Development of an Open Source Computer Vision Framework for the Leap Motion Controller

by

Daniel Hamilton

10026535

A project submitted to the

Faculty of Environment and Technology

Department of Computer Science and Creative Technologies

in partial fulfilment of the requirements for the degree of

Bachelor of Science

in

Computer Science

April 15, 2015

Abstract

Acknowledgements

Firstly I would like to thank my parents, without whom I would not have been supported throughout my time at university. To all my peers, I thank you for any time you have spared in helping me and providing motivation throughout my time at university.

Without you all, I would not be in the position I am today.

This project would not have been a success without the specific help of my supervisor Dr. Stewart Green, who supported me with guidance and feedback throughout. Along with Professor Lyndon Smith, who pointed my in the direction of literature covering the field of computer vision.

Contents

1 Introduction **12**

1.1 Computer Vision 12

1.1.1 What is Computer Vision? 12

1.1.2 Uses of Computer Vision 13

1.2 The Leap Motion Controller 14

1.2.1 What is the Leap Motion Controller? 14

1.2.2 Uses in Computer Vision 15

1.3 The Problem 15

1.4 The Purpose 16

1.5 Hypothesis 16

1.6 Solution Approach 16

1.7 Aim and Objectives 17

1.7.1 Aim 17

1.7.2 Objectives 17

1.8 Development Methodology 17

1.8.1 Planned Iterations 19

Iteration 1 19

Iteration 2 19

Iteration 3 20

1.8.2 Project Timeline 20

1.8.3 Tools 21

1.8.4 Testing Methods 22

1.9 Report Structure 22
1.10 Summary 23

2 Background Research **24**
2.1 Introduction 24
2.2 Object Recognition 24
2.2.1 What is Object Recognition? 24
2.2.2 What is Expected from Such a System? 25
2.3 Stereoscopy and Calibration 25
2.3.1 Image Correction 27
2.3.2 Camera Model 28
2.3.3 Lens Distortion 29
2.3.4 Image Rectification 30
2.3.5 Stereo Correspondence 30
2.4 Classification 31
2.4.1 Extracting Image Features 32
Edge Detection 32
Feature Detection 33
Classification of Three Dimensional Objects 33
2.5 Real World Application 34
2.6 Computer Vision Software 35
2.6.1 OpenCV 36
2.6.2 SimpleCV 37
2.6.3 Point Cloud Library 37
2.7 Summary 37

3 Requirements Analysis **39**
3.1 Introduction 39
3.2 Stakeholder Identification 40
3.3 Requirements Capture 41
3.4 Use Case Analysis 42
3.4.1 Use Case Diagram 42

3.4.2 Use Case Description 43
3.5 Functional Requirements 44
3.6 Non-Functional Requirements 44
3.7 Acceptance Criteria 45
3.8 MoSCoW Prioritisation 45
3.9 Summary 47

4 Design, Implementation and Testing 48
4.1 Introduction 48
4.2 System Architecture 49
4.3 Pre-Requisites for Implementation 50
4.4 Iterations 50
4.5 Iteration 1 52
4.5.1 Design 52
4.5.2 Implementation 52
Stage 1 - Retrieve Images From the Leap Motion Controller 52
Stage 2 - Distortion Removal 54
Stage 3 - Further Removal of Distortion with OpenCV . . 55
Stage 4 - Stereo Correspondence 61
Stage 5 - Parameter Tweaking 62
4.5.3 Test 63
4.5.4 Evaluation 66
What went well? 66
What problems were faced? 66
Lessons learned? 66
4.6 Iteration 2 67
4.6.1 Design 67
Class Diagram 67
Interaction Diagrams 69
Requirements Changes 69
4.6.2 Implementation 71

Stage 1 - The Basis of LeapCV 71
Stage 2 - Refactoring Based on New Design 72
Stage 3 - Distortion Removal Optimisation 72
4.6.3 Test 75
Performance Tests 75
Acceptance Tests 76
4.6.4 Evaluation 76
What went well? 76
What problems were faced? 76
Lessons learned? 77
4.7 Iteration 3 77
4.7.1 Design 77
Class Diagram 77
Interaction Diagram 78
4.7.2 Implementation 78
Stage 1 - Obtaining the Q Matrix 78
Stage 2 - Three Dimensional Reprojection 81
Stage 3 - Feature Matching 85
4.7.3 Test 89
Performance Test - Frame Rate 89
Performance Test - Detection Accuracy 89
Acceptance Tests 90
4.7.4 Evaluation 91
What went well? 91
What problems were faced? 91
Lessons learned? 91
4.8 Implementation Overview 91
4.9 Summary 92

5 Unachieved Requirements **93**
5.1 Introduction 93

5.2 List of Unachieved Requirements 93
5.3 Effect on the Resulting Framework 94
5.4 What it Should Have Achieved? 94
5.5 Summary 95

6 Evaluation **96**
6.1 Introduction 96
6.2 Reflection on Aim and Objectives 96
6.2.1 Aim 96
To develop an open source Computer Vision framework (LeapCV), usable by the Leap Motion Controller community. 96
6.2.2 Objectives 97
Research stereoscopic image processing methods and object classification. 97
Design and develop a framework that allows the Leap Motion Controller to be used for computer vision. 97
Enable the Leap Motion Controller to recognise simple objects. 98
Evaluate the accuracy of object recognition. 98
Release an open source framework. 98
Produce a report that communicates my findings 99
6.3 Evaluation of the Original Hypothesis 99
6.4 Reflection on Research Approach 99
6.5 Reflection on Development Methodology 100
6.6 Reflection on Tools Used 101
6.6.1 Hardware - Leap Motion Controller 101
6.6.2 Library - OpenCV 102
6.6.3 Language - Java 103
6.7 Reflection on Implementation 104
6.8 Evaluation of LeapCV 105

6.9 Summary 105

7 Conclusion **107**

7.1 Project Summary 107

7.2 Aim and Objectives 107

7.2.1 Aim 108

7.2.2 Objectives 108

7.3 Wider Significance 108

7.4 Future Work 109

Bibliography **110**

Appendices **114**

A Use Case Descriptions **115**

B Acceptance Criteria **121**

C CRCs for Iteration 2 **123**

D CRCs for Iteration 3 **125**

E Sequence Diagrams **128**

F LeapCV Javadoc Example **134**

G Implementation Code Snippets **136**

List of Figures

1.1 Infra Red Image Showing LMC Internals 14
1.2 Overview of an Iterative Development Methodology 18

2.1 Example of Radial Distortion of a Square Being Corrected (Hartley & Zisserman 2003) 27
2.2 Pinhole Camera Model as shown in Bradski & Kaehler (2008, p. 372) 28
2.3 Pinhole Camera Model with Image Plane in Front, as shown in Bradski & Kaehler (2008, p. 372) 29
2.4 Stages Leading up to Image Rectification (Gassaway 2011) . . . 30
2.5 Example Point Cloud Image from a Microsoft Kinect (Gnecco 2012) 34

3.1 UML Use Case Diagram 42

4.1 High Level System Architecture 49
4.2 Stage 1 Camera Image - Distortion Present 53
4.3 Stage 2 Camera Image - Distortion Removed 55
4.4 Stage 3 Camera Image - Distortion Further Removed with OpenCV 59
4.5 Camera Image - Distortion Present, with Highlighted Corners on a 9x6 Chessboard 59
4.6 Camera Image - Distortion Further Removed with Highlighted Corners on a 9x6 Chessboard 60
4.7 Example Disparity (Bottom Left and Right) of Synthetic Stereo Pair (Top Left and Right) (Alvarez et al. 2000) 61

4.8 GUI Created in SceneBuilder 2.0 64
4.9 GUI Being Used to Tweak Parameters 65
4.10 Class Diagram for Prototype 2 68
4.11 `nextValidFrame` Sequence Diagram 70
4.12 New UML Use Case Diagram 71
4.13 Class Diagram for Iteration 3 79
4.14 Image Showing Failed Calibration (Bottom) of a Chessboard Image (Top) 80
4.15 Top Left: Undistorted Left Image, Top Right: Undistorted Right Image, Bottom Left: Disparity Map Using StereoVar, Bottom Right: Generated Point Cloud 83
4.16 Point Cloud Showing Inaccurate Depths From Left Hand View . 84
4.17 Matches Before Outlier Removal 86
4.18 Matching Subsection of Mug in Original Left Camera Image . . 87
4.19 Matching Subsection of Mug to Right Camera Image 88
4.20 Mug to be Detected 90

E.1 `LeapCVController` Constructor Sequence Diagram 129
E.2 `getUndistortedImage` Sequence Diagram 130
E.3 `getDisparityMap` Sequence Diagram 131
E.4 `getPointCloud` Sequence Diagram 132
E.5 Updated `getDisparityMap` Sequence Diagram 133

F.1 LeapCV Javdadocs 135

List of Tables

1.1 Project Timeline 20

3.1 Documented `RetrieveCameraImages` Use Case 43
3.2 MoSCoW Task Prioritisation 46

4.1 `LeapCVController` CRC 67
4.2 Results of First Frame Rate Performance Test (FPS = Frames Per Second) 75
4.3 Acceptance Tests Iteration 2 76
4.4 Results of First Frame Rate Performance Test (FPS = Frames Per Second) 89
4.5 Results for Object Recognition Test 89
4.6 Acceptance Tests Iteration 3 90

5.1 Incomplete Use Cases 94

A.1 Documented `ManipulateByTransform` Use Case 115
A.2 Documented `ClassifyObject` Use Case 116
A.3 Documented `ChangeStereoAlgorithm` Use Case 116
A.4 Documented `ChangeStereoParameters` Use Case 117
A.5 Documented `ChangeClassifierType` Use Case 117
A.6 Documented `ChangeClassifierParameters` Use Case 118
A.7 Documented `GetDisparityMap` use case 118
A.8 Documented `ResetTrainingData` use case 119

A.9 Documented `ViewCurrentClassifications` Use Case 119
A.10 Documented `TrainClassifier` Use Case 120

B.1 Acceptance Criteria 122

C.1 `LeapCVCamera` CRC 123
C.2 `LeapCVImage` CRC 123
C.3 `LeapCVStereoUtils` CRC 124
C.4 `LeapCVImageUtils` CRC 124
C.5 `LeapCVStereoCalibrator` CRC 124

D.1 `LeapCVStereoCalibrationUtils` CRC 125
D.2 `LeapCVStereoMatcherFactory` CRC 125
D.3 `LeapCVStereoMatcher` CRC 126
D.4 `LeapCVStereoVar` CRC 126
D.5 `LeapCVStereoSGBM` CRC 126
D.6 `LeapCVStereoBM` CRC 127
D.7 `MatcherType` CRC 127

List of Listings

4.1 Loading the OpenCV Library into the Project 50
4.2 Convert an `Image` into a `BufferedImage` 53
4.3 Write `ImageList` to a `File` 54
4.4 Convert `Image` type to `Mat` type 56
4.5 Convert `Mat` type to `javafx.scene.image.Image` type 57
4.6 Crop `Mat` Image 58
4.7 Initialisation of Distortion Data 62
4.8 Initialisation of Distortion Data 74
4.9 Generate a Point Cloud from the Disparity Map 81
4.10 Writing Generated Point Cloud to a File 81
4.11 Creating Detector Extractor and Matcher from Factories 85
G.1 Slow Method of `Image` Distortion Removal 136
G.2 Example JUnit Performance Test Case 138
G.3 Example JUnit Unit Test Case 140
G.4 Failed Calibration Attempt 142
G.5 Feature Detection 145
G.6 Outlier Removal 147
G.7 Removal of Distortion from an Image using Bilinear Interpolation. Converted from C++ to Java 148

Chapter 1

Introduction

1.1 Computer Vision

1.1.1 What is Computer Vision?

It is very hard for humans to know what their own biological vision really entails, and how difficult it is to reproduce on a computer. Information from the eyes is divided into multiple channels, each streaming different kinds of information to the brain. The brain then subconsciously groups and identifies the parts of the image to examine, along with what parts to suppress. The way in which biological vision works is still largely unknown, which makes it hard to emulate on computers (Hartley & Zisserman 2003, p. xi). Computer vision systems are still relatively naive, all they "see" is a grid of numbers.

Computer vision is a vast field and hard to define. For the purpose of this report it will be defined as:

> "*The transformation of data from a still or video camera into either a decision or a new representation. All transformations are done for achieving some particular goal.*" (Bradski & Kaehler 2008, p. 2)

Even though computer vision, in terms of comparison to biological vision, still

remains an unsolved problem, there have still been many excellent achievements in the field to date.

1.1.2 Uses of Computer Vision

One of the most prominent uses of computer vision currently, is in driver less cars. In recent years, they have reached a level of sophistication at which they are being approved by governments to be used on public highways (Wakefield 2015). Complex and expensive equipment with the capability to analyse a three-dimensional scene in real-time is usually required to achieve this. Computer vision is also used on production lines. Quevedo & Aguilera (2010) set out to try and classify salmon fillets, to see if it was possible to determine whether a salmon fillet had been processed using enzymes. However, computer vision isn't only available to big companies with large amounts of money to spend on research.

> "*Computer vision is a rapidly growing field, partly as a result of both cheaper and more capable cameras, partly because of affordable processing power, and partly because vision algorithms are starting to mature.*"(Bradski & Kaehler 2008, p. ix)

One such device that has been made available is the Kinect by Microsoft. The computer vision society found that the capabilities of the Kinect could be extended beyond its intentional use for gaming, and at a much lower cost than traditional three dimensional cameras. It has been used in areas such as human activity analysis, where it is able to estimate details about the pose of the human subject in its field of vision (Han et al. 2013). It has also been used to for real time odometry whilst attached to a quadcopter, enabling the production of a three dimensional mapping of a physical space (Huang et al. 2011). Some might say a hacking culture has enabled this type of exploitation of technology to take place and is spurring the creators of these technologies to make them more open.

1.2 The Leap Motion Controller

1.2.1 What is the Leap Motion Controller?

The Leap Motion Controller (LMC) is a device aimed at providing a natural user interface through hand gestures. It is made up of two infra red cameras both with a fisheye lense, set up stereoscopically. Figure 1.1 shows an internal infra red image of the LMC cameras and infra red lighting. It provides functionality out of

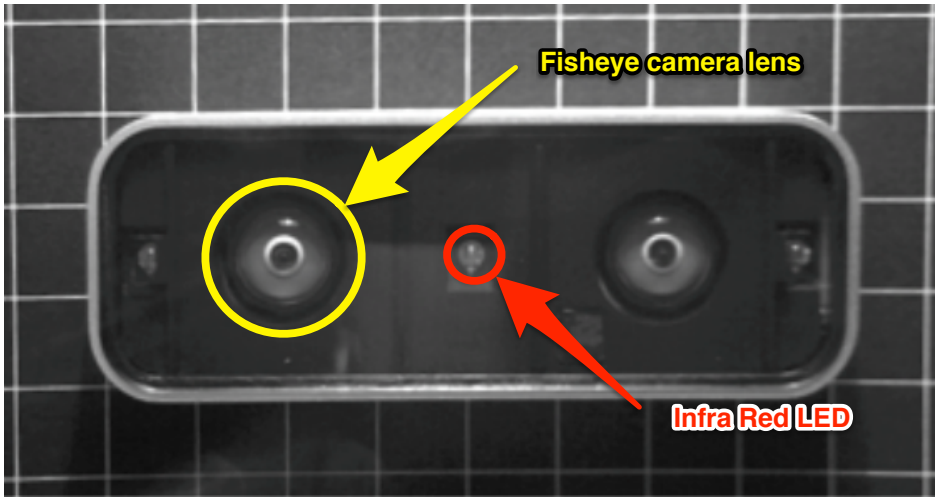

Figure 1.1: Infra Red Image Showing LMC Internals

the box that allows a developer to track movements and gestures made by a hand (Leap Motion 2015). Leap Motion have recently released a new version of their software development kit (LeapSDK version 2), allowing access to more elements of their LMC. Encompassing, images from the two cameras. This could possibly be exploited in computer vision. The LeapSDK is closed source, therefore it is not known how the current methods by which hands and gestures are classified (Bachmann et al. 2014, p. 217). With the access to the images, the LMC now allows exploration of other, custom methods for object classification and tracking. The LeapSDK can be implemented in C++, C#, Objective-C, Java, Python and Javascript. It is supported on Microsoft Windows, Mac OS X and a variety of Linux systems. There are currently iOS and Android libraries that are available

through request. It is therefore a truly multi platform device.

The LMC is an extremely accessible device and can be purchased for a price in the range of £55[1]. It is small in size with dimensions 80mm × 30mm × 11.25mm.

1.2.2 Uses in Computer Vision

The LMC has already been used in many innovative ways. An example of this is its use in recognising Australian and Arabic sign language (Potter et al. 2013, Mohandes et al. 2014). However, these solutions only make use of the default detection system in the LeapSDK. The default detection system is advanced and accurate enough to detect hand movements to this scale but raw image access is required to detect objects other than hands. As previously stated, image access has now been given in the LeapSDK, so a whole new future of possibilities has been opened. Having access to the images allows for their use in computer vision. In turn this means that such an accessible and small device can be used in a variety of new ways. By creating an open source computer vision framework (LeapCV) for the LMC, it might become of use in the field of robotics. For example, its size would make it suitable to attach to a robot as a rudimentary depth sensing system for collision avoidance.

1.3 The Problem

Currently, there is no simple solution that allows a developer to simply plug in a LMC and start developing for computer vision. While a computer vision expert might be able to start programming straight away, a non-expert would have to learn a large amount before they start to see any results.

[1] http://www.amazon.co.uk/Leap-Motion-Controller-Interacts-Airspace/dp/B00C66Z9ZC

1.4 The Purpose

The main purpose is to try and expand the areas on which the LMC can be developed and to try and find out whether it is feasible to say that a LMC can be used for computer vision in a simple way. For example, detection of objects other than hands and tools with only a few function calls.

This might start a new interest in computer vision allowing these non-experts that might have thought it too difficult, to experiment in a new and easy way. There is a large community of interest in the Leap Motion forums, with many different projects using the current LeapSDK. It might be assumed from the mere interest on threads that contain any form of reference to computer vision, that carrying out research on this will be a useful piece of work.

The framework will be open source, unlike the current LeapSDK. Providing a solution that can be changed in any way to suit a developer of the LMC. Hopefully LeapCV will lead by example and be extended further after release.

1.5 Hypothesis

It is possible to use the LMC in the field of computer vision, so that application developers can exploit the technology in different ways as to what was originally intended. This ability can then be structured into a framework, so that the effort application developers have to put in to use the LMC as a solution is low, making it an option to consider in new projects.

1.6 Solution Approach

It is envisaged that LeapCV will be made possible by taking advantage of functionality of the OpenCV library then exploiting the new image functionality of the LeapSDK.

1.7 Aim and Objectives

This section outlines the scope of the project. The success of this project will be calculated based on the completion of each.

1.7.1 Aim

- To develop an open source Computer Vision framework (LeapCV), usable by the Leap Motion community.

1.7.2 Objectives

- Research stereoscopic image processing methods and object classification.
- Design and develop a framework that allows the LMC to be used for computer vision.
- Enable the LMC to recognise simple objects.
- Evaluate the accuracy of object recognition.
- Release an open source computer vision framework.
- Produce a report that communicates my findings.

1.8 Development Methodology

There are lots of development methodologies available for consideration. It is important to follow a development methodology to:

- Divide a large problem into easy to understand tasks.
- Sharpen focus.
- Improve time estimates.
- Provide progress visibility.

- Provide better structure.
- Lead to better coding and documentation.

(Dawson 2005).

As this is not a team based project, methodologies such as SCRUM will not be considered. Neither will a conventional waterfall model be considered, as this project has a relatively high uncertainty in user requirements. Many requirements will be explored throughout the development process. Therefore, an iterative approach will be followed. The first iteration will be an example of a throw away prototype so as to explore technical uncertainties and ascertain initial requirements. It will also help to refine the scope. The later iterations will then follow more of an evolutionary prototype approach. With the first evolutionary prototype involving a large upfront requirements analysis and design. This will be based on knowledge gained in the first iteration. The aim at this stage is to have a good base structure for the rest of the iterations to be evolved from with tweaks to requirements happening in parallel with development. Figure 1.2 gives a visual representation of the iterative approach.

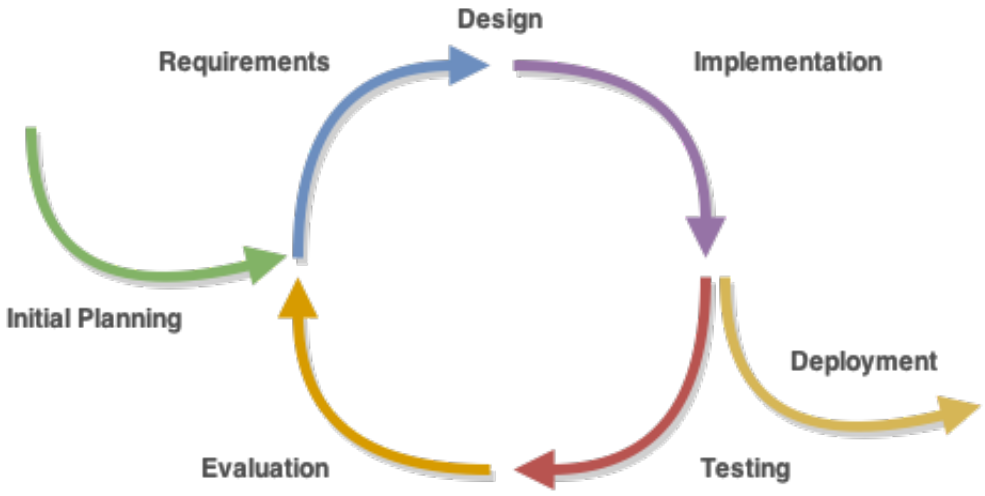

Figure 1.2: Overview of an Iterative Development Methodology

1.8.1 Planned Iterations

During each iteration a test bench application will be developed that will make use of LeapCV. The development of this test bench aims to emulate LeapCV being used in a real application. The development of this prototype will depend upon what level to which LeapCV is completed. Throughout each iteration sufficient unit tests will be implemented based on the requirements. Below is an outline of work aimed to be completed in each iteration. The work breaks down three discrete sections

1. Image processing
2. Object detection
3. Classification

Each item in the list, is dependant on the previous work being completed. Therefore, each iteration will aim to solve the work break down in the order specified. Each iteration will be fully documented in chapter 4.

Iteration 1

The first iteration will be an initial throwaway prototype. It will be used to explore the technologies that are going to be used in the project, mainly to find out how the extensive OpenCV library works, and how well it will interface with the LeapSDK. Image processing and stereo image processing will be explored. Once completed it will allow better requirements to be established for LeapCV.

Iteration 2

The second iteration will be the first of an evolutionary prototype. It will better expand on the knowledge that will be obtained in iteration 1. It will implement the foundation upon which the LeapCV will be based. The first design will be established here. It is hoped that most of the image processing and stereo

image processing functionality will be implemented in this stage, providing a solid integration of OpenCV and the LeapSDK for further development to be built upon.

Iteration 3

The third iteration will be the second stage of an evolutionary prototype. It will implement the more complex elements of the library, taking full advantage of the work that has been completed in iteration 2. Object detection will be implemented here. Any previous work will be refined based on any requirement changes. If successful, and time allows, then classification will be explored.

1.8.2 Project Timeline

Table 1.1 gives a rough outline as to when work will be completed.

Stage	**Timebox**
Carry out literature review	December 2014 — January 2015
Initial requirements analysis	December 2014 — January 2015
Iteration 1: Explore with throwaway prototype	January — February 2015
Adjust requirements analysis	January — February 2015
Iteration 2: Design, test and develop evolutionary prototype 1	February — March 2015
Iteration 3: Design, test and develop evolutionary prototype 2	March — April 2015
Finish any development and testing. Prepare report.	April 2015
Complete report	April 2015

Table 1.1: Project Timeline

1.8.3 Tools

LeapCV will be written in Java using the Java SDK. For the purpose of scope it will only be developed using version 1.8 of the Java SDK with JavaFX included. It will be developed in and built to be used within the IntelliJ IDEA integrated development environment. Development will be carried out mainly on Mac OS X 10.10 but also on Ubuntu 14.04 and Windows 8.1 to ensure multi platform capability. Due to time constraints, the project will only get fully tested on Mac OS X. JUnit 4 will be used for unit testing.

For the LMC, version 2.2.3 of the LeapSDK will be used. For OpenCV, version 2.4.10 will be used. There is a more recent version 3.0 of OpenCV. However it is a beta version and so will not be considered due to possible instability.

The code will be managed in a Git based version control system (VCS). The VCS hosting will be provided by GitHub[2]. Using a VCS enables the recording of changes to files and code over time which will be of use when discussing project implementation in chapter 4. It will also prove useful when testing on multiple platforms. By storing the code on a server, any changes can be made locally and then made available to all other platforms. This helps to ensure that each platform uses the same version and avoids platform specific versions of code being created.

[2] http://www.github.com

1.8.4 Testing Methods

Testing will be carried out to ensure that LeapCV works as intended, and that there are no bugs. This will consist of:

- **Unit testing:** these will be used to test modules of the program throughout the development process.
- **Performance testing:** this will be used to ensure that some of the non-functional requirements have been met.
- **Acceptance testing:** this will show how far the solution has gone to meeting the requirements outlined by the stakeholder.

Regression testing will also take place, but not explicitly. The unit tests that will be written in the early iterations will also be run in the later iterations to ensure that no functionality has been broken when changes are made in the code. Most of the testing results will be discussed in chapter 4. However as they hold a close relationship with the use cases, the acceptance tests will be shown in chapter 3.

1.9 Report Structure

Here is a brief summary of what will be covered in each chapter.

- **Chapter 2** is a literature review that covers the basic principles of stereo imaging and classification. It aims to provide a base knowledge of computer vision, as well as comparing some existing solutions and highlighting some real world applications.
- **Chapter 3** defines the initial requirements that have been identified for the project.
- **Chapter 4** shows the evolution of this project throughout its development. It shows how the requirements and designs have evolved over the project lifecycle. It also highlights the key decisions that were made throughout

the implementation.

- **Chapter 6** will evaluate the end product, techniques and tools used. It will also layout a call for future work based on what has been completed.
- **Chapter 7** will conclude and summarise the findings of this report.

1.10 Summary

Chapter 1 has identified the problem and suggested an approach to solving it. Following this, chapter 2 outlines more specifics in terms of the technology and background onto which the overall solution will be based.

Chapter 2

Background Research

2.1 Introduction

This chapter looks to inform the reader about the methods that will be used within this project. It aims to give a brief understanding in how Computer Vision works and what elements are applicable to enable this project to succeed. It begins with a brief overview of object recognition, stereoscopy and classification. Then continues to discuss some real world applications. Finally discussing existing libraries for computer vision.

2.2 Object Recognition

2.2.1 What is Object Recognition?

Humans can look at an object only a few times to then enable them to recognise other examples in future. However, the problem isn't as simple as just recognising an object. The difference between a human and a computer is, a human can recognise the context of that object. A human can recognise the difference in intention of an object appearing in a certain scene. An example being another person holding a gun to their face, compared to holding a bunch of flowers. They

can also choose which objects in a scene are relevant, and which are not. For example, you would generally never find a shower inside a kitchen. Therefore if a shower was in a scene it would imply that it was a bathroom (Forsyth et al. 2012).

2.2.2 What is Expected from Such a System?

Object recognition in digital images is not a new problem. Besl & Jain (1985, p. 82-83) lays out the characteristics of an ideal object recognition system. As does Forsyth et al. (2012, p.570-572). Through cross comparison, the characteristics that were looked for then are very much the same as what would be an ideal system now. Forsyth et al. (2012) says that a system such as this should:

- Recognise many different objects.
- Recognise objects seen against many different backgrounds.
- Recognise objects at an appropriate level of abstraction.
- Infer useful information about the special properties in a particular instance.
- Produce useful responses to unfamiliar objects.
- Produce responses that help achieve goals.
- Produce responses of useful complexity.

However, they then go on to highlight the fact that most current recognition strategies perform poorly when measured against these requirements.

"Not because they are bad; the problem is just very difficult".

2.3 Stereoscopy and Calibration

> "*Stereo matching is the process of taking two or more images and estimating a 3D model of the scene by finding matching pixels in the images and converting their 2D positions into 3D depths.*"(Szeliski 2010)

The idea behind this stems from the way that biological vision works. In humans, depth is perceived from the difference between the images produced by the left and the right eye. To emulate this in computer vision, the images of a scene produced by the cameras are taken from two different positions. One of the difficulties then faced with stereo matching is being able to identify the corresponding pixels in each image taken. When using two cameras, as those that are to be provided by the LMC, Bradski & Kaehler (2008, p. 415) define four steps that are needed to achieve successful correspondence between the images produced by each.

1. Mathematically remove radial and tangential lense distortion; this is called undistortion. The outputs to this step are undistorted images.

2. Adjust for the angles and distances between cameras, a process called rectification. The outputs of this step are images that are row-aligned and rectified.

3. Find the same features in the left and right camera views, a process known as correspondence. The output of this is a disparity map.

4. If we know the geometric arrangement of the cameras, then we can turn the disparity map into distances by triangulation. This step is called reprojection, and the output is a depth map.

For the system to make successful use of the images, it needs to know the set up of the cameras (calibration). Much research has already been done on this. As a result, computer vision libraries have appeared, such as OpenCV. Libraries such as this, create a layer of abstraction between the user and the underlying low-level image processing, allowing the process of calibration to be carried out with ease, without the need to know all of the mathematics behind the process. Although it is good to have a general overview of what is happening. As this is a fundamental element to this project, each stage of calibration will be covered in a little more detail.

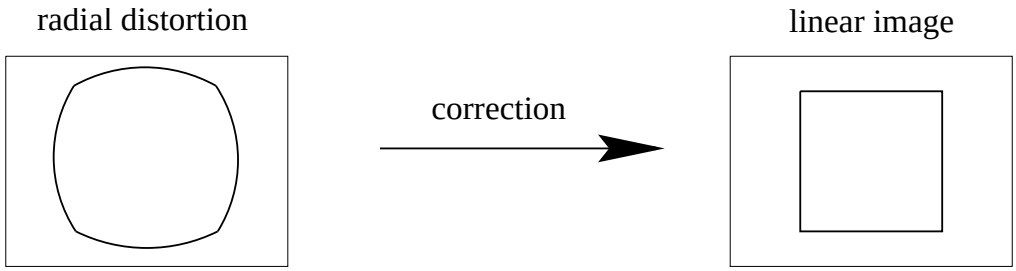

Figure 2.1: Example of Radial Distortion of a Square Being Corrected (Hartley & Zisserman 2003)

2.3.1 Image Correction

Images produced by cameras with lenses suffer from aberrations. No lens is perfect. Therefore correction is needed to get a true representation of the objects in the image. There are many different types of aberration, but the only one that will be discussed here is distortion. Distortion is an effect that changes the overall shape of an image (geometric warping) (See Figure 2.1). It is caused by the fact that different areas of a lens have slightly different focal lengths (Forsyth et al. 2012, p. 42). There is a lot of information available on how distortion happens so only the basics will be covered. The fundamental stages of processing images for use in computer vision are based on geometry. To reliably calculate depth in a scene, the system needs to know the intrinsic parameters of a camera. Essentially, the system wants to know if what the camera is producing is a true representation of the scene it is recording. Much like when you make a visit to see an optician, the optician will carry out a process to try and discover if what you see is a distortion of reality. The parameters will allow the system to take the images and pre process them (undistort) so that it can be confident they are true. This is done by producing both a camera geometry and a lens distortion model through a process of calibration. The two models will now be discussed in a little more depth.

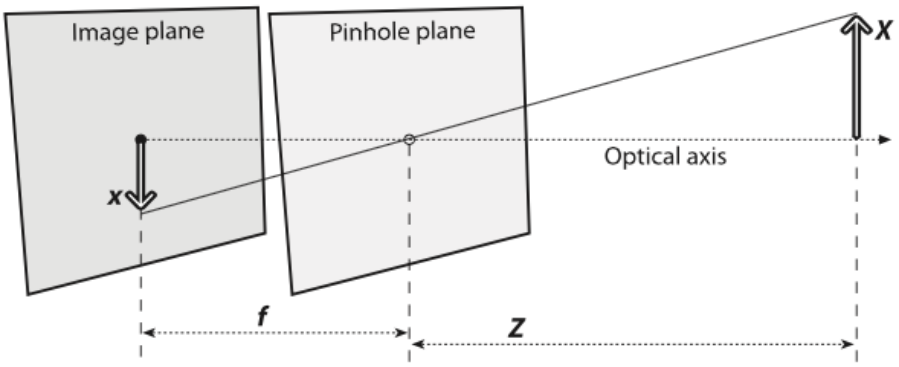

Figure 2.2: Pinhole Camera Model as shown in Bradski & Kaehler (2008, p. 372)

2.3.2 Camera Model

Most of the information available on camera models are focused around a pinhole camera. The following description and figures of this model is taken from Bradski & Kaehler (2008, p. 371-373), If more depth is needed the reader should refer to the cited material. The pinhole camera model is the most basic type of camera that still holds the fundamental mathematics on how most modern digital cameras work today. In a pinhole camera (See Figure 2.2) light reflected from the scene/object travels through a pinhole (a small aperture) that is made in the pinhole plane. This light is then projected onto the image plane. The size of this projected image is proportional to the focal length f, where Z is the distance from the pinhole plane to the object, X is the length of the object and x is the objects resulting projection on the image plane. Therefore, we can work out the size of the object using the equation $-x = f\frac{X}{Z}$. The value of x is negative due to principles of physics whereby the projection of light that has travelled through a pinhole rests, so that the resulting image appears upside down. For mathematical simplification Bradski & Kaehler (2008) states that the negative value can be omitted leaving $x = f\frac{X}{Z}$. This is done by moving the image plane in front of the pinhole plane, and then treating the pinhole as a center of projection (See Figure 2.3). This results in an image that is the correct way up. However, it would be impossible to make this camera physically, it is simply to make the mathematics less complicated. In the real world, the center of projection will not be precisely

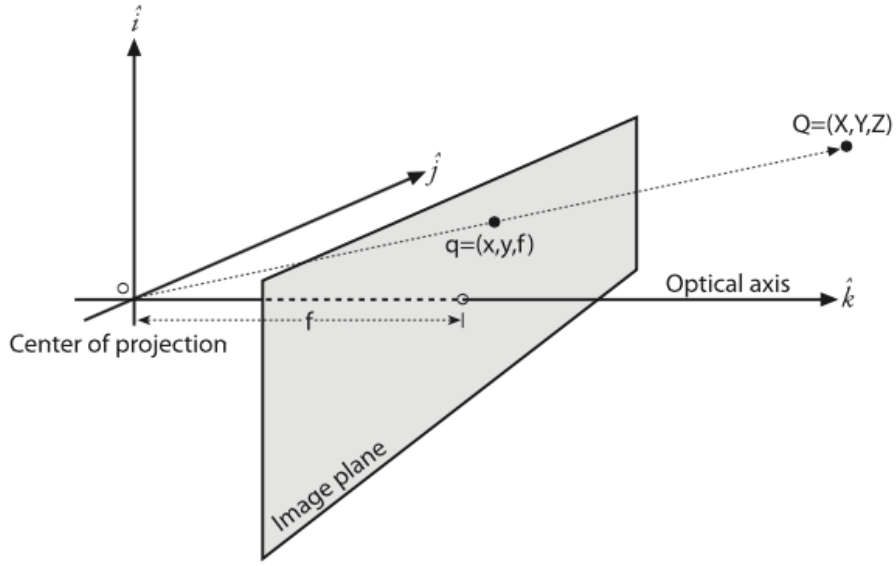

Figure 2.3: Pinhole Camera Model with Image Plane in Front, as shown in Bradski & Kaehler (2008, p. 372)

in the middle of the imaging sensor of a camera. Therefore displacement values need to be considered, c_x and c_y. The resulting model allows us to pinpoint the point Q (a point in real space) with coordinates (X, Y, Z) as a pixel location (a point on the image) where the pixel location is (x_{screen}, y_{screen}). This results in a mathematical model of $x_{screen} = f_x \left(\frac{X}{Z}\right) + c_x$, $y_{screen} = f_y \left(\frac{Y}{Z}\right) + c_y$. Having a model like this allows us to define the parameters of the camera so that false transformations of light being collected from the physical world can be corrected. This is important for methods of image rectification which will be discussed later.

2.3.3 Lens Distortion

The camera models discussed assume that cameras produce images in which the straight lines represent the same straight lines that are in the scene. However as discussed earlier a lens is never perfect and so many will create a visible curvature in the projections of straight lines in an image. Unless this distortion is taken into account, it becomes impossible to create highly accurate photorealistic reconstructions. A phenomenon known as radial distortion produces a visible curvature on the straight lines that appear in the image. This is where pixels

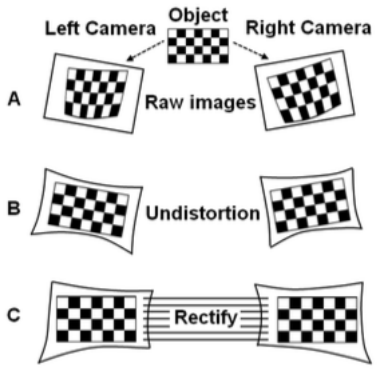

Figure 2.4: Stages Leading up to Image Rectification (Gassaway 2011)

in the image are displaced away or towards the image centre by an amount proportional to their radial distance. These types of distortion are known as barrel and pincushion respectively. The LMC has a radial distortion, more specifically it is known as a fisheye. The fisheye is a distortion where the image that is produced provides an near 180°span side-to-side (Szeliski 2010).

2.3.4 Image Rectification

To simplify the way in which pixels are matched between two images it is a good idea to align them vertically along the y-axis. This is done by aligning the corresponding epipolar lines . This means that to find a corresponding pixel in the left and right image, a search only has to take place in one dimension, along the epipolar lines. This is known as the epipolar constraint. This can be easier to understand if seen visually an example of the rectification step can be seen as item C in figure 2.4. For more in depth information on image rectification please read Bradski & Kaehler (2008, p. 419-421).

2.3.5 Stereo Correspondence

Stereo correspondence is the process of finding the same pixel in two different images or camera views. The displacement of the pixel in one image in relation to corresponding pixel in the other image is known as a disparity. The set of all

disparities between two images is known as a disparity map (Brown et al. 2003).

There are many methods of finding correspondence between a stereo pair of images. Two terms that are used in literature are "global methods" and "local methods". Both Brown et al. (2003) and Scharstein & Szeliski (2002) carry out a review on these methods. It is inferred that local methods are much more computationally efficient than global methods. However global methods generally seem to generate the best disparity.

Block matching algorithms are the most popular of the local methods but one can also use gradient methods or feature matching. Of the three, block matching seems to have become the more accepted method. Block matching methods estimate disparity of a pixel in one image (the template), with a series of small regions extracted from the other image (the search region). A statistical method, known as a match metric is then run over the pixels to determine the best match. The sum of absolute differences method (SAD) is the most popular of these match metrics (Brown et al. 2003).

2.4 Classification

> *A classifier is a procedure that accepts a set of features and produces a class label for them.* (Forsyth et al. 2012, p. 487)

Classifiers are like rules. Data gets is given, in this case a feature in an image, and the rule returns a class label. There are three basic ingredients to the recipe of classification.

1. Find a labelled dataset.
2. Build the features from within the dataset.
3. Train the classifier with the features.

The most difficult ingredient to apply is building the features. Some feature constructions are more suited to certain applications than others. It is important

to extract the features that show the variation between each class, as opposed to extract the features that show the variation within a class. As an example, one might want to extract the features of a car and the features of a van. The only features that are important are the ones that differentiate between a car and a van. The features that differentiate between one car and another variation of a car are not important. On the other hand, it is clear to see there might also be a use for differentiating between two different cars in another scenario. Showing that extraction of these features is very much application dependant.

It is worth noting that classifying an image of an object, is a different problem to detecting an object within an image. The classification of an image is stating what the object in the image is. The detection of an object within an image is stating that something exists within that image.

2.4.1 Extracting Image Features

Edge Detection

In many cases, the first stage of image analysis is to determine where the edges are. Evidence suggests that edge detection is an important part of biological vision, particularly in mammals. In most situations a predator (or prey) can be seen to be contrasted sharply with its background. By noting the edges enables the animal to quickly recognise other animals near it, which explains why camouflage is such a popular technique in the animal kingdom (Coppin 2004). An object is separated from its background in an image by an occluding contour. An occluding contour is a line where, on one side, the pixels in the image lie on the background, and on the other side, lie on the object. Occluding contours form the outlines of objects. Edges can be obtained from large image gradients. Such as those cause by sharp changes in brightness.

Feature Detection

The terms "feature", "corner" and "image point" are used interchangeably throughout literature. For example Bradski & Kaehler (2008, p. 317) describes corners as

> "*the points that contain enough information to be picked out from one frame to the next.*"

Where as, Forsyth et al. (2012, p. 179) describes corners as being a local area where the orientation of the image gradient changes sharply; a corner in the more literal sense of the word. Forsyth et al. (2012) says a feature is a more specific term of something that can be described as an image point. This report will continue using the term feature so as to avoid confusion with any code references in OpenCV later on.

Features are easy to match between different images. This makes them extremely useful when reconstructing points in three dimensions when using multiple images. Such as in a stereo image pair discussed previously. The idea is to extract unique features in an image that can be tracked in another image of the same object or scene. Imagine picking a point on a large blank wall in one image, it wont be very easy to find the exact same point in the second image. Instead unique points should be selected, such as a tap sticking out of the wall.

Classification of Three Dimensional Objects

Attempts have also been made at classifying objects in a three dimensional reconstructed scene. Besl & Jain (1985) discuss methods by which range data might be used as a method of object detection. This is one of the earliest papers that suggest the usage of range data for classification. With triangulation based finders that collect this type of data, dating back to the early seventies (Shirai 1972).

Range data comes in the form of a point cloud. A point cloud is a three

dimensional array or pixels that also hold an extra depth coordinate. This results in what looks like a cloud of points, hence the name point cloud. An example can be seen in figure 2.5. They are produced in many different ways. Devices

Figure 2.5: Example Point Cloud Image from a Microsoft Kinect (Gnecco 2012)

such as the Microsoft Kinect use a three dimensional scanner producing a dot matrix to produce them. LiDAR[1] is another example, that uses a single pulsing laser to scan a scene. Other methods include using a pair of images in a scene.

Attempts have been made at classifying point clouds. A piece of software called CANUPO has been developed based on methods outlined by Brodu & Lague (2012). This makes use of classifiers to determine the physical features of rivers, coastal environments and cliffs. Being able to classify between water, gravel vegetation and a rock surface.

2.5 Real World Application

With the brief overview of computer vision theory, the question is "where is this technology actually used?". There are companies that exist, that dedicate their

[1]Literature varies in describing what this acronym stands for. Light Detection and Ranging seems the accepted term.

mission to providing computer vision solutions. One such company is Stemmer Imaging. Stemmer Imaging is a european imaging technology company. They provide pre-built vision solutions and define some of the application areas of these in their Imaging and Vision Handbook (Stemmer n.d.). Some of these areas are:

- **Verification** - The process of verifying that a product, assembly or package has been correctly produced. For example in
- **Measurement** - The accurate measurement of component dimensions ensuring they are within pre-defined manufacturing tolerances.
- **Flaw Detection** - Ensuring that flaws such as contamination, scratches, cracks, discolouration or burn marks have not been caused by slight variations in the manufacturing process.
- **Code Readers** - The tracking of one or two dimensional barcodes for product recognition.
- **Positioning** - Picking objects up from a production line and placing them in a package for shipping. Or aligning the objects for precision machining.

These are just a few of the examples they give, but it is clear that there are many areas in which computer vision technology can be exploited.

2.6 Computer Vision Software

There are plenty of solutions available in terms of computer vision, but what solutions are there for developers to create applications with computer vision equipment? With smartphone processors often improving in each new release, as well as built in cameras, the demand for computer vision applications is increasing. Currently, devices are capable of stitching together several photographs into a high resolution panoramic image in real time, as well as having the capability of using facial recognition to unlock the device (Findling & Mayrhofer 2013).

Here is an overview of software libraries that are available to develop computer

vision solutions with.

2.6.1 OpenCV

OpenCV is an extensive Computer Vision library that has grown out of a research project, initially run by Intel. Universities were seen to be sharing internal computer vision infrastructures between students, and so OpenCV was conceived to make this type of infrastructure universally available. Its first alpha release was made available in January 1999. One of its goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly (Bradski & Kaehler 2008, p. 1). By making computer vision more accessible, Intel were increasing the need for faster processors, in turn generating themselves more income (Bradski & Kaehler 2008).

Computer vision is computationally expensive. However, many scenarios that use computer vision need to happen in real time. OpenCV was designed for computational efficiency, with a strong focus on real-time applications. Early on in its life, Intel had pushed for the requirement of more power from CPUs. Then in 2010, GPU[2] acceleration was added to OpenCV. It has been developed so that adoption from the CPU[3] library to the GPU library is easy, so that developers do not require any training. Pulli et al. (2012) have shown this extension has sped up algorithms up to four times on a GPU than a CPU.

While the original library is written in C/C++, it has also been wrapped to work on the Apple iPhone iOS and Google Android platforms. The Google Android platform version is also compatible with any Java application.

Since its first release the library has become well established in the Computer Vision field. A simple search with the term "*OpenCV*" on google scholar, returns about 38,700 results. With the Bradski & Kaehler (2008) book itself cited by 3409 others.

[2]Graphical Processing Unit
[3]Central Processing Unit

OpenCV is licensed under the BSD[4] open source licence, meaning it is open and free. The code can be embedded in other applications for research or for profit and there is no obligation that the created applications code needs to be open or free.

2.6.2 SimpleCV

SimpleCV defines itself as an open source framework for building computer vision applications (sightmachine 2014). It makes use of OpenCV and the Python programming language. It is only available to be used with Python. Its focus is to simplify the process of developing computer vision applications by providing a simple programming interface. It does this by providing a simple and semantic interface for cameras, image manipulation, feature extraction and format conversion.

2.6.3 Point Cloud Library

The point cloud library is focussed on three dimensional perception. The library is written in C++ and currently there is no supported wrapper available for using it with other languages. Although it does state that an Google Android version is in the works (Rusu & Cousins 2011). Their literature mainly highlights the use of this library with robots being able to see in three dimensions. It incorporates three dimensional processing algorithms and can perform filtering, feature estimation, surface reconstruction to name a few.

2.7 Summary

The knowledge obtained throughout the background research should provide enough of a starting point to apply the technologies to the LMC. Methods of stereoscopy will be useful throughout the development of LeapCV as will the

[4]Berkley Software Distribution

basic understanding of feature detection. OpenCV will be the library of choice throughout development, mainly due to it being the only one discovered with a Java interface. Although out of the three libraries discovered, it seemed to be the most extensive and supported, with a very good history of development. Java is the preferred language of the author, therefore a Java library was preferable. Now this paper moves on to carrying out a requirements analysis.

Chapter 3

Requirements Analysis

3.1 Introduction

This chapter attempts to lay out a set of requirements for LeapCV from the point of view of the stakeholders. They will set out a contract between the stakeholders and the LeapCV developer as a means to show what LeapCV is going to be. It should **not** show how it will be done - that is the purpose of design (Dawson 2005). They provide goals upon which a realistic scope for the project may be set and will also be the backbone of the tests that are to be written during implementation.

The elements needed to produce a good requirements analysis are as follows.

- **Stakeholder identification** is important so that the requirements are of use to who they are being developed for.
- **Use case analysis** identifies interactions between your system and users or other external systems. Also helpful in mapping requirements to your systems (Miles & Hamilton 2006).
- **Functional requirements** of a system define the system, the data and the user interface (Dawson 2005).

- **Non-functional requirements** should lay out constraints, standards, limitations, performance requirements, verification requirements and validation criteria (Dawson 2005).
- **MoSCoW Prioritisation** will set the prioritisation for each requirement. This gives a good identification as to which ones should be developed first.

The following sections will elaborate on these and show the documentation produced.

3.2 Stakeholder Identification

This project is aimed at producing an open source framework that will suit a variety of users, with the main focus being on the application developer, where the application developer will use LeapCV to develop other applications. LeapCV will not be an application that can be used as an end user. It will be designed to be a simple-to-use interface that saves the application developer a large amount of time when applying it to their own application. However, it is possible that other applications might need access to information that is produced by LeapCV, therefore another actor of user needs to be considered. Through the analysis of existing libraries and solutions that work with other imaging systems, it is possible to identify what would be suitable to provide as a simpler solution for the LMC. The stakeholder also plays a major role in discovering and keeping a project, within a defined scope. As the application developer will be the main user of the system, they will have most of the interactions.

In summary, the stakeholders are:

- **Application Developer** who will have the most interactions with the system.
- **User** who will in most cases be an external system.

3.3 Requirements Capture

Due to the exploratory nature of this project, it is hard to realise what is fully required of the system at this stage. However, the research carried out in chapter 2 can give a guide as to what might be needed. The requirements caught here can then be explored in the first iteration and then then expanded upon during the design stage of the next iteration. The problem now is to capture what requirements need to be in place to create LeapCV. In chapter 2 four steps were defined to achieve successful correspondence between images. In summary these were:

1. Remove distortion from the images.
2. Rectify the images.
3. Find corresponding features in the images and produce a disparity map.
4. Turn the disparity map into distances and produce a depth map.

However, it is arguable to say that these steps are for quite a specific problem, and a framework should be more generic. Therefore LeapCV should be able to carry out all of the operations above, without any dependency on the previous step. The application developer should be able to carry out manipulation on the images separately from the images that are getting processed through the algorithms in OpenCV. This will allow for the development of an application that might show a before and after image for example, rather than only having access to an image post processing, that might have become non-understandable to a human.

For classification it will be useful to be able to train the classifier. It will also be useful to be able to change the parameters that define the classifier. This data should have the ability to be reset, on a new set of data.

The ability to remap the pair of two dimensional images into a three dimensional form will be used to explore classification methods.

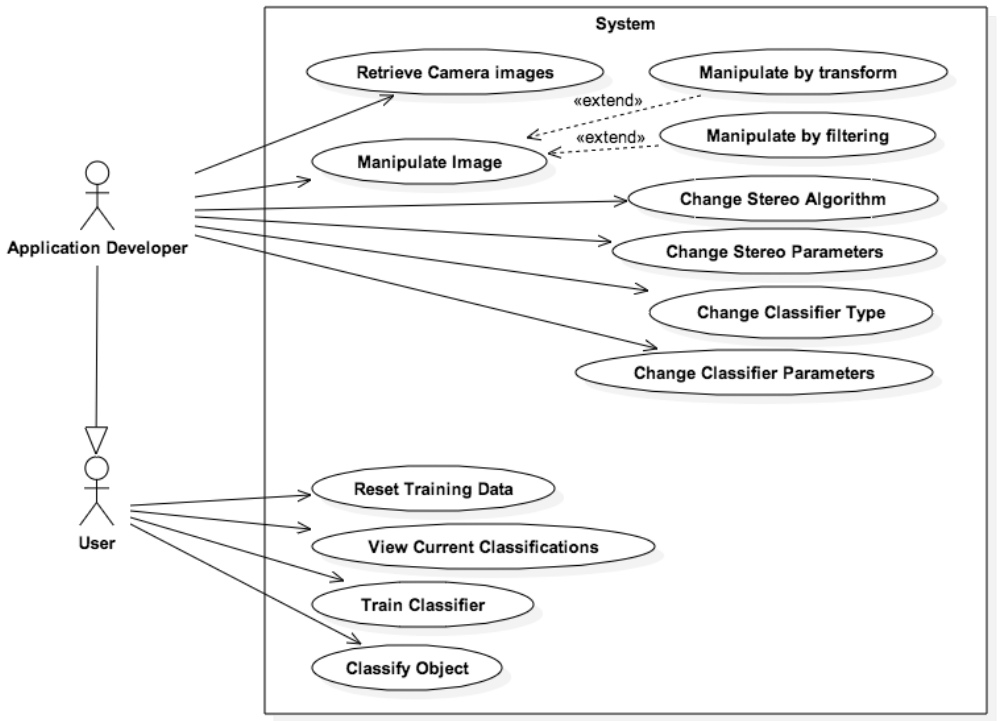

Figure 3.1: UML Use Case Diagram

3.4 Use Case Analysis

This section has been produced by following the methods outlined by Miles & Hamilton (2006) in "*Learning UML 2.0*".

3.4.1 Use Case Diagram

A use case diagram is used to capture pieces of functionality that a system will provide to the stakeholder and other users. Figure 3.1 is an example of a UML use case diagram. The stick figures represent actors on the system, the ovals represent use cases and the box enclosing the use cases show the overall system. The actors are acting upon the system through the means of a use case. Each connecting line shows the use case upon which the actor should be able to interact. In the case of a line between use cases that shows the "*extend*" relationship, it shows that the use case might want to completely reuse the connected use cases behaviour. However in *UML 2.0* this reuse is optional and dependent on a

runtime or system implementation decision. Remember at this stage we should not be stating **how** it should be implemented.

3.4.2 Use Case Description

Table 3.1 is one example of a use case description. A use case description provides the reader a little more insight into what actions are going to be performed. This is not always immediately clear from the higher level use case diagrams. Each use case that has been defined so far, has a corresponding use case description. These can be found in appendix A.

Use case	`RetrieveCameraImages`
Description	The application developer must have access to raw camera images to use with other elements of the framework. It will also allow for the images to be displayed.
Actor	Application developer
Entry Condition	The framework and the system it is being used on, must have the relevant dependencies loaded and the leap motion controller connected.
Flow of Events	1. Application developer requests an image. 2. Framework checks leap motion is ready. 3. Framework retrieves image from `LeapSDK`. 4. Framework converts image to usable format. 5. Framework returns image.
Alternative Flow	If leap motion controller is not ready , framework throws `LeapNotFound` exception.
Exit Condition	Application developer receives valid image.

Table 3.1: Documented `RetrieveCameraImages` *Use Case*

3.5 Functional Requirements

Any requirements that cannot be listed as a use case, are defined here as functional requirements. As this is a framework, user interface requirements do not need to be considered.

Automatic camera configuration. LeapCV should abstract the LMC configuration away from the application developer and should not require any explicit calibration.

Default configuration. LeapCV should have a default configuration for training and classifying three dimensional shapes.

3.6 Non-Functional Requirements

Speed. LeapCV needs to be able to process images and return a useful classification output in near real-time. A minimum of 15 frames per second should be the target.

Accuracy. LeapCV should have equal to or greater than 90% accuracy in its classification ability.

Multi-platform. LeapCV should be usable on all of the platforms that the LMC is currently supported on.

Java standards LeapCV should conform to Java standards and have a well documented interface. A Javadoc should be produced.

Semantics. LeapCV should not stray too far from terminologies and semantics used within existing computer vision libraries, unless it is essential. Ensuring users with prior experience of computer vision can use it with ease.

Extensibility. As the project will be made open source, LeapCV needs to be designed with extensibility in mind.

3.7 Acceptance Criteria

Acceptance criteria should be defined against the use cases, to provide a measure of completion. Each use case has a set of acceptance criteria defined which can be found in appendix B.

3.8 MoSCoW Prioritisation

MoSCoW is a recognised method for prioritising requirements by their importance. Ashmore & Runyan (2015) defines the categories in order of highest to lowest importance as

Must have. All features categorised in this group must me implemented for the system to work.

Should have. Features categorised in this group are of high importance but are not essential to the system working.

Could have. These features will enhance the system by giving it greater functionality, but they do not have to be delivered as a priority.

Want to have. These features only apply to a small group of users and have limited business value.

The terminology here has a business system feel. Where the term *business value* is used above, in this project it will mean it holds a limited value to the frameworks end result. Table 3.2 shows the user stories with their recognised priority.

Task	MoSCoW Prioritisation			
	Must	Should	Could	Wont
Retrieve camera images	x			
Manipulate image	x			
Manipulate by transform		x		
Manipulate by filtering		x		
Change stereo algorithm		x		
Change stereo parameters		x		
Change classifier type			x	
Change classifier parameters			x	
Reset training data			x	
View current classifications			x	
Classify object			x	
Train classifier			x	

Table 3.2: MoSCoW Task Prioritisation

The prioritisations have been worked out so that the fundamental elements of LeapCV will get created first. Although the end product should be able to classify images, there is not yet a guarantee that it is possible to do so. All of the previous requirements have to be implemented before any type of classification can be considered.

3.9 Summary

The initial requirements for the system have now been identified. Now this is completed an initial design can be created. This design will be shown in iteration 2 of chapter 4. Chapter 4 will make use of these requirements. It will aim to build upon them and turn them into a working system.

Chapter 4

Design, Implementation and Testing

4.1 Introduction

This chapter covers the implementation stage of the project. In chaper 1 the methodology of how this was going to be done was outlined. In chapter 2 research was carried out into computer vision techniques. Some of this research will now be extrapolated into a functional framework. The implementation has been planned in three iterations. Where the first iteration is a throwaway prototype and the further two iterations will follow the evolutionary prototype method. Each iteration will be further broken down into work items. The work items are loosely based on the initial requirements defined in chapter 3. These might changed slightly based on the evaluation carried out in each iteration. Implementation has been recorded through the use of blog entries. After each iteration an entry was made and elements recorded.

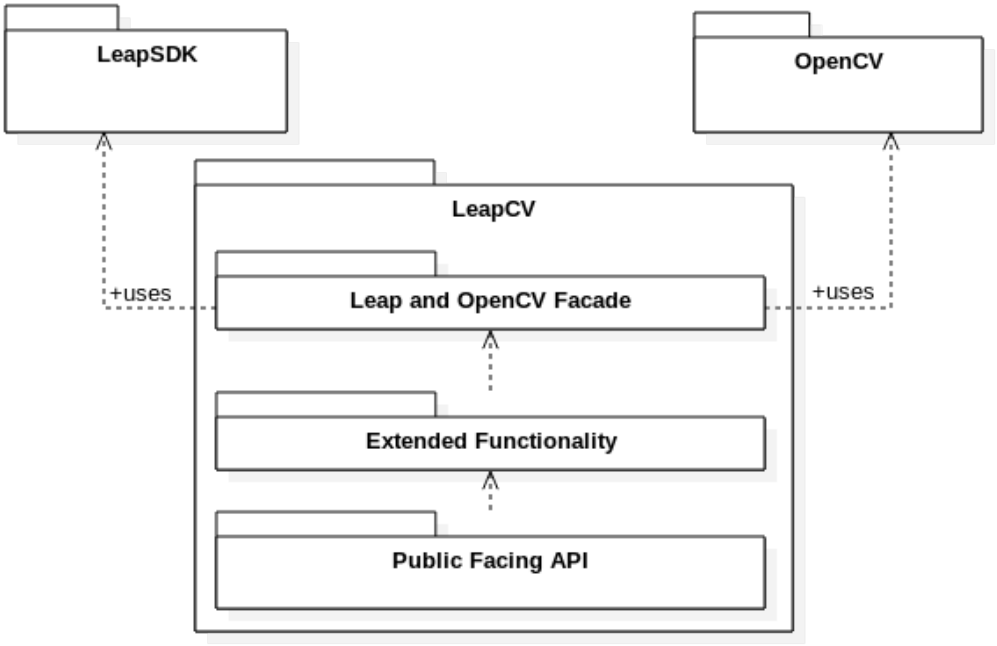

Figure 4.1: High Level System Architecture

4.2 System Architecture

Figure 4.1 outlines the intended architecture of LeapCV.

Leap and OpenCV Facade This will bring the two libraries, being OpenCV and the LeapSDK, together. This is where most of the utility style functionality will sit.

Extended Functionality This is where the functionality of each library will be put to use. Higher level functionality such as classification and object detection will sit here.

Public Facing API LeapCV will provide a public facing API that will ensure that the library is simple to use. All of the complex functionality will be occurring in the private packages.

4.3 Pre-Requisites for Implementation

Before any implementation can take place the environment needs to be set up. The Java 1.8 SDK should be installed on the platform that is being developed on ensuring that it is a version with JavaFX support. IntelliJ IDEA should be installed on the platform that is being developed on. JUnit comes as a standard package with IntelliJ IDEA.

Version 2.4.10 of OpenCV is available from `http://www.opencv.org`. Instructions for installation are also available there. It is important to ensure that OpenCV has been compiled with the Java bindings so the instructions have to be followed carefully. Once installed the library should be imported to the project. Once imported, to ensure the native libraries are loaded when the project is built it is important to include the line of code in listing 4.1.

```
1  import org.opencv.core.Core;
2
3  public static void main(String[] args){
4
5      //  The line below should be placed somewhere within your code, before
6      //  any elements of the OpenCV library are called
7      System.loadLibrary{Core.NATIVE_LIBRARY_NAME};
8
9      //  Implementation continues here...
10 }
```

Listing 4.1: Loading the OpenCV Library into the Project

The LeapSDK is available from `http://developer.leapmotion.com`. The methods for importing the LeapSDK into the project are also found here.

4.4 Iterations

The next few sections will talk in depth about the key work items that were completed to produce the prototype in each iteration. Below is a description of what will be found in each section.

Design will show the design that has been used in the implementation and highlight any key changes through discussion.

Implementation will be split into stages to highlight the thought process that went into developing the library.

Test will describe the unit tests and any key results that might have an effect on further decisions to be made. The tests will be split into:

- **Performance tests** These will be used to test some of the non-functional requirements such as the frame rate. It must be noted that these tests are specification dependant and will therefore vary machine to machine. These tests will only be run on the basic specification MacBook Pro 13inch Version 8,1. An example of a performance test written in JUnit can be seen as a listing in appendix G.2.

- **Acceptance tests** These are previously defined tests, based on the use case analysis. They will provide a measure as to how far the solution has gone to meeting the requirements. These will be performed using either a JUnit test case, or through implementation of the test bench application. The criteria are defined in appendix B.

Unit tests will be performed using JUnit and any significant failures will be highlighted throughout the discussion. The unit test cases will be created based on each individual use case. They will also be used to test classes as they are implemented. They will not be explicitly documented at the end of each iteration due to time constraints. However an example can be seen in appendix G.3.

Evaluation will review how that iteration went and what work needs to be completed or changes made before moving into the next one. It will be broken down into:

- What went well?

- What problems were faced?

- Lessons learned

4.5 Iteration 1

4.5.1 Design

The first implementation will be based on the requirements that were discovered in chapter 3. This iteration is a throwaway prototype, with the idea of exploring the functionality of the libraries, so as to elicit more requirements. These will be discussed in the evaluation section later in iteration 1. The knowledge gained through this prototype will then be applied to the first of the evolutionary prototypes to be developed in iteration 2.

4.5.2 Implementation

Stage 1 - Retrieve Images From the Leap Motion Controller

The first stage involves gaining an understanding of what type of images can be received from the LMC. The type `Image`, in package `com.leapmotion.leap`, has some useful publicly visible methods at its disposal. `Image.data()` returns a `byte[]` where each individual `byte` is a pixel in the image. Each `byte` holds an `8-bit` intensity value for the pixel. In integer form this is between 0(black) and 255(white). The `byte[]` is of length `Image.width() * Image.height()`.

The example in listing 4.2 shows how the `Image` is converted into a `BufferedImage`. Now that this is understood the first image can be retrieved from the LMC. A `Controller` must be initialised and from this we can retrieve a `Frame`. The `Frame` contains an `ImageList` which contains an `Image` for both the left and the right camera. To ensure the images can be retrieved from the camera the `PolicyFlag.POLICY_IMAGES` must be set on the `Controller` object.

Listing 4.3 is a simplified version of writing the images to two files. It makes use of the converter defined in listing 4.2. An example of one of these images is

shown in figure 4.2. The chessboard used in the examples is from the OpenCV documentation[1]. This is used in an attempt to show the distortion of the camera lens.

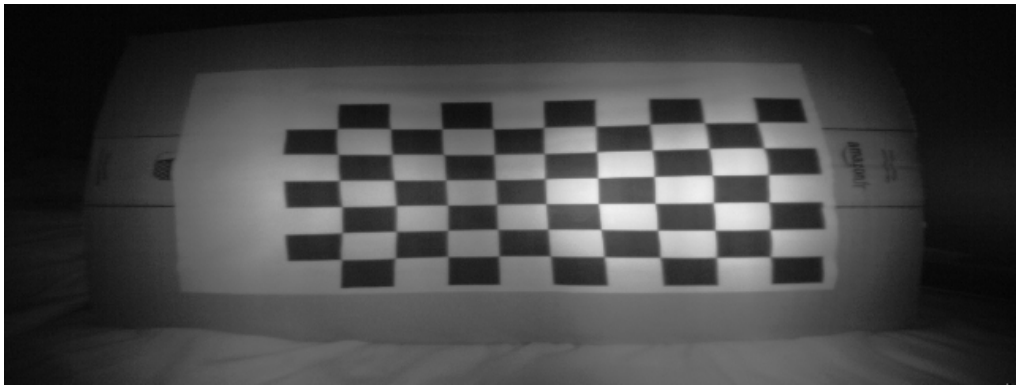

Figure 4.2: Stage 1 Camera Image - Distortion Present

```
1  public static BufferedImage toBufferedImage(Image image) {
2      int type = BufferedImage.TYPE_BYTE_GRAY;
3      byte[] imagePixels;
4
5      // Get all the pixels
6      imagePixels = image.data();
7
8      // Write all of the pixels to the buffer in a new BufferedImage
9      BufferedImage bufferedImage = new BufferedImage(image.width(),
           image.height(), type);
10     final byte[] bufferedImagePixels = ((DataBufferByte)
           bufferedImage.getRaster().getDataBuffer()).getData();
11     System.arraycopy(imagePixels, 0, bufferedImagePixels, 0,
           imagePixels.length);
12     return bufferedImage;
13 }
```

Listing 4.2: Convert an `Image` *into a* `BufferedImage`

[1] http://docs.opencv.org/_downloads/pattern.png

```
1  // Initialise the Controller
2  Controller controller = new Controller();
3
4  // Set the policy to retrieve images from the camera
5  controller.setPolicyFlag(PolicyFlag.POLICY_IMAGES);
6
7  // Get ImageList from the controller
8  ImageList images = controller.images();
9
10 // Convert both images and write to a file
11 int imageCount = 0;
12 for(Image image : images){
13         BufferedImage bufferedImage = toBufferedImage(image);
14         File outputImage = new File("image" + imageCount + ".png");
15         ImageIO.write(image, "png", outputImage);
16 }
```

Listing 4.3: Write `ImageList` *to a* `File`

Stage 2 - Distortion Removal

Now that the `Images` can be successfully taken from the LMC, the distortion must be removed. The LeapSDK provides the `Image.warp()` method for this. However the documentation states that it is not suitable for processing full images. It recommends that using a shader program with OpenGL is faster as this would allow the process to run on a GPU. However at this stage it is more interesting (and valuable to proving the hypothesis) to see whether a disparity map can be produced using OpenCV. Therefore, for now, the slow method is used. The listing in appendix G.1 shows the method by which this was done. The method results in an image as shown in figure 4.3. Unfortunately it seems that during the process of distortion removal, some of the resolution of the `Image` has been lost. We can see this from the black pixels that are left around the outside of the `BufferedImage` that is shown here. This is still a 640×240 resolution image as was seen in figure 4.2. There is a possibility this will have a detrimental effect on the use of the images for computer vision.

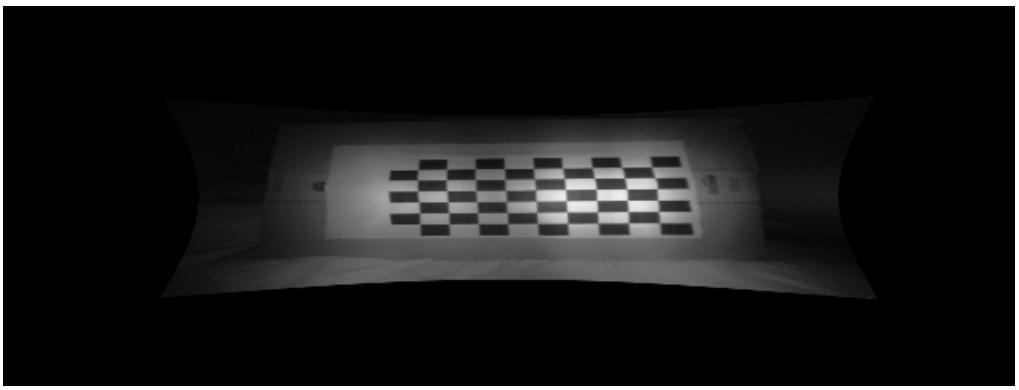

Figure 4.3: Stage 2 Camera Image - Distortion Removed

Stage 3 - Further Removal of Distortion with OpenCV

OpenCV provides functionality to resize and crop images. Therefore we shall explore further removal of distortion using OpenCV. In OpenCV, the type `Mat`, in package `org.opencv.core`, is described as "The Basic Image Container" in the documentation[2]. The `Mat` is a matrix, with the same width and height of the image. Each location in the index holds an intensity value for the pixel. Fortunately, as previously seen, the LeapSDK provides a `byte[]` of pixel values. The images are also grey scale, meaning that they are single channel. Therefore there is less complexity in the storage of the image in a `Mat` due to there being only a single integer between the values 0 and 255 representing a pixel in a grey scale image. In comparison, a colour image has three (sometimes four if including alpha) channels of colour at any pixel location. These channels being red, green and blue (RGB), each one with a value usually between 0 and 255, depending on whether it is a type that uses a `32bit` floating point representation. It is actually fortunate that the images are already grey scale, as it seems that algorithms used in feature detection favour the use of grey scale images. Listing 4.4 shows the method used to convert the `Image` into a `Mat`.

[2] `http://docs.opencv.org`

```
1  public static Mat convertToMat(Image image){
2      // Initialise the Mat with the height and width of the image
3      // CvType.CV_8UC1 is single channel unsigned 8 bit.
4      Mat convertedImage = new Mat(image.height(), image.width(),
           CvType.CV_8UC1);
5
6      // Put the image data into the Mat starting at pixel x = 0, y = 0.
7      convertedImage.put(0, 0, image.data());
8
9      return convertedImage;
10 }
```

Listing 4.4: Convert `Image` *type to* `Mat` *type*

Now that the image is in a `Mat` format it can be resized. This is done using the `resize()` method provided by the class `Imgproc` in the package `org.opencv.imgproc`. `Imgproc` is the class that handles the majority of the image processing functionality within OpenCV. The crop is performed by passing a region of interest to the method `submat()` provided by the `Mat` to be cropped, then storing the returned value in a new `Mat`. The region of interest is a `Rect` object which is a rectangle that consists of a width, a height and a pair of coordinates, at which the top left of the rectangle should begin. The code for producing a cropped image can be found in listing 4.6. The end result is an image that looks like that in figure 4.4.

One is able to see the effect further by highlighting the chessboard corners in the image. This is done using the `findChessboardCorners()` method in the `org.opencv.calib3d` package. The results can be seen in figures 4.5 and 4.6. The images show a curvature of the straight edges in figure 4.5 and a straightening of the edges in figure 4.6. The difference between the two appearing much like the effect shown in figure 2.1 that was discussed in chapter 2.

A point to note here is the discovery that the Java implementation of OpenCV does not have a method for displaying images to the screen, directly from the code. Usually this would be carried out using the `imshow()` method in the `org.opencv.Highui` package within the C++ and Python implementations. This was not foreseen under the assumption that the Java implementation had all of

the functionality of the C++ library. Therefore it was also required to develop a converter that converted the `Mat` format images back to a to a `Image` from the `javafx.scene.image` package. The code can be seen in listing 4.5. The absolute reference to this `Image` type is used to resolve the ambiguity between the JavaFX and the LeapSDK classes of the same name. This allows the images to be used with the JavaFX GUI framework that will be discussed later.

```
1  /**
2   * Turn a {@link Mat} into a {@link javafx.scene.image.Image}, useful for
        displaying in JavaFX
3   */
4  public static javafx.scene.image.Image matToWritableImage(Mat image) {
5      MatOfByte byteMat = new MatOfByte();
6      Highgui.imencode(".bmp", image, byteMat);
7
8      return new javafx.scene.image.Image(new
           ByteArrayInputStream(byteMat.toArray()));
9  }
```

Listing 4.5: Convert `Mat` *type to* `javafx.scene.image.Image` *type*

```
1   /**
2    * Crops even percentage from each side of an image
3    *
4    * @param image The {@link com.leapmotion.leap.Image} to be cropped.
5    * @param percentageCrop The percentage to which the image passed in
         should be cropped. Between 0 and 1.
6    * @return {@link org.opencv.core.Mat}
7    */
8   public static Mat crop(Mat image, double percentageCrop) {
9       int width;
10      int height;
11      int x;
12      int y;
13      Mat newImage = image;
14
15      //  Get the width and height of the image
16      width = newImage.cols();
17      height = newImage.rows();
18
19      //  Work out top left coordinates of submatrix
20      x = (int) (width * percentageCrop);
21      y = (int) (height * percentageCrop);
22
23      //  Work out the width and height of the submatrix
24      width = width - x * 2;
25      height = height - y * 2;
26
27      //  Set the rectangle of interest
28      Rect roi = new Rect(x, y, width, height);
29
30      //  Return the submatrix
31      return newImage.submat(roi);
32  }
```

Listing 4.6: Crop `Mat` *Image*

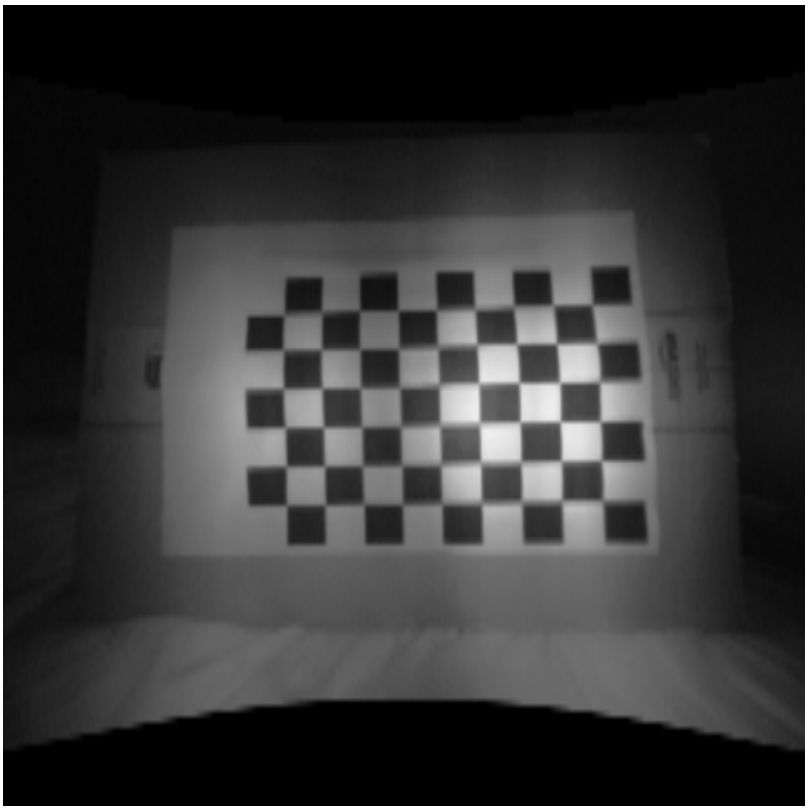

Figure 4.4: Stage 3 Camera Image - Distortion Further Removed with OpenCV

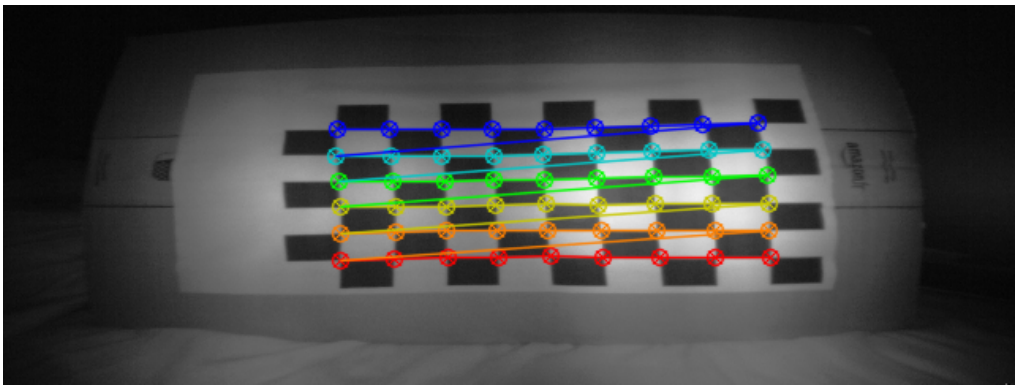

Figure 4.5: Camera Image - Distortion Present, with Highlighted Corners on a 9x6 Chessboard

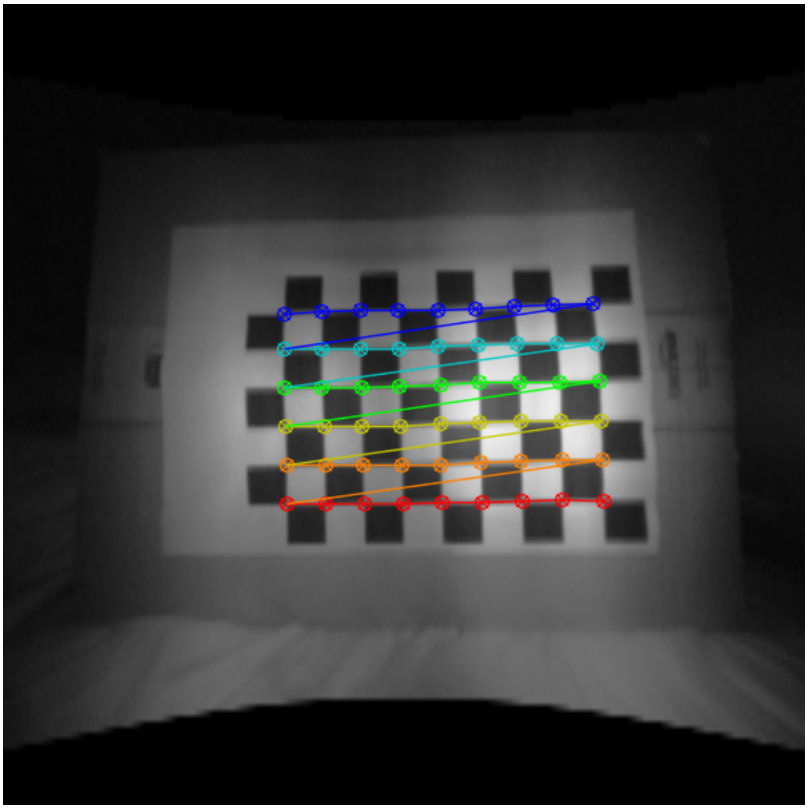

Figure 4.6: Camera Image - Distortion Further Removed with Highlighted Corners on a 9x6 Chessboard

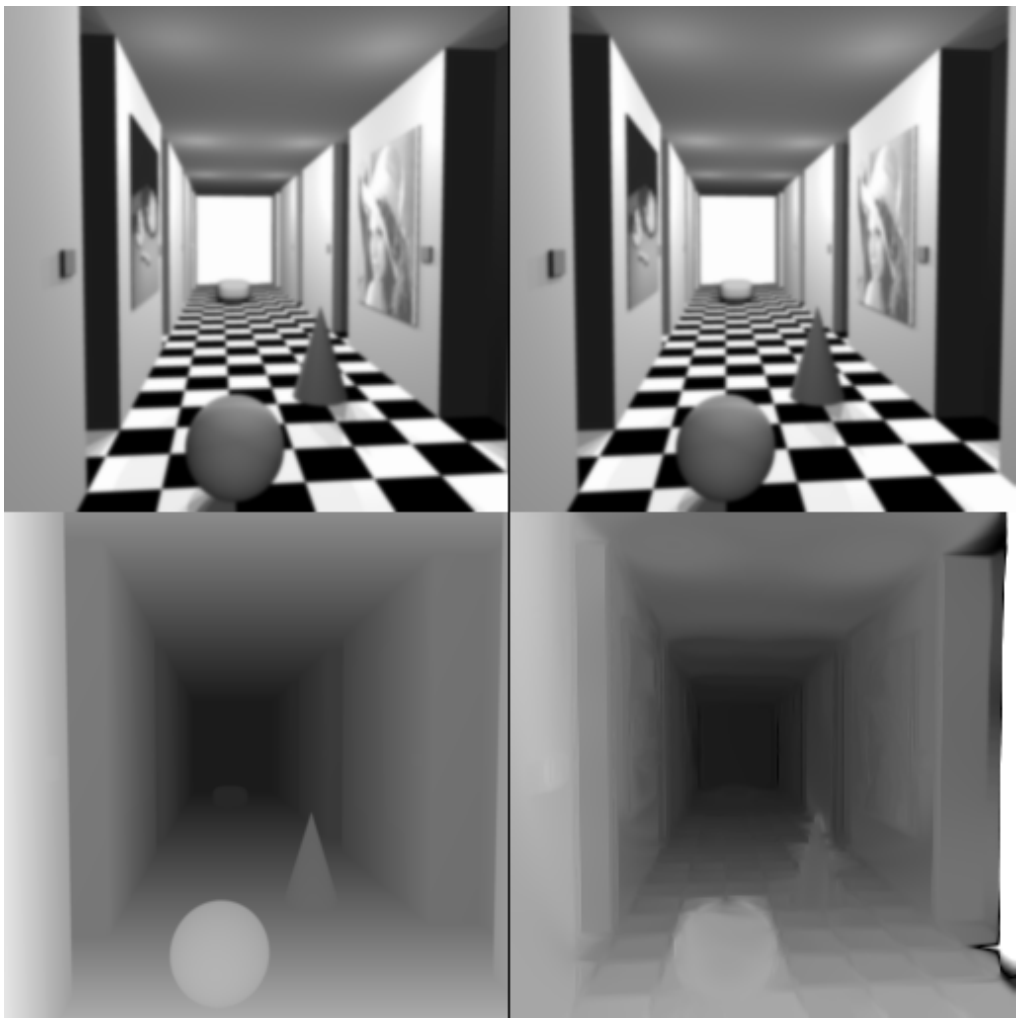

Figure 4.7: Example Disparity (Bottom Left and Right) of Synthetic Stereo Pair (Top Left and Right) (Alvarez et al. 2000)

Stage 4 - Stereo Correspondence

Stereo correspondence was briefly discussed in chapter 2. It is a technique used to match (find the correspondence) the pixels in a pair of images. The result is a grey scale image that contains a corresponding depth value at each pixel. A value of 255(white) means the area of the image is closer to the cameras. A value of 0(black) means the area is futher away. An example of a disparity map can be seen in figure 4.7.

OpenCV provides stereo correspondence functionality in the `org.opencv.calib3d`

and the `org.opencv.contrib` packages. Namely `StereoBM`, `StereoSGBM` and `StereoVar`. The first two are variations of the stereo block matching algorithms and the later is the stereo variation algorithm. `StereoSGBM` seems to be generally accepted as better than `StereoBM` in most scenarios, therefore `StereoBM` will not be attempted here.

StereoSGBM stands for semi-global block matching, an algorithm that was researched by Hirschmuller (2008). StereoVar stands for stereo variational methods, which is an algorithm produced by Kosov et al. (2009). To achieve a basic disparity map is relatively simple, the difficulty is in the parameter tweaking. Listing 4.7 shows the code for producing a disparity map with StereoSGBM.

```
1  public Mat getDisparityMapStereoSGBM(Mat left, Mat right) {
2          //  Initialise new stereo matcher
3      StereoSGBM stereo = new StereoSGBM();
4      Mat disparityMap = Mat.zeros(0, 0, type);
5
6      //  Computer disparity
7      stereo.compute(left, right, disparityMap);
8      //  Normalise output as recommended, creates a larger visual difference
9      //  in diaparities.
10     Core.normalize(disparityMap, disparityMap, 0, 255, Core.NORM_MINMAX);
11
12     return disparityMap;
13 }
```

Listing 4.7: Initialisation of Distortion Data

Stereo matching algorithms inherit from the same interface, therefore many of the methods are the same, but the type of the `stereo` variable would be changed when using a different algorithm.

Stage 5 - Parameter Tweaking

Through the exploration of the algorithms in stage 4, it became clear that it was quite difficult to know what parameters to set to get a good disparity map from the image. Therefore, as part of the prototyping methodology, a test bench graphical user interface (GUI) application will be made in JavaFX. The GUI will

not only be able to allow for parameter changing at run-time, but it will also give a good idea as to the performance that can be achieved with each of the stereo matching algorithms, through a visual analysis of the output.

JavaFX is a GUI framework for Java. It is shipped with Java 1.8 SDK and has been developed to take the place of the Java Swing framework. For speed of development, SceneBuilder will be used. SceneBuilder is a tool that allows for rapid development of GUIs. It allows for the drag and drop of GUI elements such as sliders onto the page and generates the underlying code. An example of the GUI that was quickly put together is in figure 4.8. Another example of it in use is in figure 4.9. The sliders visible are used to quickly change the stereo parameters. The labels here correspond to the parameter values for the StereoSGBM method. To allow for the processing to be done in real time, it was decided to make use of the `Listener` class within the LeapSDK, then implementing an `ObservableValue`, which is included in the JavaFX framework. The `ObservableValue` follows the observer design pattern, allowing the `Listener` to notify the main GUI thread when there is some new data for it to display. The reason for doing this is so the `Listener` could be run on a separate thread to the JavaFX GUI. It is also important to allow the GUI thread to handle when to instantiate the `Listener`, using the `Platform.runLater()` method; the `Listener` appears to lock the GUI thread if it is not done this way. This is possibly due to the fact that, by default, the LMC returns images at a very high frame rate. This resulted in finding good parameters for the correspondence algorithms.

4.5.3 Test

As this iterations prototype is intended to be thrown away, it must be noted that the testing was not intended. `StereoVar` gave the best resulting disparity map. However, it took a lot longer to process. `StereoSGBM` gave the worst resulting disparity map but the processing time was a lot faster. However, consideration needs to be made toward the fact that the slow method of distortion removal was implemented here.

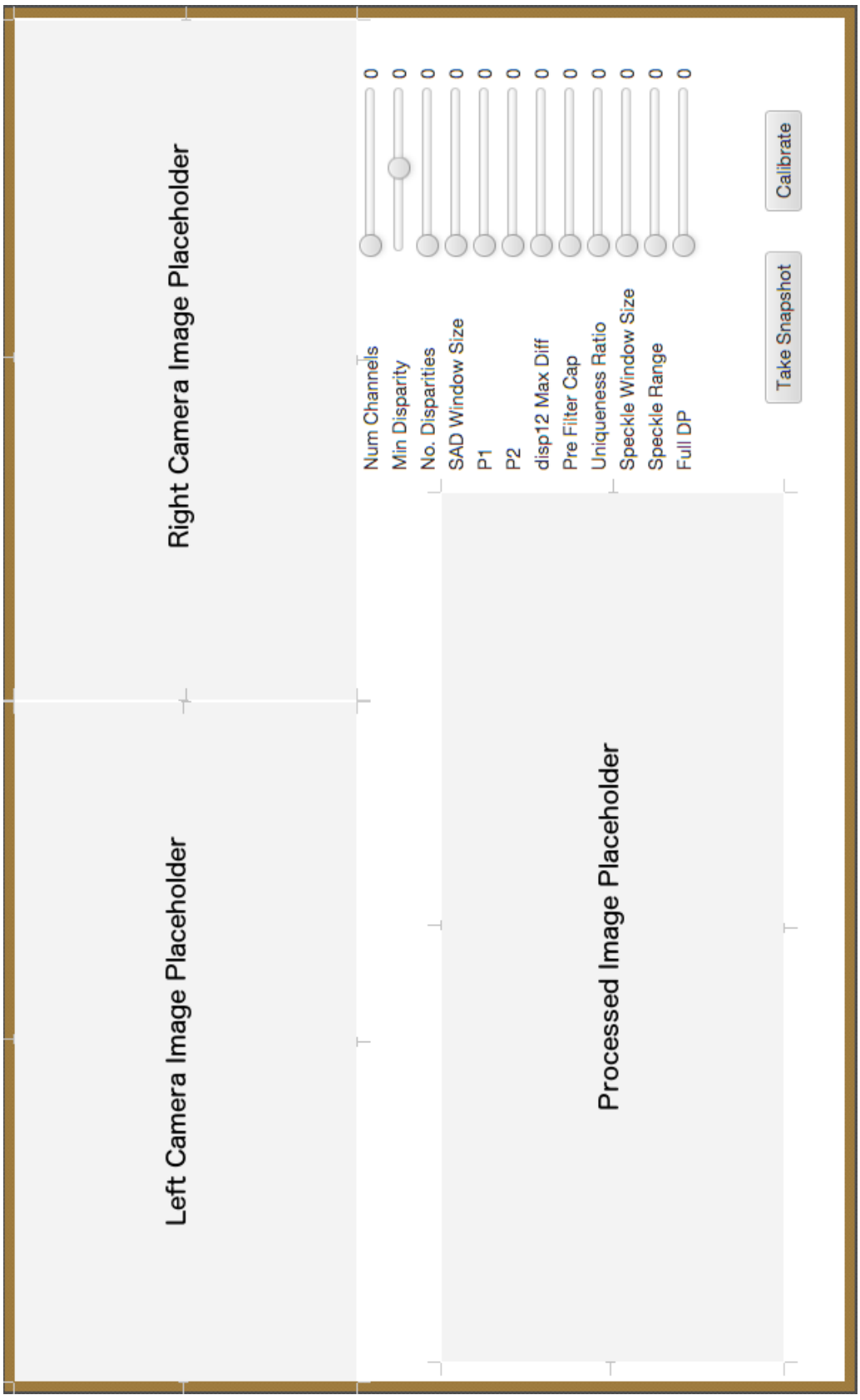

Figure 4.8: GUI Created in SceneBuilder 2.0

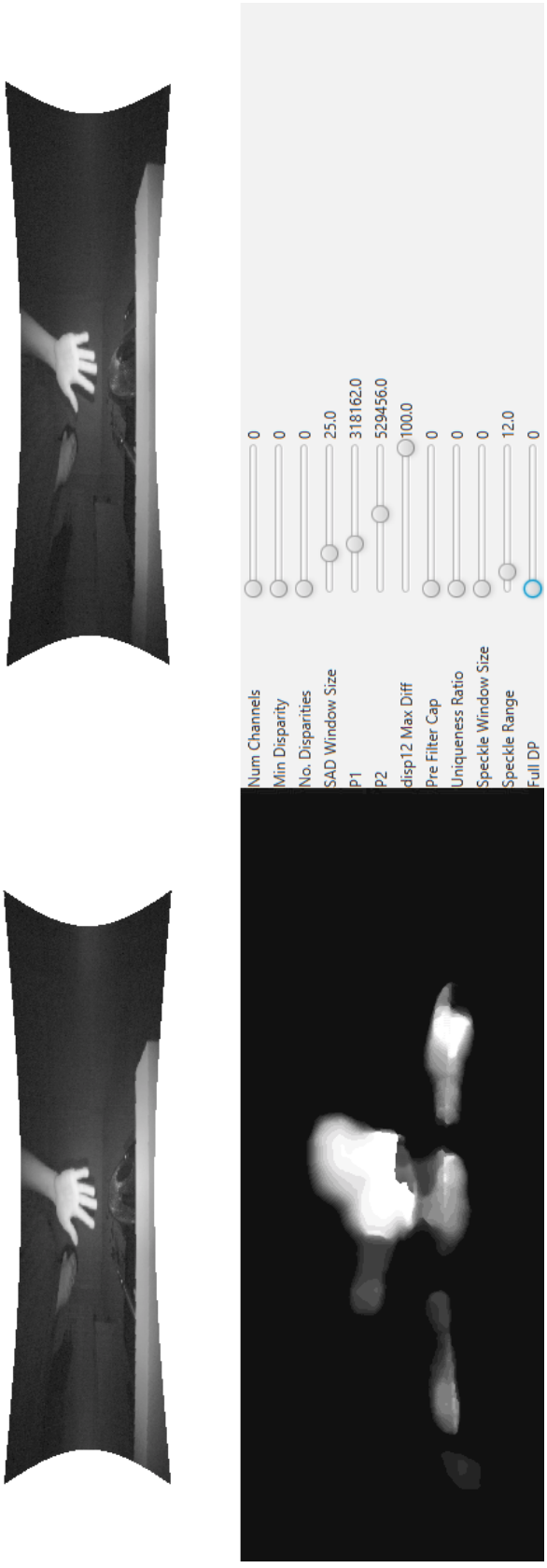

Figure 4.9: GUI Being Used to Tweak Parameters

4.5.4 Evaluation

What went well?

- The first prototype allowed for a good exploration of the OpenCV library.
- A vast amount of knowledge was gained as to how the library works, all of which will be useful going into the next iteration.

What problems were faced?

- Using the `Listener` class to access the LMC frames made the code a lot more complex to work with.
- The weighing up of priority in the optimisation of removing camera distortion. The slow method of removal was used in this iteration.
- Due to the slow frame rate, it was very difficult to achieve optimum results in the disparity map.

Lessons learned?

- The `Listener` class is designed for applications that do not have a "natural framerate". It is believed that it would be a better alternative to access the images explicitly through a method call. In turn making the system more stateful so that the application developer knows the exact stage of processing that an image is at.
- After development was carried out on the distortion removal, it was in fact discovered that the `warp()` method of distortion removal was indeed very slow and had a detrimental effect on the frame rate. Going into the next iteration, an attempt should be made at optimising the process.
- A higher frame rate should allow for further parameter tweaking in the next iteration.

4.6 Iteration 2

4.6.1 Design

Class Diagram

The class diagram shown in figure 4.10 is a basic model of LeapCV. Each class was identified using the CRC process (Sa n.d.). CRC stands for Classes Responsibilities and Collaborations. The cards contain three sections of information about a single class. These are:

- **Class:** The name of the class.
- **Responsibilities:** The information the class owns and the functionality it performs.
- **Collaborations:** The classes that this class also interacts with.

Class Name	`LeapCVController`
Responsibilities	• Interfacing with the leap motion. • Initialising the leap motion. • Retrieving image frames from the leap motion. • Base class for any interaction with the leap motion.
Collaborations	• `LeapCVCamera`

Table 4.1: `LeapCVController` *CRC*

The CRC for each class can be found in appendix C. From the CRC responsibilities one is able to infer method names on the classes. These will be created throughout implementation.

With reference to Figure 4.10, the `Utilities` package will hold most of the extended functionality. This functionality is not dependent on any particular class in the model. It is assumed that any image of any size can be passed into the utilities and be processed. The classes on the left hand side are a model of

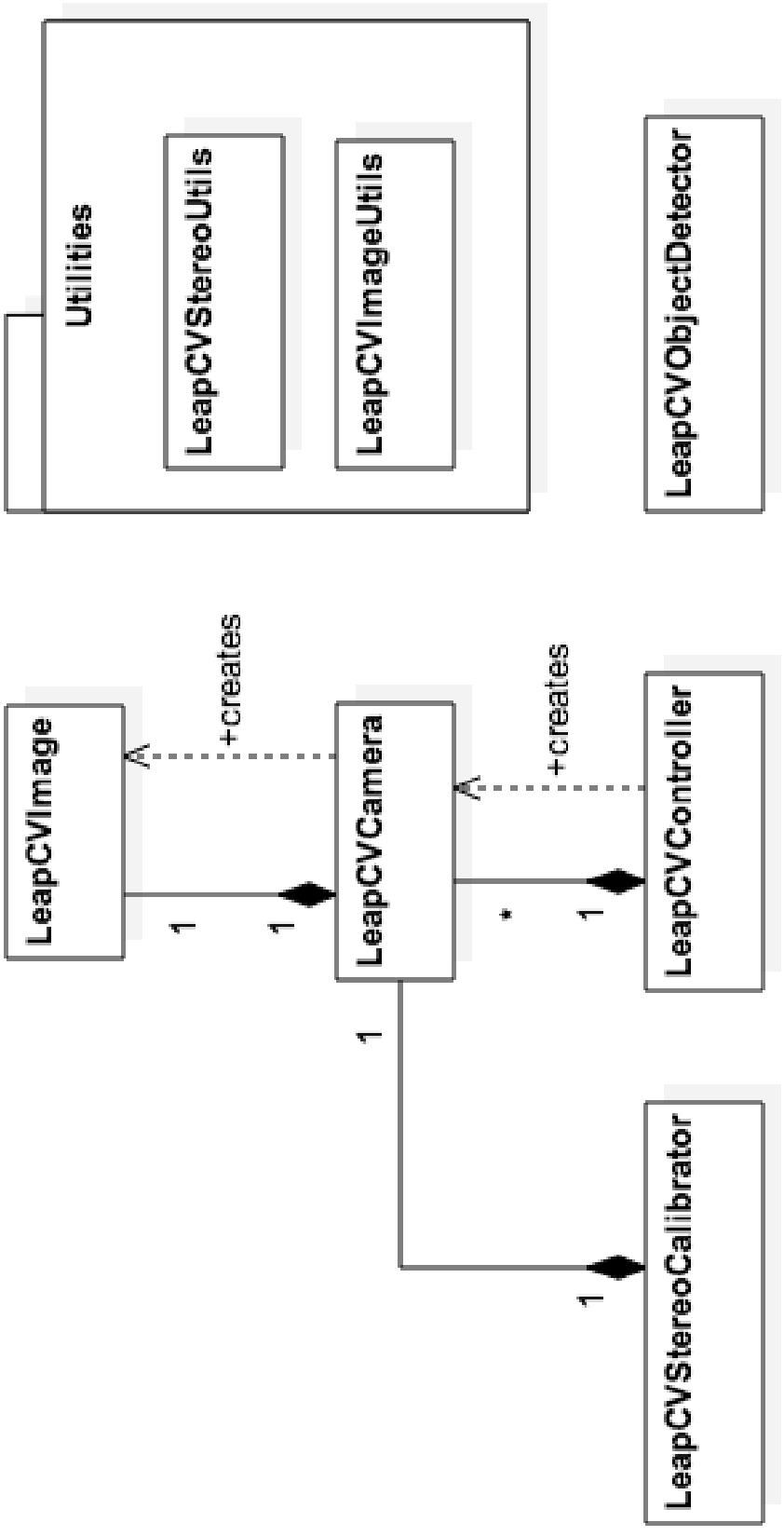

Figure 4.10: Class Diagram for Prototype 2

the LMC. Aiming to abstract it in a simple way, so retrieving information is simple. The LeapSDK does this in a way that is suited more toward tracking information, whereas here it is required to focus it more towards imaging. The `LeapCVCamera` class and `LeapCVImage` class have been included to make it this way. The model is stateful, as in, it holds one image at any one point in time, up until the application developer explicitly requests for the camera to move on to another valid frame through a method call to the controller. Note that this was decided due to the results from the previous iteration. Having a `Listener` deemed to make the library more complicated than intended.

Interaction Diagrams

Once a high level class diagram was implemented a series of interaction diagrams were put in place as an attempt to identify the methods that should exist in each class. An example of a sequence diagram can be seen in figure 4.11.

Requirements Changes

Now that much more knowledge has been gained surrounding the OpenCV library, it has been decided a few changes will be made to the requirements of LeapCV. To simplify the use of LeapCV, and keep the project within scope, it has been decided that the image manipulation functionality will be handled by LeapCV. This means that the application developer can request either the original image, or the undistorted image that LeapCV has defined. This will avoid complications in having to handle differently sized images. It will also keep LeapCV within a performance boundary. Block matching algorithms work faster on lower resolution images due to a linear complexity increase in correlation to the resolution; if an application developer resized an image to become larger than it was originally, then requested a disparity map, the frame rate will decrease. The new use case diagram reflecting these changes can be seen in figure 4.12.

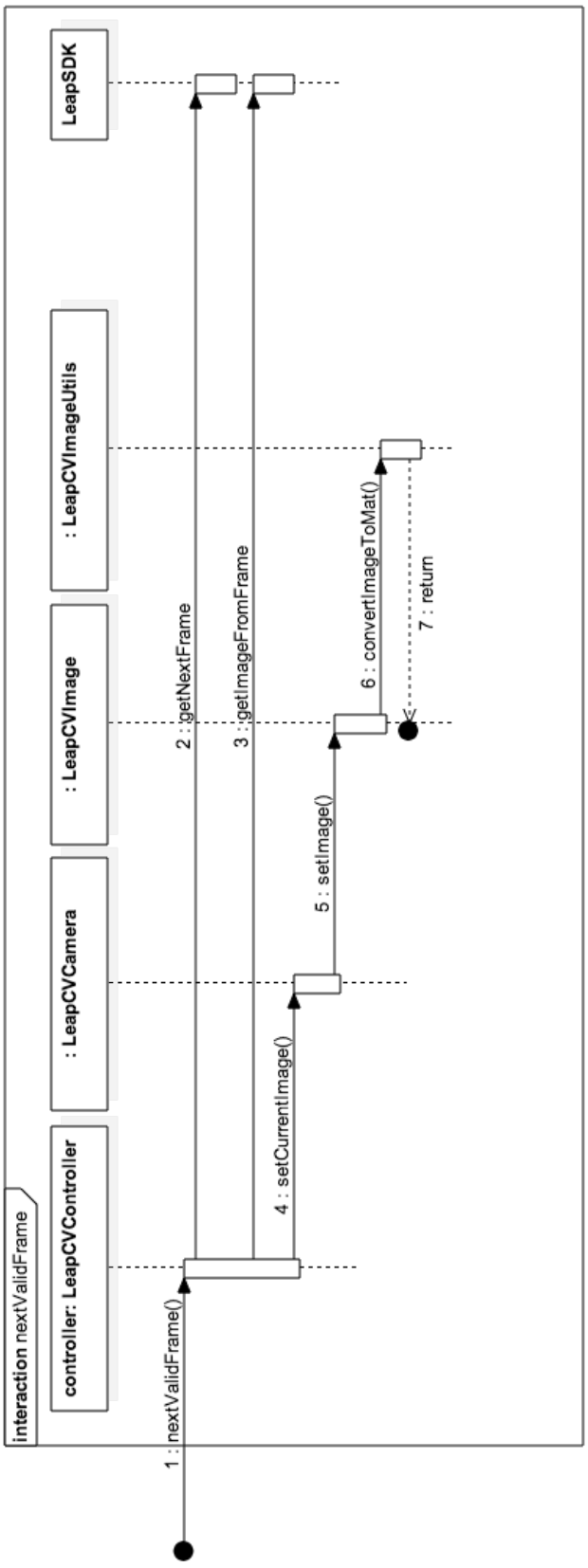

Figure 4.11: `nextValidFrame` *Sequence Diagram*

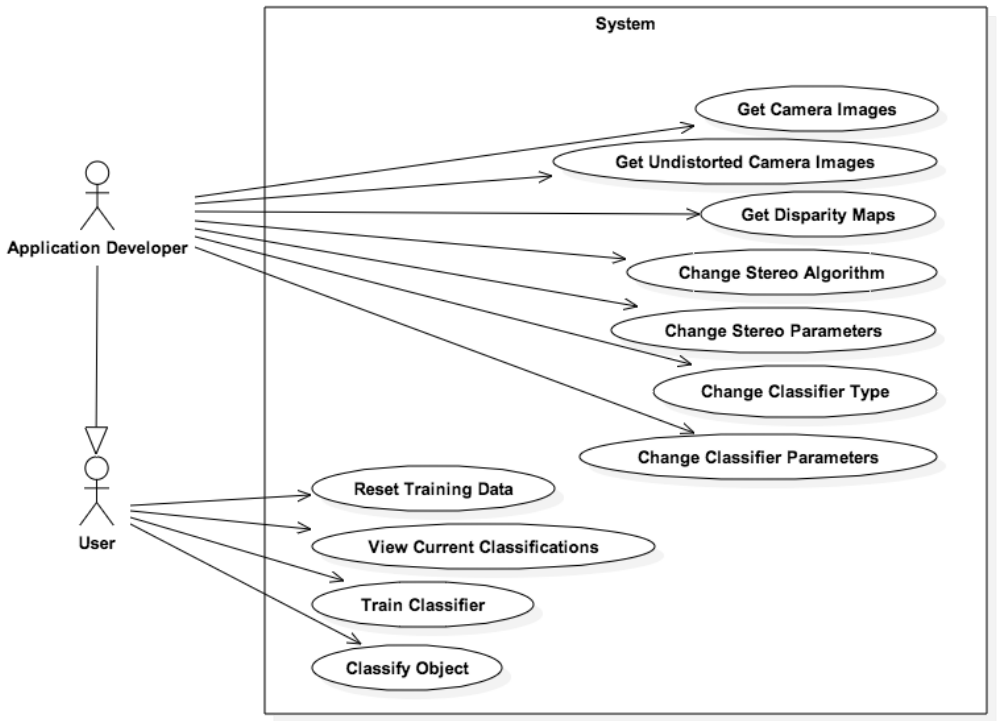

Figure 4.12: New UML Use Case Diagram

4.6.2 Implementation

Stage 1 - The Basis of LeapCV

As LeapCV is designed to be extended upon by other application developers, it is useful to think about the way in which it will be deployed. LeapCV should be IDE independent. The application developers using it should be able to easily import LeapCV into their own project. All they will need is the source code being developed here, and the OpenCV and LeapSDK dependencies to be satisfied. To do this a Git repository will be made that holds the source for LeapCV. The Git repository will have nothing but Java related files. None of the files should be related to any individual IDE.

Stage 2 - Refactoring Based on New Design

Now that the design has been created, it will be useful to create the templates for each class, also known as the skeleton classes. It is also the case that some of the functionality from iteration 1 can be transferred into the new utilities classes, primarily the segments of code that were used to manipulate and convert the images. This functionality will be heavily tested within this iteration, providing a reliable foundation on which to build the rest of the system.

Stage 3 - Distortion Removal Optimisation

As highlighted at the end of the previous iteration, it was suggested that during this iteration the distortion removal process should be optimised. The method that is currently used, calls the `warp()` method from the LeapSDK `Image` class. This method is, through self admittance by Leap Motion, a non optimised way to remove image distortion. They suggest that only a few image pixels should be used. However, in this scenario it is required that the entire picture gets used, so that it can be further processed in OpenCV. The `Image` class also has a `distortion()` method. This method contains the calibration map for the LMC. Containing values that are required to remove the distortion of each lens using a GPU shader (used to render light and special effects in images). It was suggested by Ma (2014) that these values can be used with the `remap()` function in OpenCV. Ma (2014) does state that there is no guarantee that this method is correct. With this method, the distortion map is loaded at the initialisation of the LMC. This data is set in the factory, or when the LMC is explicitly calibrated within its separate settings application. Unless the LMC is going through the calibration process, this distortion map does not change. Once the distortion map is loaded at initialisation, it can be passed with a `Mat`, to the `remap()` method. The code provided by Ma (2014) is written in the Python programming language, so analysis is required to port it over to Java. The method provided by OpenCV is highly optimised with the library, and the processing time has been noticeably

decreased. Listing 4.8 shows the method by which the distortion `Mat` has been initialised. The values that get returned are placed into the `LeapCVCamera` class. This method is quite similar to the code in listing G.1. It still uses the `warp()` method, however, this slow method is only used once throughout the run time of LeapCV, rather than every time a frame is loaded as in iteration 1.

This was not the only method of distortion removal that was created in an attempt to optimise the performance. The Leap Motion documentation also stated that faster distortion removal could be achieved through a method called bi-linear interpolation[3]. This is used to calculate intensity values to pixels within a transformed image. The code for this can be found in listing G.7. The method used here was converted from C++ code into Java, originally written by Lorenzo (2014). The method provided no benefit over using OpenCV. OpenCV already uses bi-linear interpolation to carry out the transformations. It is also highly optimised and faster at computing the transformation. It is a much more understood method with much more maintainable and readable code. Therefore the OpenCV method was kept.

[3]For an insight into this it would be useful to at least read the wikipedia article as it gives a good overview on how it works `http://en.wikipedia.org/wiki/Bilinear_interpolation`

```
1  /**
2       * Initialise the distortion matrices for use with OpenCV {@link
            Imgproc.remap} method.
3       */
4      public static Map<String, Mat> initDistortionMat(Image image) {
5
6          Mat tempX = new Mat();
7          Mat tempY = new Mat();
8
9          tempX = Mat.ones(image.height(), image.width(), CvType.CV_32F);
10         tempY = Mat.ones(image.height(), image.width(), CvType.CV_32F);
11
12         // Get distortion value for each pixel in the image
13         for (int y = 0; y < image.height(); ++y) {
14             for (int x = 0; x < image.width(); ++x) {
15                 Vector input = new Vector((float) x / image.width(), (float) y
                       / image.height(), 0);
16                 input.setX((input.getX() - image.rayOffsetX()) /
                       image.rayScaleX());
17                 input.setY((input.getY() - image.rayOffsetY()) /
                       image.rayScaleY());
18                 Vector pixel = image.warp(input);
19                 if (pixel.getX() >= 0 && pixel.getX() < image.width() &&
                       pixel.getY() >= 0 && pixel.getY() < image.height()) {
20                     tempX.put(y, x, pixel.getX());
21                     tempY.put(y, x, pixel.getY());
22                 } else {
23                     tempX.put(y, x, -1);
24                     tempY.put(y, x, -1);
25                 }
26
27             }
28         }
29
30         Map<String, Mat> retVal = new HashMap<>();
31         retVal.put(X_MAT_KEY, tempX);
32         retVal.put(Y_MAT_KEY, tempY);
33
34         return retVal;
35     }
```

Listing 4.8: Initialisation of Distortion Data

4.6.3 Test

Performance Tests

Test	Minimum Value	Actual Value	Pass
getImage	15 FPS	1112 FPS	Yes
getImageUndistorted	15 FPS	144 FPS	Yes
getDisparityStereoVar	15 FPS	11 FPS	No
getDisparityStereoBM	15 FPS	87 FPS	Yes
getDisparityStereoSGBM	15 FPS	42 FPS	Yes

Table 4.2: Results of First Frame Rate Performance Test (FPS = Frames Per Second)

Notes on table 4.2:

getImage The value in a single test varied between 800 - 1500 FPS. Over 20 tests it averaged out. This fluctuation is most likely being caused by the operating system allocating CPU priority. It is believed that this process would be bottlenecked more if the images were actually being displayed in the test. Therefore stabilising the values.

getDisparityStereoVar This was below the minimum required value, however it produces the best disparity map. This test can be passed, but not without sacrificing disparity map quality. This should be highlighted in the documentation.

getDisparityStereoSGBM It was noted that after 10 consecutive runs of this process, the frame rate dropped dramatically. It was discovered this was due to a memory leak. Each time the method `getDisparityStereoSGBM()` was called, a new StereoSGBM object was instantiated within the method scope. From the frame rate measurement we can assume that this happens 42 times per second and is therefore processor intensive. This was not leaving enough time or resource for the Java garbage collector to free the used memory. Although the test did not fail, this needs to be optimised as it causes the machine to become unresponsive.

Acceptance Tests

The acceptance tests have been checked against the acceptance criteria that were earlier defined in appendix B.

Test Reference	Pass	Notes
1	Yes	
2	Yes	
3	Yes	
4	Yes	
5	No	Not yet implemented
6	No	Not yet implemented
7	No	Not yet implemented
8	No	Not yet implemented
9	No	Not yet implemented
10	No	Not yet implemented

Table 4.3: Acceptance Tests Iteration 2

4.6.4 Evaluation

What went well?

- The optimisation of distortion removal was successful.
- Refactoring made LeapCV a lot easier to use and develop.
- As hoped for this iteration, most of the image processing and stereo functionality was completed.
- Creating a Git repository was easily done and will allow for a simple release of LeapCV.

What problems were faced?

- A memory leak was found in a performance test. The means that a redesign will be required in the next iteration.
- More parameter tweaking was required for generating disparity maps after performance testing to try and improve the frame rates.

- Whilst lots of implementation was completed in this stage, development was a lot slower than anticipated.

Lessons learned?

- Due to the problems found in testing, it means some of the work will be carried over into the next iteration.
- There will be a possible set back to the implementation of the classification functionality.
- In hindsight, it would have been useful to give the iterations slightly more slack time to cater for these problems.
- It would also have been useful to run the performance tests on the first prototype to have more of an expectation of performance throughout iteration 2.

4.7 Iteration 3

4.7.1 Design

Class Diagram

As a result of the testing and evaluation of the previous iteration, it was decided that a redesign was required to remove a memory issue found in the performance test `getDisparityStereoSGBM()`. A decision was made to implement this following the factory design pattern as defined in Freeman et al. (2004). All methods of stereo matching carry some similar methods. They also all result in a disparity map. The only thing that differs are the parameters that are passed in. Therefore a creator, also known as a factory, will be implemented. This will create a `LeapCVStereoMatcher` object. Abstracting the underlying properties of the stereo matching algorithm away from higher level code (unless an explicit cast is made

on the object to the type requested). Doing this will reduce the memory footprint, as a matcher object will only need to be created once and not each time the method `getDisparitySGBM()` is called.

Another change is to the `LeapCVStereoCalibrator` class. This class has been changed to a utilities class and renamed to `LeapCVCalibrationUtils`. This was decided due to the realisation that the calibration information should be stored with the camera. The additional CRC diagrams can be found in appendix D. The resulting design can be seen in figure 4.13.

Interaction Diagram

Due to the change in class structure, there is a slight difference in the way in which classes will interact with one another. This is reflected in the interaction diagram shown in appendix E.5.

4.7.2 Implementation

Stage 1 - Obtaining the Q Matrix

OpenCV requires the camera's intrinsic parameters be known so that images can have the distortion removed, and be rectified in a stereo set up. By carrying out the calibration, it gives what in OpenCV is known as the `Q` matrix. This is a 3x3 `Mat` that contains the distortion coefficients for the camera. In iteration 1 it was discovered that the images could have the distortion removed by using the distortion values that were provided by the LeapSDK. However, the distortion coefficients are not given by the LeapSDK. This was not realised until this iteration. An attempt was made to extract the distortion coefficients from the camera by doing stereo calibration. The distortion coefficients are the camera intrinsic parameters that were discussed in chapter 2. By carrying out stereo calibration one would expect to see an output image like the one in figure 4.3. However, from the attempts made the only calibrated image that could be

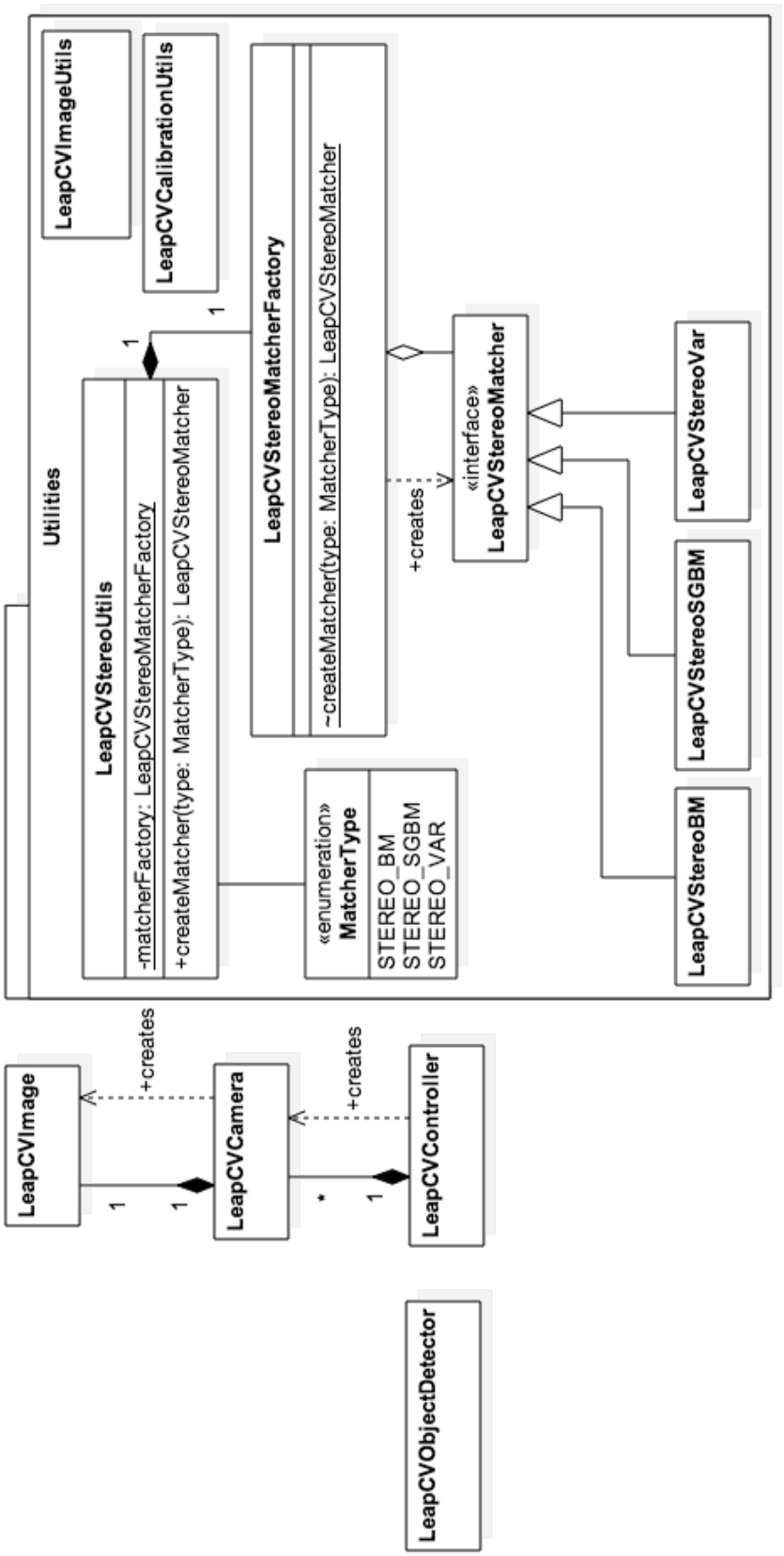

Figure 4.13: Class Diagram for Iteration 3

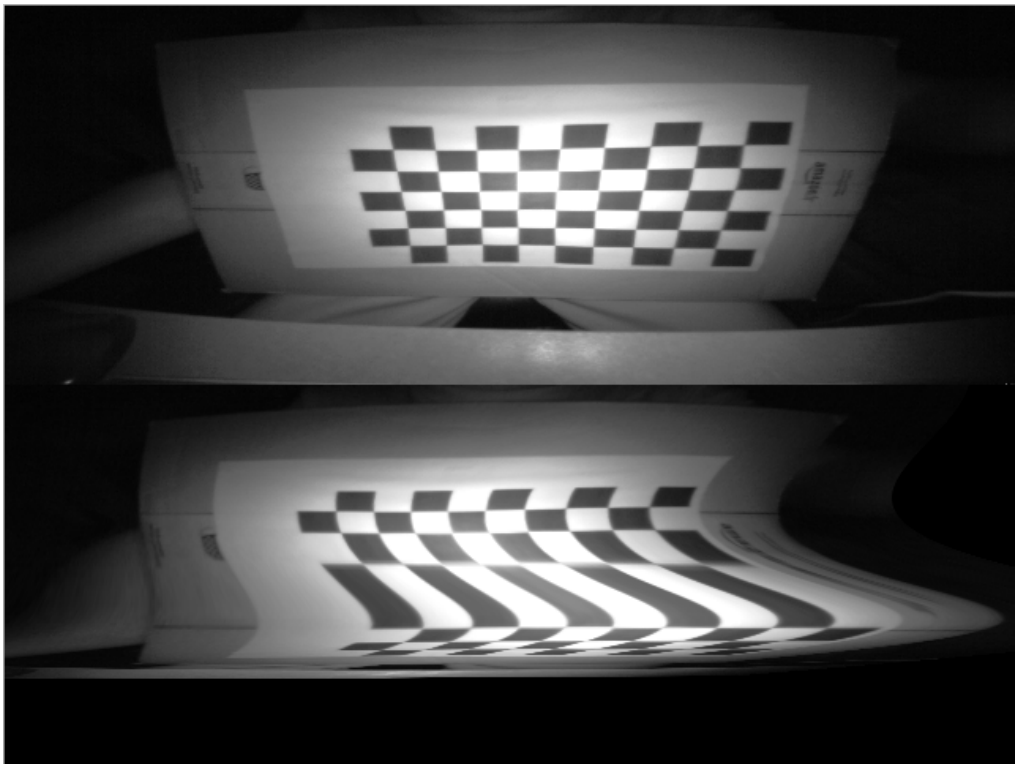

Figure 4.14: Image Showing Failed Calibration (Bottom) of a Chessboard Image (Top)

produced was like the one in figure 4.14. This attempt was unsuccessful and had to be cut short due to the risk of not completing further requirements. The code for this attempt can be found in appendix G.4. Another thought is that the distortion coefficients can be calculated from the already obtained distortion map. A method was outlined on tStackexchange (2014) might prove useful. It gives a mathematical example of how to recover the distortion coefficients from a pre calculated distortion map. This method is quite advanced and it will not be feasible to complete in the time-scale of this project. Ma (2014) does provide some values obtained from calibration of the LMC. However these values will vary slightly from camera to camera due to manufacturing precision as mentioned in chapter 2. It is therefore unlikely that the values will be exact for the LMC in use here, although for brevity these values will be used as the `Q` matrix for this project to see if any form of output can be achieved.

Stage 2 - Three Dimensional Reprojection

The process of three dimensional reprojection is to achieve a model of the real world scene that the images represent. With true distance values that are identical to the real world. This includes depth measurement. To do this with OpenCV the `Q` matrix is needed as described in stage 1. The `Q` matrix consists of parameters that allow the values in the disparity map, generated in the previous two prototypes, to be turned into true depth values. The result is known as a point cloud. The method used to generate the point cloud can be seen in listing 4.9

```
1   public Mat getPointCloud(Mat disparityMap) {
2       //  Initialise distortion coefficients in a Multi Dimensional array
3       //  Values from Ma (2014)
4       double[][] tQ = {{1.0, 0.0, 0.0,           -5.0638e+02},
5                        {0.0, 1.0, 0.0,           -2.3762e+02},
6                        {0.0, 0.0, 0.0,           1.3476e+03},
7                        {0.0, 0.0, 6.9349981e-01, 3.503271}};
8
9       Mat pointCloud = new MatOfPoint3();
10      Mat Q = new Mat(4, 4, CvType.CV_32F);
11
12      //  Put values of array tQ into Mat Q
13      for (int i = 0; i < tQ.length; ++i)
14          Q.put(i, 0, tQ[i]);
15
16      //  Generate point cloud
17      Calib3d.reprojectImageTo3D(disparityMap, pointCloud, Q);
18
19      return pointCloud;
20  }
```

Listing 4.9: Generate a Point Cloud from the Disparity Map

It was found that the point clouds varied greatly in quality based on the algorithm that was used to generate the disparity map. There was no way to test this here other than to visually inspect the output point clouds. The point clouds were written to file using the method in listing 4.10.

```
1   /**
```

```
*  Save a generated point cloud to a file, for opening in CCViewer
*/
public void savePointCloud(Mat pointCloud, File destination) {
        try {
            // open the file for writing to (.obj file)
            File output = destination;
            if (!output.exists())
                output.createNewFile();

            FileOutputStream fileOutputStream = new FileOutputStream(output);
            OutputStreamWriter writer = new
                OutputStreamWriter(fileOutputStream);
            System.out.println(pointCloud.size().width);
            System.out.println(pointCloud.size().height);

            //  For each point in the point cloud
            for (int w = 0; w < pointCloud.size().width; w++) {
                for (int h = 0; h < pointCloud.size().height; h++) {

                    //  Get XYZ from pointCloud
                    double[] values = pointCloud.get(h, w);
                    if (values != null)

                        //  Append XYZ to file
                        if (values.length >= 3)
                            writer.append("v " + String.valueOf(values[0]) + "
                                " + String.valueOf(values[1]) + " " +
                                String.valueOf(values[2]) + " " + "\n");
                }
            }
            writer.close();
            fileOutputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }


    }
```

Listing 4.10: Writing Generated Point Cloud to a File

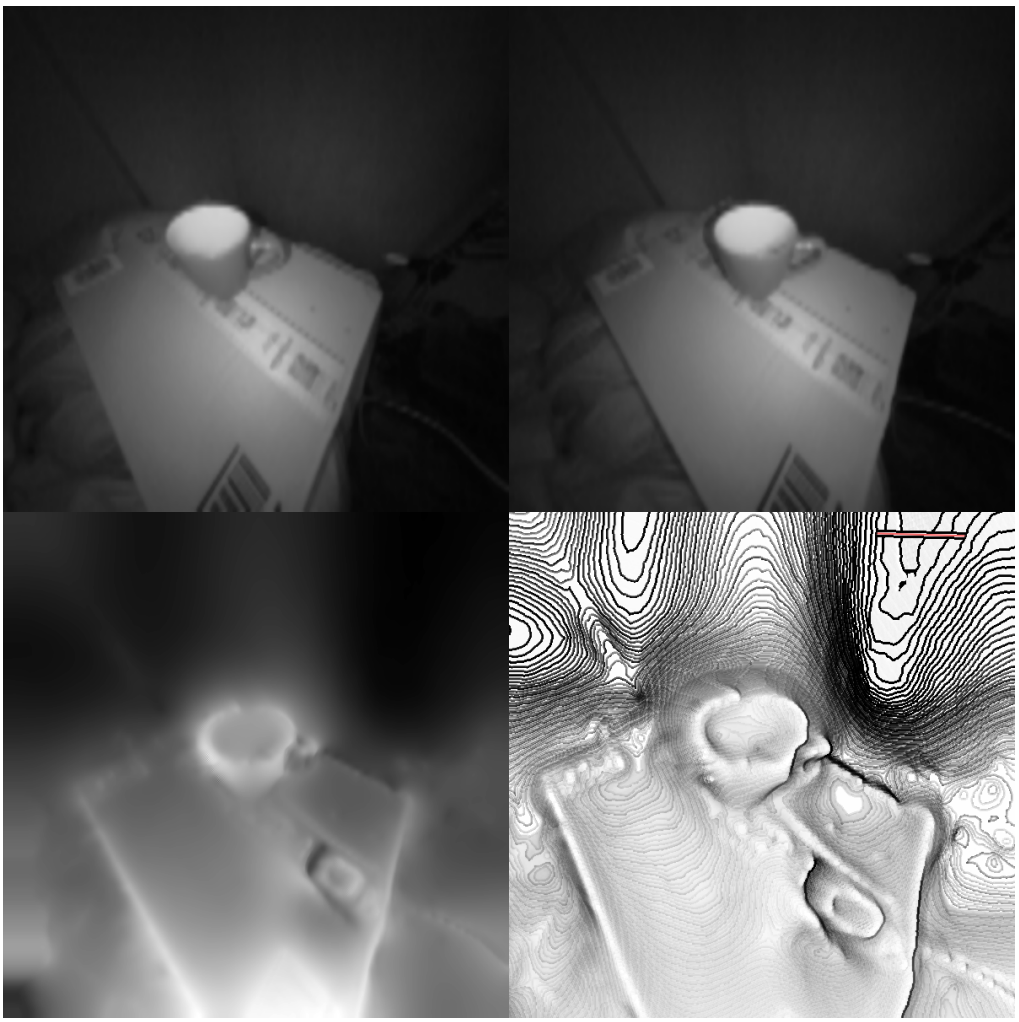

Figure 4.15: Top Left: Undistorted Left Image, Top Right: Undistorted Right Image, Bottom Left: Disparity Map Using StereoVar, Bottom Right: Generated Point Cloud

The file was then inspected with a tool called ccViewer[4]. This tool allows for the visualisation of point clouds. An example of the best point cloud produced using the `StereoVar` method can be seen in figure 4.15. The images show that the `StereoVar` Method gives a much clearer point cloud. However when looked at through multiple angles, it shows that some of the depths are non realistic. This is seen in figure 4.16. This might be due to the `Q` matrix, mentioned earlier, not being the exact one for the LMC used in this project.

[4] `http://www.danielgm.net/cc/`

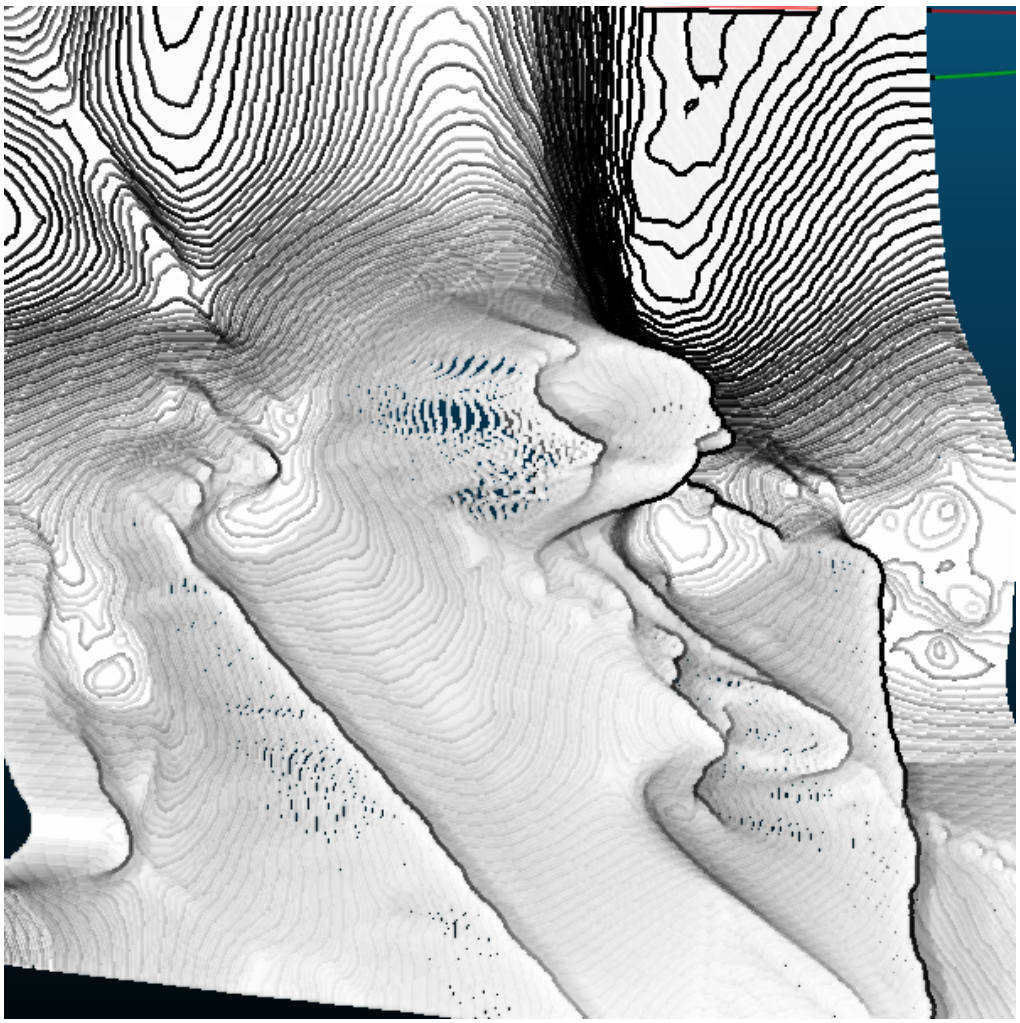

Figure 4.16: Point Cloud Showing Inaccurate Depths From Left Hand View

Some attempts were made running classification over point clouds created in this way, using the classification tool CANUPO (a plug in for ccViewer) as mentioned in chapter 2. However the results were inconclusive. Due to the difficulty in using the point cloud, and as time is running low, a different approach will be made for object recognition. Instead of making use of the disparity map functionality, then running classification on the point cloud, an attempt will be made using methods of feature recognition for classification.

Stage 3 - Feature Matching

OpenCV provides numerous classes within the `org.opencv.features2d` and `org.opencv.calib3d` packages, that are used to match features within images. The three important classes here are `DescriptorExtractor`, `DescriptorMatcher` and the `FeatureDetector`. Each of these classes follow the factory pattern. Using the `create()` method on each class, one is able to request an object specific to the type of algorithm that will be used to detect, extract and match the features within the images. An example of creating a Scale Invariant Feature Transform (SIFT) feature extractor with a Fast Approximate Nearest Neighbour Search (FLANN) matcher can be seen in listing 4.11.

```
1    public LeapCVObjectDetector() {
2        this.extractor = DescriptorExtractor.create(DescriptorExtractor.SIFT);
3        this.matcher = DescriptorMatcher.create(DescriptorMatcher.FLANNBASED);
4        this.featureDetector = FeatureDetector.create(FeatureDetector.SIFT);
5
6        this.matchedImage = new Mat();
7
8    }
```

Listing 4.11: Creating Detector Extractor and Matcher from Factories

This constructor makes it easy to swap the algorithms used to find the optimal one for object detection with the LMC. All that needs to be changed is the flag that is passed into the `create()` method. Each algorithm type gave very different results. It will not be clear which one is the best type until a classifier has been

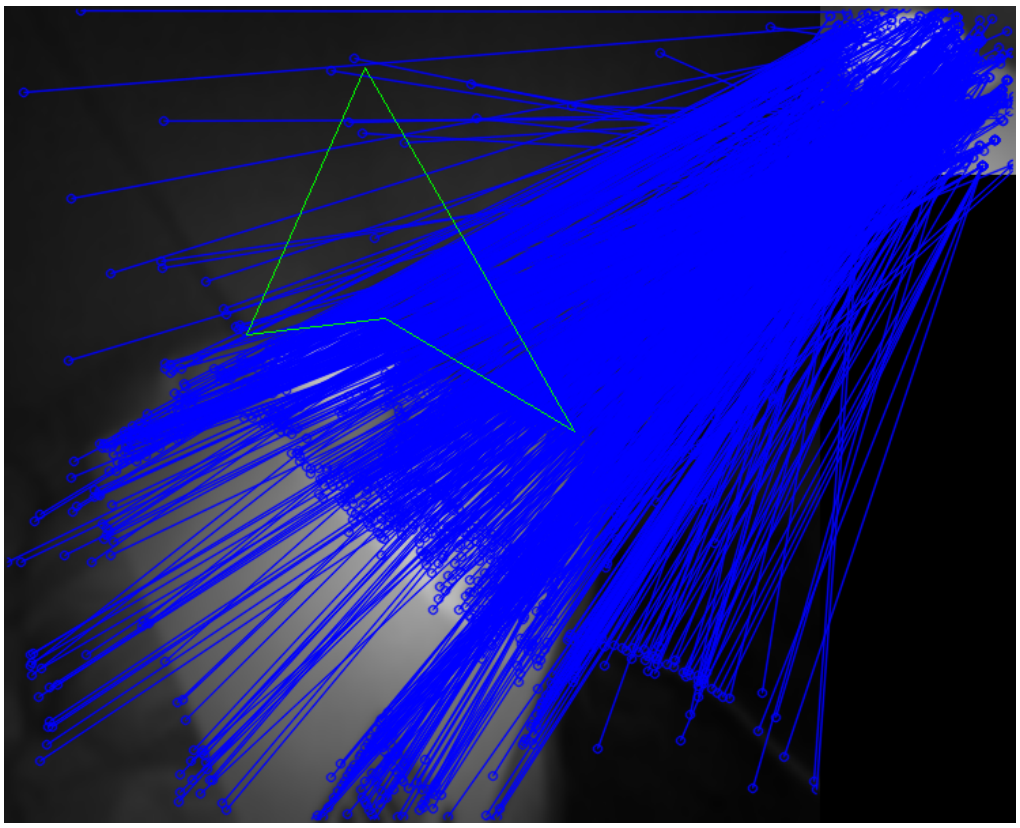

Figure 4.17: Matches Before Outlier Removal

implemented. However the best visual result so far can be achieved using a Good Features To Track (GFTT) `FeatureDetector`, a Bruteforce `DescriptorMatcher` and a SIFT `DescriptionExctractor`. Once features had been matched, something called outlier removal needed to occur. An example of and output before outlier removal takes place can be seen in figure 4.17.

A further example of the output produced after outlier removal can be seen in figures 4.18 and 4.19. This is shown along with the perspective transform box drawn around the object. Outlier removal is the process of elimination of the feature matches through statistical analysis. Based on the distance between feature matches. A full example of the implementation for object detection can be seen in appendix G.5 and G.6.

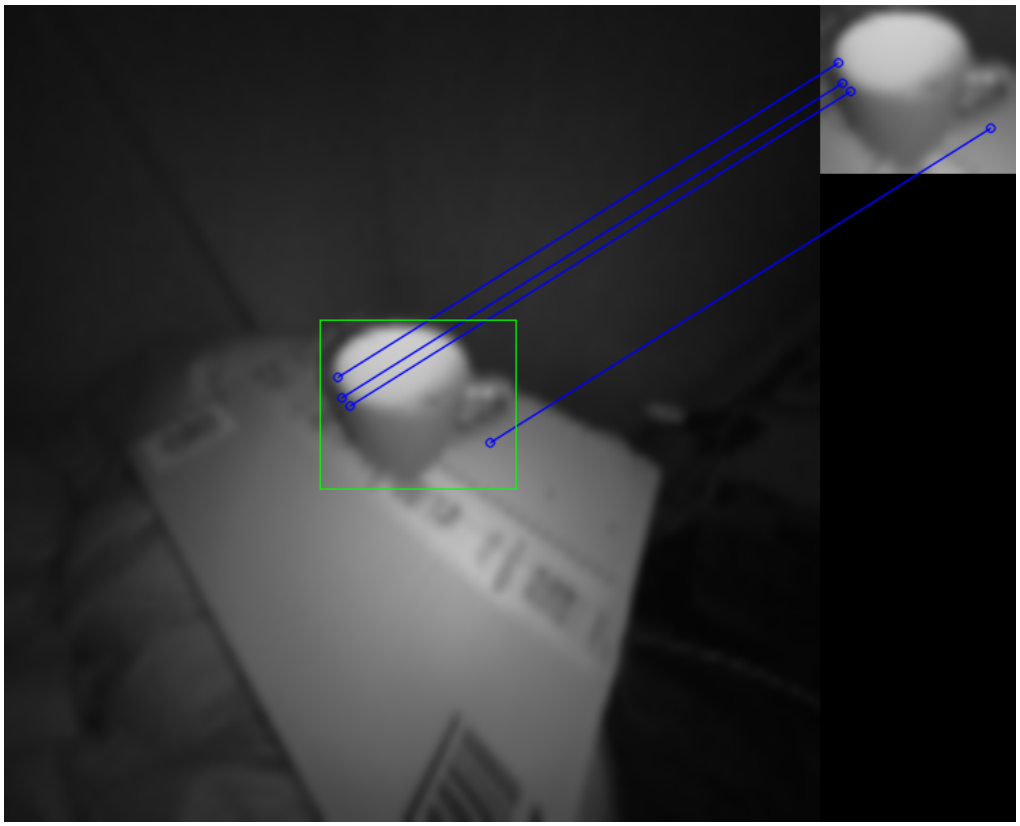

Figure 4.18: Matching Subsection of Mug in Original Left Camera Image

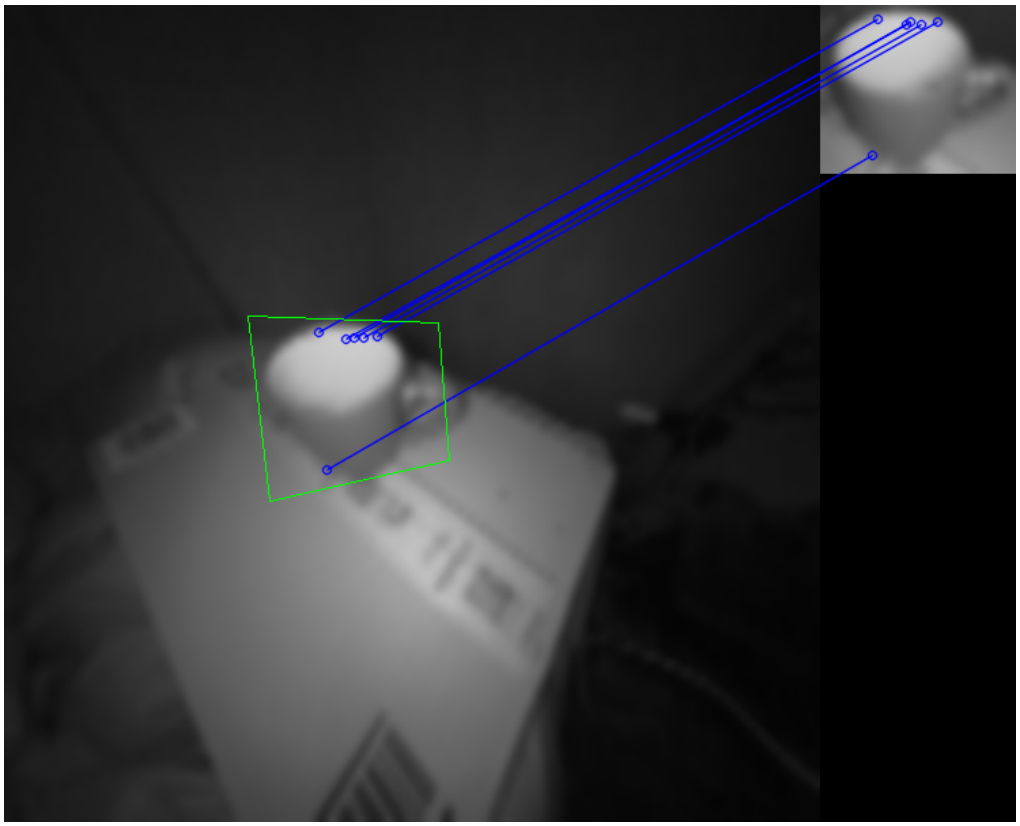

Figure 4.19: Matching Subsection of Mug to Right Camera Image

4.7.3 Test

Performance Test - Frame Rate

Test	Minimum Value	Actual Value	Pass
getImage	15 FPS	1082 FPS	Yes
getImageUndistorted	15 FPS	134 FPS	Yes
getDisparityStereoVar	15 FPS	11 FPS	No
getDisparityStereoBM	15 FPS	84 FPS	Yes
getDisparityStereoSGBM	15 FPS	41 FPS	Yes
getPointCloud	15 FPS	11 FPS	No

Table 4.4: Results of First Frame Rate Performance Test (FPS = Frames Per Second)

Notes on table 4.2:

getPointCloud This uses the StereoVar method with the default parameters and thus is bottlenecked by StereoVar. This test can be passed, but not without sacrificing the point cloud quality.

Performance Test - Detection Accuracy

Image Number	Expected Value	Actual Value
1	PASS	FAIL
2	PASS	PASS
3	PASS	PASS
4	PASS	PASS
5	PASS	FAIL
6	PASS	FAIL
7	PASS	FAIL
8	PASS	FAIL
9	PASS	FAIL
10	PASS	FAIL
11	PASS	PASS
12	PASS	FAIL
13	PASS	FAIL
14	PASS	PASS
15	PASS	PASS
16	PASS	PASS
17	PASS	FAIL
18	PASS	FAIL
19	PASS	FAIL
20	PASS	FAIL

Table 4.5: Results for Object Recognition Test

Figure 4.20: Mug to be Detected

During this iteration test was also put in place to attempt to measure the accuracy of the object detection. This was done by taking 20 still images of a detected object and visually analysing whether LeapCV had successfully found the object in the image. The object measured against can be seen in figure 4.20. Each image was measured as either a pass or a fail. The overall detection rate was 35% which is a fail when measured against the 90% accuracy requirement.

Acceptance Tests

The acceptance tests have been checked against the acceptance criteria that were earlier defined in appendix B.

Test Reference	**Pass**	**Notes**
1	Yes	
2	Yes	
3	Yes	
4	Yes	
5	No	Not yet implemented
6	No	Not yet implemented
7	No	Not yet implemented
8	No	Not yet implemented
9	No	Not yet implemented
10	No	Not yet implemented

Table 4.6: Acceptance Tests Iteration 3

4.7.4 Evaluation

What went well?

- Creation of a point cloud was successful.
- Whilst the updated design incorporating the factory pattern did not seem to improve the frame rate during the performance tests, it did successfully resolve the memory issue.
- The ability to match features between images was successful.
- The extra ability to find the projection transformation was also implemented.

What problems were faced?

- It was decided that the point cloud was not of a good enough quality to continue with classification.
- Time was lost looking for other methods of carrying out object detection.
- The ability for OpenCV to create a `Q` matrix from existing distortion maps does not exist.

Lessons learned?

- Calibration is a very difficult process. It is possible that it was more difficult due to the LMC lense having such a large radial (fisheye) distortion.

4.8 Implementation Overview

There were many challenges to be resolved throughout implementation, and some which could not be resolved. This can be seen in the previous sections.

Here some of the Functional and Non-Functional requirements that weren't measures previously, will be discussed.

Automatic camera configuration To comply with this functional requirement, the `LeapCVController` ensures that the LMC is initialised and receiving valid images before continuing.

Default configuration. To comply with this functional requirement, all of the stereo algorithms have a default set of values used in their constructor.

Multi-platform. To comply with this non-functional requirement, the LeapCV framework has been briefly tested on the three operating systems stated in chapter 1.

Java standards. To comply with this non-functional requirement, documentation has been generated in the form of a Javadoc. This can be seen in appendix F.1 and will be made available alongside the source code.

Extensibility. To comply with this non-functional requirement LeapCV was made open source using Github.

4.9 Summary

This chapter has logged the development process of the project.

It has discussed the difficulties that were faced throughout the process and highlighted the problems that were particularly interesting to overcome. It shows how the design has evolved over each iteration through an evaluation of the requirements. Moving into chapter 6, the project as a whole will be evaluated, looking at how far the project has gone in solving the original problem.

Chapter 5

Unachieved Requirements

5.1 Introduction

This chapter will briefly highlight the requirements that were not achieved during the three planned iterations. It will discuss what benefit they would have brought to LeapCV, stating what effect they have had by not being completed.

5.2 List of Unachieved Requirements

Due to the complexity of some of the problems that were faced in the iterations, and the time consuming nature of achieving good image results from the LMC, each of the iterations over ran the planned time box. It was hoped that throughout iteration 3, some time would be left to achieve some results with object classification. Unfortunately this was not a feasible option. The following table 5.1 shows use cases that were not achieved during the iterations: The test reference column corresponds to the acceptance criteria table in appendix B.

The "classify object" use case is half implemented, as seen in the last stage of iteration 3. Object detection is implemented at a very basic, inaccurate level, with no classification. It is still here as it will need to be refined for future work.

Test #	Use Case	MoSCoW Status
5	Change Classifier Type	Could
6	Change Classifier Parameters	Could
7	Reset Training Data	Could
8	View Current Classifications	Could
9	Train Classifier	Could
10	Classify Object	Could

Table 5.1: Incomplete Use Cases

5.3 Effect on the Resulting Framework

All of the unachieved requirements relate to the classification functionality of LeapCV. This this was due to get explored in iteration 3. Due to problems that were faced in iteration 3, there was no extra time to explore this. LeapCV in its current state has resulted in being a stepping stone toward this final goal. It has identified that the leap motion can in fact be used for at least some elements of computer vision, although some of the results will need to be verified in a more objective way. The use cases that have been implemented are most certainly useful toward achieving the unachieved requirements in the future.

5.4 What it Should Have Achieved?

The classification functionality was going to enable LeapCV to be used for detecting objects, other than hands, within the images produced by the LMC. With the idea that this might then enable the LMC to be considered as a viable computer vision option for new projects. The idea was to allow the application developer the ability to train LeapCV to identify a set of objects that they defined themselves. This would have involved developing a database that would hold classification data. As well as making use of classifiers that were defined in chapter 2. OpenCV has some of this functionality built in therefore it is only time that is needed to explore this further and integrate it with what has already been developed. This functionality would have been built upon the basic object detection that has already been developed in iteration 3.

5.5 Summary

It was unfortunate that some of the development was a lot more difficult than originally envisaged. However hopefully by making this project available open source the functionality will be included in the future.

Chapter 6

Evaluation

6.1 Introduction

In this chapter, the aim is to evaluate the work that has been achieved throughout this project. It will highlight which areas of work might have benefited from being done differently. It will also discuss how far the implementation has gone to fulfilling the original requirements.

To do this, each aim and objective will have its status reviewed. An evaluation of the overall system will then take place, followed by an evaluation of the technologies, techniques and tools that have been used throughout the project.

6.2 Reflection on Aim and Objectives

6.2.1 Aim

To develop an open source Computer Vision framework (LeapCV), usable by the Leap Motion Controller community.

The overall aim for this project was to develop LeapCV as a framework that can be usable by the Leap Motion community. As seen in the chapter 5 there

were requirements that were not achieved. However, the basic framework has still been developed, and can still be made open source to be developed further.

The LMC is capable of carrying out some basic computer vision tasks with LeapCV, and as such, the aim for this project has been achieved.

6.2.2 Objectives

Research stereoscopic image processing methods and object classification.

This objective was put in place to ensure that methods were researched before development of LeapCV was started. The LMC is a stereoscopic camera, and therefore the research was intended to be applied in the solution. Researching this allowed the discovery of methods for generating the disparity maps that were shown in the implementation. In hindsight, some of the information was a little out of scope. However it did allow a broader understanding of cameras on the whole and made the OpenCV library look much less daunting. Although the classification element of the implementation was not completed, some research was done, which might help anyone that would wish to continue this project in the future. This objective was met as far as this project was concerned, but research should continue for future work.

Design and develop a framework that allows the Leap Motion Controller to be used for computer vision.

The framework that has been developed in this project is a decent attempt at discovering whether the LMC is suitable to be used for computer vision. The end result here is very much a prototype and would need to be much more refined to be used successfully in a stable application. The ability to carry out computer vision functionality has been tested through the form of unit tests, and visual analysis of outputs from LeapCV. The results were then fed back into trying to

improve the outputs throughout implementation.

The design is a relatively simple model, sided by utility functionality. This modular approach means that LeapCV can be easily extended with new image processing techniques, whilst using the already existing model.

This was very much an exploratory prototyping project, as such, it is almost certain that the current implementation can be improved upon. This objective has been met as far as the description goes, however, LeapCV is most certainly not complete.

Enable the Leap Motion Controller to recognise simple objects.

Object recognition was achieved in the third iteration. However this it is only very basic. The recognition achieved does not actually classify the object that it finds. It only goes so far as to detect if features exist in two compared images. This objective has been achieved to a basic level.

Evaluate the accuracy of object recognition.

As no classification was implemented, the accuracy of the object recognition was not able to be measured with ease autonomously. Therefore this objective was not achieved.

Release an open source framework.

By using the open source focussed version control system through GitHub[1] the project is able to be released to the open source community with the click of a button. By making use of the OpenCV library, which is held under the BSD license, this allows for an extremely flexible open source project with no necessity to feed back to the original source. This project will also be released under the BSD licence. This objective has been achieved.

[1] http://www.github.com

Produce a report that communicates my findings

This report has been produced throughout the life of this project. Notes were collected using a basic blogging system throughout implementation. This was then written into the format that you see in chapter 4. The LaTeX tool was used along with a version control system, which allowed the report to be evolved modularly throughout the project. The fact that this report is being read, shows this objective was successfully completed.

6.3 Evaluation of the Original Hypothesis

The original hypothesis was as follows:

> "It is possible to use the LMC in the field of computer vision, so that application developers can exploit the technology in different ways as to what was originally intended.
>
> This ability can then be structured into a framework, so that the effort application developers have to put in to use the LMC as a solution is low, making it an option to consider in new projects."

Through the implementation of this project, it has been shown that the LMC has produced images that are able to be processed in a way that can be used in computer vision applications. This has then been structured into a framework that requires a minimal understanding of computer vision, to obtain some basic results. Therefore it is fair to say that this hypothesis is true.

6.4 Reflection on Research Approach

The initial literature survey was an important part of this project, as it was the main stage where the author could research relevant material to apply. As well as refine the problem. It also allowed the author to learn about computer vision and

cameras in general. Without this taking place it would not have been possible for the development of LeapCV to take place.

The research process used was guided by the methods outlined by Dawson (2005). One of the key notes taken from this was to know when to stop. Computer vision is such a massive field that one can get carried away with it. Therefore it was useful to define an initial scope with a good foundation of knowledge and then gather more knowledge as it was needed throughout requirements analysis and implementation. Gradually moving toward a narrower depth of focus as time went on. This process has improved the authors ability to evaluate material quickly, as well as make use of the multiple different types of knowledge sources available.

Now that the project has come to an end, it is believed that there are many more questions that should be answered, that have come to light through the hands on development of LeapCV.

6.5 Reflection on Development Methodology

The development methodology followed throughout this project was an iterative approach. Where the first iteration was an initial throwaway prototype and the later iterations evolutionary prototypes. This choice was made due to the uncertainty in the projects outcome. Therefore leaving a lot of room to adjust requirements throughout the iterations as the author learned more about the libraries being used.

It is questionable as to whether this approach is what caused there to be unachieved requirements at the end. The uncertainty at the beginning meant that there was always a possibility that all of the goals would be completed in time. However the prototyping approach meant that changes could be made throughout implementation based on the outcomes of a particular iteration.

In hindsight, whilst the first throwaway prototype was important to learn about

the basics of the OpenCV library. It is believed that a test driven approach might have been slightly better suited from then on. It was noted that throughout iteration 2 and 3, a lot of manual visual analysis was required on the images to ensure parameters were correct. This almost comes across as a bit or a paradox. Whilst the computer can generate the image, it still does not know of its wider context and whether it is right or wrong. Manual visual analysis is still needed. Raising the question as to whether a test driven approach could be used. If it could then it would most likely require a lot of test data and models of what should be expected as a perfect output from the LeapCV framework.

The MoSCoW prioritisation method did work well with this project. There was a clear breakdown in the order of work based on the research that had been done on stereoscopy in chapter 2. It allowed the project to be left in a position that could easily be picked up and carried on in future.

6.6 Reflection on Tools Used

Some of the tools used throughout development have already been discussed in other areas of this document, however this section is designed to review their usage. Without many of the tools listed below LeapCV will not have been successful.

6.6.1 Hardware - Leap Motion Controller

The LMC was described in chapter 1, however now that it has been developed with, it is useful to talk about it from a hands on perspective.

The LMC has a large online community and is a well documented and supported platform to develop with. There are frequent updates made available to the community. The LeapSDK is extensive and provides good opportunity to develop with the device. However, it was discovered only during implementation that the images that are provided by the LMC are only a low 640×240 resolution. This

is quite low in comparison to other cameras available on the market. The images are only given in an infra red gray scale format, therefore no colour information can be used for computer vision. Although generally this is not a problem as a lot of computer vision is done using grey scale images.

It is also not possible to adjust some of the camera parameters from a code level. For example to allow image access, you have to use the LMC driver user interface to turn it on. Setting set the brightness of the infra red LED's as also a small issue. On occasion, especially in very close up images, the LED's were too bright. This resulted in over exposed images. It is hoped that some of this functionality might get implemented in a future release of the LeapSDK and LMC drivers. Overall it was a robust device to use and was not difficult to get started with.

6.6.2 Library - OpenCV

OpenCV is an extensive library for computer vision. It is large and complex and required a fairly good amount of prior knowledge to feel comfortable with understanding some of the terminologies used within the library. One pitfall of using OpenCV was that most of the documentation and tutorials are written in C++ or Python. This is due to the library being written in C++ and seemingly having much better Python support than it does Java. The Java implementation is documented as an Android library, and through its usage feels like a bit of an afterthought. While there are many similarities between the two languages, it did sometimes make it quite difficult to translate between the two when looking at examples in the API.

When used with Java, OpenCV is what is known as a Java Native Interface (JNI) library. The error messages are all given in the format of C++ error messages and it was found that some of them didn't use the same terminology as the JNI. It it also not possible to view any of the JNI contents values in a memory evaluator at runtime, all that Java holds, is a memory address pointer of the JNI object. Unfortunately this occasionally made development very slow as it added

time on to any debugging.

Throughout development it was also found that elements of the JNI version of OpenCV were not actually interfaced from the C++ version. An example being the `Highgui.imshow()` functionality that the C++ version has. This allows for an image to be displayed on screen with a single function call. The JNI version does not have this functionality, which required the development of a method to display the images on the screen during prototyping, adding unforeseen time on to the projects development.

6.6.3 Language - Java

The decision to use the Java programming language was primarily due to the authors proficiency in programming with it. However, if the project was to be developed again from the ground up it would be preferable to program in either the Python or the C++ programming language. Both of these languages seem to have much more support from the OpenCV library, as well as the LeapSDK. Again adding time onto development, by having to translate examples from either C++ or Python into the Java language. It is believed that the initial time spent in learning C++ or Python to enough of an extent to complete this project, would have avoided these increases in time spend on development, throughout implementation.

A positive note is that Java has an excellent UI framework called JavaFX in version 1.8 of its SDK. JavaFX is a young and still growing framework that has made GUI development extremely easy when compared to the Java Swing framework. It follows standards of the Model-View-Controller design pattern allowing the author to understand an applications life cycle at run-time with ease.

6.7 Reflection on Implementation

LeapCV was an enjoyable project for the author to work on. However, without any prior knowledge of computer vision, learning about some of the concepts was possibly the cause of the slower development speed than was originally expected in some areas. Some of the problems faced would have most definitely benefited from having a better knowledge in the field.

The first iteration went very well. Without the initial exploration of the OpenCV library, it would not have been possible to produce any form of framework within the time allocated for this project. All of the work was relevant to bring into the next iteration and it was the biggest learning curve of the project.

Iteration 2 was much more structured and less exploratory. Having already explored OpenCV and the LeapSDK, the knowledge was able to be fed into the design of the first evolutionary prototype system. Having the design to adhere to made the implementation and re-factoring of iteration 1 an easy and satisfying task. However through testing in this iteration some issues were found that lead to a slower iteration than was originally intended. In hindsight, it might have been better to allocate some of the time from iteration 2, to iteration 1. Thus making iteration 2 a pure re-factoring task, leaving more time to build a better base system and more time for testing. It would also have been useful to run some of the performance tests in the first iteration.

Iteration 3 provided a lot of problems to overcome that weren't all explicitly about computer vision. One of these problems was to overcome a memory leak that was found through a performance test in iteration 1. This required some changes in the design of the system. Which in turn added some unforeseen development time to the project. The re-factored design did solve the memory problem but again added development time. Further to this more issues were then found in solving the computer vision problems. Resulting in having to change the approach mid way through the iteration in an attempt to achieve the requirements.

The project would have most certainly benefited from having another iteration. Although the framework is not complete, following the processes of implementation has left it in a state that could be picked up and worked upon in future.

6.8 Evaluation of LeapCV

LeapCV is a framework intended to be used by developers, requiring specific hardware to be used. As such, in its current prototype form it is difficult to distribute and have evaluated by other developers of the same technology. However, what can be said is the 50% of requirements that were either a "Must" or a "Should" in the MoSCoW table in figure 3.2, have been successfully implemented. With one of the requirements, "Classify Object" was a "Could" in prioritisation, but was only partially implemented.

The documentation has been created and the source code is also available online, therefore the state that the work is left in is suitable to say that is is "Useable".

To measure usefulness would require for LeapCV to be developed into an application by a developer, and then to be evaluated through the form of a questionnaire or change requests. With the use of Git, issues and requests can be made into the code repository by anybody that wishes to make a change.

6.9 Summary

This chapter has evaluated the elements of this project and reflected on all different aspects of the project. These were:

- Evaluation of aims and objectives.
- Evaluation of the original hypothesis.
- Reflection on the research approach.
- Reflection on the development methodology.

- Reflection on implementation

The next and final chapter of the report will summarise the project, highlighting what has been achieved, and re-stating any major points that have been made throughout.

Chapter 7

Conclusion

7.1 Project Summary

This project has researched whether the Leap Motion Controller(LMC) is able to be used in the field of computer vision, by developing a framework that has made it simple to do so. A literature review was conducted into computer vision, looking at the theory and solutions that already exist. From this a prototype framework was developed (LeapCV) that is able to carry out basic computer vision functionality.

This was carried out following a full software life cycle using an iterative prototyping approach. Establishing the requirements of such a framework, then moving on to implementing elements of them.

This report has set out to document this process, from start to finish, resulting in a reflection and evaluation on how the project went.

7.2 Aim and Objectives

The aim and objectives were set out in chapter 1 and then evaluated in chapter 6. They were as follows:

7.2.1 Aim

To develop an open source Computer Vision framework (LeapCV), usable by the Leap Motion community.

7.2.2 Objectives

- Research stereoscopic image processing methods and object classification.
- Design and develop a framework that allows the LMC to be used for computer vision.
- Enable the LMC to recognise simple objects.
- Evaluate the accuracy of object recognition.
- Release an open source computer vision framework.
- Produce a report that communicates my findings.

Through their evaluation most of these objectives were achieved. "Evaluate the accuracy of object recognition" was not possible due to the difficulties faced in developing the classification system for LeapCV.

7.3 Wider Significance

The results obtained throughout the implementation of LeapCV show that there is a possibility that the LMC could, with further development, be a viable computer vision solution in some areas.

LeapCV provides an easy way for developers to get started with computer vision, with a device that they might already have at home. It is hoped that a framework such as LeapCV will show more developers, being beginners or experts, that basic computer vision doesn't have to require large expensive imaging equipment to get started.

It is also a hope that a benefit might be found in using a system like this in robotics prototyping, where for example, a rudimentary vision system is required for prototyping. The LMC can simply be interfaced via USB and developed with.

The reason that a library such as LeapCV was made open source is so that it can be developed further, or even just looked at as a base to start another project with a similar concept. The framework might not be a perfect implementation, but it will most certainly be able to start as an example to what might be able to be achieved.

7.4 Future Work

Foremost, the hope is to implement the rest of the outstanding requirements. This would put LeapCV in a good position for extending further. It would be interesting to see if the classification is possible, and further to this whether it would prove an accurate enough device to provide real spacial measurements of objects. On top of this it would be interesting to see further research in using LeapCV in a real world application for classification. For example, whether the leap motion can be used on top of a robotic arm to carry out packaging tasks on a conveyor belt. For this capability the device would need to be perfectly calibrated and be able to provide accurate distance measurements within a scene.

LeapCV will continue to be supported as an open source project on GitHub[1]. In the hope that it will be further contributed to by members of the Leap Motion community.

[1] http://www.github.com/danhamilt1/LeapCV/

Bibliography

Alvarez, L., Deriche, R., Sánchez, J. & Weickert, J. (2000). Dense disparity map estimation respecting image discontinuities, `http://serdis.dis.ulpgc.es/~lalvarez/research/demos/StereoFlow/`. (Visited on 04/12/2015).

Ashmore, S. & Runyan, K. (2015). *Introduction to agile methods*, Pearson Education, Upper Saddle River, NJ.
URL: *www.summon.com*

Bachmann, D., Weichert, F. & Rinkenauer, G. (2014). Evaluation of the leap motion controller as a new contact-free pointing device, *Sensors* **15**(1): 214–233.

Besl, P. & Jain, R. (1985). Three-dimensional object recognition, *ACM Computing Surveys (CSUR)* **17**(1): 75–145.
URL: *www.summon.com*

Bradski, G. & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*, " O'Reilly Media, Inc.".

Brodu, N. & Lague, D. (2012). 3d terrestrial lidar data classification of complex natural scenes using a multi-scale dimensionality criterion: Applications in geomorphology, {*ISPRS*} *Journal of Photogrammetry and Remote Sensing* **68**(0): 121 – 134.
URL: *http://www.sciencedirect.com/science/article/pii/S0924271612000330*

Brown, M., Burschka, D. & Hager, G. (2003). Advances in computational stereo, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **25**(8): 993–1008.

Coppin, B. (2004). *Artificial Intelligence Illuminated*, Jones and Bartlett illuminated series, Jones and Bartlett Publishers.
URL: *https://books.google.co.uk/books?id=LcOLqodW28EC*

Dawson, C. (2005). *Projects in Computing and Information Systems: A Student's Guide*, Addison-Wesley.
URL: *http://books.google.co.uk/books?id=EAfUdoRsjQ0C*

Findling, R. D. & Mayrhofer, R. (2013). Towards pan shot face unlock, *International Journal of Pervasive Computing and Communications* **9**(3): 190–208.
URL: *http://dx.doi.org/10.1108/IJPCC-05-2013-0012*

Forsyth, D. A., Ponce, J., Mukherjee, S. & Bhattacharjee, A. K. (2012). *Computer vision: a modern approach*, Pearson, Harlow.
URL: *www.summon.com*

Freeman, E., Freeman, E., Sierra, K. & Bates, B. (2004). *Head First Design Patterns*, Head first series, O'Reilly Media, Incorporated.
URL: *https://books.google.co.uk/books?id=GGpXN9SMELMC*

Gassaway, J. (2011). Local bundling of disparity maps for improved dense 3d visual reconstruction.

Gnecco, P. (2012). first point cloud. kinect//trapcode form. !, `https://www.youtube.com/watch?v=s3wl2y9ANDs`.

Han, J., Shao, L., Xu, D. & Shotton, J. (2013). Enhanced computer vision with microsoft kinect sensor: A review, *Cybernetics, IEEE Transactions on* **43**(5): 1318–1334.

Hartley, R. & Zisserman, A. (2003). *Multiple view geometry in computer vision*, Cambridge university press.

Hirschmuller, H. (2008). Stereo processing by semiglobal matching and mutual information, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **30**(2): 328–341.

Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D. & Roy, N. (2011). Visual odometry and mapping for autonomous flight using an rgb-d camera, *International Symposium on Robotics Research (ISRR)*, pp. 1–16.

Kosov, S., Thormählen, T. & Seidel, H.-P. (2009). Accurate real-time disparity estimation with variational methods, *Advances in Visual Computing*, Springer, pp. 796–807.

Leap Motion, I. (2015). What's new with v2 tracking.
URL: *https://developer.leapmotion.com/features*

Lorenzo, C. (2014). Visualize un-distorted images received from the leap motion cameras using opencv.
URL: *http://stackoverflow.com/questions/27260163/visualize-un-distorted-images-received-from-the-leap-motion-cameras-using-opencv*

Ma, R. (2014). Leap motion as a rudimentary depth sensor.
URL: *http://blog.rymnd.com/articles/leap-motion-depth/*

Miles, R. & Hamilton, K. (2006). *Learning UML 2.0*, O'Reilly Media.
URL: *https://books.google.co.uk/books?id=QhiA6vT56E4C*

Mohandes, M., Aliyu, S. & Deriche, M. (2014). Arabic sign language recognition using the leap motion controller, *Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on*, pp. 960–965.

Potter, L. E., Araullo, J. & Carter, L. (2013). The leap motion controller: A view on sign language, *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, OzCHI '13, ACM, New York, NY, USA, pp. 175–178.
URL: *http://doi.acm.org/10.1145/2541016.2541072*

Pulli, K., Baksheev, A., Kornyakov, K. & Eruhimov, V. (2012). Real-time computer vision with opencv, *Commun. ACM* **55**(6): 61–69.
URL: *http://doi.acm.org/10.1145/2184319.2184337*

Quevedo, R. & Aguilera, J. M. (2010). Computer vision and stereoscopy for estimating firmness in the salmon, *Food and Bioprocess Technology* **3**(4): 561.

Rusu, R. B. & Cousins, S. (2011). 3D is here: Point Cloud Library (PCL), *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

Sa, J. (n.d.). Crc process.
URL: *http://goo.gl/c0mFWr*

Scharstein, D. & Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms, *International Journal of Computer Vision* **47**(1): 7–42.
URL: *www.summon.com*

Shirai, Y. (1972). Recognition of polyhedrons with a range finder, *Pattern Recognition* **4**(3): 243 – 250.
URL: *http://www.sciencedirect.com/science/article/pii/0031320372900039*

sightmachine (2014). Simplecv, `https://github.com/sightmachine/SimpleCV`.

Stackexchange (2014). How to calculate the lens distortion coefficients with a known displacement vector field?
URL: *http://math.stackexchange.com/questions/302093/how-to-calculate-the-lens-distortion-coefficients-with-a-known-displacement-vect*

Stemmer (n.d.). The imaging and vision handbook.

Szeliski, R. (2010). *Computer vision: algorithms and applications*, Springer Science & Business Media.

Wakefield, J. (2015). Driverless car review launched by uk government.
URL: *http://www.bbc.co.uk/news/technology-31364441*

Appendices

Appendix A

Use Case Descriptions

Use case	`ManipulateByTransform`
Description	The application developer should be able to manipulate the image by transformation. (Such as resize or warp). For use if overriding calibration methods.
Actor	Application developer
Entry Condition	The image needs to be instantiated and valid for a successful transformation. `RetrieveCameraImages` needs to have been successful.
Flow of Events	1. Application developer requests a transformed image by transformation type. 2. Framework checks image is valid. 3. Framework validates transformation. 4. Framework transforms image. 5. Framework returns transformed image.
Alternative Flow	If the image or transformation is not valid a `TransformationException` should be thrown.
Exit Condition	Application developer receives a transformed image as a new instance.

Table A.1: Documented `ManipulateByTransform` *Use Case*

Use case	`ClassifyObject`
Description	The user should be able to ask the framework to start classifying images it receives
Actor	User
Entry Condition	The leap motion controller must be successfully calibrated and receiving valid images. The classifier must have been trained before detecting objects.
Flow of Events	1. User requests classification of image. 2. Framework checks image is valid. 3. Framework runs object detection algorithm. 4. Framework returns classification of image.
Alternative Flow	If the image is not valid then the framework will notify the User. If no classification can be given then a pre defined value will be given to the user stating this.
Exit Condition	User receives a classification for the object in the image

Table A.2: Documented `ClassifyObject` *Use Case*

Use case	`ChangeStereoAlgorithm`
Description	The application developer should be able to change the stereo algorithm used to generate the disparity map.
Actor	Application developer
Entry Condition	The application developer should pass in a valid pre defined flag to a constructor, stating what type of algorithm should be used when generating a disparity map.
Flow of Events	1. Application developer passes in pre defined flag. 2. Framework checks flag is valid. 3. Framework constructs object based on flag. 4. Framework returns a valid object.
Alternative Flow	If the flag is not of a correct value then an exception should be thrown.
Exit Condition	Application developer receives valid object for generating a disparity map.

Table A.3: Documented `ChangeStereoAlgorithm` *Use Case*

Use case	`ChangeStereoParameters`
Description	The application developer should be able to change the stereo algorithm parameters used to generate the disparity map.
Actor	Application developer
Entry Condition	The application developer should pass in a valid set of parameters for the stereo algorithm. The stereo algorithm to be used should already be set.
Flow of Events	1. Application developer passes in parameters. 2. Framework checks parameters are valid. 3. Framework sets parameters. 4. Framework notifies application developer of a successful parameter set.
Alternative Flow	If the parameters are not valid then an exception should be thrown.
Exit Condition	Application developer receives valid object for generating a disparity map.

Table A.4: Documented `ChangeStereoParameters` *Use Case*

Use case	`ChangeClassifierType`
Description	The application developer should be able to change the type of classifier that gets used to classify objects.
Actor	Application developer
Entry Condition	The application developer should pass in a valid pre defined flag to a constructor, stating what type of classifier will be used.
Flow of Events	1. Application developer passes in pre defined flag. 2. Framework checks flag is valid. 3. Framework constructs object based on flag. 4. Framework returns a valid object.
Alternative Flow	If the flag is not valid then an exception should be thrown.
Exit Condition	Application developer receives valid object for generating a disparity map.

Table A.5: Documented `ChangeClassifierType` *Use Case*

Use case	`ChangeClassifierParameters`
Description	The application developer should be able to change the classifier parameters used to classify an object.
Actor	Application developer
Entry Condition	The application developer should pass in a valid set of parameters for the classifier. The classifier to be used should already be set.
Flow of Events	1. Application developer passes in parameters. 2. Framework checks parameters are valid. 3. Framework sets parameters. 4. Framework notifies application developer of a successful parameter set.
Alternative Flow	If the parameters are not valid then an exception should be thrown.
Exit Condition	Application developer receives valid object for classifying objects.

Table A.6: Documented `ChangeClassifierParameters` *Use Case*

Use case	`GetDisparityMap`
Description	The application developer should be able to receive a disparity map, generated by running both camera images through a stereo correspondence algorithm.
Actor	Application developer
Entry Condition	The application developer must have passed in a valid image from both of the cameras in the same frame.
Flow of Events	1. Application developer passes in an image from both cameras. 2. Framework checks the images are valid. 3. Framework runs images through stereo correspondence algorithm. 4. Framework returns disparity map.
Alternative Flow	If the images are not valid then an exception should be thrown.
Exit Condition	Application developer receives valid disparity map.

Table A.7: Documented `GetDisparityMap` *use case*

Use case	`ResetTrainingData`
Description	The user should be able to reset the classification data.
Actor	User
Entry Condition	The leap motion controller must be successfully calibrated and receiving valid images. The images being passed into the classifier must be valid.
Flow of Events	1. User requests for the classification data to be removed. 2. Framework removed the . 3. Framework notifies user on successful classification.
Alternative Flow	If the images are not valid then the framework will notify the User. If training the classifier fails then the user will be notified.
Exit Condition	User is notified on success of training.

Table A.8: Documented `ResetTrainingData` *use case*

Use case	`ViewCurrentClassifications`
Description	The user should be able to view all of the previous classifications that have been made by the framework.
Actor	User
Entry Condition	The leap motion controller must hold existing classification labels of previous classifications.
Flow of Events	1. User requests to view all existing classification labels. 2. Framework attempts to retrieve classification labels. 3. If successful the framework will return a list of all current classifications available.
Alternative Flow	If there is no classification history then the user will be notified.
Exit Condition	User receives a list of all existing classifications that the framework holds.

Table A.9: Documented `ViewCurrentClassifications` *Use Case*

Use case	`TrainClassifier`
Description	The user should be able to ask the framework to classify new object in images.
Actor	User
Entry Condition	The leap motion controller must be successfully calibrated and receiving valid images. The images being passed into the classifier must be valid.
Flow of Events	1. User passes image or a group of images into the classifier. 2. Framework checks images are valid. 3. Framework trains the classifier with the new images. 4. Framework notifies user on successful classification.
Alternative Flow	If the images are not valid then the framework will notify the User. If training the classifier fails then the user will be notified.
Exit Condition	User is notified on success of training.

Table A.10: Documented `TrainClassifier` *Use Case*

Appendix B

Acceptance Criteria

Test #	**Use Case**	**Acceptance Criteria**
1	Retrieve Camera Images	• Can retrieve images from the camera. • Provides images in a useful format.
2	Manipulate Image	• Can manipulate images that are taken from the camera. • Can manipulate images with a transformation. • Can manipulate images with a filter.
3	Change Stereo Algorithm	• Can select which method of stereo correspondence to use.
4	Change Stereo Parameters	• Can change the parameters for a particular stereo correspondence algorithm.

5	Change Classifier Type	• Can change types of classifier used.
6	Change Classifier Parameters	• Can change the parameters on a particular classifier.
7	Reset Training Data	• Any previous training data can be removed.
8	View Current Classifications	• Existing trained classification labels can be viewed.
9	Train Classifier	• The classifier can be trained on a new set of classification data. • Classification data can be collected at run time with the leap motion, prior to training.
10	Classify Object	• The framework can segment an object within an image. • The framework will provide a classification label when provided with data to classify.

Table B.1: Acceptance Criteria

Appendix C

CRCs for Iteration 2

Class Name	`LeapCVCamera`
Responsibilities	• Hold calibration information about each camera in the controller. • Hold the current image for this camera, requested from the controller.
Collaborations	• `LeapCVController` • `LeapCVImage`

Table C.1: `LeapCVCamera` *CRC*

Class Name	`LeapCVImage`
Responsibilities	• Holds two different image formats post processing.
Collaborations	• `LeapCVCamera`

Table C.2: `LeapCVImage` *CRC*

Class Name	`LeapCVStereoUtils`
Responsibilities	• All types of stereo functionality.
Collaborations	• `OpenCV` • `LeapSDK`

Table C.3: `LeapCVStereoUtils` *CRC*

Class Name	`LeapCVImageUtils`
Responsibilities	• All types of image manipulation. • Conversion between image types. • Shared functionality between classes.
Collaborations	• `OpenCV` • `LeapSDK`

Table C.4: `LeapCVImageUtils` *CRC*

Class Name	`LeapCVStereoCalibrator`
Responsibilities	• Carry out functionality to calibrate each camera. • Store the calibration information for each camera.
Collaborations	• `LeapCVCamera`

Table C.5: `LeapCVStereoCalibrator` *CRC*

Appendix D

CRCs for Iteration 3

Class Name	`LeapCVStereoCalibrationUtils`
Responsibilities	• Perform calibration functionality for the Leap Motion Controller. • Create the calibration functionality for the `LeapCVCamera` class.
Collaborations	• `OpenCV` • `LeapCVCamera`

Table D.1: `LeapCVStereoCalibrationUtils` *CRC*

Class Name	`LeapCVStereoMatcherFactory`
Responsibilities	• Handle the creation of stereo matchers.
Collaborations	• `OpenCV` • `LeapCVStereoUtils` • `LeapCVStereoMatcher` • `LeapCVStereoBM` • `LeapCVStereoSGBM` • `LeapCVStereoVar`

Table D.2: `LeapCVStereoMatcherFactory` *CRC*

Class Name	`LeapCVStereoMatcher`
Responsibilities	• Generalisation for stereo matchers. • Provides interface to implement for StereoMatchers.
Collaborations	• `OpenCV` • `LeapCVStereoUtils` • `LeapCVStereoMatcher` • `LeapCVStereoBM` • `LeapCVStereoSGBM` • `LeapCVStereoVar`

Table D.3: `LeapCVStereoMatcher` *CRC*

Class Name	`LeapCVStereoVar`
Responsibilities	• Implements `LeapCVStereoMatcher` interface. • Handles all default parameters and parameter updating for `org.opencv.contrib.StereoVar` class.
Collaborations	• `org.opencv.contrib.StereoVar` • `LeapCVStereoMatcher`

Table D.4: `LeapCVStereoVar` *CRC*

Class Name	`LeapCVStereoSGBM`
Responsibilities	• Implements `LeapCVStereoMatcher` interface. • Handles all default parameters and parameter updating for `org.opencv.calib3d.StereoSGBM` class.
Collaborations	• `org.opencv.calib3d.StereoSGBM` • `LeapCVStereoMatcher`

Table D.5: `LeapCVStereoSGBM` *CRC*

Class Name	`LeapCVStereoBM`
Responsibilities	• Implements `LeapCVStereoMatcher` interface. • Handles all default parameters and parameter updating for `org.opencv.calib3d.StereoBM` class.
Collaborations	• `org.opencv.calib3d.StereoBM` • `LeapCVStereoMatcher`

Table D.6: `LeapCVStereoBM` *CRC*

Class Name	`MatcherType`
Responsibilities	• Enumeration of implemented `LeapCVStereoMatcher` types. • Provides flag for `LeapCVStereoMatcherFactory`.
Collaborations	• `org.opencv.calib3d.StereoSGBM` • `LeapCVStereoUtils` • `LeapCVStereoMatcherFactory`

Table D.7: `MatcherType` *CRC*

Appendix E

Sequence Diagrams

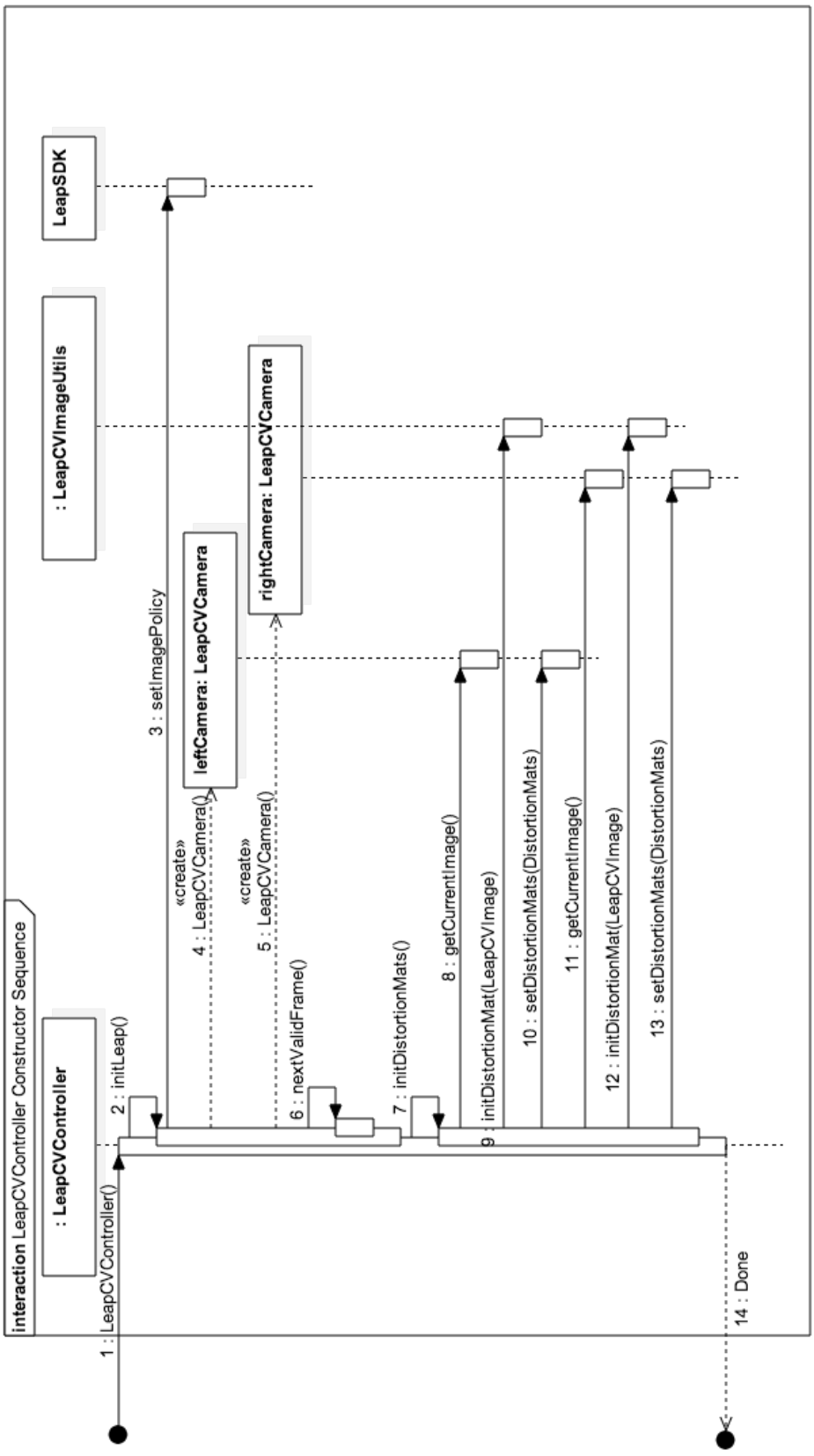

Figure E.1: `LeapCVController` *Constructor Sequence Diagram*

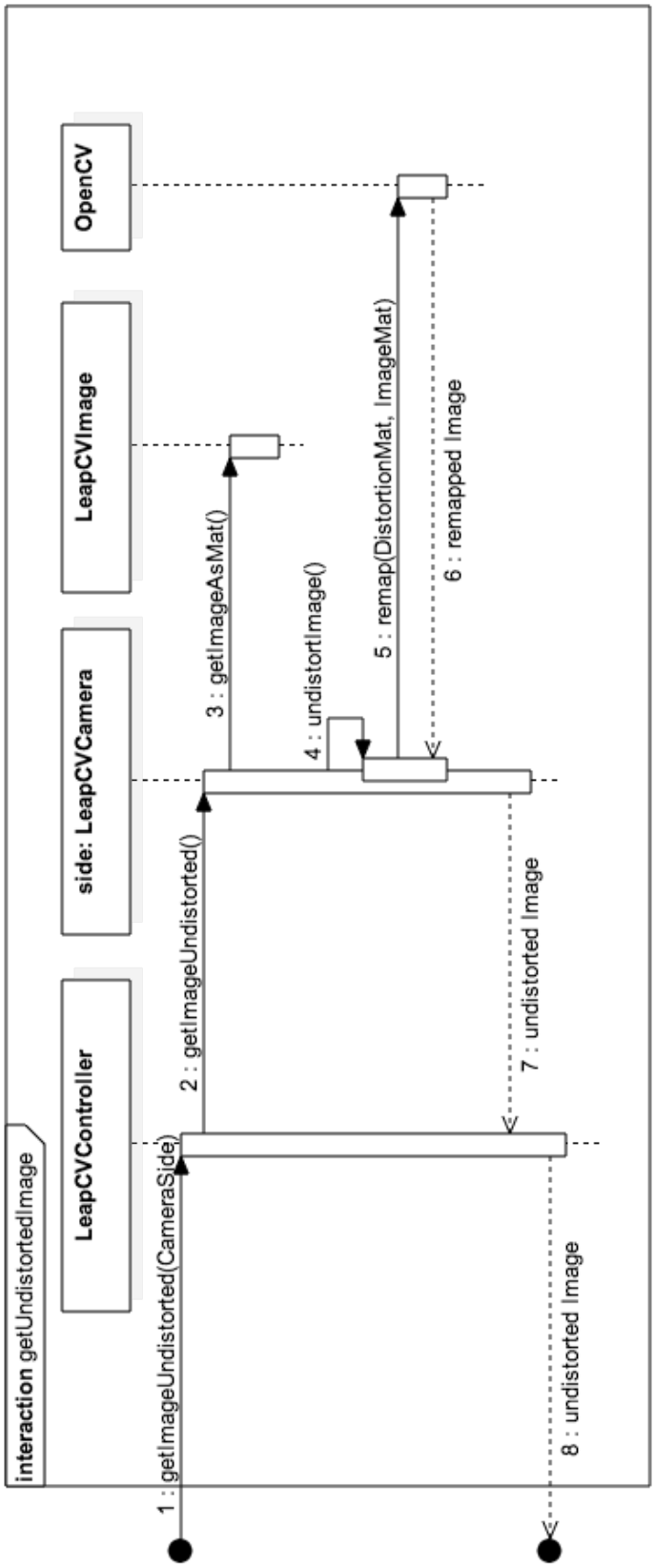

Figure E.2: `getUndistortedImage` *Sequence Diagram*

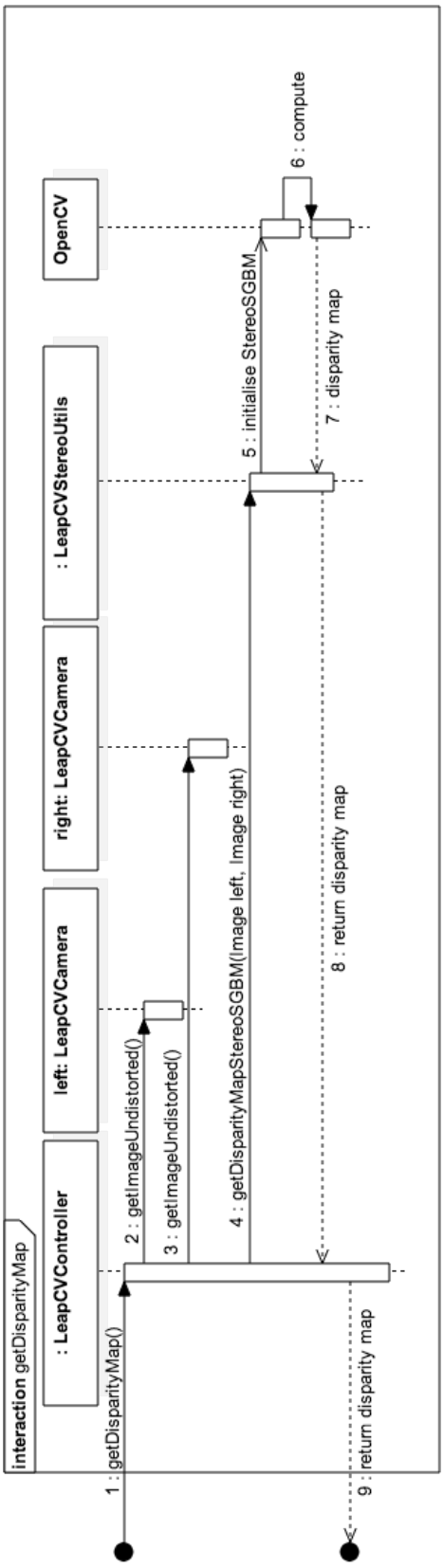

Figure E.3: `getDisparityMap` *Sequence Diagram*

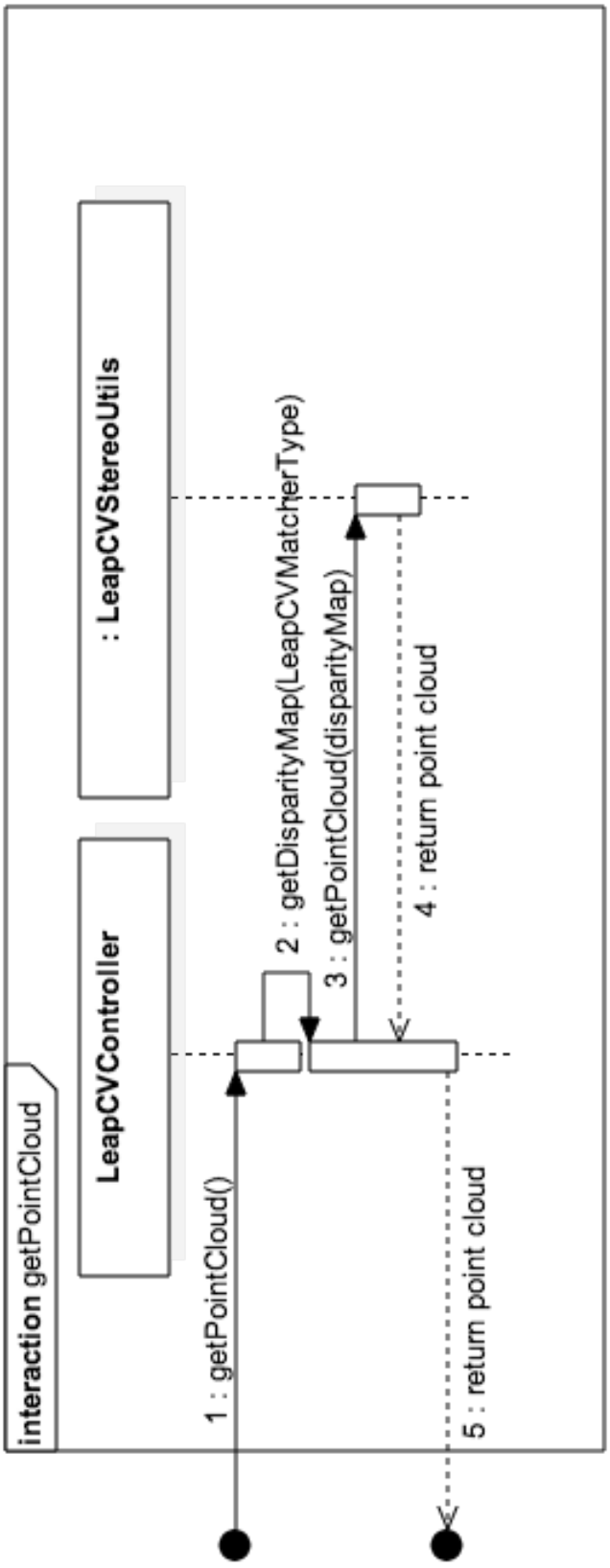

Figure E.4: `getPointCloud` *Sequence Diagram*

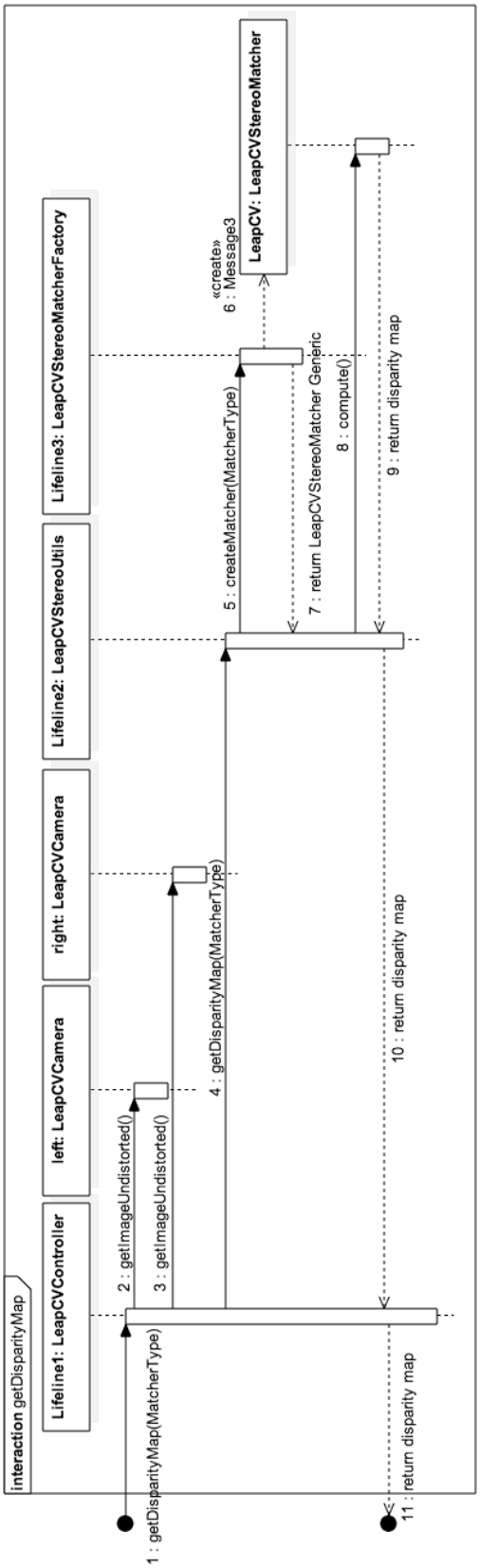

Figure E.5: Updated `getDisparityMap` *Sequence Diagram*

Appendix F

LeapCV Javadoc Example

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.leapcv

Class LeapCVImageUtils

java.lang.Object
com.leapcv.utils.LeapCVImageUtils

```
public class LeapCVImageUtils
extends java.lang.Object
```

Field Summary

Fields

Modifier and Type	Field and Description
static int	IMAGE_HEIGHT
static int	IMAGE_WIDTH
static java.lang.String	LEFT_IMAGE_KEY
static java.lang.String	RIGHT_IMAGE_KEY
static java.lang.String	X_MAT_KEY
static java.lang.String	Y_MAT_KEY

Constructor Summary

Constructors

Constructor and Description
LeapCVImageUtils()

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method and Description
static org.opencv.core.Mat	convertToMat(com.leapmotion.leap.Image image) Convert a leap type Image to an OpenCV Mat
static org.opencv.core.Mat	crop(org.opencv.core.Mat image, double percentageCrop) Crops even percentage from each side of an image
static org.opencv.core.Mat	denoise(org.opencv.core.Mat image, double denoisingFactor)
static org.opencv.core.Mat	gaussianBlur(org.opencv.core.Mat image)
static java.util.Map<java.lang.String,org.opencv.core.Mat>	initDistortionMat(com.leapmotion.leap.Image image) Initialise the distortion matrices for use with OpenCV Imgproc method.
static javafx.scene.image.Image	matToWritableImage(org.opencv.core.Mat image) Turn a Mat into a WritableImage, useful for displaying in JavaFX
static org.opencv.core.Mat	medianBlur(org.opencv.core.Mat image)
static java.awt.image.BufferedImage	toBufferedImage(com.leapmotion.leap.Image image) Turn a leap motion Image type into a BufferedImage
static javafx.scene.image.WritableImage	toWritableImage(java.awt.image.BufferedImage image) Turn a BufferedImage into a WritableImage, useful for displaying in JavaFX

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Figure F.1: LeapCV Javdadocs

Appendix G

Implementation Code Snippets

```
1  public BufferedImage toUndistortedImage(Image image){
2      int targetWidth = image.width();
3      int targetHeight = image.heigh();
4      int type = BufferedImage.TYPE_BYTE_GRAY;
5      int bufferSize = targetWidth*targetHeight;
6      byte[] pixels = new byte[bufferSize];
7      int brightness = 0;
8
9      // Loop through each pixel in the image
10     for(double y = 0; y < targetHeight; y++){
11         for(double x = 0; x < targetWidth; x++){
12
13             // Normalise the slope for the current pixel
14             Vector input = new Vector((float)x/targetWidth,
                   (float)y/targetHeight, 0);
15
16             //Convert from normalized [0..1] to slope [-4..4]
17             input.setX((input.getX() - image.rayOffsetX()) /
                   image.rayScaleX());
18             input.setY((input.getY() - image.rayOffsetY()) /
                   image.rayScaleY());
19
20             // Look up the pixel coordinates in the raw image
21             // corresponding to the slope values.
22             Vector pixel = image.warp(input);
23
24             // Check that the pixel coordinates are actually valid...
25             if((pixel.getX() >= 0) &&
```

```
26                  (pixel.getX() < targetWidth) &&
27                  (pixel.getY() >= 0) &&
28                  (pixel.getY() < targetHeight)) {
29
30                   // Convert the discrete x and y coordinates into a data buffer
                         index
31                   int data_index = (int)(
32                       Math.floor(pixel.getY())
33                       * targetWidth
34                       + Math.floor(pixel.getX())
35                   );
36
37                   // Look up brightness value at current pixel
38                   brightness = image.data()[data_index] & 0xff;
39
40              } else {
41                   //Display invalid pixels as red
42                   brightness = 255;
43
44              }
45              // Copy the brightness value into the new undistorted image
46              pixels[(int) Math.floor(y * targetWidth + x)] =  (byte) brightness;
47          }
48      }
49
50      // Turn the undistorted image into a buffered image
51      BufferedImage bufferedImage = new BufferedImage(targetWidth, targetHeight,
            type);
52      final byte[] targetPixels = ((DataBufferByte)
            bufferedImage.getRaster().getDataBuffer()).getData();
53      System.arraycopy(pixels, 0, targetPixels, 0, pixels.length);
54
55      return bufferedImage;
56  }
```

Listing G.1: Slow Method of `Image` *Distortion Removal*

```
1  package com.leapcv;
2
3  import com.leapcv.utils.LeapCVStereoUtils;
4  import junit.framework.TestCase;
5  import org.opencv.core.Core;
6  import org.opencv.core.Mat;
7  import org.opencv.highgui.Highgui;
8
9  import java.util.List;
10
11 /**
12  * Test the performance of the LeapCV functionality
13  */
14 public class LeapCVPerformanceTest extends TestCase {
15     LeapCVController controller;
16     Mat leftImage = null;
17     Mat rightImage = null;
18     List<LeapCVCamera> cameras = null;
19     final int NUM_FRAMES = 100;
20     final int MIN_FRAMERATE = 15;
21
22     public void setUp() throws Exception {
23         super.setUp();
24         System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
25         this.controller = new LeapCVController();
26     }
27
28     public void testFrameRateGetPointCloud(){
29         long start = 0;
30         long stop = 0;
31         Mat dispMat = null;
32         Mat pointMat = null;
33         LeapCVStereoUtils utils = new LeapCVStereoUtils();
34
35         //  Start timer
36         start = System.nanoTime();
37
38         for(int i = 0; i < NUM_FRAMES; ++i){
39             leftImage = this.controller.getLeftImageUndistorted();
40             rightImage = this.controller.getRightImageUndistorted();
41             dispMat = utils.getDisparityMap(leftImage,rightImage);
42             pointMat = utils.getPointCloud(dispMat);
43             this.controller.nextValidFrame();
44         }
```

```
45
46          //  Stop timer
47          stop = System.nanoTime();
48
49          // Calculate Framerate
50          long elapsed = stop - start;
51          double seconds = (double)elapsed/1000000000.0;
52          double fps = NUM_FRAMES/seconds;
53
54          //  Print Framerate
55          System.out.println("FPS GET PC: " + fps);
56
57          Highgui.imwrite("/Volumes/macintosh_hdd/Users/daniel/Desktop/dispTest.png",
                dispMat);
58
59          assertTrue(fps > MIN_FRAMERATE);
60      }
61  }
```

Listing G.2: Example JUnit Performance Test Case

```
1  package com.leapcv;
2
3  import com.leapcv.LeapCVController;
4  import junit.framework.TestCase;
5  import org.opencv.core.Core;
6  import org.opencv.core.Mat;
7  import org.opencv.highgui.Highgui;
8
9  import java.util.List;
10
11 public class LeapCVControllerTest extends TestCase {
12     LeapCVController controller;
13     Mat leftImage = null;
14     Mat rightImage = null;
15     Mat leftImageUndistorted = null;
16     Mat rightImageUndistorted = null;
17     List<LeapCVCamera> cameras = null;
18
19     public void setUp() throws Exception {
20         super.setUp();
21         System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
22         this.controller = new LeapCVController();
23     }
24
25     public void testNextValidFrame() throws Exception {
26         this.controller.nextValidFrame();
27
28     }
29
30     public void testGetLeftImage() throws Exception {
31         leftImage = this.controller.getLeftImage();
32         assertNotNull(leftImage);
33     }
34
35     public void testGetRightImage() throws Exception {
36         rightImage = this.controller.getRightImage();
37         assertNotNull(rightImage);
38     }
39
40     public void testGetLeftImageUndistorted() throws Exception {
41         leftImageUndistorted = this.controller.getLeftImageUndistorted();
42         assertNotNull(leftImageUndistorted);
43     }
44
```

```
45      public void testGetRightImageUndistorted() throws Exception {
46          rightImageUndistorted = this.controller.getRightImageUndistorted();
47          assertNotNull(rightImageUndistorted);
48      }
49
50      public void testGetCameras() throws Exception {
51          this.cameras = this.controller.getCameras();
52          assertNotNull(cameras);
53
54          assertEquals(2, cameras.size());
55          assertEquals("Cameras in wrong order", LeapCVCamera.LEFT_ID,
                cameras.get(LeapCVCamera.LEFT_ID).getSide().getSideId());
56          assertEquals("Cameras in wrong order",LeapCVCamera.RIGHT_ID,
                cameras.get(LeapCVCamera.RIGHT_ID).getSide().getSideId());
57          assertNotSame("Cameras same", cameras.get(0), cameras.get(1));
58
59          leftImage = this.controller.getLeftImage();
60          rightImage = this.controller.getRightImage();
61          leftImageUndistorted = this.controller.getLeftImageUndistorted();
62          rightImageUndistorted = this.controller.getRightImageUndistorted();
63
64          this.writeImages();
65      }
66
67      private void writeImages(){
68          final String path =
                "/Volumes/macintosh_hdd/Users/daniel/Desktop/disp/testData/";
69          Highgui.imwrite(path + "leftImage.png", leftImage);
70          Highgui.imwrite(path + "rightImage.png", rightImage);
71          Highgui.imwrite(path + "leftImageUndistorted.png",
                leftImageUndistorted);
72          Highgui.imwrite(path + "rightImageUndistorted.png",
                rightImageUndistorted);
73
74      }
75  }
```

Listing G.3: Example JUnit Unit Test Case

```
1  public class LeapCVCalibrationUtils {
2
3      private List<LeapCVCamera> cameras = null;
4      private List<Mat> objectPoints = null;
5      private Map<Integer, List<Mat>> imagePoints = null;
6      private Map<Integer, List<Mat>> corners = null;
7      private Size patternSize = null;
8
9      public LeapCVCalibrationUtils(LeapCVCamera left, LeapCVCamera right) {
10         this.cameras = new ArrayList<>();
11         this.objectPoints = new ArrayList<>();
12         this.imagePoints = new HashMap<>();
13         this.corners = new HashMap<>();
14         this.patternSize = new Size(9, 6);
15         this.cameras.add(left);
16         this.cameras.add(right);
17
18         System.out.println("Size 1: " + left.getImageUndistorted().size());
19         System.out.println("Size 2: " + right.getImageUndistorted().size());
20     }
21
22     /**
23      * Find the corners of a 9x6 chessboard
24      */
25     public void findChessboardCorners() {
26         for (LeapCVCamera camera : cameras) {
27             Mat image = new Mat();
28             camera.getImageUndistorted().copyTo(image);
29
30             Mat corners = new MatOfPoint2f();
31             boolean cornersFound = Calib3d.findChessboardCorners(image,
                   patternSize, (MatOfPoint2f) corners);
32             Calib3d.drawChessboardCorners(image, patternSize, (MatOfPoint2f)
                   corners, cornersFound);
33             Highgui.imwrite("chess_" + image.toString() + ".png", image);
34
35             if (this.corners.get(camera.getSide().getSideId()) == null) {
36                 this.corners.put(camera.getSide().getSideId(), new
                       ArrayList<Mat>());
37             }
38
39             this.corners.get(camera.getSide().getSideId()).add(corners);
40             this.objectPoints.add(create3dChessboardCorners(patternSize, 3));
41             System.out.println(camera.getSide().getSideId());
```

```
42              System.out.println(cornersFound);
43          }
44      }
45
46      /**
47       * Draw corners on the chessboard in an image
48       * @param image
49       * @return
50       */
51      public static Mat getImageWithChessboard(Mat image) {
52              Size patternSize = new Size(9,6);
53              Mat corners = new MatOfPoint2f();
54              Mat chessImage = Mat.zeros(image.size(), image.type());
55
56              image.copyTo(chessImage);
57
58              boolean cornersFound = Calib3d.findChessboardCorners(chessImage,
                    patternSize, (MatOfPoint2f) corners);
59              Imgproc.cornerSubPix(chessImage, (MatOfPoint2f)corners, new
                    Size(11,11), new Size(-1,-1), new
                    TermCriteria(TermCriteria.MAX_ITER,30,0.1));
60
61              Imgproc.cvtColor(image,chessImage,Imgproc.COLOR_GRAY2RGB);
62              Calib3d.drawChessboardCorners(chessImage, patternSize,
                    (MatOfPoint2f) corners, cornersFound);
63              return chessImage;
64      }
65
66      /**
67       * Generate Mat based on size of chessboard
68       * @param boardSize size of the chessboard
69       * @param squareSize size of the chessboard square sides in cm
70       * @return {@link org.opencv.core.Mat}
71       */
72      public Mat create3dChessboardCorners(Size boardSize, float squareSize) {
73          MatOfPoint3f corners = new MatOfPoint3f();
74
75          for (int i = 0; i < boardSize.height; ++i) {
76              for (int j = 0; j < boardSize.width; ++j) {
77                  corners.push_back(new MatOfPoint3f(new Point3(j * squareSize,
                        i * squareSize, 0)));
78              }
79          }
80          return corners;
```

```
81      }
82
83      /**
84       * Carry out calibration procedure on the leap motion cameras
85       */
86      public void calibrateLeapCameras() {
87          Mat rotationMatrix = new Mat();
88          Mat fundamentalMatrix = new Mat();
89          Mat essentialMatrix = new Mat();
90          Mat tranMat = new Mat();
91          Mat leftCameraMat = new Mat();
92          Mat leftCameraCoeffs = new Mat();
93          Mat rightCameraMat = new Mat();
94          Mat rightCameraCoeffs = new Mat();
95
96          //  Criteria for calibration
97          TermCriteria tc = new TermCriteria(TermCriteria.MAX_ITER +
                TermCriteria.EPS, 100, 1e-5);
98
99          //  Calibrate cameras
100         Calib3d.stereoCalibrate(objectPoints,
101                 this.corners.get(CameraSide.RIGHT.getSideId()),
102                 this.corners.get(CameraSide.LEFT.getSideId()),
103                 leftCameraMat,
104                 leftCameraCoeffs,
105                 rightCameraMat,
106                 rightCameraCoeffs,
107                 cameras.get(0).getImageUndistorted().size(),
108                 rotationMatrix,
109                 tranMat,
110                 essentialMatrix,
111                 fundamentalMatrix,
112                 tc,
113                 Calib3d.CALIB_FIX_ASPECT_RATIO +
114                         Calib3d.CALIB_ZERO_TANGENT_DIST +
115                         Calib3d.CALIB_SAME_FOCAL_LENGTH +
116                         Calib3d.CALIB_RATIONAL_MODEL);
117
118     }
119
120 }
```

Listing G.4: Failed Calibration Attempt

```
/**
  * Match image features
  */
 public Mat match(Mat left, Mat right) {
     MatOfDMatch matches = new MatOfDMatch();
     MatOfKeyPoint leftKeyPoints = getFeatures(left);
     MatOfKeyPoint rightKeyPoints = getFeatures(right);
     Mat leftDescriptors = getFeatureDescriptors(left);
     Mat rightDescriptors = getFeatureDescriptors(right);

     //  Only try and match if some features have been found
     if ((!leftKeyPoints.empty()) && (!rightKeyPoints.empty())) {
         //  Match the features
         matcher.match(leftDescriptors, rightDescriptors, matches);
     }

     //  Remove any outliers from the matches
     matches = this.removeOutliers(matches);

     //  Prepare keypoints for finding perspective transform
     List<DMatch> matchesList = matches.toList();
     List<KeyPoint> leftKeyPointsList = leftKeyPoints.toList();
     List<KeyPoint> rightKeyPointsList = rightKeyPoints.toList();
     List<Point> objectList = new ArrayList<>();
     List<Point> sceneList = new ArrayList<>();

     for (int i = 0; i < matchesList.size(); i++) {
         objectList.add(rightKeyPointsList.get(matchesList.get(i).trainIdx).pt);
         sceneList.add(leftKeyPointsList.get(matchesList.get(i).queryIdx).pt);
     }

     MatOfPoint2f objects = new MatOfPoint2f();
     MatOfPoint2f scene = new MatOfPoint2f();
     objects.fromList(objectList);
     scene.fromList(sceneList);

     //  Find the perspective transformation between the images
     Mat H = Calib3d.findHomography(objects, scene, Calib3d.RANSAC, 5);
     Mat objectCorners = new Mat(4, 1, CvType.CV_32FC2);
     Mat sceneCorners = new Mat(4, 1, CvType.CV_32FC2);

     //  Initialise perfect square of original image
     objectCorners.put(0, 0, 0, 0);
     objectCorners.put(1, 0, right.cols(), 0);
```

```
45          objectCorners.put(2, 0, right.cols(), right.rows());
46          objectCorners.put(3, 0, 0, right.rows());
47
48          //  Transform the square in objectCorners to match the transformation
                given in H
49          Core.perspectiveTransform(objectCorners, sceneCorners, H);
50
51          //  Draw the matches in both images, with joining line between them,
                into a new image
52          Mat image = new Mat();
53          Features2d.drawMatches(left, leftKeyPoints, right, rightKeyPoints,
                matches, image,
54                  new Scalar(255, 0, 0), new Scalar(0, 255, 255),
55                  new MatOfByte(), Features2d.NOT_DRAW_SINGLE_POINTS);
56
57          //  Draw square with transformed corners from sceneCorners
58          Core.line(image, new Point(sceneCorners.get(0, 0)), new
                Point(sceneCorners.get(1, 0)), new Scalar(0, 255, 0), 1);
59          Core.line(image, new Point(sceneCorners.get(1, 0)), new
                Point(sceneCorners.get(2, 0)), new Scalar(0, 255, 0), 1);
60          Core.line(image, new Point(sceneCorners.get(2, 0)), new
                Point(sceneCorners.get(3, 0)), new Scalar(0, 255, 0), 1);
61          Core.line(image, new Point(sceneCorners.get(3, 0)), new
                Point(sceneCorners.get(0, 0)), new Scalar(0, 255, 0), 1);
62
63
64          return image;
65
66      }
```

Listing G.5: Feature Detection

```
1   /**
2    * Remove outliers from matched features
3    *
4    * @param matches
5    * @return
6    */
7   private MatOfDMatch removeOutliers(MatOfDMatch matches) {
8       MatOfDMatch goodMatches = new MatOfDMatch();
9       double max_dist = 0;
10      double min_dist = 100;
11
12      for (int i = 0; i < matches.rows(); i++) {
13          double dist = matches.toList().get(i).distance;
14          System.out.println(dist);
15          if (dist < min_dist)
16              min_dist = dist;
17          if (dist > max_dist)
18              max_dist = dist;
19      }
20
21      for (int i = 0; i < matches.rows(); i++) {
22          if (matches.toList().get(i).distance < 3.5 * min_dist) {
23              MatOfDMatch goodMatch = new
                    MatOfDMatch(matches.toList().get(i));
24              goodMatches.push_back(goodMatch);
25          }
26      }
27
28      return goodMatches;
29  }
```

Listing G.6: Outlier Removal

```
1  /**
2  *  Code to remove distortion from image using bilinear interpolation,
       converted from C++, (Lorenzo 2014).
3  */
4  public BufferedImage undistortImage(Image image){
5          double destinationWidth = 640;
6          double destinationHeight = 240;
7          byte[][] destination = new
               byte[(int)destinationWidth][(int)destinationHeight];
8
9          //define needed variables outside the inner loop
10         double calX, calY, weightX, weightY, dX1, dX2, dX3, dX4, dY1, dY2,
               dY3, dY4, dX, dY;
11         int x1, x2, y1, y2, denormalizedX, denormalizedY;
12         int x, y;
13
14         final byte[] raw = image.data();
15         final float[] distortion_buffer = image.distortion();
16
17         //Local variables for values needed in loop
18         final int distortionWidth = image.distortionWidth();
19         final int width = image.width();
20         final int height = image.height();
21
22         for (x = 0; x < destinationWidth; ++x) {
23             for (y = 0; y < destinationHeight; ++y) {
24                 //Calculate the position in the calibration map (still with a
                       fractional part)
25                 calX = 63 * x/destinationWidth;
26                 calY = 63 * y/destinationHeight;
27                 //Save the fractional part to use as the weight for
                       interpolation
28
29                 weightX = calX - Math.floor(calX);
30                 weightY = calY - Math.floor(calY);
31
32                 //Get the x,y coordinates of the closest calibration map
                       points to the target pixel
33                 x1 = (int)calX; //Note truncation to int
34                 y1 = (int)calY;
35                 x2 = x1 + 1;
36                 y2 = y1 + 1;
37
38                 //Look up the x and y values for the 4 calibration map points
```

```
                    around the target
39                  // (x1, y1)  ..  .. .. (x2, y1)
40                  //    ..                ..
41                  //    ..    (x, y)      ..
42                  //    ..                ..
43                  // (x1, y2)  ..  .. .. (x2, y2)
44                  dX1 = distortion_buffer[x1 * 2 + y1 * distortionWidth];
45                  dX2 = distortion_buffer[x2 * 2 + y1 * distortionWidth];
46                  dX3 = distortion_buffer[x1 * 2 + y2 * distortionWidth];
47                  dX4 = distortion_buffer[x2 * 2 + y2 * distortionWidth];
48                  dY1 = distortion_buffer[x1 * 2 + y1 * distortionWidth + 1];
49                  dY2 = distortion_buffer[x2 * 2 + y1 * distortionWidth + 1];
50                  dY3 = distortion_buffer[x1 * 2 + y2 * distortionWidth + 1];
51                  dY4 = distortion_buffer[x2 * 2 + y2 * distortionWidth + 1];
52
53                  //Bilinear interpolation of the looked-up values:
54                  // X value
55                  dX = dX1 * (1 - weightX) * (1- weightY) + dX2 * weightX * (1 -
                        weightY) + dX3 * (1 - weightX) * weightY + dX4 * weightX *
                        weightY;
56
57                  // Y value
58                  dY = dY1 * (1 - weightX) * (1- weightY) + dY2 * weightX * (1 -
                        weightY) + dY3 * (1 - weightX) * weightY + dY4 * weightX *
                        weightY;
59
60                  // Reject points outside the range [0..1]
61                  if((dX >= 0) && (dX <= 1) && (dY >= 0) && (dY <= 1)) {
62                      //Denormalize from [0..1] to [0..width] or [0..height]
63                      denormalizedX = (int) (dX * width);
64                      denormalizedY = (int) (dY * height);
65
66                      //look up the brightness value for the target pixel
67                      destination[x][y] = raw[denormalizedX + denormalizedY *
                            width];
68                  } else {
69                      destination[x][y] = -1;
70                  }
71
72
73              }
74          }
75          byte[] b = new byte[640*240];
76          int i = 0;
```

```
77          for(y = 0; y < destinationHeight; ++y){
78                  for(x = 0; x < destinationWidth; ++x){
79                          b[i++] = destination[x][y];
80                  }
81          }
82      BufferedImage bufferedImage = new
            BufferedImage(image.width(),image.height(),
            BufferedImage.TYPE_BYTE_GRAY);
83      final byte[] targetPixels = ((DataBufferByte)
            bufferedImage.getRaster().getDataBuffer()).getData();
84      System.arraycopy(b, 0, targetPixels, 0, b.length);
85      return bufferedImage;
86  }
```

Listing G.7: Removal of Distortion from an Image using Bilinear Interpolation. Converted from C++ to Java