Department of Mechanical and Aerospace Engineering

Development of a Computer-Vision Based Orientation System for CubeSats

Author: Daniel Hingston

Supervisor: Dr Stuart Grey

A thesis submitted in partial fulfilment for the requirement of degree in

Master of Science in Advanced Mechanical Engineering

2019

Copyright Declaration

.

Abstract

There is much potential for the development of orientation systems for CubeSats. This project covers the creation of an experimental open source computer vision orientation system, specifically for use on CubeSats. A critical literature review studied existing orientation systems, neural network uses in astronomy, and specific tools and approaches that would be applicable. The *OpenCV* library functionality for cascade classifier training was employed, and HAAR and LBP feature types were investigated. Different training and detection settings were studied to tune the detection system for the best performance. A final set of trained cascades was produced for the northern celestial hemisphere, and tested as a complete set. Further experiments were performed to investigate the chief causes of error. It was found that the primary area for improvement is to increase the rotationally invariant performance of detection, which may be an inherent weakness of an approach utilising HAAR and LBP feature types. Future work is suggested to adopt the successful aspects of this project, develop the idea further, and progress towards a deployable experimental system.

Acknowledgements

I would like to thank Dr Grey for taking on my project idea to develop into an MSc Thesis. The interest and enthusiasm that he has shown in my work has been much appreciated. His specialist knowledge was vital in shaping the course of this project, and setting achievable limits in what was a challenging topic area for me.

Thank you too to Virginia, my fiancée, for her constant support through the long months of hard work that this project involved. Thank you in particular for listening to me talk endlessly about cascade tests, keeping me cheery when nothing seemed to work, and for many really excellent ideas that helped me see this project through to conclusion.

Lastly, I would like to thank my parents Peter and Charlotte Hingston for supporting me, in many ways including financially, through this MSc which has been so interesting for me. I am glad that I have had the opportunity to study this course and I hope that you will enjoy reading this conclusion to it, and hopefully not notice too many errors.

Table of Contents

1 Introduction 9
2 Project Objectives and Scope 10
3 Methodology 11
3.1 Overview 11
4 Literature Review 12
4.1 Existing Star-Tracking Technologies 12
4.2 Neural Networks and Machine Learning in Astronomy 13
4.3 Raspberry Pi Single-Board Computer 15
4.3.1 Dedicated Camera Hardware 16
4.4 Open-Source Computer Vision 17
4.4.1 HAAR, LBP, and HOG Feature Classifiers 17
4.4.2 OpenCV Library 18
4.4.3 Image Databases for Training 20
4.5 Fiducial Markers and Machine Learning 21
4.6 Astronomy Coordinate Systems and Projections 22
4.6.1 Projections 22
4.6.2 WCS, FITS, and Astropy Python Package 24
5 System Development 26
5.1 Development Hardware and Methodology 26
5.1.1 Use of a Raspberry Pi 3B and V2 Camera 26
5.1.2 Use of Stellarium 28
5.1.3 Use of a Linux Virtual Server for Cascade Training 29
5.2 Experimental Programme 30
5.2.1 Bright Star Identification 30
5.2.2 Initial Cascade Test Results 32
5.2.3 The Need for Fiducial Markers 33
5.2.4 BASIC or ALL Mode 37
5.2.5 Local Binary Pattern (LBP) 37
5.2.6 Testing for Rotational Invariance 39
5.2.7 Expansion of Negative Image Dataset 39
5.2.8 Further Rotation Testing 41

5.2.9 Confidence Value Utilising ‘levelWeights’ Parameter 42
5.2.10 Orientation Calculation 42
5.3 Overview of Final System 42
5.3.1 Target Selection and Final Cascade Training 42
5.3.2 Pixel to RA/Dec Conversion 47
6 Results 49
6.1 Verification of System Performance 49
7 Results Discussion 51
7.1 Cause of System Failure 51
7.2 Modifications to Improve System Performance 52
7.2.1 Fiducial Marker Placement Program 52
7.2.2 Improving Cascade Accuracy 53
7.2.3 Rotational Invariancy of LBP-feature Cascades 54
8 Conclusions 56
9 Future Work 57
9.1 GitHub Repository 57
10 References 58
11 Appendices 62
11.1 Appendix 1: Stellarium Script for Image Capture 62
11.2 Appendix 2: Positive Sample Creation Program (Python) 64
11.3 Appendix 3: Descriptor (bg.txt) File Creation (Python) 66
11.4 Appendix 4: Negative Dataset Creation Program (Python) 67
11.5 Appendix 5: Final Orientation Test Program (Python) 68

List of Figures

Figure 1: Representation of the construction of a gnomonic projection (Source: Author, simplified and modified from a diagram presented by Calabretta and Greisen, 2002)24
Figure 2: Processed (left) and raw (right) images captured using V2 Camera. 26
Figure 3: Simulated V2 Camera Field-of-View (Stellarium) 28
Figure 4: Identification of the brightest stars in a Stellarium-generated star image using functionality of OpenCV library. 31
Figure 5: Example of star pattern with overlaid fiducial markers, overlapping Type 6. 36
Figure 6: Zenithal Equidistant plot of northern celestial hemisphere, showing the RA/Dec coordinates of the 55 cascdes trained initially. 43
Figure 7: Final cascade testing process. 44
Figure 8: Zenithal Equidistant plot of northern celestial hemisphere, showing RA/Dec coordinates of the 31 down-selected cascades. 46
Figure 9: Successful example of two true positive detections made on sample image. 49
Figure 10: How pixel resolution of image affects detection of a given star. 52
Figure 11: Graph showing rotational performance of four cascades. 54

List of Tables

Table 1: Test results of fiducial marker type investigation...35
Table 2: Cascade training '-mode' setting comparison test results..............................37
Table 3: LBP and HAAR feature type comparison test results. 38
Table 4: Results of negative dataset comparison tests. 40
Table 5: Training Properties for Four Selected Cascades. 54

Nomenclature

In the context of this Thesis, the following specialist terms are described as follows:

Acceptance Ratio	A measure of the precision of a trained cascade classifier, typically measured on the final training stage.
Apparent Magnitude	A scale of astronomical object brightness as seen by an observer on Earth.
Cascade	A classifier trained to identify a specific target object.
CubeSat	Miniature satellite standard comprised of multiples of 10cm x 10cm x 10cm units (U).
Fiducial Marker	A marker adding a point of visual reference to an image.
FITS	Flexible Image Transport System, a digital file format used in astronomy.
FOV	Field of view.
HAAR	Haar-like features are a type of image feature used for classifier training.
HOG	Histogram of Oriented Gradients - a further type of feature used for classifier training.
LBP	Local Binary Pattern - another type of image feature used for classifier training.
levelWeight	An *OpenCV* term analogous to a confidence value for a given detection performed by a cascade.
OpenCV	‘Open Source Computer Vision’, a library of computer vision functions.
RA/Dec	Right Ascension and Declination, a coordinate pair for a point on the equatorial celestial coordinate system.
Stellarium	Open-Source astronomy software used to simulate star views.
WCS	World Coordinate Systems, a standard for FITS keywords for pixel to world coordinate transformations.

1 Introduction

In this MSc Thesis it was intended to develop and test (in a simulated environment) a computer vision-based orientation system specifically, but not exclusively, suitable for use on CubeSat miniature satellites.

A literature review was undertaken on the subjects of related astronomy and neural network computer vision projects, relevant neural network technologies and astronomy software, as well as suitable methods and tools employed in the creation of a neural network trained classifier for identification of a given target object.

Through a thorough experimental programme the multiple factors that can influence the training and detection use of a cascade classifier were analysed. The findings of this programme led to the creation of a final set of 31 tested cascades to fully cover the northern celestial hemisphere. This project made use of the *OpenCV* (Open Source Computer Vision) library.

Testing of the final cascade set against simulated views of the stars from space revealed that the cascades were highly prone to false positive detection. Minor differences in the star patterns presented to the cascades due to distortion, rotation, and placement of fiducial markers radically reduced their performance. Further experimentation was undertaken to provide more focussed investigation into the chief causes of unreliability that were identified, and measures were taken to improve performance.

It was concluded that whilst the principle of this Thesis was valid, and an orientation sensor of this type may provide some real benefits for a CubeSat application, the use of HAAR and LBP feature types for classifier training provided issues associated with detection. The main area for improvement is to ensure consistent detection confidence regardless of the angle of the provided target.

A major project objective was to open source the project, to ensure full capability for future development of the ideas presented. To meet this objective a full copy and record of the project was uploaded to a project *GitHub* repository, under an MIT license.

2 Project Objectives and Scope

General objective: To design and test a computer vision system for orientation sensing, employing neural network techniques to perform a star identification process.

i. Through a literature review, determine suitable methods to investigate.
ii. Experimentally determine optimum settings for training and detection functions, through an iterative testing process.
iii. Create and test a final set of classifiers using simulated test conditions.

General objective: To develop the system with consideration for deployment on a CubeSat satellite.

i. Consider the constraints that the CubeSat design requirement imposes upon the design of the orientation sensor.
ii. Refer to existing CubeSat orientation methods in the literature review, and incorporate findings of those sources into the development process.

General objective: To open source the project, to allow for future development of the concept, and make available all methodology, code, and findings accessible to others.

i. Publish all code created, results found, and this MSc Thesis itself on *GitHub* with an MIT License to allow permissive use for other students and researchers.
ii. Ensure that all code is annotated, and detailed notes are created to accompany the above files, so that that this work can be reproduced and expanded upon.

The scope of the project is naturally limited by the time and funding constraints of an MSc Thesis, so as to maximise the worth of the project it was decided to focus purely on the object recognition machine learning tools offered by the *OpenCV* library. A lengthier project could involve creation of customised neural networks for the training stages. The purpose of this project was also not to discuss the mathematics or computational process of specific machine learning processes, rather to apply them to a real-world problem. Detailed discussion of the background of any processes used in this Thesis are covered in the literature discussed in Section 4.

3 Methodology

The primary purpose of this project was to develop, test, and open-source a computer vision orientation system for CubeSats. A detailed literature review improved understanding of the main concepts involved, and allowed a specific methodology or approach to be selected for experimental investigation. An experimental programme guided the creation of the programs and methods needed to create a functioning system, which were further tested to thoroughly analyse system performance. An appreciation of the design constraints imposed by designing for a CubeSat application guided the technology selection, and provide additional depth for discussion of the viability of the system for deployment outside of a purely simulated environment.

3.1 Overview

Section 4 provides a detailed discussion of the relevant literature that was drawn on in the creation of this MSc Thesis. Section 5 details the development and test process that was followed in order to create a computer vision spacecraft orientation system, and system test results are presented in Section 6. Section 7 provides analysis of the test results, discusses the limitations of the system, and provides additional test results to examine the main causes of unreliability. Section 8 contains the conclusion to the paper, and Section 9 provides suggestions for future work based on this Thesis, and discusses the open sourcing of this project. Additionally, Section 10 contains the references used for this research project, and copies of the main Python and *Stellarium* programs used are reproduced as appendices to this Thesis paper in Section 11.

4 Literature Review

This section contains a summary of the key findings of the literature reviewed for this Thesis, organised in to the significant topic areas.

4.1 Existing Star-Tracking Technologies

Star trackers of different forms have been utilised in space since the 1960s (Trask, 2002), and a full analysis of the evolution and nature of their technology is outside the scope of this Thesis paper. In this literature review, focus was placed upon sources that have specific relevance to this project, for instance sharing the basis of being designed for a CubeSat application.

McBryde (2012) describes the development of a star tracker specifically for CubeSats, employing a CCD sensor. The system described by McBryde (2012) was able to return satellite orientation to "tens of arcseconds or returned an error. Only twice out of 200 times did the star tracker return a false positive". At the time of that paper, no CubeSat had been flown with a functioning star tracker, and McBryde suggests that CubeSat attitude determination technology is behind that of larger satellites. The methodology employed by McBryde (2012) for star identification was an adaptation of a voting method, which naturally places greater emphasis on real stars, and supresses false positives. The challenges of miniaturisation of orientation sensors for a CubeSat application is further discussed by Lindh (2014), who describes the need to develop "a cheaper and smaller substitute", as "most of the available trackers are aimed for large satellites and are as previous mentioned rather expensive.".

Of specific relevance to this MSc Thesis is the work by Gutiérrez, Fuentes and Díaz (2017), who developed a star tracker for a CubeSat application (minimum of 2U size), utilising a Raspberry Pi single-board computer to provide the processing power. The system they created can achieve an average accuracy of 5 seconds of arc, however suffers from a long processing time of 75 seconds. In recognition of this, their system also offers a lower accuracy mode which can calculate attitude to 3 minutes of arc within 35 seconds. This long processing time is perhaps partly due to their adoption of the Raspberry Pi 2, a model of the Raspberry Pi which as of 2019 is technically obsolete. The methodology employed by Gutiérrez, Fuentes and Díaz (2017) makes use of the

‘Source Extractor’ software suite to identify stars, and the ‘Match’ program to correspond the results obtained from ‘Source Extractor’ to a catalogue of known objects. As a sensor, Gutiérrez, Fuentes and Díaz (2017) employed the Raspberry Pi V2 Camera. Their project is also open-sourced.

A commercially-available satellite attitude sensor is described by Brown, Clements and Evans (2010), in which they discuss a ‘star-sensor’. They differentiate ‘star-sensors’ from ‘star-trackers’ as “Star sensors vary significantly from star trackers. Instead of using imagers, some star sensors operate by using a photo detector to produce pulses when stars transmit the detector’s FOV” (Brown, Clements and Evans, 2010), which is a concept developed from the original 1960s instruments. They describe a limitation of the technology which is that star magnitude calibration “must be based on orbit characterisation”. According to Brown, Clements and Evans (2010), reasonable approximations during testing can be obtained with a very dark sky on Earth, but this would not be possible for this MSc thesis. Instead, a simulated test environment was employed using *Stellarium* to provide star images (see Section 5.1.2). A similar star-sensor is discussed by Kruijff and van der Heiden (2003), the performance was an accuracy of < 10 arc seconds, and a 99% success rate.

4.2 Neural Networks and Machine Learning in Astronomy

“Conventional on-board systems face severe computational bottlenecks introduced by serial microprocessors operating on inherently parallel problems. New computer architectures based on the anatomy of the human brain seem to promise high speed and fault-tolerant solutions to the limitations of serial processing.”, wrote Alvelda and Martin (1988). Indeed, the interest in applying a neural network approach to spacecraft orientation determination has existed for multiple decades. Although the references in Alveda and Martin’s paper to computing are long obsolete, the process they outline for a computer vision-based spacecraft orientation system remains largely applicable. According to Alveda and Martin (1988), the process involves:

i. “Image a portion of the sky,
ii. Compare the characteristic pattern of stars in the sensor field-of-view to an on-board star catalogue,
iii. Thereby identify the stars in the sensor FOV [Field of View],

iv. Retrieve the identified star coordinates,

v. Transform and correlate FOV and real-sky coordinates to determine spacecraft attitude.”

According to Alveda and Martin (1988), at the time of writing no planetary spacecraft had flown with such a system, mainly due to the limitations of on-board memory.

The computational power of modern computers makes such advanced methods as neural networks possible, and many successive papers have examined the potential for neural network application for not only spacecraft orientation, but wider astronomy applications. Lindblad et al. (1997) analyse a range of algorithms that can be applied to star identification using a neural network. It was concluded that “the basic nearest neighbour method, using only one prototype per star … implement in neural network type architecture … such a method would provide a fast and robust solution for small FOV star trackers.” (Lindblad et al., 1997). Additionally, they provide an interesting insight into the technical specifications of typical spacecraft star trackers of the time, suggesting that a typical model would “have a field of view (FoV) of 20 x 20 degrees, and have a 512 pixel/row 512 rows MMP CCD. The sensitivity is typically in a range of magnitude M =+0.1 to +4.5.” (Lindblad et al., 1997). Furthermore, they suggest that these sensors would be able to determine attitude to within a few arcseconds. In a companion paper, Lindsey, Lindblad and Eide (1997) present a more detailed methodology for star constellation identification using a neural network. The basis of their work relies on transforming the stars viewed through the sensor lens into a histogram of the radial distances from the most central star visible to the other visible stars of the constellation, which is identified by a trained neural network. The key advantage that Lindsey, Lindblad and Eide (1997) present in such a system is the memory required would be approximately 1/15th of the size of that required of a star tracker system employing a conventional non neural network ‘tri-star’ method.

A further example of a neural network system employing imagery from an onboard CCD camera is that studied by Trask (2002). The neural network proposed by Trask (2002) has the benefit that it uses “a new technique of star pattern encoding that removes the star magnitude dependency.”. The paper serves to support the case that a neural network based star tracker device can offer faster processing times than a non-neural network based conventional star tracker, citing that the conventional CT-633 tracker from *Ball Aerospace* requires 60 seconds to obtain the orientation, whereas the prototype system proposed by Trask requires 12 seconds worst-case (Trask, 2002). The sensitivity of the instrument Trask developed to star magnitude is claimed to be better than 5.0 instrument magnitude (Trask, 2002), not dissimilar to the 4.5 magnitude limit of the commercially available sensors discussed by Lindblad et al. (1997). A further source of limit magnitude for similar sensors is Qian et al. (2013) who suggest that a typical star tracker camera of a CMOS type can observe stars up to a visual magnitude of 5.0. Trask (2002) approached the star identification problem using a 'convex hull' method that allowed all stars in the instrument's FOV to be used as part of the orientation calculations. The findings of Trask (2002) are given added authenticity as the performance specifications were taken from real night sky field trials, rather than just a simulated software test process.

Kim and Brunner (2017) presented a study in using a neural network approach to classify images of stars and galaxies as collected by robotic Earth-based telescopes. The neural network approach allows the fast and accurate processing of large data volumes whilst requiring minimal human input. The training set that Kim and Brunner (2017) employed contained 40,000 sample images, which was further increased through applying rotation, translation, reflection, and the addition of Gaussian noise to these positive images to generate further samples. According to Kim and Brunner (2017), "training our network takes about forty hours on an NVIDIA Tesla K40 GPU", a demonstration of the computationally demanding nature of neural network training that has prevented widespread exploitation of the technology until relatively recently.

4.3 Raspberry Pi Single-Board Computer

The Raspberry Pi range of single-board computers are just some of the large number of rival single board computers on the market today. Many of these offer superior performance to the Raspberry Pi and some even offer sufficient computing power to be able to run full versions of mainstream operating systems such as *Windows 10*. However, the reason for the Raspberry Pi's success can probably be largely attributed to the very low retail price and the simplicity of operation. As discussed in Section 4.1, Gutiérrez, Fuentes and Díaz (2017) have developed a star-tracker for CubeSat use based on a Raspberry Pi computer, and there have been other examples of Raspberry Pi usage in astronomy. A *Debian*-based operating system, *Raspbian*, was developed for the Raspberry Pi, and offers functionality familiar to a Linux user. There are three main reasons why the Raspberry Pi was suitable for this CubeSat application:

i. Power consumption and size – the Raspberry Pi is highly compact and offers excellent computing capability for minimal power consumption (< 0.5A Raspberry Pi Model 3B).
ii. Simplicity – the Raspberry Pi is simple to operate and due to the wider success of the computer there is a vast array of documentation available which will provide support to this project.
iii. Open source – the Raspberry Pi is commonly used by the open source community, and so by developing this project for the Raspberry Pi, this project offers better suitability for the open source nature of some CubeSat projects.

4.3.1 Dedicated Camera Hardware

The Raspberry Pi Foundation offers a board-mounted CMOS camera. The camera is built around a Sony IMX219 sensor which offers 8 megapixels and can record a static image with the resolution of 3280 by 2464 pixels (IMX219 Product Brief, n.d.). The lens is fixed focus and offers a field of view of 62.2° in the horizontal axis and 48.8° in the vertical axis (Camera Module - Raspberry Pi Documentation, 2019).

A consideration is that of digital camera technology; as described, the Raspberry Pi V2 Camera features a CMOS sensor as opposed to a CCD type sensor. CCD sensors have been commonly used in astronomy due to their greater sensitivity to low luminance objects, however CMOS sensors have a major advantage which is that they can record a digital image much more quickly. This is considered an important characteristic for the satellite-mounted application given that the satellite body may be in motion and a quick image capture results in the captured data being closer to real time than it would be if there was a long delay present in the capture process. Additionally, CMOS sensors are significantly less expensive.

Although the Raspberry Pi V2 camera is basic and does not offer the benefits of a more costly high performance camera sensor, for the purpose of this MSc project that was considered the most suitable option, given the perfect compatibility with the Raspberry

Pi computer and the high level of documentation published relating to its use and performance including a wide variety of dedicated software libraries. Of great relevance to this project was the work authored by Pagnutti, et al. (2017) which accounts a detailed study into the optical performance of the Raspberry Pi V2 Camera.

4.4 Open-Source Computer Vision

4.4.1 HAAR, LBP, and HOG Feature Classifiers

Viola and Jones, (2001), developed a “visual object detection framework that is capable of processing images extremely rapidly while achieving high detection rates.” The methodology in question is the HAAR-feature classifier, which uses “features which are reminiscent of Haar Basis functions” (Viola and Jones, 2001), to train a classifier to recognise a supplied positive sample. A good overview of the HAAR method is provided by Commin (2009).

A development of HAAR-feature training was made by Lienhart and Maydt (2002), who added additional, rotated, HAAR-like features that now added the capability to recognise HAAR-like features at 45° angles. “With these new rotated features our sample face detector shows off on average a 10% lower false alarm rate at a given hit rate.” (Lienhart and Maydt, 2002). It was anticipated that the false selection of false positives would be a key issue for this project, given that for a correct spacecraft orientation to be calculated, false positives must be reliably ignored. For this reason, it was thought that the enhanced HAAR method proposed by Lienhart and Maydt (2002) should be investigated – as it may have potentially proved beneficial to providing additional rotational invariant performance for the trained cascade classifiers.

An alternative but related method for classifier training depends on Local Binary Patterns (LBP), as first developed by Ojala, Pietikainen and Harwood (1994). The LBP feature type approach was selected for investigation in this MSc Thesis due to the fact that “The texture measures based on local binary patterns are also computationally extremely simple.” (Ojala, Pietikainen and Harwood, 1994). It is anticipated that cascade classifiers trained on an LBP feature set may offer superior performance in the

computationally-limited environment of the Raspberry Pi. In contrast to the HAAR feature method, the LBP classifier makes use of texture analysis.
Additionally, the HOG (Histogram of Oriented Gradients) method was identified as being applicable for the satellite orientation method of computer vision as investigated in this MSc Thesis. Dalal and Triggs (2005), were not the first to describe the HOG principles, but were responsible for much of its development, showing "experimentally that grids of Histograms of Oriented Gradients (HOG) descriptors significantly outperform existing feature sets for human detection". Although their work was applied to images of pedestrians, the HOG methodology is of interest as Dalal and Triggs (2005) showed that HOG was "reducing false positive rates by more than an order of magnitude relative to the best Haar wavelet based detector".

4.4.2 OpenCV Library

The Open Source Computer Vision (*OpenCV*) library, originating from an Intel research project of 1998, is "aimed at providing the tools needed to solve computer-vision problems. It contains a mix of low-level image-processing functions and high-level algorithms" (Pulli et al., 2012). According to Pulli et al. (2012), "One of the major design and implementation goals for *OpenCV* has always been high performance." which is a key factor for selecting a computer vision method to use on a relatively low-powered device such a Raspberry Pi single-board computer.

The training of HAAR and LBP classifiers is supported by the *OpenCV* CascadeClassifier class (OpenCV: cv::CascadeClassifier Class Reference, 2019), and is optimised for use within a Python 3 Linux environment. The HAAR functions include the ability to employ the extended HAAR feature set as developed by Lienhart and Maydt (2002). It should be noted that *OpenCV* does not necessarily reflect the most recent developments in machine learning and object recognition, presumably to place greater emphasis on the stability and reliability of each version release. According to Howse, Joshi and Beyeler (2016), *OpenCV* offers functionality for employing other feature detection methods, such as SIFT (Scale-Invariant Feature Transform), and the faster to process SURF (Speeded-Up Robust Features). Additionally, *OpenCV* "has support for even more feature descriptors, such as Features from Accelerated Segment Test (FAST), Binary Robust Independent Elementary Features (BRIEF), and Oriented FAST and Rotated BRIEF (ORB)", (Howse, Joshi and Beyeler, 2016).

The selection of the most appropriate feature types for classifier training to investigate in this MSc project was driven mainly by the requirements of the CubeSat orientation sensing application, the key requirements deemed to be:

i. Minimal processing time. As each recognisable star pattern will have a unique classifier, an input image will be subjected to a detection process by a large number of classifiers, so the method must be fast to process, particularly given the limited computational power of the Raspberry Pi computer.
ii. Minimal file size. Closely related to processing time, as large complex cascade files take more processing time to perform a detection, the most suitable classifier will use a feature type that provides accurate and clear detection based on readily apparent image features. Minimising file size also saves the limited memory of the Raspberry Pi.
iii. Resistance to false positives. To perform the orientation calculations, the system must offer high resistance to the identification of false positives.

It was decided to further investigate HAAR and LBP type features for classifier training. HAAR due to the straightforward implementation of HAAR classifier training within *OpenCV*, and LBP as a partner due to the very similar nature of the classifier training process. This would permit a full analysis of the performance of the HAAR and LBP feature type classifiers for this application. The benefits of HAAR and particularly LBP in terms of processing speed as described by Ojala, Pietikainen and Harwood (1994) are highly desirable for the CubeSat orientation application, given the need to rapidly obtain data to determine the attitude of a manoeuvring or spinning craft.

A key functionality of the *OpenCV* CascadeClassifier class is that of the 'create_samples' function, for which further description can be found in the class documentation (OpenCV: cv::CascadeClassifier Class Reference, 2019). The 'create_samples' function has the ability to generate positive samples from only a single positive image. The worth of this function to this project is apparent; for instance, if a classifier to recognise red minibuses was desired, the positive image dataset would compromise of thousands of unique images of red minibuses from different angles, lightings, and vehicle models. In the case of the star recognition problem, it is not feasible to capture thousands of images of the stars equivalent to that which would be seen from low Earth orbit for the sake of this MSc thesis, and hence *Stellarium* is employed to produce simulated sky images. The 'create_samples' function can be employed to manipulate the single positive image by varying image distortions, orientations, skews, discolourations etc. to generate an artificial dataset of varied positive images (OpenCV: cv::CascadeClassifier Class Reference, 2019).

The *OpenCV* cascade classifier training process is well described by Howse et al. (2015), who describe how "the cascade classifier training process follows an iterative process to train subsequent stages of weak classifiers (1…N). Each stage consists of a set of weak classifiers, until the criteria for that specific stage have been reached." The *OpenCV* CascadeClassifer class documentation (OpenCV: cv::CascadeClassifier Class Reference, 2019), provides full information on the parameters that govern the training process. Investigation into the effect of altering these parameters was made as part of this MSc Thesis project, as a full understanding of these parameters was vital for the creation of reliable and accurate cascade files for star pattern recognition.

4.4.3 Image Databases for Training

To train a classifier, a large dataset of negative images is required. Russakovsky et al. (2015) describe the *ImageNet* online dataset of categorised images for neural network training, and suggest that it is larger than any other image classification dataset. "ImageNet contains 14,197,122 annotated images organised by the semantic hierarchy of WordNet" (Russakovsky et al., 2015). Although commonly employed for neural network training, at the time of writing of this MSc Thesis the *ImageNet* website was under maintenance and hence an alternative had to be sought.

Alternatives to ImageNet are the datasets created by Krizhevsky (2009), who developed two categorised datasets: "the CIFAR-10 set has 6000 examples of each of 10 classes and the CIFAR-100 set has 600 examples of each of 100 non-overlapping classes." These are both downloadable from the internet, and are relatively compact; the CIFAR-10 set is only 355 MB. As an additional option, Kinsley (2016), uploaded a downloadable subfolder of *ImageNet*, containing images of sports scenes. This contains nearly 2000 images which are potentially useful for classifier training.

4.5 Fiducial Markers and Machine Learning

The term ‘fiducial marker’ refers to a marker overlaid onto an existing image, to provide additional visual reference. Fiducial markers provide points which are designed to be easily recognised by the relevant computer vision process, and have widespread usage in Augmented Reality technology as well as computer vision (Lightbody, Krajník and Hanheide, 2017). Lightbody, Krajník and Hanheide (2017) also provide a list of criteria by which a fiducial marker’s worth is judged; the key priorities are ‘robustness’, ‘distinguishability’, ‘economic feasibility’ – not relevant given that in this application the markers are purely software-based, and ‘precision’. It is proposed that fiducial markers could be employed to provide additional visual information in the star pattern recognition application, by overlaying fiducial markers onto star patterns within the input image from the sensor camera. An example of employing fiducial markers to provide additional visual reference for computer vision is described by Stathakis (2011), in which “fiducial markers are used as a means of providing the scene with a number of specific key-points, or features, such that computer vision algorithms can quickly identify them within a captured image”. Stathakis describes great success with this methodology, however it is noted that the aim of his investigation was to find individual fiducial markers within an indoor scene, rather than identify a specific pattern of fiducial markers, and it does not necessarily follow that such a task would be simple for a computer vision system.

A vital requirement for the fiducial markers that could be employed to be overlaid upon star patterns was that of requiring infinite rotational symmetry, as they should appear identical from any angle. This eliminated the large number of square or otherwise non-circular markers that are described in the work of Stathakis (2011). A variety of circular marker types were found to have been discussed and testing in existing literature, including the ‘Fourier Tag’, developed originally by Sattar et al. (2007), and mentioned by both Stathakis (2011), and Sagitov et al., (2017). It was hypothesised that the visual complexity of the Fourier Tag, designed to encode multi-bit information, would provide a good set of identifiable features for classifier training. Many types of fiducial markers are designed to be able to encode information within their patterns, but this is not required for the application of computer vision discussed in this paper. Gatrell, Hoff and Sklair (1991) proposed the concentric circle or ‘CCTag’ marker which consists of several concentric rings, developed and tested for identifying target panels on a spacecraft body. According to Gatrell, Hoff and Sklair, 1991, the CCTag "is easily extracted from the image, its extraction is robust … and its centroid is completely invariant to the three translational and one rotational degrees of freedom". This suggested that the CCTag could be an appropriate type for this application, as the main requirement is that it should be easily identifiable by the computer vision system. A third and quite visually dissimilar circular marker is the gradient type described as "a circular fiducial with one single ring painted with a continuous gradation of gray hues" by Calvet et al. (2016).

4.6 Astronomy Coordinate Systems and Projections

To design an orientation system for a spacecraft, a sound appreciation of astronomical coordinate systems is vital, in order to be able to perform the calculations required to identify the attitude and angle at which the spacecraft lies. The right ascension and declination (RA/Dec) coordinates refer to coordinates on the equatorial celestial coordinate system, and the coordinates given for a certain astronomical object are given for a specific epoch, currently J2000.

4.6.1 Projections

Projection refers to the way in which points on a certain surface are mapped onto another. In the case of a camera mounted on a spacecraft viewing the stars, the projection created by the rectilinear lens of a camera can be approximated as a that of a 'Perspective' or 'Gnomonic' projection, in which the plane of projection is tangent to the sphere at a specific declination. According to Beck (2010), a gnomonic projection "is the CCD 'camera view' … is the perspective of an observer hovering over a particular location on the surface and looking down at it." A gnomonic projection can approximate either the outside of a sphere (such as the Earth's surface) or the inside of a sphere (such as the equatorial celestial coordinate system). It is therefore imperative to understand the equations that govern the nature of the projection. Calabretta and Greisen (2002) present a very thorough overview of the mathematics of projections applicable to astronomy. Calabretta and Greisen (2002, pp.9-12) describe a gnomonic projection as a type of zenithal projection, "constructed with the pole of the native coordinate system at the reference point." The following equations (1 to 7), reproduced from Calabretta and Greisen (2002, pp.9-12), describe the relationship between points on the celestial coordinate system (where ϕ represents right ascension, and θ represents declination) to cartesian points (x, y) on the tangent plane:

$$(\phi_0, \theta_0)_{zenithal} = (0, 90^{\circ}) \tag{1}$$

$$x = R_{\theta} \sin(\phi) \tag{2}$$

$$y = -R_{\theta} \cos(\phi) \tag{3}$$

Inverted, these give us:

$$\phi = \arg(-y, x) \tag{4}$$

$$R_{\theta} = \sqrt{x^2 + y^2} \tag{5}$$

Specifically, for the gnomonic projection:

$$R_{\theta} = \frac{180^{\circ}}{\pi} \cot(\vartheta) \tag{6}$$

And inverted:

$$\theta = \tan^{-1}\left(\frac{180^{\circ}}{\pi R_{\vartheta}}\right) \tag{7}$$

Using the above equations, it is possible to construct further relationships to describe the mapping of points from any tangent plane onto the pole-centred gnomonic coordinate system. The following figure, overleaf, illustrates a simple representation of the formation of a gnomonic projection for a projection plane tangent to the pole of the celestial sphere, with the projection origin at μ radii from the centre of the sphere.

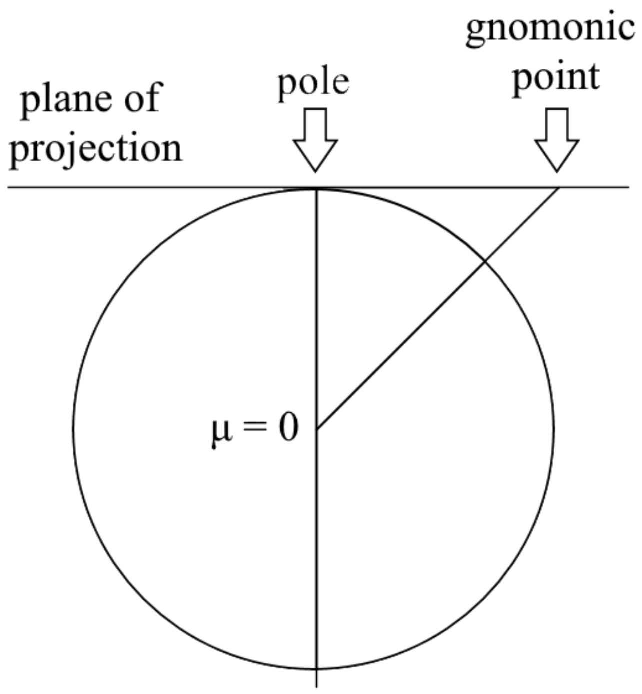


Figure 1: Representation of the construction of a gnomonic projection (Source: Author, simplified and modified from a diagram presented by Calabretta and Greisen, 2002)

An additional projection of use in this project is the 'Zenithal Equidistant' projection, discussed by Calabretta and Greisen (2002, pp.14), in which "the native meridians are uniformly divided to give equispaced parallels". The following equation, reproduced from Calabretta and Greisen (2002, pp.14), describes the key relationship for this projection:

$$R_\theta = 90^\circ - \theta \tag{8}$$

4.6.2 WCS, FITS, and Astropy Python Package

FITS is a file format commonly used in astronomy. A FITS image contains astronomy-specific information within the header. Some of this data relates to the concept of WCS (World Coordinate System), a standard for data in FITS headers which relates to pixel to coordinate system mapping. Beck (2010), discusses how FITS supports a Gnomonic projection, with a dedicated WCS code – 'TAN'. *Astropy* is an astronomy functions library for Python which supports the usage of WCS data within FITS headers. Although the Raspberry Pi V2 Camera would not capture FITS images, the functionality of the *Astropy* library can still be employed as the equivalent of a FITS header can be created programmatically, as described by the official documentation (Astropy Documentation — Astropy v3.2.1, 2019). *Astropy* has the capability to create a custom coordinate system for a specific point (such as the camera axis), provided that certain data relating to the global coordinate system is known. The *Astropy* library was investigated for use in determining spacecraft orientation given certain parameters. An Open-Source package for *Astropy*, *GWCS* (Generalized World Coordinate System), has the functionality to determine the coordinate system parameters of the central axis, given a list of known pixel to coordinate mappings, as described by the *GWCS* documentation (GWCS Documentation — gwcs v0.10.1.dev468, 2019).

5 System Development

This section outlines the experimentation that was performed in order to develop and iteratively test the neural network based orientation system.

5.1 Development Hardware and Methodology

5.1.1 Use of a Raspberry Pi 3B and V2 Camera

The Raspberry Pi single-board computer and compatible camera (as discussed in Section 4.3) was selected as the hardware on which to base this project. In particular, the 3B model of Raspberry Pi was selected, as it offers a good compromise between performance (desired to be able to perform fast detections) and power consumption (necessary to minimise for a CubeSat application).

5.1.1.1 Image Processing Pipeline

The process automatically applied to images captured by the Raspberry Pi V2 Camera is as follows (typical of a standard CMOS digital camera capture process): *Transposition, black level composition, lens shading, white balance, digital gain, Bayer de-noise, de-mosaic, YUV de-noise, sharpening, colour processing, distortion removal, and resizing.*

For images with unencoded output, the pipeline ends at this point. Unencoded output most commonly refers to images in the YUV or RGB formats. If exporting an image in a Bayer format, this is performed prior to the stages listed above. A Python script was used to extract a Bayer format image from a still taken using the V2 camera. This image was exported from the Raspberry Pi to a *Windows 10* PC using *WinSCP* and *Adobe Photoshop* was used to transform the image from a raw format into a JPEG

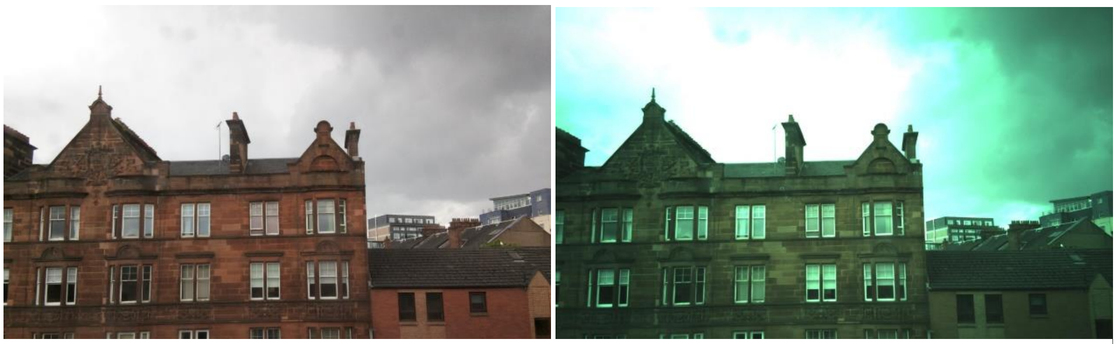


Figure 2: Processed (left) and raw (right) images captured using V2 Camera.

format image. This image is displayed in Figure 2, on the right. As can be seen from the raw image, there are some very clear flaws, most notably the poor colour balance and the strong vignette effect around the image borders. Additionally, it is clear to note that a sharpening process has been performed – clearly visible around the margin between the roofline of the building and the sky. For comparative purposes, a photograph of the identical view was captured several seconds previously using the basic still command 'raspistill'. As a basic image capture this image will include four of the standard corrective processes that are performed automatically for basic image captures using the Raspberry Pi Camera V2.

The investigation into the raw Pi Camera V2 images showed that a very high level of automatic image processing exists in the Pi image capture pipeline. It was decided that for the purpose of this project, the full processing pipeline should remain in place, i.e. full default image processing should be carried out on all images captured. To attempt to improve on the image processing exhibited above would be outside of the scope of this project. It is recognised however that there are multiple challenges that the Pi Camera-based star sensor would face when employed on a CubeSat in low Earth orbit. Primarily, these would involve the following issues:

i. Glare from the Sun or Moon preventing the sensor from detecting stars.
ii. The presence of other solar system bodies, especially prominent planets such as Venus, in recorded images which may be mistaken for stars and confuse the star identification process.
iii. Additional noise being added to the recorded images.
iv. Occlusion of the stars by the Earth itself, which may also provide a source of false positives due to artificial lighting being observed from orbit.

In order to guarantee that the star sensor has been thoroughly tested in a simulated environment, it was important to investigate these factors as fully as possible. *Stellarium* can simulate the glare effect from the Sun and the Moon, as well as place the other solar system bodies amongst the test images in order to test the resilience of the system to their presence.

5.1.2 Use of Stellarium

Stellarium is an open source astronomy software designed primarily to help amateur astronomers locate and identify objects in the night sky. It has a detailed database of objects and can display their position in the sky for observer position on the Earth's surface for any given time period. Additionally, it can be used to simulate the view one would expect to see through different optical devices including telescopes, binoculars, and cameras. It is therefore a very powerful tool for generating views of the night sky.

The following figure, below, shows a screenshot taken within *Stellarium* demonstrating the indicated rectangular field of view of the Raspberry Pi V2 camera. Overlaid on the image are traces of the outlines of the major constellations to provide some reference of the scale. This exercise was useful as it provided an early indication of the visual information which would be contained in each frame that the camera of the spacecraft can observe. This was taken into account when generating the sample data to simulate the images that the camera would take when in operation onboard a CubeSat.

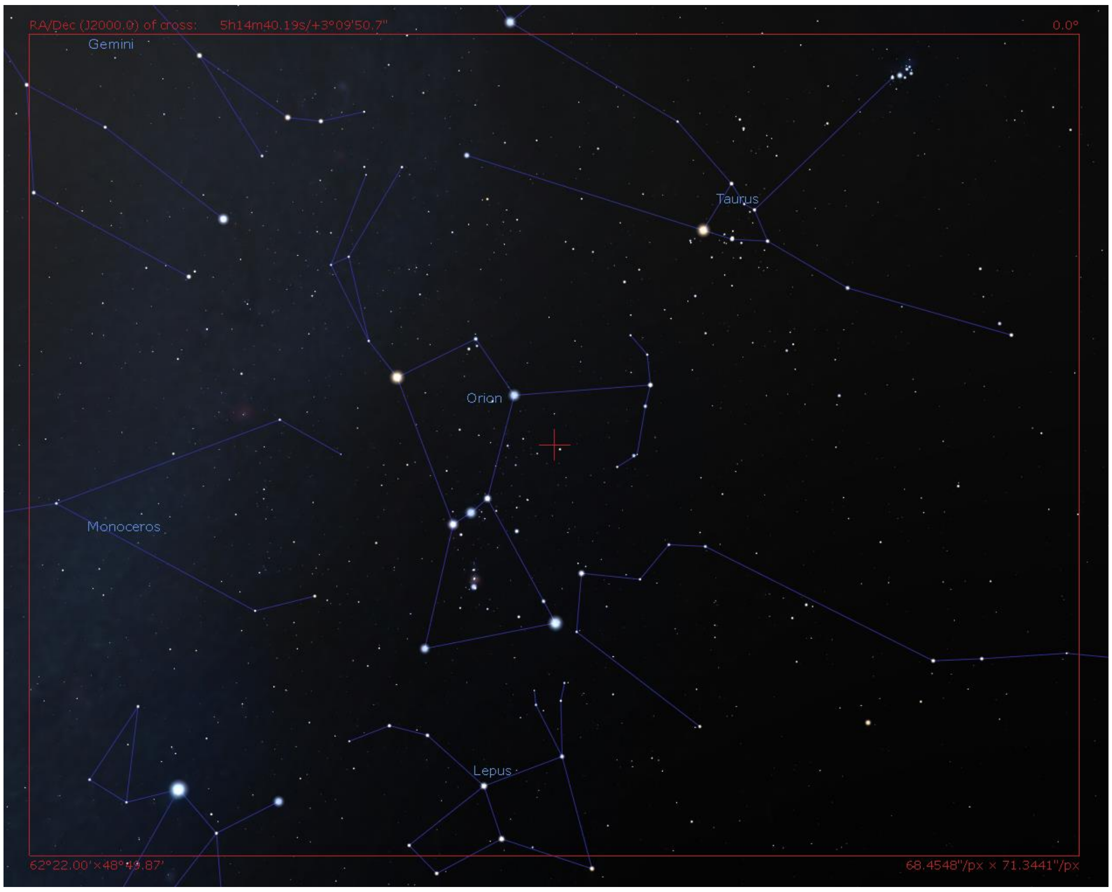

RA/Dec (J2000.0) of cross: 5h14m40.19s/+3°09'50.7"
0.0°
Gemini
Taurus
Orion
Monoceros
Lepus
62°22.00'×48°49.87'
68.4548"/px × 71.3441"/px

Figure 3: Simulated V2 Camera Field-of-View (Stellarium)

5.1.3 Use of a Linux Virtual Server for Cascade Training

As well as the Raspberry Pi 3B, a *Windows 10* PC was used for creation of the negative and positive datasets, and the use of *Stellarium*. It was originally intended to run the cascade training processes, which are computationally demanding, on this Windows PC as it is equipped with a powerful CPU, GPU, and 8GB of DDR4 RAM, sufficient to run HAAR training with acceptably short training times. The training could be performed on the Raspberry Pi itself, but the minimal 1GB of RAM in particular would be an unacceptable bottleneck for the process, and would result in long training times.

However, whilst the main functionality of *OpenCV* was used on *Windows 10* with success, it was not possible to use the 'create_samples' function required for the creation of the positive image samples. This is a known issue and the simplest solution is to hire a Linux Virtual Server (VPS) to run the training procedures on. A VPS with 8GB of RAM and *Ubuntu* 16.04.6 (LTS) x64 pre-installed was hired. 8GB of RAM was judged to be sufficient to provide acceptably short training times (to allow for quick testing and iteration), but windows without raising the cost prohibitively. The SSH client *Putty* was used to control the VPS via command line, and *WinSCP* was used to perform file transfer from the *Windows 10* PC. The following libraries and bindings were installed, following the guidance of Kinsley (2016):

- *cmake*
- *git*
- *libgtk2.0-dev*
- *pkg-config*
- *libavcodec-dev*
- *libavformat-dev*
- *libswscale-dev*
- *python-dev*
- *python-numpy*
- *libtbb2*
- *libtbb-dev*
- *libjpeg-dev*
- *libpng-de*
- *libtiff-dev*
- *libjasper-dev*
- *libdc1394-22-dev*
- *libopencv-dv, the OpenCV development library*
- *dos2unix*

After initial experimentation to confirm that the training process operated on the VPS was running correctly, a second VPS with identical specifications and setup was hired in order to double the rate of cascade creation and testing.

5.2 Experimental Programme

5.2.1 Bright Star Identification

It was identified that a Python program would be needed to identify the bright stars within a given star image, which was eventually amalgamated into the positive image creation program (see appendix for code, Section 11.2). A program was written in Python 3 using the functionality of the *OpenCV* library which employed the following process:

Initially the images are transformed to grayscale images, so that every pixel value is now between zero (black), and 255 (white), shown below as Equation 9 (OpenCV: Color conversions, 2019). An appropriate threshold value was chosen to preserve the stars themselves but remove other visual information. The threshold function converts all pixels which have a value of less than a specified value to either black, or if preferred, white, and all pixels above that value to the opposite colour. This stage outputs a binary colour image where each pixel is either white or black. In this case, the stars are made into black pixels and the sky is made to be white pixels, which facilitates the following stage.

$$RGB[A]\ to\ Gray: \quad Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{9}$$

The erosion function is employed next, to remove the minor stars and also further de-noise the image. The erosion function works by identifying each group of black pixels and removing rings of the outer pixels of each group step by step for a specified number of iterations. The effect of this is to remove all from the image except the brightest and most prominent stars. The erosion function cannot function for an iteration value < 1, but this was found to remove too many stars. To counter this, the image was temporarily resized prior to the erosion stage to double the resolution in order to prevent the erosion step from removing nearly all stars present.

The next step used in the program utilises the ‘findContours’ function of the *OpenCV* library, which determines each distinct group of black pixels on the white background by looking for the clear edge present between them. After this stage, the number of bright stars in the image is known, however their pixel locations are not. In order to do this, a bounding rectangle is drawn around each of the identified contours from the previous stage. The centre-point of each rectangle, which are the tightest fit possible around each contour, is known, in terms of the x and y pixel values from the top left corner of the image (0, 0). At this point in the program the program has successfully identified the brightest stars in the image and their locations in the image according to the pixel value that is closest to the centre of each point.

The following figure shows a screenshot of a debugging stage in the calibration and development of the above code. The red dots indicate the locations of the brightest stars, which are overlaid on top of the original full colour image, and text has been added to the top left-hand corner to display how many bright stars have been identified. The blue lines on the image are drawn from pixel (0, 0), top left, to each of the identified centre-points for the bright stars.

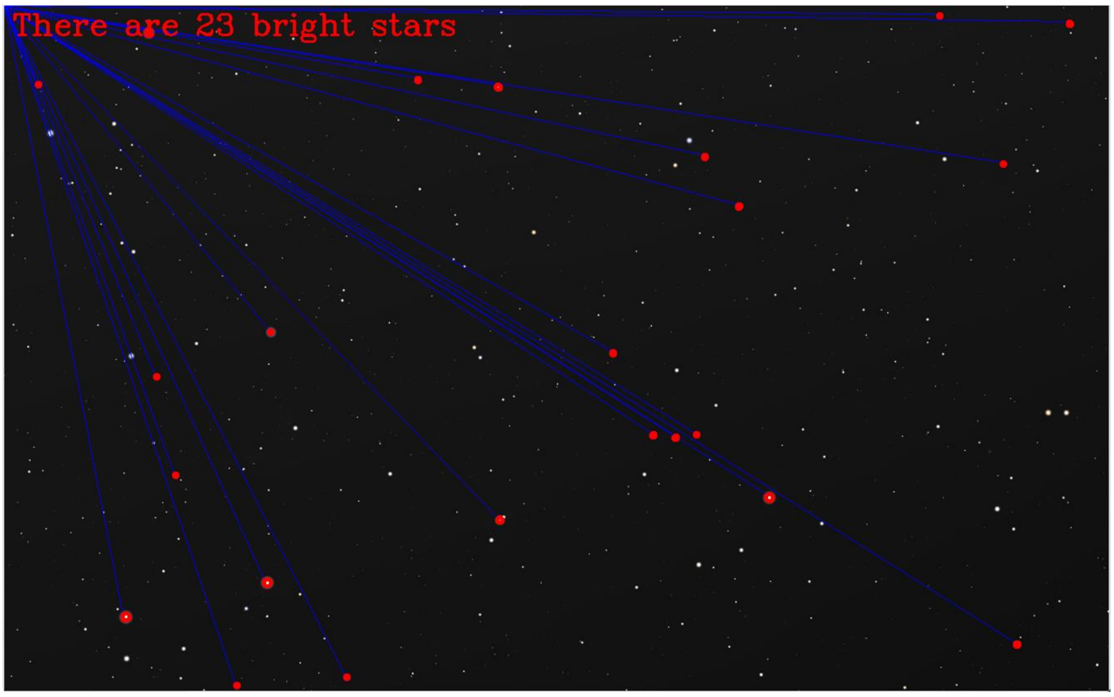


Figure 4: Identification of the brightest stars in a Stellarium-generated star image using functionality of OpenCV library.

5.2.2 Initial Cascade Test Results

Initial cascades that were trained were highly unreliable in providing a good true detection. Detection accuracy was improved by tuning of the ‘detectMultiScale’ function as described in Section 5.2.2.2.

5.2.2.1 Testing of Alternative Negative Image Datasets

It was considered that a potential fault with the initial test cascades was that of the negative image dataset. This had been generated from images captured from *Stellarium* using an automatic script to configure *Stellarium* for observation representative of the view of the Pi V2 Camera, and to take screenshots at a range of coordinates on the equatorial celestial coordinate system. The *Stellarium* script is reproduced in full as an appendix to this report (Section 11.1).

The first negative image dataset consisted purely of nearly 2000 images of the stars, grey-scaled and resized to 100 x 100 pixels. It was considered that the visual information of the tiny pale stars on this black image would be lost at such small resolutions, and so a second negative image dataset was produced which artificially exaggerated the size of the stars before the image was reduced to the same 100 x 100 pixels. This did not in fact offer any advantage at all when the cascade classifiers trained using this dataset were tested.

Due to the poor performance of the *Stellarium*-generated negative image datasets, an alternative was sought. *ImageNet*, a source of categorised images for neural network training (Russakovsky et al., 2015), was under maintenance at the time of this MSc Thesis, however a small sample of 2000 images of sports scenes was downloadable from another source (Kinsley, 2016). This was used as the negative image dataset for the following tests until it was enhanced by additional images (see Section 5.2.7).

5.2.2.2 OpenCV ‘detectMultiScale’ Function

The *OpenCV* function used to employ the cascade .xml files is called ‘detectMultiScale’, and takes multiple inputs used to control the behaviour of the function. The two main parameters used to tune the object detection are as follows:

i. scaleFactor – The detection process functions, as described by Nguyen (2018) by sweeping the detection area (20 x 20 pixels in every test run up to this point in the research) across the whole image. This is looking to detect a target object that is that particular size, and so it is necessary to scale the input test image for each successive ‘run’ of the detector, in order to detect true positives in the image that are larger than this window. A scaleFactor value of 1.05 for instance will shrink the test image by 5% following each detection run, until the smaller dimension of the test image (assuming rectangular) equals or is smaller than the 20-pixel specified detection size. The smaller this value, the more detection runs that are performed on the test image, and hence the processing time for the detection stage is increased.

ii. minNeighbours – This is a useful parameter to assist in the elimination of false positives. A ‘neighbour’ is referred to as another positive detection bordering the positive detection in question. The assumption is that a true positive will be identifiable as one which is a concentration of positive detections, whereas lone or small concentrations of positive detections are likely to be false. The higher this value, the more demanding the minNeighbour count required – and use of this parameter is a challenge in finding the optimum point where false positives are disregarded, but the true positive is not also rejected by an over-demanding minNeighbour requirement.

Through the manual tuning of these parameters, it was possible to achieve moderate identification success with the first batch of trained cascade files. However, it was clear that the cascades were moderately unreliable, and very prone to incorrectly selecting false positives or completely missing the true positive. It was considered necessary to improve the methodology employed with the aim of improving detection reliability.

5.2.3 The Need for Fiducial Markers

The testing of the initial batch of cascades gave rise to the additional hypothesis that it was the positive files employed in the classifier training that resulted in poor eventual cascade detection quality. The type of positive images used, namely simulated star-field images (from *Stellarium*), and the same star-field images with dilated bright stars, were assumed to not contain sufficiently identifiable HAAR-type features for a successful classifier training process to take place. As a result, it was considered necessary to alter the positive images supplied to the classifier training by adding additional visual information to the images.

The theory of adding additional information to the simple star images to aid classifier training by providing more HAAR-type features, as discussed in Section 4.5, was demonstrated with a simple test involving the placement of a simple 70 x 82-pixel greyscale image of a human face onto the brightest stars in an example image from *Stellarium,* using a photo-editing program (*paint.net*). The positive image taken from this sample image was used to train a cascade .xml file. When tested against the test image, this showed improved detection reliability over the previous tests, giving confidence to the hypothesis that it was necessary to enhance the positive images. Such markers placed on an image are known as 'fiducial markers', and are commonly used in object recognition or augmented reality applications. It was recognised that there were several key requirements in the selection of markers, namely:

i. Infinite rotational symmetry – ultimately the star identification cascades must be shown to be rotationally invariant – i.e. able to provide a successful identification regardless of the orientation of the supplied input image. As the input image provided to the identification pipeline when in a simulated real use-case will be at a different orientation to the original sample image from which the positive sample was extracted and the cascade trained, it is important that the markers will appear identical from any angle – hence the requirement for infinite rotational symmetry.

ii. The markers should contain feature types readily identifiable by the classifier training process. This should be verified by testing prior to selection of the most appropriate marker type.

5.2.3.1 Types of Fiducial Markers Tested

A Python script (see appendix for full code, Section 11.2) using functionality of the *OpenCV* library was used to automatically apply these markers to an input image from *Stellarium,* simulating the view of the Pi V2 Camera when on-orbit. The script employed the bright star identification methodology discussed previously in this report, and placed the markers on these locations. From this image, the positive image for training purposes was produced, as well as the marker-studded output test image which would be used to verify the performance of the trained cascade .xml file.

5.2.3.2 Fiducial Marker Test Results

Five cascades using different input positive images were tested for each of the six types of fiducial marker, based on those discussed in the reference material analysed in Section 4.5. The results of these tests are summarised in the table, below:

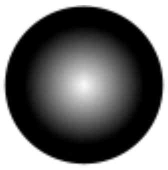
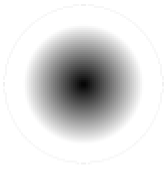
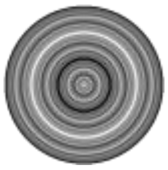
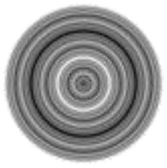
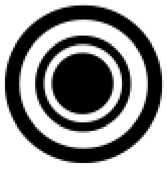
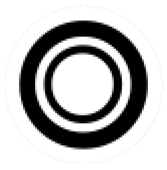
Fiducial Marker	**Marker Image**	**Mean Acceptance Ratio of Final (9th) Stage**
1 – Gradient type		0.07248353
2 – Inverse gradient type		0.00306776
3 – Fourier tag type		0.02088470
4 – Inverse Fourier tag type		0.01176640
5 – CCTag-type		0.01921776
6 – Inverse CCTag-type		0.00567280

Table 1: Test results of fiducial marker type investigation.

Markers types 1, 3, 4 and 5 could be immediately discounted as viable options due to the unacceptably poor acceptance ratio (analogous to precision) seen on the final (9th) training stage. When tested against an input test image, all three of these fiducial marker types resulted in highly unreliable detection of the true positive in the image.

Marker types 2 (Inverse gradient tag) and 6 (Inverse CCTag) were shown to exhibit very satisfactory performance against input test images, with clear identification of the true positive and good resilience against accidental selection of false positives. Either of these marker types are judged to be appropriate for implementation in this computer vision star identification project.

However, a further requirement is for the markers placed on an image to not be placed on-top of each other (overlapping), but rather merged or blended together. At the beginning of the star identification pipeline the input image is received from the Pi V2 Camera, and the fiducial markers are applied to this. If the markers are allowed to overlap each other rather than blended, then it cannot be guaranteed that the markers will always be applied in the same order by the Python program. Given this, although the markers may be applied in the same locations, the visual pattern presented to the identification cascades may differ – resulting in poor detection reliability. It is therefore vital to blend the markers together so that the apparent pattern will be identical, regardless of the order in which the Python program identifies the bright stars.

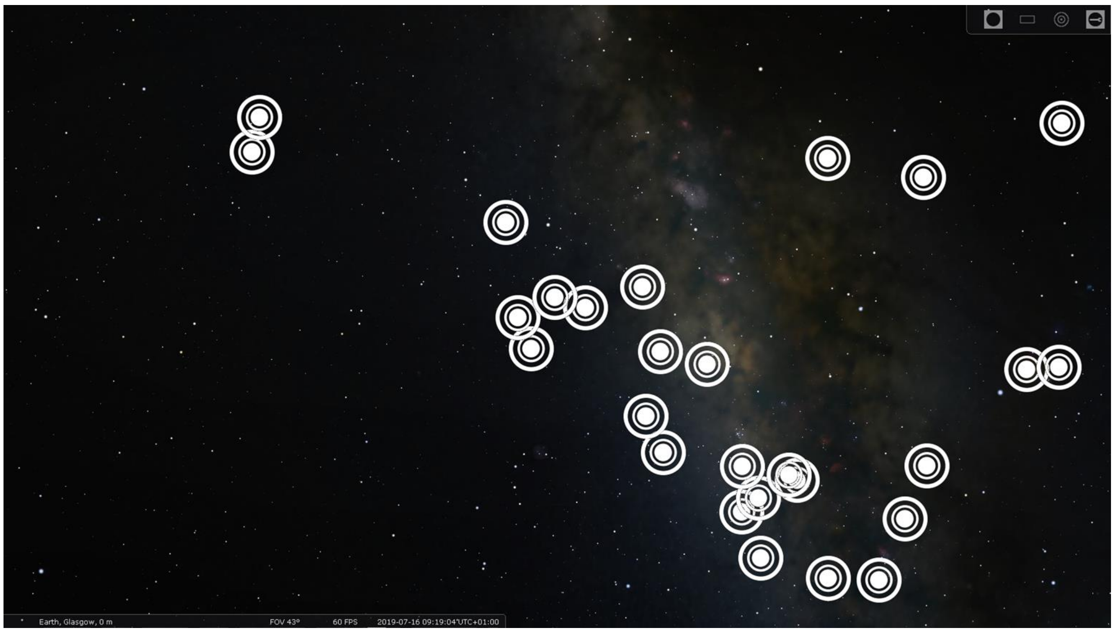


Figure 5: Example of star pattern with overlaid fiducial markers, overlapping Type 6.

Fiducial marker type 6, an inversion of the CCTag developed by Gatrell, Hoff and Sklair (1991) was more suitable for this application as the black rings of the marker could be easily treated as transparencies by the *OpenCV* image placement function, and so can be used to create an overlapped fiducial marker pattern, as can be seen in the above image, an example of the test images used for fiducial type 6.

5.2.4 BASIC or ALL Mode

A parameter of the *OpenCV* 'train_cascade' function that was hypothesised to be worthy of investigation was the '-mode' parameter, which determines the HAAR features that the training procedure will utilise. Unless otherwise specified, this defaults to 'BASIC', which only utilises vertical HAAR features, the original type of HAAR features as discussed by Viola and Jones (2001). Lienhart and Maydt (2002) developed an additional set of HAAR features that are at 45º angles, and these can be employed in *OpenCV* by the following 'train_cascade' parameter: '-mode ALL'. To examine whether the use of the extended feature set would result in more reliable cascades, comparative tests were performed in which that was the only variable. The results are displayed in the following table:

-mode Setting	**Acceptance Ratio of Final (9th) Stage**	**Duration of training process (seconds)**	**File size of cascade.xml file (kB)**
BASIC (default)	0.00263295	720	55
ALL	0.00294665	1680	56

Table 2: Cascade training '-mode' setting comparison test results.

The extended feature set resulted in a significantly longer training duration, more than double that of the training process utilising the basic set of HAAR features, but did not result in a lower acceptance ratio value. Testing of both cascades on the test star image in fact showed that the cascade trained using the extended feature set resulted in more false positives than that trained using the basic set. As a result of this test, it was decided to continue employing the basic (default) HAAR feature set in all further tests.

5.2.5 Local Binary Pattern (LBP)

A further parameter of the *OpenCV* 'train_cascade' function is that of 'featureType'. This refers to the type of image features that the training process utilises, and by default it is set to HAAR features. Alternatively, by employment of the '-featureType LBP' command, the training process can be set to instead utilise Local Binary Pattern features, as discussed in Section 4.4.1. Multiple comparative tests were performed to compare the specifications and performance of cascades trained using the rival methods. The expectation was that LBP-trained cascades would be faster to train and employ, but would offer lower identification accuracy.

Due to this very fast training time, this permitted the exploration of improving cascade accuracy and reliability through training with greater image resolutions, or by using more training stages – both measures which would have resulted in unacceptably long training durations if employed in conjunction with a HAAR-type cascade. After comparison to HAAR using the original settings of 10 stages and an image size of 20 x 20 pixels, sizes of 30 x 30, 40 x 40 and 50 x 50 were trialled. It was noted that although these yielded improvements in the Acceptance Ratio parameter, training time increased exponentially. It was found that a more efficient means of increasing Acceptance Ratio with a less significant training time trade-off was through the utilisation of more than 10 training stages, however the potential number of stages utilisable was limited by the availability of positive samples (1800) to only 12 stages. The file sizes of the LBP-type cascade .xml files were noted to be considerably smaller than the equivalent HAAR-type cascade .xml files, which would result in faster processing times and also make less demand on memory allocation when ultimately deployed on the Raspberry Pi.

The following table contains details of comparative HAAR and LBP training tests:

Feature Type	Resolution (pixels)	Stages	Training Time (s)	Acceptance Ratio of Final Stage	Cascade .xml file size (kB)
HAAR	20 x 20	10	1320	0.00483941	66
LBP	20 x 20	10	7	0.00717194	22
LBP	30 x 30	10	31	0.00511454	23
LBP	40 x 40	10	210	0.00554799	21
LBP	50 x 50	10	804	0.00425359	19
LBP	30 x 30	12	49	0.00204217	27

Table 3: LBP and HAAR feature type comparison test results.

5.2.6 Testing for Rotational Invariance

The *OpenCV* ‘create_samples’ function, as previously discussed in Section 4.4.2, automatically creates a large set of positive images from a single input positive image. The function overlays the positive image sample onto the negative images provided, creating an artificial set of positive images, containing the object of interest. A number of parameters can apply various manipulations to the input positive image in order to improve the resilience of the trained cascade to variations in the target object. Examples of this are the ‘maxxangle’, ‘maxyangle’ and ‘maxzangle’ parameters. These set a limit of rotation in those respective axes, and the input positive image will be randomly distorted or rotated within those limits. The z axis in this case refers to the axis projecting directly out from the face of the image, and is therefore the axis of interest when considering the rotation of the view of a given star pattern for an observer, as it is analogous to the central axis of the camera lens.

It is vital to ensure that the cascades can perform a successful and reliable positive identification for any given angle at which the target star pattern is presented to the relevant cascade. It was hypothesised that by setting the maxzangle (which takes an argument in radians) to the value of a full rotation, that the training process would create positive samples over the limit of a full rotation, therefore creating a cascade file able to recognise that particular positive image in any orientation.

In practice, the maxzangle parameter returned an error if provided with an input value of greater than 2 radians. A cascade trained with this value also had an acceptance ratio value of 0.0132771 (compared to the 0.00497771 of the same cascade trained with maxzangle 0.5) on the final (9th) stage, which is normally an indicator of poor performance. Indeed, testing showed that this cascade was unable to correctly identify the target star pattern, even if no rotation was induced in the input image.

5.2.7 Expansion of Negative Image Dataset

It was seen that when using a higher value of the maxzangle parameter in the ‘create_samples’ function this lowered the performance of the trained cascade considerably. This was to be expected, given that the same number of positive samples (1800) were now randomly distributed over a rotation range four times larger, and so the ability of the cascade to successful discriminate the true positive result from the false positives was degraded. In order to overcome this, more training stages were required to produce a more accurate cascade, however as discussed previously this number of positive samples limited the cascade training to a maximum of 12 stages.

As the positive samples are created using the 'create_samples' function from overlaying the single positive sample onto the negative image dataset, it was necessary to expand the size of the negative image dataset. The dataset of *ImageNet* origin of sports scenes was enhanced by the addition of a wide variety of images from the author's own photographs, bringing the total number of images in the database from 1964 images to 5010. Training was performed using 4000 positives samples, and 2000 negative samples (increased from 1800 positive and 900 negative), maintaining the 2:1 ratio of positive to negative images.

A direct comparison was made between the two dataset sizes to show how the adoption of the larger dataset improved performance. Although 20 training stages was the initial goal, this proved to be impractical due to the exponential increase of training time of each additional stage, and so 15 training stages was deemed to be good compromise between cascade accuracy and training duration. The results of the comparative test are shown in the following table *(the training times were not recorded for these tests)*:

Negative Dataset	**maxzangle (radians)**	**Acceptance Ratio of Final (9th) Stage**	**File size of .xml file (kB)**
Original (1964 images)	0.5	0.00497771	20
Enhanced (5010 images)	0.5	0.00028917	46
Original (1964 images)	2.0	0.01327710	28
Enhanced (5010 images)	2.0	0.00105548	73

Table 4: Results of negative dataset comparison tests.

From the data displayed above it can be confirmed that the original dataset size of 1964 images was unable to provide sufficient positive samples to support the larger rotation angle of 2.0 radians, with the acceptance ratio dropping unacceptably to 0.013 (3 decimal places). This particular cascade was unable to provide a successful positive ID on the test input image. The larger dataset, however, allowed the training of a total of 15 stages for each cascade, which pushed the acceptance ratio significantly lower than those seen from the 10 stage cascades trained on the smaller dataset. A significant increase in file size of the cascade .xml files can also be observed, reflecting the additional complexity of the training process that was performed.

5.2.8 Further Rotation Testing

The larger negative image dataset permitted the training of functional test cascades with higher maxzangle values. Although the 2.0 radians training input is only the equivalent of 32% of a full rotation, the cascade trained with this maxzangle value was shown to be able to perform a successful ID on test input images showing the same star pattern in images distributed over a range of 360º rotation, with varying accuracy. As expected, the reliability of successful target identification was much improved over the cascade trained for a maxzangle of 0.5 radians.

A successful true positive could be identified in an image provided at varying angles over a full 360-degree rotation, but not for the same detectMultiScale parameters. It was shown that the reliability of successfully discriminating the true positive from surrounding false positives varied, raising the need for a more complex automatic identification procedure making use of a confidence value to assess the quality of each detection made by the cascade on the test image. Despite these weaknesses being observed, these tests led to mistakenly high confidence in the rotational performance of the cascades. Further analysis of rotational performance is presented in Section 7.2.3.

Some rotated test images led to no successful star pattern identification for any ‘detectMultiScale’ setting by the otherwise successful cascades. It was observed that in these particular images, the *OpenCV* script responsible for applying the fiducial markers to the test image had selected a different set of stars to apply the markers to. The LBP-type cascades proved to be highly reliant on being shown exactly the same pattern of markers – small variances in marker pattern resulted in failed detections.

A potential solution to this problem would be to train cascades on a number of positive samples, each showing the same star pattern with minor variations in the pattern of fiducial markers displayed, but this would be a more convoluted procedure, and a large inconvenience considering the number of cascades that would need to be trained to provide a workable model for identification over the whole of both celestial hemispheres. This issue was later addressed in Section 7.2.1.

5.2.9 Confidence Value Utilising ‘levelWeights’ Parameter

To be able to reliably and automatically identify the true positive amongst the false positives in any given detection, a measure of the confidence of the cascade in the detection of each identified positive would be required. The detectMultiScale can provide a value name ‘levelWeight’, which is the confidence value in the detection, at the final stage. Additional code was developed in the test script to record this value for each detection, and allow the code to select the result with the greatest confidence value.

5.2.10 Orientation Calculation

In order to be able to be able to use the computer vision star identification cascades to sense the orientation of the CubeSat, it is necessary to perform some mathematical conversion steps. A minimum of two star pattern targets must be successfully identified within any given input image for successful orientation sensing. The right ascension and declination celestial coordinate values for the J2000 epoch for any given star target (from which an individual cascade file was trained) is known, and this data is supplied to the Python script from the file name of the cascade itself (see Section 5.3.1). It is therefore known that a certain pixel coordinate on the image is matched to that particular celestial coordinate, so therefore the celestial coordinate aligned with the axis of the camera (the central pixel in the image) can be found.

The Pi V2 Camera features a rectilinear lens, like the majority of commercially available cameras. A lens such as this will produce an image in a perspective, or gnomonic projection, in which all great circles are presented as straight lines. The image taken by the Pi V2 camera of the stars will therefore be a perspective image, which is an oblique aspect tangent to the equatorial celestial coordinate system.

5.3 Overview of Final System

This section discusses the development of the final system using the processes developed in Section 5.2.

5.3.1 Target Selection and Final Cascade Training

It was decided that purely the northern celestial hemisphere would be considered for testing of the computer vision system. The justification for this choice was that a large number of cascades would need to be trained for each hemisphere, and to perform testing on both hemispheres would double the training and testing time required – it was considered more profitable to instead perform more rigorous tests on the cascade set for the northern hemisphere alone. There are only minor differences in the code required to perform the orientation task on the southern celestial hemisphere (in relation to handling negative declination values), but the methodology would be identical.

Figure 6 shows a Zenithal equidistant plot of the northern celestial hemisphere with red dots indicating the RA/Dec coordinates of the centre-points of the final trained cascades. Whilst their locations appear randomised, they are all located on the location of recognisable star patterns with absolute magnitudes that would be most likely to be detected by a small CMOS camera chip. The density of the chosen coordinates was

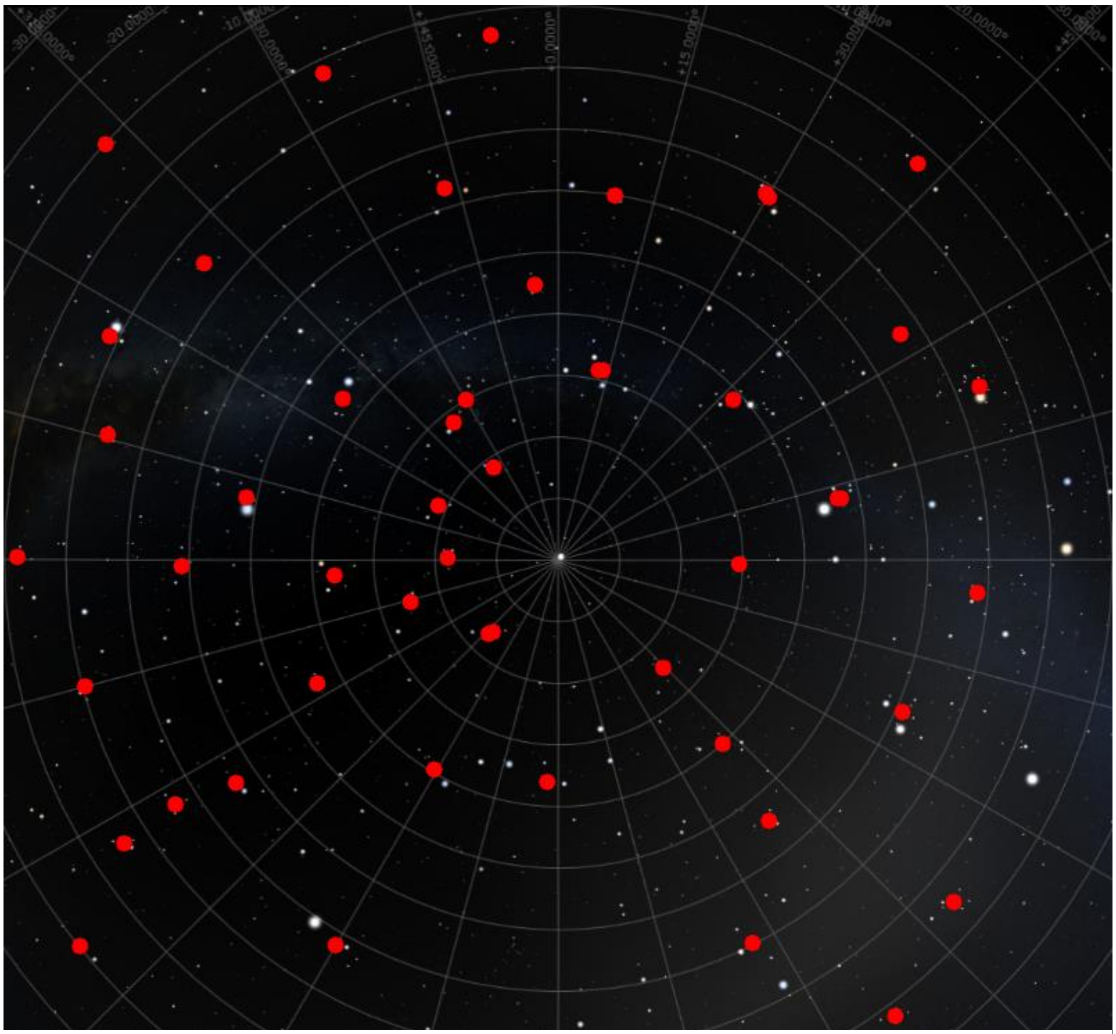


Figure 6: Zenithal Equidistant plot of northern celestial hemisphere, showing the RA/Dec coordinates of the 55 cascdes trained initially.

designed to reflect the need of the orientation calculation stage in requiring two correct positive identifications. The final commands containing the parameters selected to use for the training of the final cascade set are reproduced below. These parameters were selected based on the findings of the iterative tests performed in Section 5.2.

```
opencv_createsamples -img “image name”.jpg -bg bg.txt -info info/info.lst -pngoutput info -maxxangle 0.5 -maxyangle 0.5 -maxzangle 2 -num 4800
```

```
opencv_createsamples -info info/info.lst -num 4800 -w 30 -h 30 -vec positives.vec
```

```
opencv_traincascade -data data -vec positives.vec -bg bg.txt -numPos 4000 -numNeg 2000 -numStages 15 -w 30 -h 30 -featureType LBP
```

Following the training of these 55 cascades, their performance was tested in order to verify their individual performance prior to combining them into a single set:

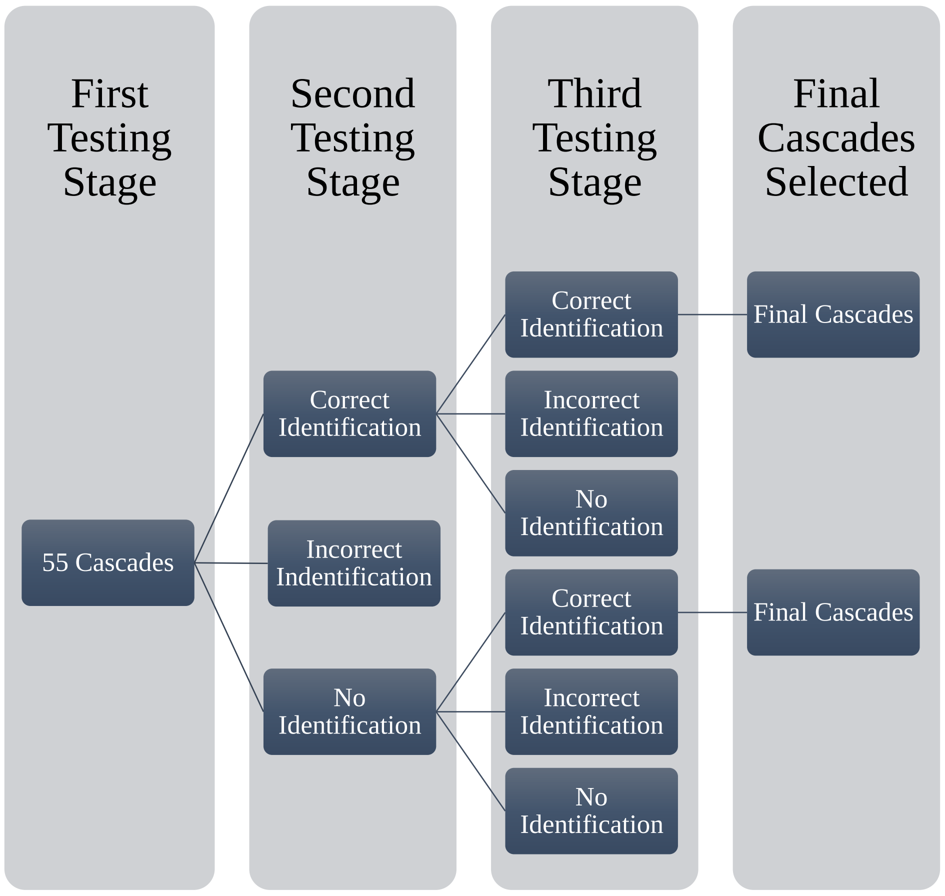


Figure 7: Final cascade testing process.

The test stages shown in Figure 7 are described fully below:

i. A Python script was written to run each cascade against an input image containing the positive star pattern in a central position. Of the 55 cascades, the cascades that performed correct identifications, incorrect identifications, and no identifications were recorded, for the following 'detectMultiScale' parameters: scaleFactor = 1.05, minNeighbors = 17, minSize = (200, 200), maxSize = (400, 400). The cascades that returned a false positive were removed from the set.

ii. The second testing stage was to modify the detectMultiScale parameters for those that did not perform any detection during the previous test stage. The minNeighbors requirement was lowered to 10. This time, those that did not return a correct detection or any detection at all were removed from the set.

iii. To not give undue precedence to any individual cascade, it is imperative that they are assigned a weighting to ensure that their sample levelWeights (or confidence value) are equivalent for a sample detection. This was calculated as the number that the levelWeights value for a successful detection performed by that cascade needed to be multiplied by in order to obtain the nominal value of '5'. Following the creation of these values, the remaining set were tested against their respective sample images containing the positive star pattern, and once again the cascades returning false positives or no detections were removed.

iv. The remaining cascades numbered 31. Testing the entire set of cascades against each individual image containing an example of the positive star pattern (relating to an individual cascade within the set) in a central position, the set returned 100% identification of the true positive, with zero false positives and zero failed detections.

The file naming format for the cascade.xml files was as follows:

"1,83.92080,-1.14830,17,1.078.xml"

The Python test script identifies each parameter separated by a comma. The order of parameters is cascade identification number, right ascension of target object, declination of target object, minNeighbors parameter tuned for that cascade, and the weighting factor applied to the levelWeights value produced by that individual cascade.

The 31 down-selected cascades that exhibited reliable performance in test conditions are shown as their associated RA/Dec coordinates in Figure 8. Although this does not appear to offer many identification points in the northern celestial hemisphere, due to the relatively wide FOV of the Raspberry Pi V2 Camera (62.2° in the horizontal axis and 48.8° in the vertical axis (Camera Module - Raspberry Pi Documentation, 2019)), these are deemed to be sufficient to guarantee a minimum of two detections for any given FOV within the northern celestial hemisphere.

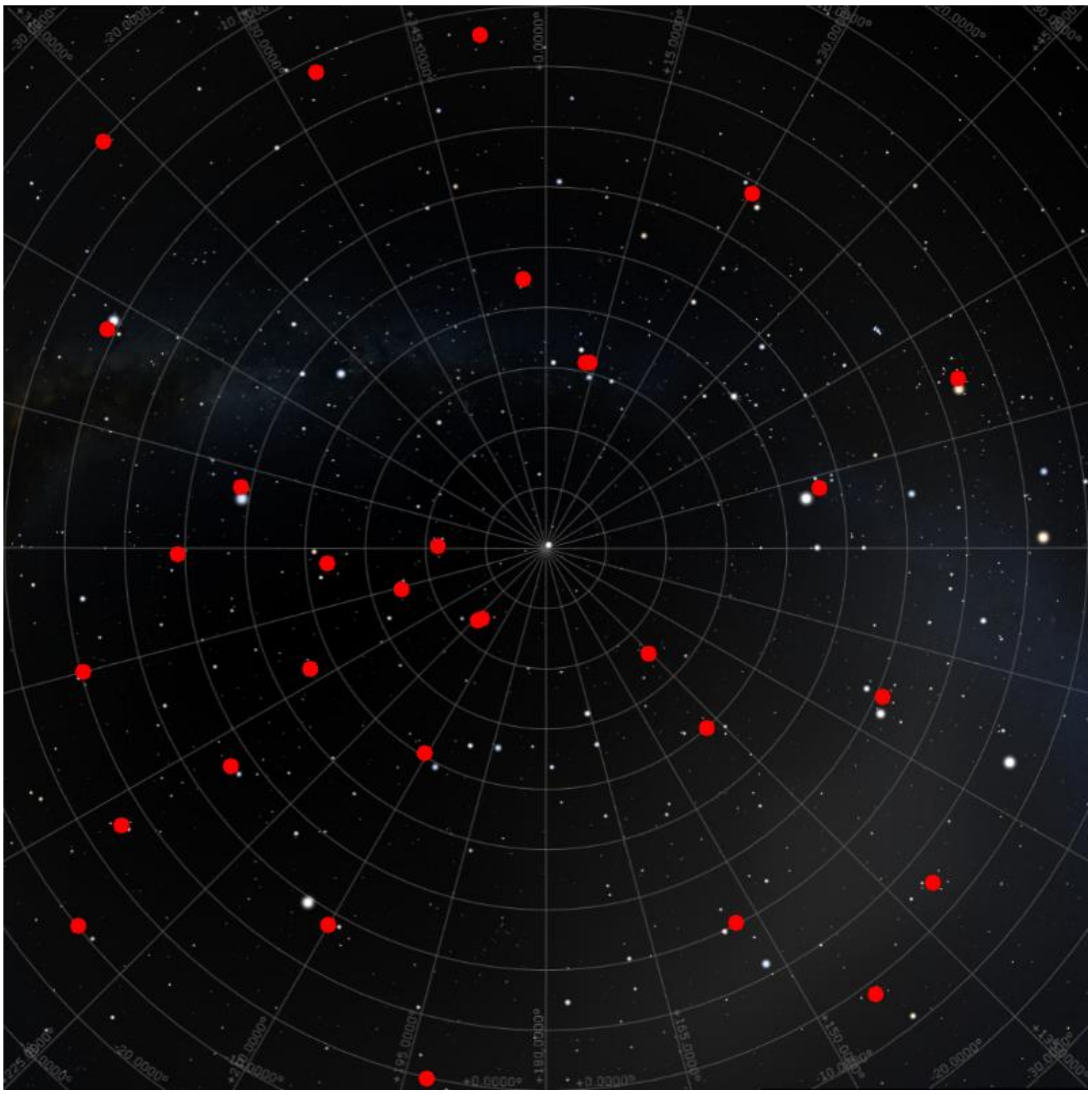


Figure 8: Zenithal Equidistant plot of northern celestial hemisphere, showing RA/Dec coordinates of the 31 down-selected cascades.

5.3.2 Pixel to RA/Dec Conversion

Once a pair of star patterns are successfully identified by the system, by testing an input image (from the Pi V2 Camera in real-world usage, from *Stellarium* in simulated testing) against the 31 final cascades, the following data is known to the program: pixel coordinates of the camera axis (image centre), pixel coordinates of two detected patterns, for each of which is known a RA/Dec coordinate pair. The next step is to determine the RA/Dec coordinate of the central pixel, corresponding to the alignment of the camera axis, and the orientation of the camera body (and by extension the CubeSat) relative to the equatorial celestial coordinate system.

Employing the equations described by Calabretta and Greisen (2002), as discussed more fully in Section 4.6.1, the conversion is provided by the following means:

i. For the first pixel coordinate and RA/Dec pairing, from the first identified target, the following equations are employed, using modifications of the equations given by Calabretta and Greisen (2002):

$$R_\theta = k \cdot \tan(90 - \theta) \tag{9}$$

Where k = 311, a scale value, θ = declination of target, and ϕ = right ascension of target.

$$pole\ plot\ x = 960 - \ R_\theta \sin(\phi) \tag{10}$$

$$pole\ plot\ y = 540 + \ R_\theta \cos(\phi) \tag{11}$$

Where x and y are cartesian coordinates of a gnomonic projection plot centred on the pole of a celestial hemisphere. The values of 960 and 540 relate to the central pixel of this plot, created in the same resolution as the input image, on which the detections were made.

ii. The procedure to find a pair of new cartesian coordinates on a polar centred gnomonic projection of the same resolution as the input image is now repeated for the second pixel coordinate and RA/Dec pairing.

iii. On the original input image, a triangle is found which links the two pixel coordinates of the detections, and the central pixel (camera axis).

iv. This triangle is scaled to fit onto the new pixel coordinates that relate to the polar centred gnomonic projection. The pixel coordinate of the third point of the triangle (that related to the camera axis on the input image) is converted to RA/Dec coordinates (using the equations described by Calabretta and Greisen (2002)).

v. The rotation of the camera body relative to the equatorial celestial coordinate system can be determined by using the *Astropy* library by creating a customised WCS centred on the RA/Dec value of the camera axis, and rotating the detection points incrementally until the WCS function successfully maps the coordinate values of the two detections onto their known RA/Dec coordinates.

Following this stage, the orientation of the CubeSat's camera central axis should be known, and this can be transformed to any chosen reference frame relative to the satellite's body.

6 Results

This section covers results of tests performed to attempt identification of star patterns in test images from *Stellarium*.

6.1 Verification of System Performance

The final set of cascades trained to identify 31 star patterns within the northern celestial hemisphere were tested using a set of images captured within *Stellarium* representing the FOV of the Raspberry Pi V2 Camera. For each test image, all 31 cascades were tasked to performed detections, and the Python program would distinguish between true and false positives based upon the weighted levelWeights value, the size of the bounding rectangle, and the specified minNeighbors and scaleFactor parameters of the 'detectMultiScale' function. The results were displayed graphically, as exhibited in the example of a successful identification of two true star patterns in Figure 9, below. The red text above the bounding rectangle displays the central pixel coordinate of that rectangle, the identifying number of that specific cascade file, and below the rectangle the text displays the weight levelWeights (confidence value) result for that detection.

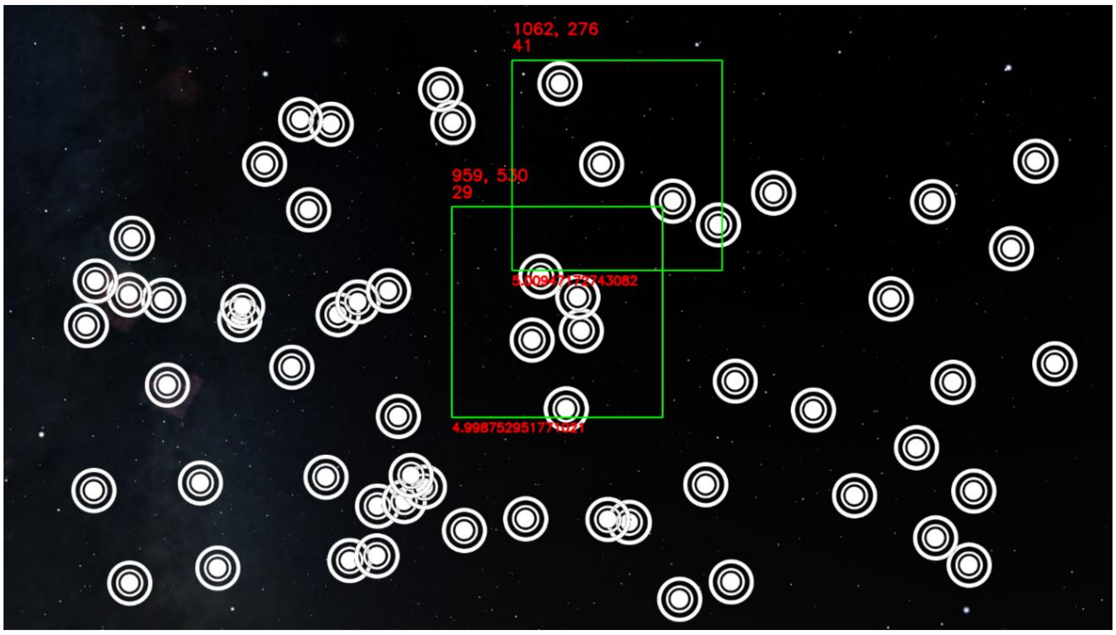


Figure 9: Successful example of two true positive detections made on sample image.

However, this successful result was seen in < 10% of the test images provided to the detection program (presented in full as an appendix to this paper, Section 11.5). When the target star patterns appeared near the peripheries of the image, rather than centrally, the cascades failed to perform a correct detection. Additionally, it was seen that the levelWeights value for a successful detection by a specific cascade could vary by a large degree depending on the location of the detection within the image, presumably as the minor distortions and rotations seen in the marker pattern due to the distortion of the gnomonic projection affected the performance of the detection. The issues seen during these tests can be summarised as:

i. Poor rotational invariance – cascades would frequently fail to spot a visible target star pattern if rotated by more than approximately 90°.

ii. Marker pattern variation – the program used to apply the fiducial markers to the brightest stars in the image would not reliably place markers on the same bright stars, resulting in minor variation in the markers patterns seen. This would affect the ability of a cascade to spot the desired target star pattern.

iii. Distortion caused by projection effects – the 100% reliability of the cascade set to identify to identify their target star patterns when the star patterns were located centrally in the image degraded to < 10% when the target star patterns were located closer to the peripheries of the image, due to the distortions induced by the gnomonic projection.

iv. Great significance given to false positives – the cascade set generated false positives with levelWeights (confidence values) greater than those seen when testing the cascades against their true target star patterns, resulting in it being challenging to programmatically distinguish between true and false positives.

In summary, the full cascade set was almost totally unable to correctly identify the target star patterns when tested on randomised northern celestial hemisphere sample images from *Stellarium*.

7 Results Discussion

This section adds further analysis to the results of the final system test.

7.1 Cause of System Failure

The results of the testing of the final system as discussed in Section 6.1 showed a high degree of unreliability of the system to correctly identify target star patterns that the 31 final cascades had been trained to identify. Further tests were made to determine the root cause of the unreliability.

A main cause for error is the repeatability of the method used to apply the fiducial markers to the input image, to match the fiducial marker patterns that the cascades were trained against. Tests showed that the program would show variation in the bright stars identified, and hence variation existed in the fiducial marker pattern produced for any given star group. The cascades showed adaptability to situations where one or more expected markers were missing from the image, but this also hints at great susceptibility of the cascades to identifying false positives in other similar marker patterns, a trend that was exhibited strongly during overall system testing. This also suggests that the accuracy of the cascades is not sufficient, which is possibly reflected by the Acceptance Ratio values for the final stages as seen during the cascade training process – these values, analogous to cascade accuracy, deteriorated significantly depending on variation in the positive images, and other training parameters such as rotation about the z axis. It is possible that by training for a larger dataset of positive and negative images, and employing more training stages, then more accurate and resilient cascades could be produced.

The adoption of the *OpenCV* 'create_samples' function to produce a large dataset of positive images from a single input positive image may be a further root of unreliability. Although the 'create_samples' function can apply a variety of distortions and changes to the sample image, these artificial simulations of a large dataset of positive images may not provide sufficient variation to train a classifier which is able to perform a reliable positive identification even under rotated and distorted conditions. Those rotated and distorted conditions are seen due to the projection that approximates the lens distortion onto the equatorial celestial coordinate system.

There is a possibility that the method may simply require a significantly more rigorous training procedure, utilising far more samples, such as the 40,000 used by Kim and Brunner (2017), or many more stages, could improve cascade accuracy to the degree needed to avoid the false positives case. It appears from testing of the final system that the cascades can produce highly variable levelWeights values following a detection (whether true or false)

7.2 Modifications to Improve System Performance

To address the weaknesses of the system as described in Section 6.1, a number of further experiments were performed, to provide a more thorough understanding of the cause of the characteristics exhibited by the complete system.

7.2.1 Fiducial Marker Placement Program

It was observed that small variations occurred within the marker patterns depending on the location of the target star pattern within the image. This caused problems in detection by the cascades, as the appearance of the target star pattern would vary from that supplied to the cascade training process.

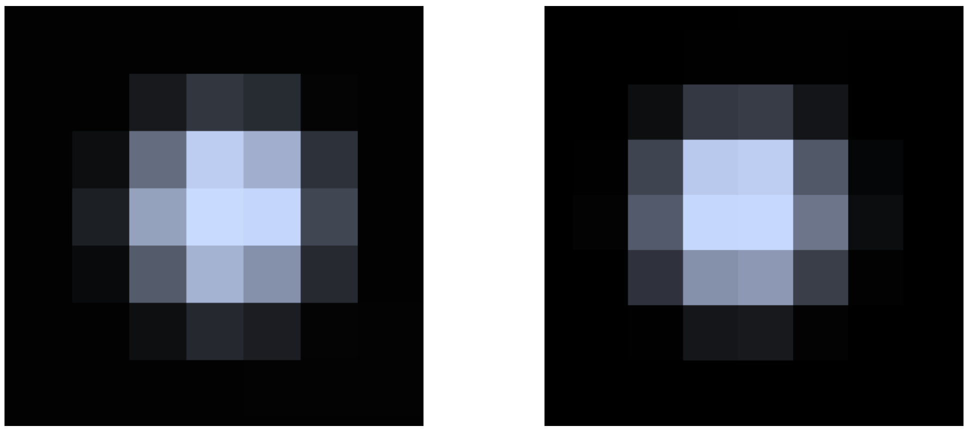


Figure 10: How pixel resolution of image affects detection of a given star.

Figure 10, above, illustrates the cause of the issue. The two images show the same star, taken from two different images of approximately the same area of the northern celestial hemisphere. The left-hand image shows the star placed with a central pixel. This star will still be present in the processed image following the erosion process (described further in Section 5.2.1), however in the right-hand image the star does not fall so that it has a central pixel, and hence will be removed from the image following the erosion process. This is a clear weakness of implementing the *OpenCV* erosion function to remove minor stars, as it will perform inconsistently dependent on pixel resolution.

An alternative methodology was sought, which made use of the research by Lindblad et al. (1997), Trask (2002), and Qian et al. (2013), which describes the limit magnitude of typical star sensors as being in the region of 5 magnitude. The test images for this Thesis were captured in *Stellarium*, and a limit magnitude of 5 was applied, so only stars with an apparent brightness greater than this were displayed, to simulate the sensitivity of the Pi V2 Camera when in use. The erosion function was removed from the marker placement program, so that fiducial markers were applied to all stars visible within the image. This resulted in an increased number of markers being present, but removed any variance in the marker placement patterns as seen previously.

7.2.2 Improving Cascade Accuracy

It was hypothesised that the accuracy as gauged by the 'Acceptance Ratio' value of the final stage of the cascade training process was not sufficient to provide reliable detection. To improve this, a larger negative image dataset was required. A new dataset of 8898 images was formed, as a combination of the previous largest dataset employed, with the addition of a set of images depicting images of the southern celestial hemisphere complete with placed fiducial markers. This permitted the training of cascades with 8000 positives images (as created using the *OpenCV* 'create_samples' function), and 4000 negative images, double the number as used to train the full cascade set as described in Section 5.3.1.

To reduce the significantly increased training time, a pair of more powerful Linux Virtual Servers with 16GB of RAM were hired, a RAM size increase of 100%. By employing 19 training stages, and the larger image dataset, a cascade with an acceptance ratio on the 19th stage of 1.44886×10^{-5} was created, an acceptance ratio smaller than the mean acceptance ratio of the full cascade set as discussed in Section 5.3.1 by a factor of approximately 10x. The performance of the cascades trained to a higher degree of accuracy (as measured by the Acceptance Ratio) exhibited marginally improved reliability. Whilst this dataset was twice as large as that used previously in the experimentation process of this MSc Thesis, it should be noted that it is still only 1/5th of the size of the dataset employed by Kim and Brunner (2017), see Section 4.2.

7.2.3 Rotational Invariancy of LBP-feature Cascades

To identify the true positive amongst the false positive detections, it was desirable that the confidence value (as gauged by the *OpenCV* 'levelWeights' parameter as discussed previously in Section 5.2.9) would remain relatively constant for a given cascade irrespective of the angle at which the target star pattern is presented to the detection process. Further rotation tests were performed to analyse the consistency of the confidence value for a true positive at different image angles, for a number of cascades trained using different parameters, four of which are presented in Table 5, below. The rotational performance of these cascades is graphed in Figure 11, below.

Cascade Number	Positive Samples	Training Stages	Acceptance Ratio of Final Stage	Cascade File Size (kB)	maxzangle (radians)
1	8000	19	0.000066	85	1.0
2	8000	19	0.000014	63	0.5
3	3000	15	0.000162	31	0.5
4	8000	18	0.000294	100	2.0

Table 5: Training Properties for Four Selected Cascades.

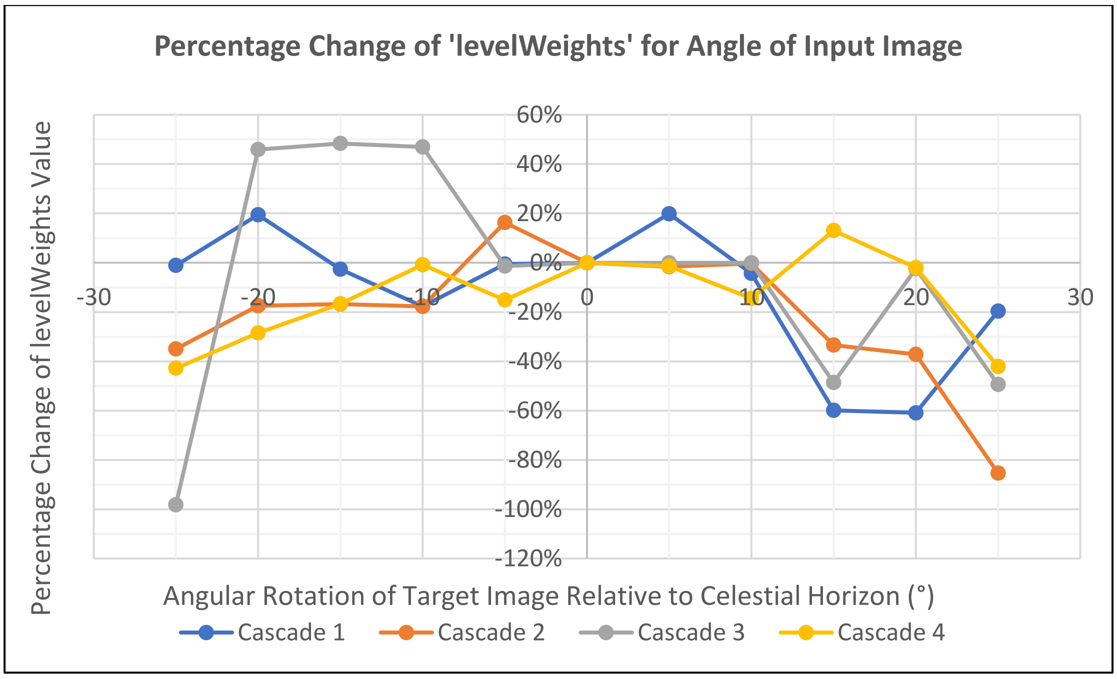


Figure 11: Graph showing rotational performance of four cascades.

The purpose of the above figure is to demonstrate how the levelWeights value (analogous to a confidence value) of a true positive detection for any given cascade will degrade as the rotational angle of the input image increases relative to the level image which was use as the positive sample for the training process. The nature of the degradation is not representable by a function that could be used to predict degradation, as it is highly irregular and unpredictable. The result of this is that the trained cascades are only dependable for a narrow rotational range, which is considerably smaller than the rotational range for which the cascades were trained for (maxzangle, see Table 5).

The widest angular range on which a true detection was made was seen from ‘Cascade 1’ (Table 5), trained using the expanded image dataset (Section 7.2.2). It performed a correct detection over a range of 75° (outside of the range displayed in Figure 11), with a sharply degrading confidence value. The cascades trained using the expanded image dataset (Cascades 1, 2, and 4 in Figure 11) showed a more consistent confidence value over a slightly wider range than the cascades trained using smaller datasets. On average, an acceptably consistent confidence value was achieved over a rotational range of 25°.

A potential solution to the problem of rotational variation would be to train a number of cascades for a particular target star pattern, for different rotation ranges. For instance, with consistent performance over 25°, 15 cascades could be employed each with a range of 24° to cover a full 360° rotation of a specific target object. A set of 15 cascades for the same target would be able to return a consistent confidence value irrespective of target rotation, at the expense of considerably greater processing time, and file storage requirements.

8 Conclusions

In conclusion, this Thesis provided a summary of experimentation to develop a spacecraft orientation system based on star pattern recognition using neural networks, which would offer good suitability for a CubeSat application, due to the hardware's small physical size, low cost, open-source nature, and minimal power consumption. The methodology chosen was to utilise the neural network training tools included in the *OpenCV* library, and an iterative test process refined the classifier training and detection processes. A full set of cascade classifiers was produced for the northern celestial hemisphere, which revealed limitations of the system, which were addressed by additional tests and modifications. In particular, the classifiers (trained using LBP feature types) were found to have limited performance in respect to image rotation.

Although the time constraints of this MSc Thesis did not permit the creation of a full working system, the methodologies developed in this Thesis for the creation of a neural network based star tracker could form the basis for future improvement and enhancement of this project. Future development of this project has been easily facilitated by the open sourcing of this Thesis, with a permissive copyright license. Despite there being several known limitations of the methodology, the small file sizes of the cascades, and quick processing times, suggest that the neural network approach may offer very real benefits as a star tracker in terms of processing speed, if future development could be performed to further mitigate the system weaknesses. Suggested additional developments of this project are provided in Section 9.

9 Future Work

There is good potential for expansion and improvement of the work presented in this Thesis paper. Given the time and budgetary constraints of an MSc Thesis, it was not possible to investigate all relevant aspects of the project, or to take the work to the conclusion of a fully-working, deployable system. The main areas for future work are identified as follows:

i. Explore fully other alternative object recognition machine learning methods supported by *OpenCV*, such as HOG, which may offer superior rotational invariance, a primary area for improvement of system performance.

ii. Work to identify and mitigate the main sources of error within the system methodology, which will mainly involve the training of cascade files with improved resistance to false positives, which were the main source of failure in the system. Employment of larger datasets, and more training stages to improve accuracy should be trialled.

iii. Continue to develop the project towards a version that could potentially be deployed as a trial system onboard a University-launched CubeSat mission. Key areas of modification would be mechanical design of system mounting and installation within a CubeSat chassis, adaptation of the programs to perform with real camera imagery rather than *Stellarium*-generated simulations, and a rigorous testing procedure to ensure system reliability prior to deployment.

9.1 GitHub Repository

The *GitHub* repository for this project, located at the following link (https://github.com/raspberrystars/CV-Star-Sensor), contains the full code created and used in this project. Additionally, there is a detailed set of instructions to allow the reader to replicate and improve upon this work. The appendices to this paper (Section 11) contain the key programs written for this project, for reference.

10 References

Alvelda, P. and Martin, A.M.S., 1988. Neural Network Star Pattern Recognition for Spacecraft Attitude Determination and Control. In: *Advances in Neural Information Processing Systems 1*. [online] Neural Information Processing Systems 1988. Neural Information Processing Systems Foundation, Inc. Available at: <https://papers.nips.cc/paper/177-neural-network-star-pattern-recognition-for-spacecraft-attitude-determination-and-control> [Accessed 21 Jun. 2019].

Anon 2019. *Astropy Documentation — Astropy v3.2.1*. [online] Available at: <https://docs.astropy.org/en/stable/> [Accessed 6 Aug. 2019].

Anon 2019. *Camera Module - Raspberry Pi Documentation*. [online] Available at: <https://www.raspberrypi.org/documentation/hardware/camera/> [Accessed 6 Aug. 2019].

Anon 2019. *GWCS Documentation — gwcs v0.10.1.dev468*. [online] Available at: <https://gwcs.readthedocs.io/en/latest/> [Accessed 6 Aug. 2019].

Anon 2019. *OpenCV: cv::CascadeClassifier Class Reference*. [online] Available at: <https://docs.opencv.org/4.1.1/d1/de5/classcv_1_1CascadeClassifier.html> [Accessed 6 Aug. 2019].

Anon 2019. *OpenCV: Color conversions*. [online] Available at: <https://docs.opencv.org/4.1.0/de/d25/imgproc_color_conversions.html> [Accessed 15 Aug. 2019].

Anon 2019. *Stellarium: Scripting Engine*. [online] Available at: <http://stellarium.org/doc/head/scripting.html#script_console> [Accessed 6 Aug. 2019].

Anon n.d. *IMX219 Product Brief*. Available at: <https://github.com/rellimmot/Sony-IMX219-Raspberry-Pi-V2-CMOS/blob/master/IMX219%20Product%20Brief.pdf> [Accessed 7 Jun. 2019].

Beck, J., 2010. *HMI & WCS Coordinates, Projections And Arrays For Dummies*. [online] Stanford. Available at: <http://sun.stanford.edu/~beck/JSOC/HMI_WCS_Dummies.pdf> [Accessed 31 Jul. 2019].

Brown, K.Z., Clements, T. and Evans, J., 2010. Cubesat Orientation Control and Matching to Communications System Requirements. [online] IAA Conference on Dynamics and Control of Space Systems (DYCOSS). . Available at: <http://iaaweb.org/iaa/Scientific%20Activity/conf/dycoss14/IAA-AAS-DyCoSS2-14-10-09.pdf> [Accessed 20 Feb. 2019].

Calvet, L., Gurdjos, P., Griwodz, C. and Gasparini, S., 2016. Detection and Accurate Localization of Circular Fiducials under Highly Challenging Conditions. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.562–570.

Commin, H., 2009. *Robust Real-time Extraction of Fiducial Facial Feature Points using Haar-like Features*. [online] Imperial College London. Available at: <https://www.researchgate.net/publication/276923418_Robust_Real-time_Extraction_of_Fiducial_Facial_Feature_Points_using_Haar-like_Features> [Accessed 16 Jul. 2019].

Dalal, N. and Triggs, B., 2005. Histograms of Oriented Gradients for Human Detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. [online] 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). San Diego, CA, USA: IEEE.pp.886–893. Available at: <http://ieeexplore.ieee.org/document/1467360/> [Accessed 6 Aug. 2019].

Gatrell, L.B., Hoff, W.A. and Sklair, C.W., 1991. Robust image features: concentric contrasting circles and their image extraction. In: *Proceedings of SPIE - The International Society for Optical Engineering*. [online] Cooperative Intelligent Robotics in Space II. Boston: The International Society for Optical Engineering. Available at: <https://www.researchgate.net/publication/253321542_Robust_image_features_concentric_contrasting_circles_and_their_image_extraction> [Accessed 17 Jul. 2019].

Greisen, E.W. and Calabretta, M.R., 2002. Representations of world coordinates in FITS. *Astronomy & Astrophysics*, 395(3), pp.1061–1075.

Gutiérrez, S., Fuentes, C.I. and Díaz, M.A., 2017. *Open Hardware and Software Star Tracker: An Opportunity for Collaboration in the Emerging Cubesat Community*. [online] University of Chile. Available at: <https://www.preprints.org/manuscript/201703.0111/v1> [Accessed 21 Jun. 2019].

Harrison Kinsley, 2016. *Creating your own Haar Cascade OpenCV Python Tutorial*. [online] Available at: <https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/> [Accessed 5 Aug. 2019].

Howse, J., Joshi, P. and Beyeler, M., 2016. *OpenCV: Computer Vision Projects with Python*. Packt Publishing Ltd.

Howse, J., Puttemans, S., Hua, Q. and Sinha, U., 2015. *OpenCV 3 Blueprints*. Packt Publishing Ltd.

Kim, E.J. and Brunner, R.J., 2017. Star-galaxy Classification Using Deep Convolutional Neural Networks. *Monthly Notices of the Royal Astronomical Society*, 464(4), pp.4463–4475.

Krizhevsky, A., 2009. *Learning Multiple Layers of Features from Tiny Images*. [online] Available at: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf> [Accessed 21 Jun. 2019].

Kruijff, M. and van der Heiden, N., 2003. Automated Star Sensor Performance Assessment using Real-Sky Data of MEFIST II. International Astronautical Congress.

Lienhart, R. and Maydt, J., 2002. An Extended Set of Haar-like Features for Rapid Object Detection. In: *International Conference on Image Processing*. [online] Rochester: Institute of Electrical and Electronics Engineers. Available at: <https://ieeexplore.ieee.org/document/1038171> [Accessed 18 Jul. 2019].

Lightbody, P., Krajník, T. and Hanheide, M., 2017. A Versatile High-Performance Visual Fiducial Marker Detection System with Scalable Identity Encoding. [online] The Symposium. University of Lincoln. Available at: <https://www.researchgate.net/publication/317611637_A_versatile_high-performance_visual_fiducial_marker_detection_system_with_scalable_identity_encoding> [Accessed 16 Jul. 2019].

Lindblad, T., Lindsey, C.S., Eide, Å., Solberg, Ö. and Bolseth, A., 1997. *Star Identification using Neural Networks*. [online] Available at: <http://gluon.particle.kth.se/~lindblad/Lindblad_StarDet.pdf> [Accessed 21 Jun. 2019].

Lindh, M., 2014. *Development and Implementation of Star Tracker Electronics*. [online] Available at: <http://www.diva-portal.org/smash/get/diva2:737909/FULLTEXT01.pdf> [Accessed 21 Jun. 2019].

Lindsey, C.S., Lindblad, T. and Eide, Å., 1997. A Method for Star Identification using Neural Networks. *Proceedings of SPIE - The International Society for Optical Engineering*. [online] Available at: <https://www.researchgate.net/publication/252328364_Method_for_star_identification_using_neural_networks> [Accessed 17 Jun. 2019].

McBryde, C.R., 2012. A Star Tracker Design for CubeSats. [online] University of Texas. Available at: <https://ieeexplore.ieee.org/document/6187242> [Accessed 17 Jun. 2019].

Nguyen, Q., 2018. *Mastering Concurrency in Python: Create faster programs using concurrency, asynchronous, multithreading, and parallel programming*. Packt Publishing Ltd.

Ojala, T., Pietikainen, M. and Harwood, D., 1994. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In: *Proceedings of 12th International Conference on Pattern Recognition*. Proceedings of 12th International Conference on Pattern Recognition. pp.582–585 vol.1.

Pagnutti, M., Ryan, R.E., Cazenavette, G., Gold, M., Harlan, R., Leggett, E. and Pagnutti, J., 2017. Laying the foundation to use Raspberry Pi 3 V2 camera module imagery for scientific and engineering purposes. *Journal of Electronic Imaging*, [online] 26(1)(013014). Available at: <https://www.spiedigitallibrary.org/journals/Journal-of-Electronic-Imaging/volume-26/issue-1/013014/Laying-the-foundation-to-use-Raspberry-Pi-3-V2-camera/10.1117/1.JEI.26.1.013014.full?SSO=1> [Accessed 10 Jun. 2019].

Pulli, K., Baksheev, A., Kornyakov, K. and Eruhimov, V., 2012. Real-time computer vision with OpenCV. *Communications of the ACM*, 55(6), p.61.

Qian, X., Yu, H., Chen, S. and Low, K.S., 2013. An Adaptive Integration Time CMOS Image Sensor With Multiple Readout Channels. *IEEE Sensors Journal*, 13(12), pp.4931–4939.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei, 2015. *ImageNet Large Scale Visual Recognition Challenge*. [online] Available at: <http://image-net.org/challenges/LSVRC/> [Accessed 21 Jun. 2019].

Sagitov, A., Shabalina, K., Sabirova, L., Li, H. and Magid, E., 2017. ARTag, AprilTag and CALTag Fiducial Marker Systems: Comparison in a Presence of Partial Marker Occlusion and Rotation. In: *Informatics in Control, Automation and Robotics*. Cham: Springer International Publishing.

Sattar, J., Bourque, E., Gigu`ere, P. and Dudek, G., 2007. Fourier tags: Smoothly degradable fiducial markers for use in human-robot interaction. [online] Computer and Robot Vision, 2007. CRV '07. Available at: <https://www.researchgate.net/publication/4252625_Fourier_tags_Smoothly_degradable_fiducial_markers_for_use_in_human-robot_interaction> [Accessed 17 Jul. 2019].

Stathakis, A., 2011. *Vision-Based Localization using Reliable Fiducial Markers*. [online] University of Ottawa. Available at: <https://pdfs.semanticscholar.org/11f2/a68d1332851067898088cb33075566396d7c.pdf> [Accessed 17 Jul. 2019].

Trask, A.J., 2002. *Autonomous Artificial Neural Network Star Tracker for Spacecraft Attitude Determination*. [online] University of Illinois. Available at: <https://www.researchgate.net/publication/234493665_Autonomous_artificial_neural_network_star_tracker_for_spacecraft_attitude_determination> [Accessed 21 Jun. 2019].

Viola, P. and Jones, M., 2001. Robust Real-time Object Detection. [online] Second International Workshop On Statistical And Computational Theories Of Vision – Modeling, Learning, Computing, And Sampling. Vancouver. Available at: <https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-IJCV-01.pdf> [Accessed 18 Jul. 2019].

11 Appendices

11.1 Appendix 1: Stellarium Script for Image Capture

The below code is for a *Stellarium* script. No syntax formatting is shown, as the code was written using *Microsoft Notepad*. To allow the file to be executable as a *Stellarium* script, it must be saved with a '.ssc' file extension, rather than the default '.txt'.

```
// Name: Thesis test script 4
// License: MIT License
// Author: raspberrystars (GitHub user)
// Last Modified: 06/07/19
// Description: Collects images of northern celestial hemisphere.

//Initial messages to user
LabelMgr.labelScreen("Now running 'thesis_4'.", 200, 200, true, 20,
"#ff0000");
core.wait(2);
LabelMgr.deleteAllLabels();
LabelMgr.labelScreen("Setting up test environment.", 200, 200, true,
20, "#ff0000");

//Stellarium environment setup
core.setTimeRate(0);
LandscapeMgr.setFlagLandscape(false);
LandscapeMgr.setFlagAtmosphere(false);
LandscapeMgr.setFlagFog(false);
LandscapeMgr.setFlagCardinalsPoints(false);
SolarSystem.setFlagLabels(false);
Satellites.setFlagHints(false);
Satellites.setFlagLabels(false);
MeteorShowers.setEnableMarker(false);
MeteorShowers.setEnableLabels(false);
GridLinesMgr.setFlagGridlines(false);
StarMgr.setLabelsAmount(0);
StelMovementMgr.toggleMountMode();
core.setGuiVisible(false);
core.setProjectionMode("ProjectionPerspective");
StelMovementMgr.zoomTo(48.55, 1);
core.wait(3);

//Movement code
LabelMgr.deleteAllLabels();
LabelMgr.labelScreen("Capturing sample images.", 200, 200, true, 20,
"#ff0000");
core.wait(2);
LabelMgr.deleteAllLabels();
core.wait(2);

var j = 0;
while (j < 360) {
```

```
for (i = 0; i < 27; i++)
{
core.moveToRaDecJ2000(j, i * 2.5 + 25, 0);
core.screenshot("check", false,
"E:/Documents/Strathclyde/Thesis/Stellarium Screenshots", false,
"png");
core.wait(0.5);
}
j = j + 5;
i = 0;
core.wait(1);
}
core.wait(2);

//Test complete message
LabelMgr.deleteAllLabels();
LabelMgr.labelScreen("Test complete, you may exit Stellarium.", 200,
200, true, 20, "#ff0000");
core.wait(2);
LabelMgr.deleteAllLabels();
LabelMgr.labelScreen(j, 200, 200, true, 20, "#ff0000");
```

11.2 Appendix 2: Positive Sample Creation Program (Python)

Please note that this code includes the bright star identification process (including the erosion function), as discussed initially in Section 5.2.1, not the modified methodology as presented in Section 7.2.1.

```python
#Imports required libraries.
import imutils
import cv2

#Loads images of the fiducial marker, the background star image, and a black rectangle.
#The black rectangle is of the same resolution as bg_img.
fg_img = cv2.imread("fiducial_6_80x80.png", cv2.IMREAD_UNCHANGED)
bg_img = cv2.imread("filename.png")
rectangle = cv2.imread("rectangle_black.png")

#Enlarges bg_img to prevent erosion stage from removing too many stars.
enlarged_big = cv2.resize(bg_img, (3840, 2160))

#Greyscales the enlarged background star image.
graybg = cv2.cvtColor(enlarged_big, cv2.COLOR_BGR2GRAY)

#Thresholds, erodes, and dilates the stars on this image.
thresh = cv2.threshold(graybg, 175, 255, cv2.THRESH_BINARY)[1]
mask1 = thresh.copy()
mask1 = cv2.erode(mask1, None, iterations=1)
mask2 = mask1.copy()
mask2 = cv2.dilate(mask2, None, iterations = 2)

#Resizes back to original resolution, and finds the dimensions.
mask2resized = cv2.resize(mask2, (1920, 1080))
h2, w2 = mask2resized.shape[:2]

#Identifies 'contours' within the processed image.
cnts = cv2.findContours(mask2resized.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

#Finds the dimensions of the fiducial marker image.
#This image is a circular markers on a rectangular transparent background.
fgy, fgx = fg_img.shape[:2]

#Handles the alpha channel of the fiducial marker image, needed for the transparency.
alpha_s = fg_img[:, :, 3] / 255.0
alpha_l = 1.0 - alpha_s

rectangle_build = rectangle.copy()

#For each identified contour (bright star), applies a fiducial marker to that location.
#Adds the fiducial markers to the black rectangle.
for c in cnts:
        x,y,w,h = cv2.boundingRect(c)
        y1 = int((y + 0.5 * h) - 0.5 * fgy)
        y2 = y1 + fgy

        x1 = int((x + 0.5 * w) - 0.5 * fgx)
        x2 = x1 + fgx

        rectangle_temp = rectangle.copy()

        if y1 > 0 and y2 < h2 and x1 > 0 and x2 < w2:
                for c in range(0, 3):
                        rectangle_temp[y1:y2, x1:x2, c] = (alpha_s * fg_img[:, :, c] + alpha_l * rectangle[y1:y2, x1:x2, c])

                rows,cols,channels = rectangle_temp.shape
                roi = rectangle_build[0:rows, 0:cols ]
                img2gray = cv2.cvtColor(rectangle_temp,cv2.COLOR_BGR2GRAY)
                ret, mask = cv2.threshold(img2gray, 10, 255, cv2.THRESH_BINARY)
                mask_inv = cv2.bitwise_not(mask)
                img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)
                img2_fg = cv2.bitwise_and(rectangle_temp,rectangle_temp,mask = mask)
                dst = cv2.add(img1_bg,img2_fg)
                rectangle_build[0:rows, 0:cols ] = dst

#Merges the rectangular image with fiducial markers onto the background sky image.
rows,cols,channels = rectangle_build.shape
roi = bg_img[0:rows, 0:cols ]
img2gray = cv2.cvtColor(rectangle_build,cv2.COLOR_BGR2GRAY)
ret, mask = cv2.threshold(img2gray, 10, 255, cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)
img1_bg = cv2.bitwise_and(roi,roi,mask = mask_inv)
img2_fg = cv2.bitwise_and(rectangle_build,rectangle_build,mask = mask)
dst = cv2.add(img1_bg,img2_fg)
bg_img[0:rows, 0:cols ] = dst

#Specifies area of main image to crop for positive image.
y=390
x=810
h=300
w=300

#Crops section of main image with markers, names it positive file and saves.
crop_img = bg_img[y:y+h, x:x+w]
grey_img = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)
shrunk_img = cv2.resize(grey_img, (50, 50))
small_image_name = 'positive.jpg'
cv2.imwrite(small_image_name, shrunk_img)

#Names and creates a test file which cascades trained on this positive image can be checked against.
big_image_name = 'filename2.png'
cv2.imwrite(big_image_name, bg_img)
```

11.3 Appendix 3: Descriptor (bg.txt) File Creation (Python)

n.b. If this script is run on a Windows PC, but the classifier training is performed in a Linux environment, it will be necessary to use the dos2unix function to convert the file into an ASCII format suitable for use in the training procedure, that solves bugs arising from line ending issues when transferring from one operating system to another.

```python
#Imports required libraries.
import cv2
import os

#Specifies location of neg folder.
if not os.path.exists('neg_southern2'):
    os.makedirs('neg_southern2')

#Loops over each negative file in that folder.
#Creates .txt file with names of these files.
for file in ['neg_southern2']:
    for img in os.listdir(file):
        line = img+'\n'
        with open('bg.txt','a') as f:
            f.write('neg/' + line)
```

11.4 Appendix 4: Negative Dataset Creation Program (Python)

```python
#Imports required libraries
import cv2
import os, os.path

#Assumes 1920x1080 pixel input Stellarium screenshot.
#The following coordinates allow a 1080x1080 square image to be cropped.
y=0
x=420
h=1080
w=1080

#A stage counter used to name files.
picno = 1

#Location of the Stellarium screenshots to be processed.
path = "E:/Documents/Strathclyde/Thesis/Stellarium Screenshots/Northern Celestia
l Hemisphere/"
path_list = []

#Creates full file location for each file within that specified folder.
for file in os.listdir(path):
    extension = os.path.splitext(file)[1]
    path_list.append(os.path.join(path, file))

#For each file, crops to square, greyscales, and shrinks to 100x100 pixels.
#Each image is then saved as a number .jpg
for imagePath in path_list:
    imgstart = cv2.imread(imagePath)
    crop_img = imgstart[y:y+h, x:x+w]
    grey_img = cv2.cvtColor(crop_img, cv2.COLOR_BGR2GRAY)
    shrunk_img = cv2.resize(grey_img, (100, 100))
    image_name = str(picno)+'.jpg'
    cv2.imwrite(image_name, shrunk_img)
    #cv2.imshow(imagePath, shrunk_img)
    picno += 1
```

11.5 Appendix 5: Final Orientation Test Program (Python)

```python
#Imports required libraries.
import cv2
import numpy as np
import os, os.path

#Creates variables for position and RA/Dec coordinate for two detections.
d1x, d1y, d1ra, d1dec, d2x, d2y, d2ra, d2dec = (0,)*8

#Loads an input test image (which has had fiducial markers applied).
#In practice an image would be received from the RPi camera and markers applied.
img = cv2.imread('stellarium-169-markers.png')

#Creating a detection function.
def stardetection(cascade, ra, dec, minn, sf):
    #Specifies the cascade file to be loaded.
    stars_cascade = cv2.CascadeClassifier('realcascades/weighted_purged/'+cascade+','+ra+','+dec+','+minn+','+sf+'.xml')

    #Applies the detectMultiScale3 function with the appropriate parameters.
    stars, rejectLevels, levelWeights = stars_cascade.detectMultiScale3(
        img,
        scaleFactor = 1.05,
        minNeighbors = int(minn),
        flags = 0,
        minSize = (300, 300),
        maxSize = (400, 400),
        outputRejectLevels = True
        )

    #Create some additional variables = 0 for use in later 'for loops'.
    i = 0
    highweight = 0
    big_w = 0
    weighted = 0

    #The purpose of this if statement is to see if any detection has been made.
    if(len(stars) > 0):
        for (x,y,w,h) in stars:

            #This if statement will find the detection with the largest bounding box.
            if w > big_w:
                highweight = levelWeights[i]
                weighted = float(highweight)*float(sf)
                x1 = x
                y1 = y
                w1 = w
                h1 = h

        #The if statement below sets the levelWeights value bounds for a 'successful' detection.
        if (weighted > 4) and (weighted < 6):
            font = cv2.FONT_HERSHEY_SIMPLEX
            cv2.putText(img,cascade,(x1,y1-16), font,0.9,(0,0,255),2,cv2.LINE_AA)
            cv2.putText(img,str(weighted),(x1,y1+h1+25), font,0.7,(0,0,255),2,cv2.LINE_AA)
            cv2.rectangle(img,(x1,y1),(x1+w1,y1+h1),(0,255,0),2)
            cenpixx = int(x1 + 0.5 * w1)
            cenpixy = int(y1 + 0.5 * h1)
            cv2.putText(img,str(cenpixx)+', '+str(cenpixy),(x1,y1-45), font,0.9,(0,0,255),2,cv2.LINE_AA)
            shrunk_img = cv2.resize(img, (1344, 756))
            cv2.imshow("Star Pattern Detections",shrunk_img)
            print('Cascade number '+cascade+' DETECTS')
            print(weighted)
            print()

            #Pulls in the global variables for the pixel and world coordinates of the detections.
            global d1x, d1y, d1ra, d1dec
            global d2x, d2y, d2ra, d2dec

            #The following statements assign the parameters of two successful detection to those variables.
            if (d1x == 0):
                d1x = cenpixx
                d1y = cenpixy
                d1ra = ra
                d1dec = dec
            else:
                d2x = cenpixx
                d2y = cenpixy
                d2ra = ra
                d2dec = dec
        else:
            print('Cascade number '+cascade+' POOR DETECTION')
            print(weighted)
            print()
    else:
        print('Cascade number '+cascade+' NO DETECTION')
        print()
    return(d1x, d1y, d1ra, d1dec, d2x, d2y, d2ra, d2dec)

#Runs the detection function for each cascade file within the specified folder.

for file in ['realcascades/weighted_purged']:
    for item in os.listdir(file):
        item = os.path.splitext(item)[0]
        items = item.split(',')
        first = int(items[0])
        second = str(items[-4])
        third = str(items[-3])
        fourth = int(items[-2])
        fifth = str(items[-1])

        stardetection(cascade = str(first), ra = second, dec = third, minn = str(fourth), sf = fifth)

    #Prints the returned pixel coordinates and RA/Dec coordinates for two positive IDs from the stardetection fn.
    print(d1x, d1y)
    print(d1ra, d1dec)
    print()
    print(d2x, d2y)
    print(d2ra, d2dec)
    print()
```