes in frames from the camera, sets camera parameters and sends frames on.

On Preview Frame... Take in Images

YUV 420 to RGB

Convert frames from YUV colour format to RGB.

Feature Tracker

Find Feature Points and track.

Display Feature Points

OpenGl Renderer

On Preview Frame Take in Images

Compute the Texture coordinates based on sensors and feature tracking.

Create and display Textures

Save Frames as Images

Compress frames into Video

Record

Save video to file

Sensor Handler

Takes in and processes data from the sensors.

Sensor Layer

Displays sensor graphs.

Capture

Save image to file directly from camera.

Figure 9 Project Top Level Schematic
4.2 My OpenGL Renderer

In this section I will go into detail as to how my OpenGL renderer takes in the frame data from the camera, converts it to a usable format and displays the corrected and uncorrected frames on screen.

First off, OpenGL needs to be set up to render in 2D.

4.2.1 Setup Parameters

View

In my code, the OpenGL “camera” and “window” for the view are set up in the “onSurfaceChanged” and “setupDraw2D” functions in my OpenGL Renderer (glrend).

For this project, all I was interested in was mapping the devices camera frames on to the screen. I did this by first defining a “viewport” [15] in the onSurfaceChanged function. This basically defines the size of the window we are looking through and it was set to the size of the devices screen. The “setupDraw2D” function is then called. In this function I call glOrtho which sets up the orthographic projection. This basically scales the projection and sets it to a size. In mine it was 200.0f high and 200.0f*(devicewidth/deviceheight) wide. This means that an object at the centre of the screen that is 200.0f high and 200.0f*(devicewidth/deviceheight) wide will be full screen (assuming it is mapped correctly). I left the position of the OpenGL camera at the default location (0.0f, 0.0f, 0.0f in OpenGL float coordinates).

Other Setup Parameters

Depth is very important in 3D rendering. However it is not needed in 2D rendering so we simply get rid of it. “glClearDepth” in the “onSurfaceCreated” function. We also clear the screen completely of everything “glClearColor” in the “onSurfaceCreated” function. This just leaves us with a black 2D screen to work with. We also set up very standard OpenGL lighting so the frames can become visible.

The renderer receives the frames by calling “onPreviewFrame” which receives a preview frame in the form of an array of bytes data every time “PictureCallback” is called in the preview class. The preview class also changes the image size to 256*256 pixels. This just makes the image a power of two for use as a texture as explained in chapter 3.

Some conversion is still needed though before OpenGL can use this data.

4.2.2 YUV to RGB

As previously stated, Android cameras by default record in YUV colour format. Now for most applications this is fine, but not for OpenGL. OpenGL only works with RGB colour format and passing it anything else will result simply in an incredibly distorted image (See the figure below).
There are several methods to solve this. The simplest would be to set the camera to record in RGB except this doesn’t work on most Android devices. Another simple option would be to just write a function in Android that breaks the byte data down and then puts it back together in RGB format. However on testing it was found that this was simply too slow for my application so another solution was required.

The solution was to make a library in C that would do the conversion and implement it in native code using the JNI. Thus I compiled the library into my project and created the Android.mk and Application.mk files for it.

With the frames taken in and in a format OpenGL can handle they are bound to a texture.

4.2.3 Textures
The application has two textures, the background and foreground textures. The vertex coordinates of the background are set to match the orthographic projection such that it is full screen. The texture coordinates are then mapped such that the frame matches the vertex perfectly so the frame appears full screen. This texture is then darkened using “glColor4f”. glColor4f basically defines the colour balance of the rendered image. By making the RGB values for the image (0.5), (0.5), (0.5) respectively, I reduced the colour intensity of them and made the texture darker.
The foreground Texture is the more complicated one. This texture is made such that it shifts with the image based on sensors and feature tracking. The vertex coordinates are set to be 0.8 times the coordinates of the background texture. This simply makes this texture 4/5 the size of the background texture and as it is on top from a user perspective it means that the background texture appears as a frame around the foreground texture.

Figure 11 OpenGL Textures in App

The texture coordinates for the foreground texture are defined as follows:

```java
    texcords2 = new float[] {
        0.09375f+c, a-0.064f+d,
        b-0.09375f+c, a-0.064f+d,
        0.09375f+c, 0.064f+d,
        b-0.09375f+c, 0.064f+d
    };
```

Figure 12 Code Snippet of Texture Coordinates for Foreground OpenGL Texture

These coordinates are defined inside the onDraw function of the OpenGL renderer. The reason for this is simple, the onDraw function updates every frame and we need these coordinates to be constantly updating in order to shift the image.

If c and d are equal to zero the foreground texture lines up perfectly with the background texture and apart from the background texture being darker they appear as 1 image. But, c and d are what defines how much the image is shifted by the sensors and feature tracking.
They contain the magnitude for how much the device is being moved based on both of these, I will go through how these are modified further on.

It is worth noting that this is not the only way this could be done. It would be possible to shift the image with a “glTranslate” and “glRotate”. glTranslate moves the defined area across the screen by some amount and glRotate rotates the defined area. However, if this was implemented it might also have been necessary to change the vertex coordinates with the translate and rotate. I decided to use the texture mapping because it seemed like it might be more accurate and less unnecessary overhead as the texture has to be mapped anyway. It may also have been more complicated if changing the vertex coordinates too became necessary but using glTranslate and glRotate should be a perfectly valid solution too.

4.3 Sensor Shifting

As said earlier, shifting is done by altering the values of the floats c & d in the renderer. As such what needs to happen is to extract some float values from the sensors that correctly correspond to the amount we want to shift the image by on screen. I played around with quite a few methods for doing this (most were much like how the sensor graphs are calculated which I’ll go into later) but in the end decided on a relatively simple method which gave the best result of the methods I tried.

In the “sensorhandler” class, the “onSensorChanged” method gives a “SensorEvent”. “SensorEvent.values” gives us the float values for how much the sensor has moved since it last output values. What most sensor applications do is put these into a function and use a time variable to turn these into a velocity and that works quite well. However, for my project we are largely just interested in magnitudes of shifting as we want to know how much to instantaneously shift the image by instantaneously in real time. This is the method I chose which output some surprisingly accurate results (although they become increasingly inaccurate over time):

1. A float[] variable sendis is set up.
2. Each time the onSensorChanged method is called, the values for SensorEvent.values[0] are added to sendis[0] and event.values[1] are added to sendis[1] (SensorEvent.values outputs values in the x, y and z ([0], [1] and [2]) directions. We ignore z because we are not interested in it. Thus we are storing the magnitudes in the x and y directions in sendis).
3. These values need to be divided by some value as they are too large as float values to be used for my texture. I divided them by 100 but what would be ideal is some sort of calibration to calculate how much to divide them by exactly.
4. When the renderer is mapping the foreground texture it puts c= sendis[0] and d= sendis[1] and maps the texture.
5. sendis is then cleared of all values and the cycle continues

Thus this is basically how the image is shifted based on the sensors.
4.3.1 Calibration

One thing which is really necessary for the sensors to be accurate is some sort of calibration for the sensors. The plan I had was to use the feature tracking to basically calibrate the sensors as follows:

\[(\text{Values from Sensor}) \times (\text{Some value } x) = (\text{Values from Feature Tracking})\]

\[x = \frac{(\text{Values from Feature Tracking})}{(\text{Values from Sensor})}\]

Therefore:

\[(\text{Correct Values from Sensor}) = (\text{Values from Sensor}) \times x\]

Theoretically, what would happen is that when the application first started, the sensors would be calibrated by the feature tracking. Several value for \(x\) would be found as shown above and an average worked out to get the most accurate value possible.

This value for \(x\) would then be multiplied by all the values for the sensors in x and y directions. There would likely be separate values for \(x\) in the x and y directions as they would have very different magnitudes.

Figure 13: Sensor Shifting Screenshot
4.4 Feature Shifting

Similar to the Sensor Shifting, the final result wanted from the feature tracking is 2 float values to put c and d to in order to shift the image.

There are likely many different methods that could be used to shift the image based on feature tracking. In my project I did the following to get my values for c and d in feature tracking:

1. My “getFeatures” function searches the image region we are interested in to find keypoints and saves them to the KeyPoints vector list.
2. I then pass this list to the “processKeyPoints” function which compares all the keypoints based on their responses to find the best 10. In doing so it removes all keypoints but the top ten.
3. My onDraw function is set up to draw all of these keypoints on screen as red circles so the user can easily see what keypoints are being tracked.
4. It keeps doing this until the user touches one of the keypoints on screen. When the user touches the screen, if there is a keypoint there, the application stops tracking new keypoints and focuses on that. It draws a green circle on it instead of the red ones to indicate that this point is indeed the “Anchor Point”.
5. From then on the application starts searching the image for that specific keypoint. If it finds it, it sets the “Match Point” as that keypoint.
6. The movement in the device in the x and y direction is then calculated quite simply as:
   
   $$(\text{anchor point location}) - (\text{match point location})$$

7. This is done in the getDisplacement function. What it does is searches a region around a template (the template is set around the anchor point) and finds the point that matches the anchor point. The location of the template is moved based on the gyroscope data in order to improve accuracy. The matching point is set as the match point and the difference is computed as stated above.
8. If no match is found in this region it once again searches for keypoints.
9. The output of the getDisplacement function is a “Point” variable with x and y values.
10. The values for this point are put into a float[] variable fdis in the onDraw function.
11. When the renderer is mapping the texture it puts c= fdis[0] and d= fdis[1] and thus the texture is shifted based on feature tracking.

The above method is how the feature tracking is designed to work and how it would work if the project was fully functional. Admittedly I never got the feature tracking completely working but I will get on to that later on in the report.
4.5 Sensor Graphs

One of the features of my project is a real time graph of the sensor data over time. But before I go into that I want to mention a structure class that I used in my project to help handle readings from the sensors and feature tracking.

4.5.1 Readings

Reading is a structure class in my project designed to make handling of readings from the sensors a little easier to plot. There is little special about it other than the functions like “smoothReadings” and “smoothList”. These allow us to create the second “smoothed” version of the chart which creates a nicer view of the sensor data over time rather than the pure jagged results given by the first graph. Two arrays of readings are created to handle the data from the sensor, “mReadings” and “gyroReadings”.

Each time the sensorhandler’s “onSensorChanged” function is called, the readings for the linear accelerometer are added into mReadings and the readings for the gyroscope are added into gyroReadings. mReadings is used to create the top left chart which is a graph of the raw readings from the linear accelerometer over time. The bottom left chart is also a graph of the linear accelerometer over time. However, the Reading function “smoothList” is used to smooth out the graph. The bottom right graph is the graph of the gyroscope data (gyroReadings).
In the left graphs the green line represents movement in the y direction, red represents movement in the x direction and blue the z direction.

In the right graph, blue represents rotation around the z axis, red represents rotation around the x axis and green rotation around the y axis.

4.6 Save Corrected Video

As stated several times previous, Android has no build in code to save a modified video to file. As such it was necessary for me to come up with one myself. The plan I had to do this was to first save each frame to file as Jpeg images and then compile all of these images into a single video.

4.6.1 Saving Edited Frames

I successfully saved all of the frames as Jpegs as follows:

1. Render the modified texture in OpenGL
2. Use glReadPixels to read the values of all pixels on the screen (Or just a certain area) and save them to a buffer “screenshotBuffer”.
3. Carry out some necessary pixel shifting on each pixel in the buffer (This is necessary to prevent a distorted image).
4. Create a bitmap out of the buffer using “Bitmap.createBitmap” [16].
5. Compress the bitmap into a jpeg using Bitmap.compress.

This is done in the OpenGL renderer at every frame.
There are a number of problems with this method. The most prominent being overhead. This method is slow and cuts the frame rate hugely. It might be possible to speed it up using buffers when the texture is first created to increase efficiency or by implementing parts of it in a separate thread.

4.6.2 Saving These Frames to Video
This is definitely the difficult part. There is no function in Android that will allow us to compile those frames into a video so use of the JNI is required.

FFMPEG
FFmpeg [17] is a free software available to developers. It is designed to give all sorts of functions for use with editing video. There are versions of it for Android but they are still at a relatively early stage in development and aren’t that well developed yet. As a result of this, although there are some helpful tips online, there are not a lot of resources to aid in the implementation of FFmpeg for Android and a lot of difficulties can ensue with its use.

FFmpeg like OpenCV is a C library that uses the JNI to implement itself in Android. Unlike OpenCV, it is not all that developed and it does not have its own precompiled library that can simply be linked to the project. On top of that there are quite a few problems with it even if it does get implemented correctly. As it currently stands it often only works with certain processors and certain versions of Android.

It seems then that this is a very poor choice of library to use but the fact is that this function in Android is just completely lacking and there isn’t any real better options out there at the moment. There are other C libraries too that might work but they would all have similar problems.

It is quite possible that Android in the near future might implement a feature for exactly this purpose. After all, Android is constantly being updated and new features being added. For now though, if anyone wishes to save edited video in Android, they have to make do with the JNI and C libraries like this or else give up their attempt.

4.7 Camera Capture
It is also worth mentioning just in terms of the functionality of my project that there is a capture button on the right hand side of the app to the left of the record button (which instigates recording all frames as Jpegs). This capture button uses a simple, fairly standard, camera capture process to capture an image directly from the camera and save it to a file. It actually works (perhaps unsurprisingly) relatively similarly to my code for capturing all the frames on screen for the record functionality although capturing one frame straight from the camera is somewhat simpler. The only reasons really why I bother having a capture class specifically for this purpose is to perhaps slightly increase efficiency but more so to just keep the code neat.

My capture process works as follows:
1. The capture button is pressed. This calls the camera “TakePicture” function which uses the “Capture” class in order to implement a picture callback to send the data of the image to the Capture class.

2. The Capture class receives the byte data of the image and decodes it into a bitmap.

3. This bitmap is compressed to a Jpeg file.

4. A location is set for the image to be stored to using the MediaStore class. The image is then saved to this location.

It is largely just based on the standard Android method for capturing images as stated in chapter 2.
Chapter 5: Design Outcomes

So how successful was the project? What exactly was achieved? Well let’s first look at each feature in turn and see how successful I was in implementing it and how successful each was.

5.1 Rendering Camera Frames in OpenGL

This area of my project was very successful.

I wasn’t sure about using OpenGL at first because I worried about adding too much complexity to the project and possibly adding unnecessary overhead but in the end it ended up solving many of my problems. It allowed me to render and modify the camera frames in real time with no major drop in frame rate. It allowed me to shift the image in real time with no major increase in overhead. It also meant I didn’t have to work with bitmaps at all in my project except when saving images. Bitmaps I found from other projects add a serious amount of overhead.

I had a lot of difficulties implementing OpenGL with the camera at first due to serious issues with the colour format of the camera which forced me to make use of the JNI and a C library. This perhaps added a layer of complexity that I would have rather avoided but it was worth it when I successfully got it all working.

One of the major drawbacks to using OpenGL is that I did have a drop in image quality. In order for textures to map efficiently in OpenGL the image size needs to be a power of 2 as previously stated. As such my image is size 256*256 pixels. This is smaller than the default size of the image which on my device was around 1020*768 pixels. However the quality of the image was still I believe within acceptable quality for images on a mobile device, it was still a very clear image just not as high quality and crisp as before.

It should also have been very possible to have the image as 512*512 without any issues as this is also a power of 2. However as stated at the beginning of my project, I wanted it to run on a multitude of Android devices and the cameras on many Android devices do not have image quality as high as 512*512 so I went for accessibility and decided on 256*256 which should be fine on most devices. It would have been very possible to write a function to detect the quality of the camera (In fact I have one written) and then if this quality is higher than 512*512 set it to that, if it is lower then set the quality to 256*256. However due to time constraints I simply never got around to implementing this.

One point worth noting is that in order to get the frames from the camera I use a camera preview class with a “PreviewCallback”. For the most part this is a relic of my code from back before I used OpenGL where I was using the preview class to display the camera image. This preview class should be removed to save overhead and the PreviewCallback called in the main activity “CameraActivity”. Again due to time constraints I never implemented this.
5.2 Using the Sensors

The sensors again were another area that I believe to be relatively successful.

I was successful in getting real time data from the sensors to the renderer for shifting the image based on the movement of the device. However there is definite room for improvement here.

The calibration as mentioned previously would dramatically improve the performance of the sensors, however this is reliant on the feature tracking which I will mention next.

There is some other room for improvement for application of the sensors if time had allowed. The Gyroscope for modifying the orientation of the texture on rotation could be used to greater affect, possibly by implementing a “glRotate” when mapping the texture or having more complex texture mapping algorithms.

5.3 Feature Tracking

This was one area of the project that was not as successful as I would have liked and it did unfortunately hold back the project as a whole as it was a key part of the functionality.

I was very successful in tracking the feature points, displaying them in real time on screen so the user could select them. Where I fell down was the processing of the distance the device has moved based on feature tracking so it can be effectively shifted. There was an error in my code that when it found an anchor point it correctly stopped tracking other features but it failed to compare this to the match point each frame and provide information to shift the image accordingly. This meant that I was unable to implement real time accurate image shifting based on the feature tracking but I do know it is possible, it was simply lack of time that caused me to not implement it correctly.

However, it is worth mentioning that I was successful in applying the feature tracking incredibly efficiently. In the past it has been shown that feature tracking uses a lot of overhead and can cause a huge drop in frame rate but not in my project. It still is a lot of overhead yes, but it runs alongside the OpenGL renderer without any real noticeable drop in frame rate. This is actually largely due to the use of OpenGL. The main thing that slowed down feature tracking in the past was the use of bitmaps to scan and shift the image but this was unnecessary in my code due to the use of the OpenGL frames so it runs very smoothly.

5.4 Shifting the Image in Real Time

My method for shifting the image based on OpenGL texture mapping has proven I believe to be incredibly effective and in my opinion is a success.

As said previously, it may have been better to have implemented the gyroscope a little bit more but I have still shown that this method works to seamlessly shift the image in real time based on information provided to it.
5.5 Saving Video to File
Again this was an area where I had some successes but not as much as I would have liked.

I was successful in having the frames of the renderer saved to a file but it runs slowly. Had I had more time I would have liked to have implemented this better using buffers more efficiently and possibly using a separate thread to increase efficiency.

In terms saving a video in general or saving these frames to a video, I was unsuccessful in my attempt to implement a C library such as FFmpeg to carry out this application, but again, it is something I know is possible to implement and given more time I know this could be achieved. It also in my opinion is only a matter of time before Android or OpenCV releases some method to do this either in Android code or through OpenCV4Android as both of these are constantly being updated and this does seem like a very obvious missing feature in both applications.
Chapter 6: Conclusion and Personal Reflection

6.1 Conclusion

So overall, my Project was very successful in showing that this application is not only possible but possible to run efficiently, and in real time.

Overall, it currently functions in shifting the image based on the sensors (although somewhat inaccurately). It shows exactly how the feature tracking operates although it does not shift the image based on this as of yet. It saves the frames from the OpenGL renderer which proves that saving corrected frames is possible although the current method is somewhat slow but can be improved. I have also outlined exactly how it would be possible to save these frames to a video although it is very awkward to implement. It also does all of this (except saving the frames) seamlessly with little or no drops in frame rate so one of my main goals (efficiency) was largely accomplished.

So I believe my biggest success is showing almost without a shadow of a doubt that this application is possible and I am supremely confident that given more time I could achieve full functionality with this application.

6.2 Future Work

There are a number of steps required that are necessary to achieve full functionality.

Firstly the issues with the feature tracking need to be solved to get that working correctly. With these errors removed the image should shift based on feature tracking.

With both the feature tracking and sensor tracking functioning the calibration should be implemented as previously outlined to calibrate the sensors based on the feature tracking. This would significantly improve the accuracy of the sensors and allow them to accurately work in real time with every frame.

A clock cycle should be implemented with the sensors shifting the image every frame and the feature tracking shifting and possibly recalibrating the sensors every say 50 frames or so.

The method of saving the frames from the renderer to file should be made more efficient to reduce the drop in frame rate as stated previously.

A C library capable of saving multiple frames into a video (Like FFmpeg) should be implemented in order to save to a video.

With these five steps complete this application would achieve the full functionality I set out to accomplish.

There are other possible improvements however.
Accuracy with the sensors could be improved by making better use of the gyroscope for rotation and also by using the z axis of the accelerometer to correct for movements in distance.

If a user is recording video they would likely want audio also so some method for recording audio might make the app more tempting to a user. The main difficulty here would be matching audio with the video so the video should use some timestamp to ensure the frames are recorded with time taken correctly but libraries like FFmpeg should be able to implement this without too much difficulty.

6.3 Other Uses for Application

There are also countless other possibilities for this app.

The app runs frames through OpenGL and saves them. OpenGL can be used for all sorts of image altering applications such as enhancing quality of the image, changing colour balance and adding in colour, adding in graphics etc. As the groundwork is done with using the camera with OpenGL it would be possible to easily alter and record the video in countless different ways.

The fact that OpenGL is implemented with OpenCV with the camera also opens up all sorts of augmented reality possibilities. Perhaps recording video and overlaying some sort of other object in order to change the scene in some way, there are endless possibilities. The fact that device also uses OpenCV and sensors also allows easy implementation of extra controls through the sensors and any sort of 2d vision application with OpenCV.

Those last few points are outside the bounds of what I was attempting to accomplish with this project but I still felt it was worth noting the worth of the work I have done with this project.

6.4 Personal Reflection

The sum of what I have learnt throughout this project cannot easily be summed up in a couple of paragraphs on this page. My degree is electronic & computer engineering so I had very little prior knowledge or experience working with all of the very complex, very different and very specific features necessary for this project.

I had previously had some experience working with Android but very limited experience to the extent that I had to do a lot of research at the beginning on simple things like how and why an Android project is structured the way it is as I had no experience creating an Android application from scratch. I had to quickly learn the difference between mobile development and desktop development including learning how implementations and extensions work in Android as they are very different from simple class structures as used in desktop applications. The layout of an Android application is defined by its layout XML file. XML is a language that I had never used before so I had to learn how that worked as lack of knowledge of this caused some problems early on.
I had no prior experience working with mobile platforms of any sort so attempting to figure out how the Android camera and sensors worked the way they did at first caused some difficulty, particularly with the recording of images and video and figuring out how to use the output from the sensors. In fact one of the first things I did when creating my own project from scratch was create a simple standard Android camera application from scratch to record images and video and make a simple application that uses sensors to move objects on screen.

I had some knowledge of OpenGL when it came to using it in the project as I had previously used it for designing 3D graphics. However, learning how to work in 2D and map the textures in such a specific and relatively complex way was something I hadn’t previously done and had to learn.

I started using OpenCV for the first time in my course as I was beginning the project so I gained some useful knowledge of how OpenCV worked throughout my project that was somewhat helpful but also confusing as OpenCV for Android is very different to standard OpenCV for Windows which is what I was using in my course.

I had little to no previous experience with working with concurrent systems, As such I had difficulty working with the multiple threads my application was running simultaneously. I had to learn about concurrent threads in Android (a lot of my learning actually came through decoding strange error messages) in order to get my application working and working reliably.

Working with the JNI in Android was definitely something I previously had absolutely no knowledge of. As such I found it very difficult to work with at first to the extent that I installed an entirely new operating on my computer (Ubuntu, Linux) as it made working with the JNI easier. All compiling and running scripts and such necessary for compiling JNI libraries into the application could be done with Linux Terminal rather than complex third party software in Windows. However I had never used terminal before so I had to learn largely from scratch how it operated.

I now consider myself quite a competent programmer in Android. When I began I spent quite a bit of time bugging friends of mine from outside college who were very experienced in Android for advice, and I now feel I can competently help them go through problems they have with their applications. Despite all the major difficulties I have faced with this project, it is extremely satisfying making an application that you can see working, that you can use on your phone or on your tablet and can see other people use it too. I am very tempted to spend the required time over the summer to finish this project so I can see it fully working. If not I do definitely think Android development is something I will revisit at some point in the future in some form. The advantage of the project making use of so many different features of Android is that I now know how to use all these different features and could use that knowledge for other applications.

I have learnt a huge amount about difficulties faced when making any sort of computer application. When I first began I started by trying to modify the project of someone who
previously did a similar project. I struggled with it for weeks before throwing it out and starting my own project from scratch. I learned that it is a very common occurrence to make one’s own code from scratch rather than trying to understand code made by someone else and it is something that I would seriously recommend another student to do. If in the future a student does a project about Android image stabilization, I would recommend to them to not use my project. I would tell them to look at mine for an idea of how to carry it out and start their own project from scratch. A project that they did and that they understand everything in as it will save them a lot of time in the long run as it will be done the way they want to do it and they are happy with.

I have learnt the importance of proper time management and planning for problems. I don’t believe there is a single section of my project that just worked as I expected it to. There was always something that didn’t work for some obscure reason that needed resolving. It took me perhaps longer than it should have to anticipate this and to never expect what you are doing to just work.

Overall I now consider myself a far more competent programmer than I ever did before, not just in Android but in general. I know that the experience I have gained from doing this project will serve me well with whatever I end up doing in the near future and despite the frustration, I am very glad that I chose this project.
# Chapter 7: Figures, References & Appendices

## 7.1 Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Image Stabilization Example</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Android Camera Overview</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Android Sensor Axis Diagram. Source: Android website</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>OpenGL Perspective View Source: <a href="http://libcinder.org/about/">http://libcinder.org/about/</a></td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>OpenGL Texture Coordinates</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>Code Snippet of Full Screen Texture Vertex Coordinates.</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>Screenshot of App</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>Project Top Level Schematic</td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>OpenGL Renderer using YUV format image</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>OpenGL Textures in App</td>
<td>26</td>
</tr>
<tr>
<td>12</td>
<td>Code Snippet of Texture Coordinates for Foreground OpenGL Texture</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>Sensor Shifting Screenshot</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>Feature Tracking in App</td>
<td>30</td>
</tr>
<tr>
<td>15</td>
<td>Sensor Graphs Screenshot</td>
<td>31</td>
</tr>
</tbody>
</table>


7.2 References


7.3 Appendices

Appendix A:

See disc submitted with project for code.