An Open Source Computer Vision Framework for Leap Motion

Daniel Hamilton 10026535,

Computer Science,

University of the West of England.

April 4, 2015

Abstract

Acknowledgements

Contents

List of Figures

List of Tables

List of Listings

Word Count: 10823.

Chapter 1

Introduction

1.1 Computer Vision: An Intellectual Frontier

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). 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?

The Leap Motion Controller 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. It provides functionality out of 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
SDK, allowing access to more elements of their Leap Motion Controller. Namely,
images from the two cameras. This could possibly be exploited in computer
vision. The SDK is closed source, therefore it is not known the current methods
by which hands and gestures are classified (Bachmann et al. 2014, p. 217). With
the access to the images it allows exploration of other, custom methods for object
classification and tracking. The SDK is for C++, C#, Objective-C, Java, Python
and Javascript. It works 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 leap motion is an extremely accessible device and it 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 Leap Motion Controller 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 SDK. 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 stated previously,

---

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

image access has now been given, and so a whole new future of possibilities have 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 for the leap motion, it might become of use in the field of robotics. For example, it's size would make it suitable to attach to a robot as a rudimentary depth sensing system for collision avoidance.

1.3   Current State-of-the-Art

1.3.1   OpenCV

OpenCV is an extensive Computer Vision framework 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 acceleration was added to OpenCV. It has been developed so that adoption from the CPU 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 iOS and Java/Android platforms. 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).

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.

OpenCV is licensed under the BSD 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.

1.4   The Problem

Currently, there is no simple solution that allows a developer to simply plug in a leap motion 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.

1.5   The Purpose

The main purpose is to try and expand the areas on the leap motion can be developed with. To try and find out whether it is feasible to say that a leap motion can be used for computer vision in a simple way. For example, to detect 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 before, 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 SDK. 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.

1.6   Hypothesis

It is possible to use the leap motion 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 leap motion as a solution, is low, making it an option to consider in new projects.

1.7   Solution Approach

It is envisaged that the framework will be made possible by taking advantage of functionality of the OpenCV library. Then exploiting the new image functionality of the leap motion SDK as mentioned earlier. The next few sections will outline the processes that will be followed to keep the project within a realistic scope and enable its completion.

1.8   Aim and Objectives

This section outlines what the scope of the project is to be. The success of this project will be calculated based on the completion of each of the aim and objectives.

1.8.1   Aim

- To develop an open source Computer Vision framework usable by the leap motion community.

- Enable the leap motion to recognise simple three dimensional shapes.

1.8.2   Objectives

- Research methods for stereoscopic image object recognition.

- Research stereoscopic image processing methods.

- Design and develop a framework that allows the leap motion to be used for computer vision.

- Evaluate the accuracy of object recognition.

- Release an open source framework.

- Produce a report that communicates my findings

1.9   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. With the first iteration being an example of a throw away prototype so as to explore technical uncertainties and ascertain initial requirements and scope estimate. The later iterations will then follow more of an evolutionary prototype approach. Where a large upfront requirements analysis and design takes place in the first of the evolutionary prototypes. This will be based on knowledge gained in the first iteration but not so much the implementation from the first iteration. It will aim to have a good base structure for the rest of the iterations to be evolved from. With tweaks to requirements happening in later iterations based on the evaluation of the previous implementation. Figure 1.1 gives a visual representation of the iterative approach.
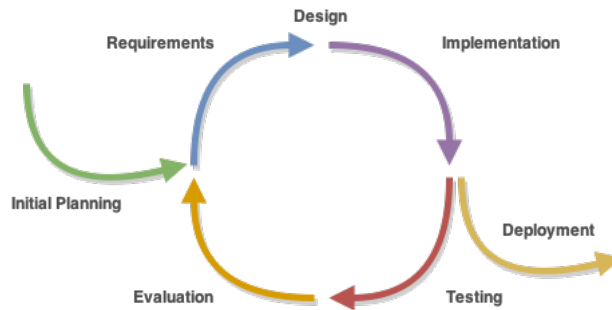
Figure 1.1: Overview of an iterative development methodology

1.9.1 Planned Iterations

During each iteration a test bench application will be developed that will make use of the framework. The development of this prototype will depend upon what level to which the framework is completed. Throughout each iteration sufficient unit tests will be implemented based on the requirements. Below is an outline of

what work is aimed to be completed in each iteration. The work breaks down three discrete sections

1. Image processing

2. Object detection

3. Classification

Each section in the order of the list, is dependant on the previous work being completed. Therefore each iteration will aim to solve the work break down in order. 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 leap motion SDK. Image processing and stereo image processing will be explored here. Once completed it will allow better requirements to be established for the framework.

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 whole framework will be based. The initial requirements will be established here so that a high level design can be accomplished. 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 leap motion SDK.

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 here.

1.9.2   Project Timeline

Table 1.1 gives a rough outline as to when work will get 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

1.9.3 Tools

The framework 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 Ubuntu 14.04 but also on Mac OS X 10.10 and Windows 8.1 to ensure multi platform capability. Due to time constraints the project will only get fully tested on Ubuntu. JUnit 4 will be used for unit testing.

For the leap motion, version 2.2.3 of the SDK 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). Using a VCS, records 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.

1.9.4 Testing Methods

Testing will be carried out to ensure that the framework 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 as they are developed.

- **Performance testing** this will be used to ensure that some of the non-functional requirements have been met.

- **Acceptance testing** 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 get run in the later iterations, to ensure that no functionality has been broken when changes get made in the code. Most of the testing results will be discussed in chapter 4 however due to the close relationship with the use cases the acceptance tests will be shown in chapter 3.

1.10 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.

- **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 5** will evaluate the work and layout a call for future work based on what has been completed.

- **Chapter 6**

1.11 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.

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 of a similar object at a later date. However the problem isn't as simple as just recognising an object. Humans can also recognise the context of an object. A human can recognise the difference in intention of 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 (Forsyth et al. 2012).

Object recognition in digital images is not a new problem. And neither is the specific problem of three dimensional object detection. 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). 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   Stereopsis 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 Leap Motion, 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 step is a disparity map, where the
   disparities are the differences in x-coordinates on the image planes of the
   same feature viewed in the left and right cameras: $x^l - x^r$.

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 like
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,
not needing 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.

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
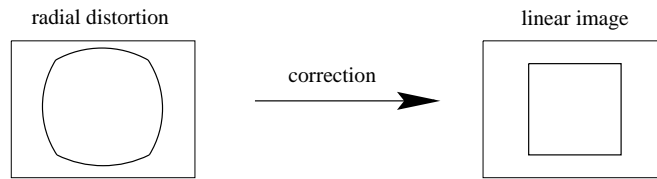
Figure 2.1: Example of radial distortion of a square being corrected (Hartley & Zisserman 2003)

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 here. 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.

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 I advise you read 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
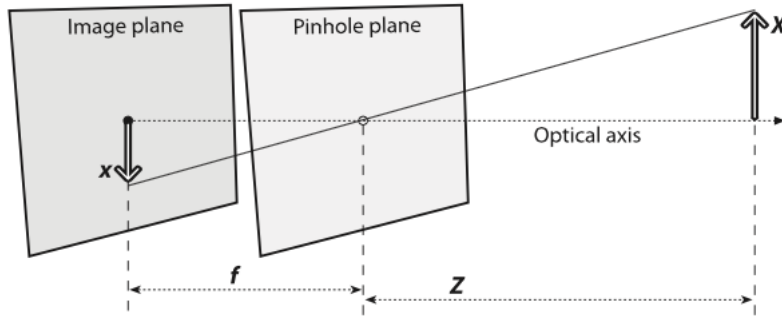
Figure 2.2: Pinhole camera model as shown in Bradski & Kaehler (2008, p. 372)

travels through a pinhole that is made in the pinhole plane. This light then gets 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 image on the image plane. Therefore we can work out the size of the object using the equation $-x = f\dfrac{X}{Z}$. For mathematical simplification Bradski & Kaehler (2008) states that the negative value can be ommited leaving $x = f\dfrac{X}{Z}$. 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 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 mathematical model of $x_{screen} = f_x \left(\dfrac{X}{Z}\right) + c_x, y_{screen} = f_y \left(\dfrac{Y}{Z}\right) + c_y$. Having a model like this allows us to define the parameters of the camera so that false transformations of light
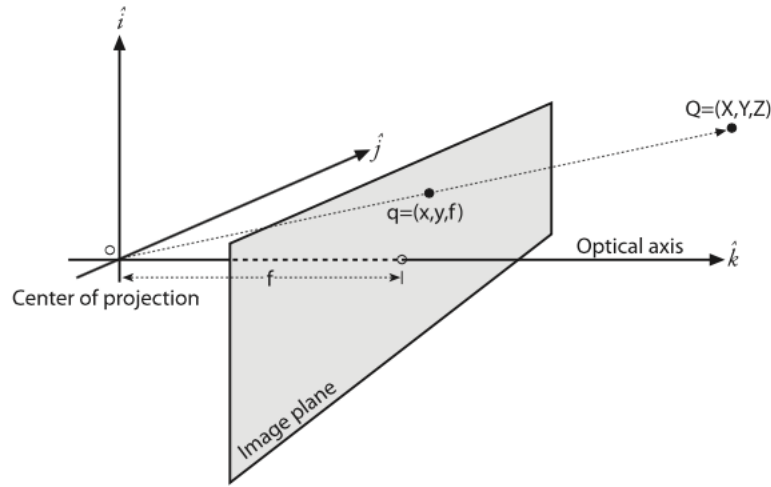
Figure 2.3: Pinhole camera model with image plane in front, as shown in Bradski & Kaehler (2008, p. 372)

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 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 Leap Motion 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

TODO

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 passed in, 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.

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.

Corner 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 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 corner is a more specific term of something that can be described as an image point. This report will continue using the term corner as Bradski & Kaehler (2008) has, so as to avoid confusion with any code references in OpenCV later on.

Corners are easy to match between different images. This makes them extremely useful when reconstructing points in three dimensions when using multiple images.

Such as the images produced in stereopsis. The idea is to extract unique points 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.

2.5   Summary

Chapter 3

Requirements Analysis

3.1 Introduction

This chapter attempts to lay out a set of requirements for the framework, from the point of view of the stakeholders. They will set out a contract between the stakeholders and the framework developer as a means to show what the framework 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 next few 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 the framework to develop other applications. The framework being developed here 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 the framework, 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 leap motion. 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 the framework. 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 this framework 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.

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
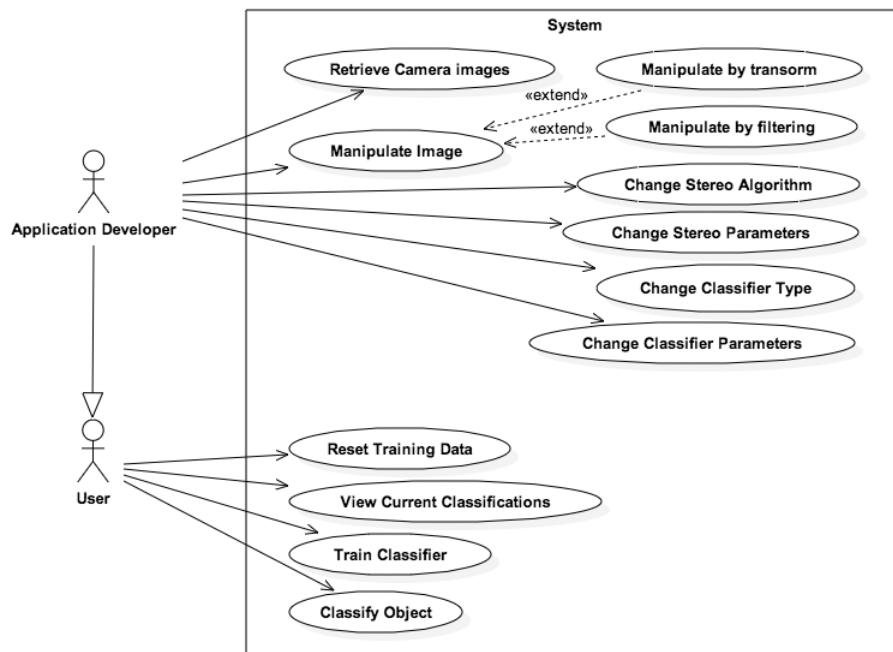
Figure 3.1: UML use case diagram

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

As this is a framework, user interface requirements do not need to be considered.

Automatic camera configuration. The framework should abstract the leap motion configuration away from the application developer and should not require any explicit calibration.

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

3.6   Non-Functional Requirements

Speed. The framework 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. The framework should have equal to or greater than 90% accuracy in its classification ability.

Multi-platform. The framework should be usable on all of the platforms that the leap motion is currently supported on.

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

Semantics. The framework 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, the framework needs to be designed with extensibility in mind.

3.7   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 terms of this project it will just mean a limited value to the libraries 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		
Reset training data		x		
View classification		x		
Classify new object			x	
Classify image			x	

Table 3.2: MoSCoW task prioritisation

The prioritisations have been worked out so that the fundamental elements of the framework 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.

3.8 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 prototype 1 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. 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.

4.2   System Architecture
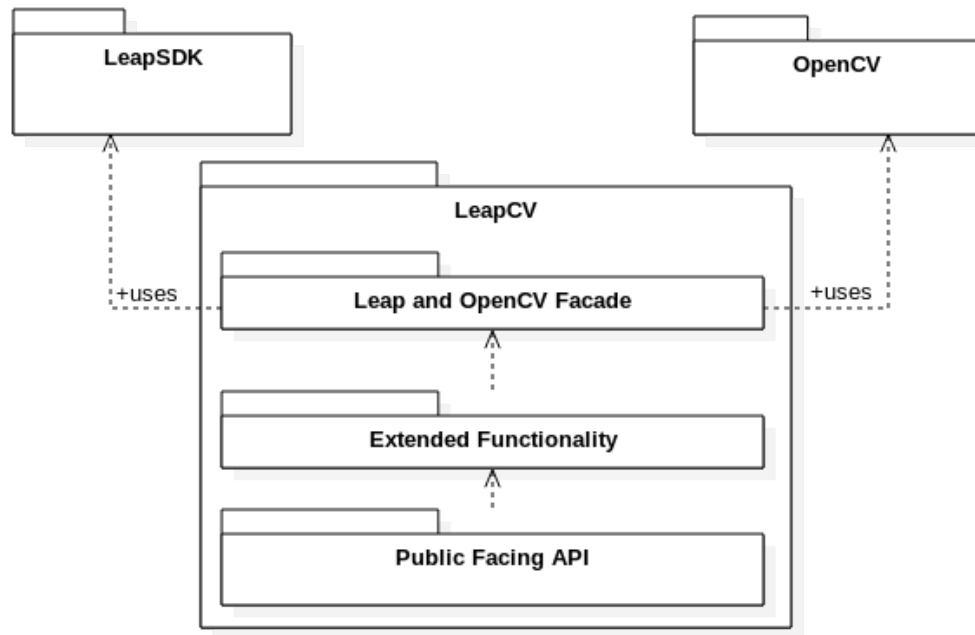
Figure 4.1: High level system architecture

Figure 4.1 outlines the intended architecture of the framework.

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 The framework 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.

The latest version of OpenCV should be downloaded from the website[1]. 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 get loaded when the project gets built it is important to include the line 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 the leap motion website[2]. 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

---

[1]http://www.opencv.org

[2]http://developer.leapmotion.com

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.

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 went badly" and "lessons learned".

4.5   Prototype 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 prototype 4.5. The knowledge gained through this prototype will then be applied to the first of the evolutionary prototypes, prototype 4.6.

4.5.2   Implementation

Stage 1 - Retrieve Images From the leap motion

The first stage involves gaining an understanding of what type of images can be received from the leap motion. 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 leap motion. A Controller must be initialised, 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 (INSERT FIGURE HERE).

```java
public static BufferedImage toBufferedImage(Image image) {
    int type = BufferedImage.TYPE_BYTE_GRAY;
    byte[] imagePixels;

    // Get all the pixels
    imagePixels = image.data();

    // Write all of the pixels to the buffer in a new BufferedImage
    BufferedImage bufferedImage = new BufferedImage(image.width(),
        image.height(), type);
    final byte[] bufferedImagePixels = ((DataBufferByte)
        bufferedImage.getRaster().getDataBuffer()).getData();
    System.arraycopy(imagePixels, 0, bufferedImagePixels, 0,
        imagePixels.length);
    return bufferedImage;
}
```

Listing 4.2: Convert an Image *into a* BufferedImage

```java
// Initialise the Controller
Controller controller = new Controller();

// Set the policy to retrieve images from the camera
controller.setPolicyFlag(PolicyFlag.POLICY_IMAGES);

// 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 leap motion, 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. They suggest 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 will suffice. The listing as appendix C.1 shows the method by which this was done. This method then results in an image as shown in figure (INSERT FIGURE HERE). 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 has been given here.

Stage 3 - Further Removal of Distortion with OpenCV

OpenCV provides functionality to resize and crop images. Therefore here we shall explore the 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[3]. The `Mat` is a matrix, with the same width and height

---

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

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 between 0 and 255. 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 4.5 shows the method used to convert the `Image` into a `Mat`.

```
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. This is quite a generic process to carry out on images and can be viewed in appendix (INSERT APPENDIX HERE). The end result is an image that looks like that in figure (INSERT FIGURE HERE).

Stage 4 - Stereo Correspondence

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 method(TODO) and the later is the stereo variation method. `StereoSGBM` seems to be generally accepted as better than `StereoBM` in most scenarios, therefore `StereoBM` will not be attempted here. (INSERT CODE AND DESCRIPTIONS ON STEREO)

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. It will also give a good idea as to the performance that can be achieved with each of the stereo matching algorithms. 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, SwingBuilder will be used. SwingBuilder 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 (INSERT FIGURE HERE). The sliders visible are used to quickly change the stereo parameters. To allow for the processing to get 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 was 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` seems to lock the GUI thread if it is not done this way. This is possibly due to the fact that by default the leap motion returns images at a very high frame rate. This resulted in finding good parameters for the correspondence algorithms.

4.5.3 Test

As this prototype is intended to be thrown away, it must be noted that the testing of this prototype 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 it needs to be considered that the slow method of distortion removal was implemented here.

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. This will be useful in the next iteration.

What did not go so well?

Using the `Listener` seemed to make the code a lot more complex to work with. It seems more suited to applications that are heavily making use of the LeapSDK with its tracking functionality alone. In this case it is believed it would be better to access the images, by explicitly requesting them from the leap motion. This will make it a lot simpler for development as each image needs to be processed individually. The `Listener` is intended for use when the camera is working at it highest frame rate. However through the frame rate results we have here, we can see that it will never be working at full capacity with the methods intended to be used.

Lessons learned?

In stage 2 of this iteration, it was suggested that the slow method was going to be used for distortion removal, as speed was not important at this stage. Going into the next iteration, an attempt should be made at optimising the process. This is due to the fact using the slow method did indeed have a large hit on the frame rate.

4.6   Prototype 2

4.6.1   Design

Class Diagram

The class diagram shown in figure 4.2 is a basic model of the system that is going to be built upon. Each class was identified using the CRC process. (CITE) CRC
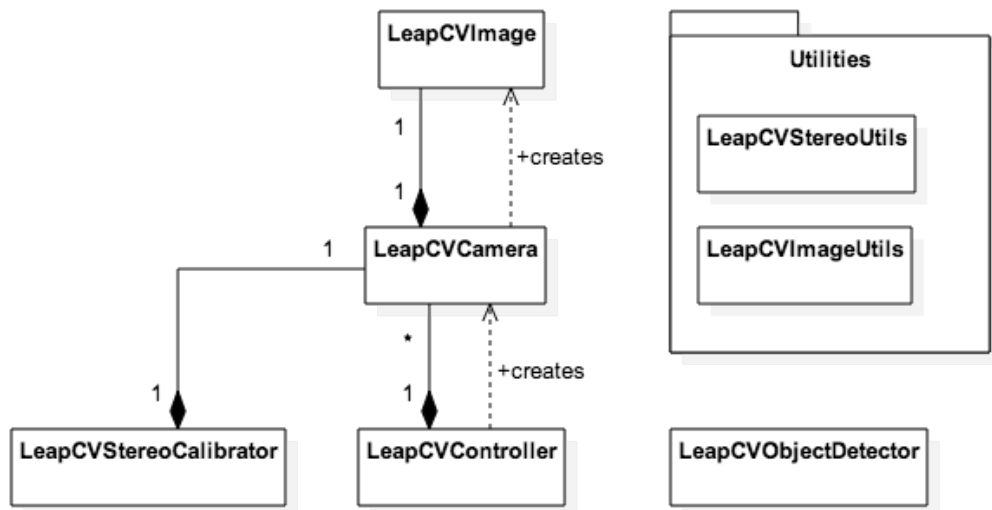
Figure 4.2: Class Diagram for Prototype 2

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 B. The `Utilities` package will be where most of the extended functionality will sit.

Requirements Changes

(TODO)

4.6.2 Implementation

Stage 1 - The Basis of the Framework

As the framework is designed to be extended upon by other application developers, it is useful to think about the way in which it will be deployed. The framework should be IDE independent. The application developers using it should be able to easily import the framework 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 the framework. 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 sometimes known as the skeleton classes. It is also the case that some of the functionality from prototype 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 `distortion()` method. This method contains the calibration map for the leap motion. 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 leap motion. This data is set in the factory, or when the leap motion is explicitly calibrated within its separate settings application. Unless the leap motion 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 4.5 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 C.1. It still uses the `warp()` method however this slow method is only used once throughout the run time of the using the LeapCV framework. Rather than every time a frame is loaded as in iteration 1.

```
/**
     * Initialise the distortion matrices for use with OpenCV {@link
           Imgproc.remap} method.
     *
     * @param image
     * @return {@code Map<String,Mat>} of X and Y matrix.
     */
    public static Map<String, Mat> initDistortionMat(Image image) {

        Mat tempX = new Mat();
        Mat tempY = new Mat();

        tempX = Mat.ones(image.height(), image.width(), CvType.CV_32F);
        tempY = Mat.ones(image.height(), image.width(), CvType.CV_32F);

        // Get distortion value for each pixel in the image
        for (int y = 0; y < image.height(); ++y) {
            for (int x = 0; x < image.width(); ++x) {
                Vector input = new Vector((float) x / image.width(), (float) y
                        / image.height(), 0);
                input.setX((input.getX() - image.rayOffsetX()) /
                        image.rayScaleX());
                input.setY((input.getY() - image.rayOffsetY()) /
                        image.rayScaleY());
                Vector pixel = image.warp(input);
                if (pixel.getX() >= 0 && pixel.getX() < image.width() &&
                        pixel.getY() >= 0 && pixel.getY() < image.height()) {
                    tempX.put(y, x, pixel.getX());
                    tempY.put(y, x, pixel.getY());
                } else {
                    tempX.put(y, x, -1);
                    tempY.put(y, x, -1);
                }
```

```
29
30                }
31            }
32
33            Map<String, Mat> retVal = new HashMap<>();
34            retVal.put(X_MAT_KEY, tempX);
35            retVal.put(Y_MAT_KEY, tempY);
36
37            return retVal;
38        }
```

Listing 4.5: Initialisation of Distortion Data

4.6.3 Test

Performance Tests

```
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 4.6: Example JUnit Performance Test Case

Date	Test Reference	Pass	Notes

Table 4.2: Acceptance Test Template

Unit Tests

Acceptance Tests

4.6.4   Evaluation

What went well?

The optimisation of distortion removal was successful. Refactoring made the framework a lot easier to use and develop. Creating a Git repository was easily done and will allow for a simple release of the open source framework.

What did not go so well?

Lessons learned?

4.7   Prototype 3

4.7.1   Design

4.7.2   Implementation

Stage 1 - Obtaining the Q Matrix

OpenCV requires the camera's intrinsic parameters to 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. This was not realised until this iteration. An attempt was made to extract the distortion coefficients from the camera by doing stereo calibration. However it was an unsuccessful attempt. The code for this attempts can be found in appendix (INSERT APPENDIX HERE). Another thought is that the distortion coefficients can be calculated from the already obtained distortion map. A method was outlined on the internet that might prove useful by Stackexchange (2014). 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 leap motion. 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 leap motion in use here. Although for speed of development these values will be used as the `Q` matrix for this project. In hope that it will provide some form of output in the next stage.

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 is made of values 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 4.7

```
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 4.7: 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 (LISTING HERE). They were then opened in a tool called ccViewer[4]. This tool allows for the visualisation of point clouds. An example of a point cloud using the `StereoSGBM` method can be seen in figure (INSERT FIGURE HERE). And example of a point cloud using the `StereoVar` method can be seen in figure (INSERT FIGURE HERE). The images show that the `StereoVar` Method gives a much clearer point cloud. However when looked at multiple angles It shows that some of the depths are non realistic. This might be due to the `Q` matrix mentioned earlier, not being the exact one for the leap motion used in this project. As time is running low and it is difficult to achieve a good point cloud without a good calibration method, a different approach will be made for object recognition. Instead of making use of the disparity map functionality, an attempt will be made using methods of feature recognition for classification.

---

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

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 SIFT feature extractor with a FLANN matcher can be seen in listing 4.8.

```
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 4.8: 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 leap motion. 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 implemented. An example of the output produced by the method in listing 4.8 can be seen in figure (INSERT FIGURE HERE).

Date	Test Reference	Pass	Notes

Table 4.3: Acceptance Test Template

4.7.3 Test

Performance Tests

Unit Tests

Acceptance Tests

4.7.4 Evaluation

What went well?

Managed to create a point cloud, building upon successful functionality created in prototype 2. Managed to get features matched between images.

What did not go so well?

The point cloud was not of high enough quality for classifying; it was noisy and unclear. Time was lost looking up other options of object detection.

Lessons learned?

4.8 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 5, the project as a whole will be evaluated. Looking at how far the project has gone in solving the original problem.

Chapter 5

Evaluation

5.1 Introduction

In this chapter the aim is to evaluate the work that has been done toward 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.

(COPY AIM AND OBJECTIVES HERE)

5.2 Summary

Chapter 6

Conclusion

6.1   Introduction

6.2   Summary

Bibliography

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.”.

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*

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.

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.

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

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.

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*

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*

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	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.7: Documented `TrainClassifier` *use case*

Appendix B

CRC Diagrams

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 B.1: `LeapCVCamera` *CRC*

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

Table B.2: LeapCVImage *CRC*

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

Table B.3: LeapCVStereoUtils *CRC*

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

Table B.4: LeapCVImageUtils *CRC*

Appendix C

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) &&
```

73

```
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 C.1: Slow Method of Image *Distortion Removal*