# An Open Source Computer Vision Framework for Leap Motion

Daniel Hamilton 10026535,
Computer Science,
University of the West of England.

March 25, 2015

**Abstract**

**Acknowledgements**

# Contents

# List of Figures

# List of Tables



Word Count: 7487.

# 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.

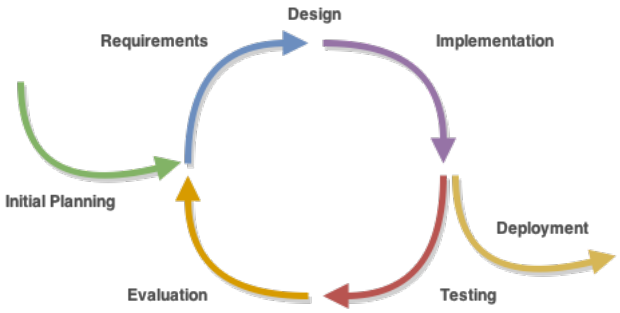


*Figure 1.1: Overview of an iterative development methodology*

### 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.

## 1.10 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 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.

### 2.2.1 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

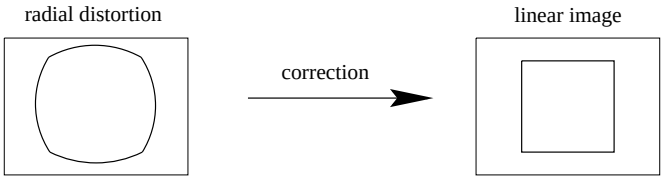


*Figure 2.1: Example of radial distortion of a square being corrected (Hartley & Zisserman 2003)*

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.

### 2.2.2 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 travels through a pinhole that is made in the pinhole plane. This light then gets

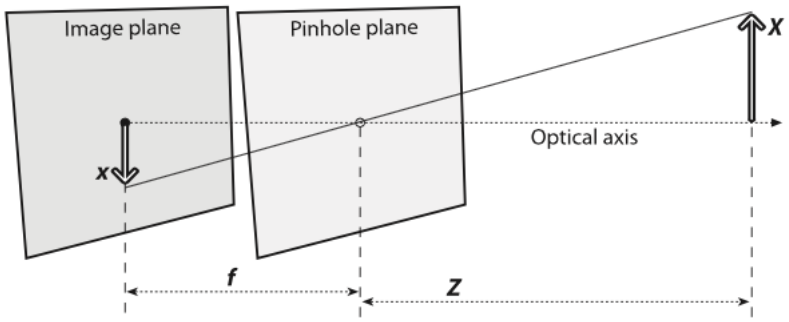


*Figure 2.2: Pinhole camera model as shown in Bradski & Kaehler (2008, p. 372)*

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 $-x = f\frac{X}{Z}$. For mathematical simplification Bradski & Kaehler (2008) states that the negative value can be ommited 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

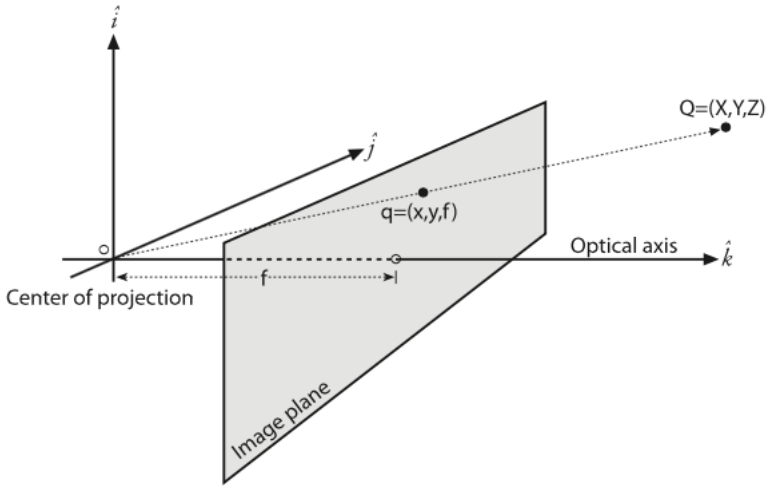


*Figure 2.3: Pinhole camera model with image plane in front, as shown in Bradski & Kaehler (2008, p. 372)*

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 $x_{screen} = f_x \left(\frac{X}{Z}\right) + c_x$, $y_{screen} = f_y \left(\frac{Y}{Z}\right) + c_y$. 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.

### 2.2.3 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).

### 2.2.4 Image Rectification

TODO

## 2.3 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.

### 2.3.1 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.

## 2.4 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 for **??**.

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

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.
4. Turn the disparity map into distances.

## 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

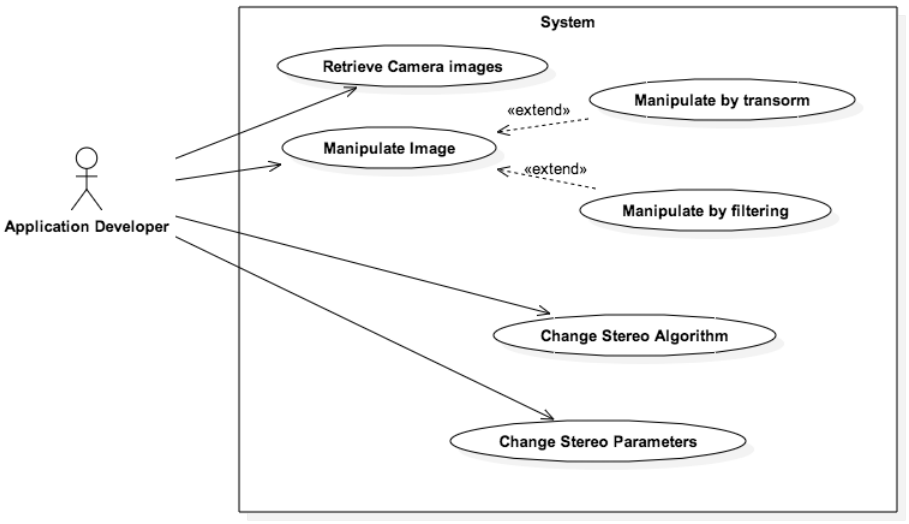


*Figure 3.1: UML use case diagram*

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.<br>2. Framework checks leap motion is ready.<br>3. Framework retrieves image from `LeapSDK`.<br>4. Framework converts image to usable format.<br>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. 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 iteration 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

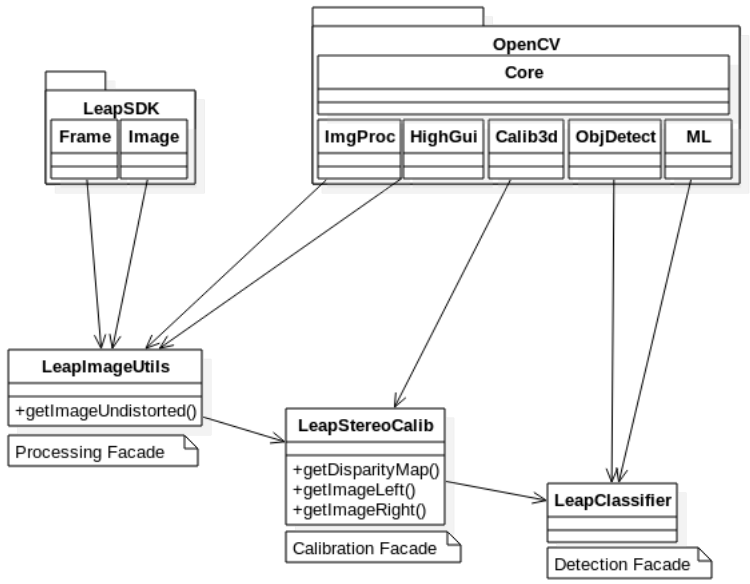


*Figure 4.1: High level system architecture*

## 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. To ensure the library gets loaded when the project gets built it is important to include the line in listing 4.1.

```
import org.opencv.core.Core;

public static void main(String[] args){

    //  The line below should be placed somewhere within your code, before
    //  any elements of the OpenCV library are called
    System.loadLibrary{Core.NATIVE_LIBRARY_NAME};

    //  Implementation continues here...
}
```

*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.

**Implementation** will be split into stages to highlight the though 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.

## 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()`.

```
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`

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.

```
// 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
ImageList images = controller.images();

// Convert both images and write to a file
int imageCount = 0;
for(Image image : images){
        BufferedImage bufferedImage = toBufferedImage(image);
        File outputImage = new File("image" + imageCount + ".png");
        ImageIO.write(image, "png", outputImage);
```

```
}
```

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

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).

### Stage 2 - Distortion Removal

Now that the `Images` can be successfully taken from the leap motion, the distortion must be removed from them. The LeapSDK provides a 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. This would allow the process to run on a GPU in a much faster time. 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. Listing 4.4 shows the method by which this was done.

```
public BufferedImage toUndistortedImage(Image image){
    int targetWidth = image.width();
    int targetHeight = image.heigh();
    int type = BufferedImage.TYPE_BYTE_GRAY;
    int bufferSize = targetWidth*targetHeight;
    byte[] pixels = new byte[bufferSize];
    int brightness = 0;

    // Loop through each pixel in the image
    for(double y = 0; y < targetHeight; y++){
        for(double x = 0; x < targetWidth; x++){

            // Normalise the slope for the current pixel
            Vector input = new Vector((float)x/targetWidth,
                (float)y/targetHeight, 0);

            //Convert from normalized [0..1] to slope [-4..4]
            input.setX((input.getX() - image.rayOffsetX()) /
                image.rayScaleX());
            input.setY((input.getY() - image.rayOffsetY()) /
                image.rayScaleY());

            // Look up the pixel coordinates in the raw image
            // corresponding to the slope values.
            Vector pixel = image.warp(input);

            // Check that the pixel coordinates are actually valid...
            if((pixel.getX() >= 0) &&
               (pixel.getX() < targetWidth) &&
               (pixel.getY() >= 0) &&
               (pixel.getY() < targetHeight)) {

                // Convert the discrete x and y coordinates into a data buffer
                    index
                int data_index = (int)(
                    Math.floor(pixel.getY())
                    * targetWidth
                    + Math.floor(pixel.getX())
                );

                // Look up brightness value at current pixel
                brightness = image.data()[data_index] & 0xff;
```

```
            } else {
                //Display invalid pixels as red
                brightness = 255;

            }
            // Copy the brightness value into the new undistorted image
            pixels[(int) Math.floor(y * targetWidth + x)] =  (byte) brightness;
        }
    }

    // Turn the undistorted image into a buffered image
    BufferedImage bufferedImage = new BufferedImage(targetWidth, targetHeight,
        type);
    final byte[] targetPixels = ((DataBufferByte)
        bufferedImage.getRaster().getDataBuffer()).getData();
    System.arraycopy(pixels, 0, targetPixels, 0, pixels.length);

    return bufferedImage;
}
```

*Listing 4.4: Slow Method of* `Image` *Distortion Removal*

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 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.

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

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 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`.

```
public static Mat convertToMat(Image image){
    // Initialise the Mat with the height and width of the image
    // CvType.CV_8UC1 is single channel unsigned 8 bit.
    Mat convertedImage = new Mat(image.height(), image.width(),
        CvType.CV_8UC1);

    // Put the image data into the Mat starting at pixel x = 0, y = 0.
    convertedImage.put(0, 0, image.data());

    return convertedImage;
}
```

*Listing 4.5: 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 handled 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`. 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).

### 4.5.3 Test

### 4.5.4 Evaluation

## 4.6 Prototype 2

### 4.6.1 Design

### 4.6.2 Implementation

### 4.6.3 Test

### 4.6.4 Evaluation

## 4.7 Prototype 3

### 4.7.1 Design

### 4.7.2 Implementation

### 4.7.3 Test

### 4.7.4 Evaluation

## 4.8 Summary

# Chapter 5

# Evaluation

## 5.1 Introduction

## 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.

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*

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.

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.<br>2. Framework checks image is valid.<br>3. Framework validates transformation.<br>4. Framework transforms image.<br>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.<br>2. Framework checks image is valid.<br>3. Framework runs object detection algorithm.<br>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.<br>2. Framework checks flag is valid.<br>3. Framework constructs object based on flag.<br>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.<br>2. Framework checks parameters are valid.<br>3. Framework sets parameters.<br>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.<br>2. Framework checks flag is valid.<br>3. Framework constructs object based on flag.<br>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.<br>2. Framework checks parameters are valid.<br>3. Framework sets parameters.<br>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.<br>2. Framework checks images are valid.<br>3. Framework trains the classifier with the new images.<br>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*