

Vertical Camera Stabilization using Computer Vision Techniques



Prepared by:
Shai Kadish
KDSSHA001

Department of Electrical Engineering
University of Cape Town

Prepared for:
Jarryd Son

Professor in the Department of Electrical and Electronics Engineering
University of Cape Town

October 2019


Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in mechatronics engineering

Key Words: Computer vision, Vertical stabilization, System Control, Detection, Tracking, algorithm design, autonomous

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this final year project report from the work(s) of other people, has been attributed and has been cited and referenced.
3. This final year project report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof

Name: Shai Kadish

Signature: 

Date: 13 October 2019

Acknowledgements

The completion of this project is due in part to the support and expertise provided by a number of very generous people to be acknowledged here.

First and foremost, thank you to Jarryd Son for your guidance throughout this project. The support and expertise that you provided to me throughout the development of this report were invaluable, and I am sure the final product would not be the same without your help. Additionally, I would like to acknowledge that Professor Son took me on as an additional student, allowing me to pursue a topic of extreme interest to myself. For that, I cannot thank you enough.

Thank you to my parents, Cathy and Craig, Josh, Gael, Hugh and Nola for supporting me throughout this project.

Thank you to my friend Paz Shina, who generously lent to me his camera slider, which was a great help during the testing phase of this report.

Thank you to Jamie, for keeping calm while I was configuring the Raspberry Pi, and all the other times that I needed that.

Finally, thank you to the friends who I worked alongside, both in the development of this report, and throughout my undergraduate degree. This journey has had its highs and lows, but going through it with such a great group of people has made it that much more bearable.

Abstract

Handheld video recordings are prone to unstable, shaky images as a result of human error. There are multiple well explored methods of resolving this issue, such as digital image stabilization and with the use of camera gimbals.

This report details the design process behind the invention of a new method for achieving image stability. This novel method of stabilizing a camera utilizes computer vision techniques, as it solves the problem by detecting and tracking the subject within an image by deploying the relevant algorithms for these techniques, and actuating a camera in vertical space in order to keep this subject at the centre of the image frame. The system achieves vertical stability in real time in order to make vertical jitters and shakes unperceivable when watching the recorded footage.

Multiple configurations of the system are tested and deliberated upon in order to produce the final design found within this report. The system designed within this report showed significant improvements to vertical stability when compared with a camera with no stability measures applied to it. These improvements were determined by using multiple performance measuring parameters, and the final system designed performed at least twice as well in terms of these parameters when compared to a system with no stability measures applied to it.

Table of contents

1. Introduction.....	1
1.1 Background to the study	1
1.2 Objectives of this study.....	1
1.3 Scope and Limitations.....	1
1.4 Plan of development.....	2
1.5 Report Outline.....	2
2. Literature Review	3
2.1 Stabilization.....	3
2.2 A brief overview of relevant computer vision concepts.....	4
2.2.1 Image Features.....	5
2.2.2 Bounding Boxes	5
2.3 Object Detection	6
2.3.1 Neural Network Approach for Classification.....	6
2.3.2 Haar cascades.....	10
2.4 Tracking algorithms.....	11
3. Methodology	13
3.1 Research	13
3.2 Design Process	13
3.3 System level Requirements Analysis.....	13
3.3.1 User Requirements.....	13
3.3.2 Functional Requirements.....	14
3.3.3 Acceptance Test Procedures (ATPs)	14
3.4 Software selection	15
3.5 Hardware Selection.....	15
3.5.1 Micro-Controller selection.....	15
3.5.2 Camera selection	16
3.5.3 Vertical actuator selection.....	17
3.6 Computer vision algorithm selection	19
3.6.1 Object Detection algorithm selection.....	20
3.6.2 Object tracking algorithm selection.....	20
3.7 Workstation Setup.....	21
4. Design.....	24
4.1 System Design	24
4.2 Hardware Design and Implementation	24
4.2.1 Hardware design process	25
4.2.2 Hardware performance testing and design iterations.....	27
4.3 Software Design and Implementation.....	28
4.3.1 High level code design.....	28
4.3.2 Computer vision algorithm design and integration.....	30
4.3.3 Computer Vision Algorithm Optimization	31
4.3.4 Motor Control Design	32
4.3.5 Function design.....	33
5. Testing.....	37
5.1 Testing environment setup.....	37
5.2 Test design.....	37
5.2.1 Stepped output disturbance test.....	38

5.2.2	Sinusoidal output disturbance test.....	38
5.3	Performance measures.....	39
5.3.1	The Integral Squared Error (ISE).....	39
5.3.2	The Integral Absolute Error (IAE).....	39
5.3.3	Integral Time-weighted Absolute Error (ITAE)	40
5.3.4	Tracking Failures.....	40
5.4	ATPs.....	40
6.	Results.....	42
6.1	Stepped output disturbance test results	42
6.1.1	ISE data for stepped output disturbance test.....	42
6.1.2	IAE data for stepped output disturbance test	43
6.1.3	ITAE data for stepped output disturbance	44
6.1.4	Tracking failure data for stepped output disturbance.....	44
6.2	Sinusoidal output disturbance test results	45
6.2.1	ISE data for sinusoidal output disturbance test	46
6.2.2	IAE data for stepped output disturbance test	46
6.2.3	Tracking failure data for sinusoidal output disturbance	47
6.3	ATP test results.....	47
7.	Discussion.....	50
8.	Conclusions	52
9.	Recommendations	53
10.	References	54
11.	Appendix	60
11.1	The GitHub repository containing the final code design of the project.....	60
11.2	The complete data set generated in the testing process, saved to GitHub.....	60
11.3	A video showcasing the final system design.....	60

List of Figures

Figure 1: Rotational capability of a 3-axis gimbal.....	3
Figure 2: The Pedevator	4
Figure 3: Black White intensity of a feature window	5
Figure 4: Bounding Boxes	6
Figure 5: A single neuron or "perceptron"	7
Figure 6: A regular 3-layer ANN.....	7
Figure 7: The general architecture of a CNN.....	8
Figure 8: Detection Methods Performance Comparison.....	10
Figure 9: Haar features	11
Figure 10: V-Model used in the design Process	13
Figure 11: The Raspberry Pi 3 connected to the Raspberry Pi Camera Module	17
Figure 12: The NEMA 17 Stepper Motor	18
Figure 13: A Rack and Pinion	18
Figure 14: Interfacing a Microcontroller and Stepper motor with the DRV8825	19
Figure 15: Box plot revealing averages and distributions of image processing speeds and accuracy for different trackers.....	20
Figure 16: The hardware workstation	22
Figure 17: Circuit diagram of RPi interfacing with the DRV8825 and NEMA 17.....	23
Figure 18: The classic control feedback loop for the system.	24
Figure 19: 3D printed NEMA 17 rack and pinion design, showing each component and the system assembled.....	25
Figure 20: 3D printed Raspberry pi camera case.	26
Figure 21: The newly designed rack	26
Figure 22: The fully assembled system.....	27
Figure 23: Microstepping Current Waveform of the DRV8825 motor driver.....	27
Figure 24: Flowchart explaining black-box design of code.	30
Figure 25: Analysis of a typical input image, with relevant tracker information.....	32
Figure 26: Flowchart describing look (left) and its usage (right)	34
Figure 27: Flow chart for a stepmode function	35
Figure 28: Flow diagram of the choostep function	35
Figure 29: The testing environment.....	37
Figure 30: Error vs. Time for the simulation of a step in output disturbance	38
Figure 31: Error vs. Time for the simulation of a sinusoidal output disturbance	39
Figure 32: A screenshot from a video recording.....	40
Figure 33: Comparison of the results from the step test between when the system on and when it is disabled.	42
Figure 34: Box and whiskers charts from ISE data generated by the step test.....	42
Figure 35: Box and whiskers charts from IAE data generated by the step test	43
Figure 36: Box and whiskers charts from ITAE data generated by the step test.....	44
Figure 37: Number of tracking failures during step test	44
Figure 38: Error vs. time for KCF step response	45
Figure 39: Comparison of the results from the sinusoid test between when the system on and when it is disabled	45
Figure 40: Box and whiskers charts from ISE data generated by the sinusoid test	46
Figure 41: Box and whiskers charts from IAE data generated by the sinusoid test.....	46
Figure 42: Number of tracking failures during sinusoid test.....	47

Figure 43: Error vs. time for the step test using the MFT tracker..... 47

Figure 44: A screenshot from a video recording..... 48

Figure 45: The system being run as a hand-held device..... 48

Figure 46: Error vs. time for a MFT step test..... 49

Figure 47: Error vs. time for a MFT sinusoidal test. 49

1. Introduction

1.1 Background to the study

Hand held cameras are inherently prone to human error. This human error can be present in the form of jitters from the camera man's shaky hands, horizontal instability from undesirable movements in the horizontal plane, or vertical instability from undesirable movements in the vertical plane. This problem has been solved by devices such as gimbals and camera rigs, and software such as digital image stabilization programs.

Many of these established tools to solve camera stability problems make use of sensors such as inertial measurement movements to detect disturbances, and the post processing of a video recording to simulate stability. These commonly used methods will be ignored in this report, where a new method of achieving vertical stability is to be developed.

This report will create a novel solution to this problem, by developing a rig which counteracts vertical instability with the aid of computer vision techniques to be deployed on an embedded device. These techniques include tracking and detection algorithms, and will employ the use of real-time footage from a camera. These techniques will be used to stabilize the camera in vertical space relative to a pre-determined subject: A human face. In this way, the system will ensure that the face being filmed by the camera is kept at the centre of the image even if vertical disturbances produce instability in the rig.

1.2 Objectives of this study

The objective of this report is to design, build and test a novel solution to the problem of vertical instability faced by hand-held cameras. The system under design must make use of computer vision techniques in order to solve this problem. The objectives of the report are therefore to both solve the problem of vertical instability for a hand held device, and to solve it in a novel way by using computer vision techniques. A successful design will achieve better results in terms of vertical stability than a camera with no stability measures employed.

The objective of the study separate from the development of the system, is to detail the process followed throughout its development. This will hopefully make the results from the project repeatable should a reader wish to recreate the system.

1.3 Scope and Limitations

This project is limited by a R1500 budget as is prescribed to all students developing an undergraduate thesis in engineering for UCT. This limits the materials and equipment at the disposal of the author of this report. The report is also limited by time, as there was a four month period in which this report could be developed. These limitations lead to the limiting of the scope of the project to be more problem specific, rather than producing a system which can broadly solve issues of vertical instability. The final design must also be portable and hand held, which limits some of the hardware options available to the system, as the computers available to meet such requirements include underpowered embedded device.

In terms of the scope of the report and subsequent system designed within, the computer vision techniques applied are only tested and designed for stabilizing an image around a human face, with constant lighting, at a range under 2m. The scope is also limited to apply only computer vision techniques, meaning that the system to be designed cannot be integrated with other image stabilization systems such as camera gimbals.

1.4 Plan of development

An initial schedule for the development of this report can be found here. The time allocation for each aspect of the report was deliberated and decided upon in developing the schedule found here. This schedule accounts for every week within the 12 weeks available to the author to develop this report.

1. Perform a detailed literature review and research phase for around three weeks.
2. Develop a project outline including functional requirements, user requirements, and ATPs within one week
3. Select and order relevant hardware and software components required for the project within one week
4. Set up a hardware and software workstation within one week. This includes installing all the relevant software and packages, as well as interfacing and testing the various sub-systems
5. Design and build hardware to be used by the system within one week
6. Design and implement the software and control algorithms to be used by the system for three weeks. This time period is where hardware and software tuning will occur, as well as full system integration and testing.
7. One week will be allocated to designing a test environment as well as the tests and performance metrics by which the final design will be judged. This week also includes the analysis and generation of the test results.
8. The final week of the project will be spent editing the report, and writing the conclusion.

Throughout the development of this project, the report will be updated as each section has been completed. In this way, the report can be improved over a long period of time rather than written up last minute.

1.5 Report Outline

The following report will begin with a review of the literature and concepts relevant to the development of the task at hand. These topics include camera stabilization and computer vision concepts. The ideas and explanations found in this section will inform the designs and methodologies undertaken in the proceeding sections.

Next, a detailed description of the methodologies and project management techniques utilized in the development of this report is presented. This section also includes justifications regarding why each software and hardware component found in the design section was chosen.

The following section explains the design process following in creating the final system. This includes descriptions of both the hardware and software designs undertaken, as well as the algorithms developed to control the system.

A testing section follows the design section, in which the methods by which the final design is to be tested is described. This section is followed by the results from the testing process. A discussion and subsequent conclusion follows the results section, where the level of success achieved by the developed project is determined.

2. Literature Review

The designs and methods used throughout the project must be informed by appropriate literature. This chapter details the state of the art concepts and techniques utilized in the fields of image stabilization and computer vision, as well as the tracking and detection algorithms commonly used for such vision based tasks. These topics are discussed with an inflection towards creating a system which is hand-held (and subsequently, deployed using an embedded device), and one which is designed around stabilizing the camera around a human face.

2.1 Stabilization

Image stabilization is a longstanding and ongoing problem when it comes to filming a video. Hand held cameras especially suffer from random camera motions, which could come in the form of translations or rotations in the image.

In modern digital cameras, it is common to find *Optical Image Stabilization* (OIS). This form of mechanical stabilization reduces the blur from shaky hand jitters by applying control techniques [1], [2] to centre a camera within its housing, in order to compensate for $\pm 1^\circ$ of unwanted rotation [3]. These systems have been implemented for over 30 years, but this form of stabilization will not be useful in compensating for sustained vertical movements, as is required. OIS may be an interesting way to stabilize the sensor itself, but not the entire camera body. An image with noise can be stabilized completely digitally [4], performing the task of an OIS, but research in achieving this stability in real time is lacking.

While OIS is used to account for shaky hands, *Electronic Image Stabilization* (EIS), also known as digital image stabilization, is used to account for inter-frame jitter when recording video. The shaky jitter between frames is smoothed with the use of software, aided by sensors such as gyroscopes [3]. The software shifts the image in the frame digitally, by making use of parts of the image which extend past the visible frame, to act as a buffer for the shaky motion seen between frames of the video [5], and can employ a multitude of motion modelling techniques, such as using optical flow [6], or inter-frame homography with a Kalman filter [7]. Most video cameras today make use of EIS for video stabilization. The Raspberry Pi camera module has built in EIS functionality [8], so by enabling this functionality, the image used for detection, and thus vertical stability, will be more stable.

Another stabilizing technique, which takes the form of an external rig to the camera, is a gimbal. Gimbals use a system of motors and sensors in order to counteract unwanted camera movements, and keep the camera stable and balanced within the rig [9].

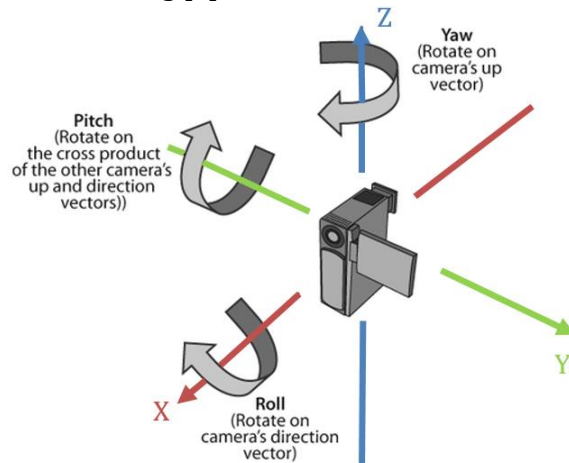


Figure 1: Rotational capability of a 3-axis gimbal; A 3-Axis gimbal counteracts rotational motion along the X,Y and Z axis, also known as rotation in the *Euler angles* of the system, being Pitch, Yaw, and Roll [10]. For a 4-axis gimbal, translational motion along the Z-axis is also counteracted

For a 3-Axis gimbal, the motion that is being counteracted is effected by rotation in the Pitch (also referred to as the *tilt*) axis. This axis is used to keep the camera facing forward when filming an object moving vertically [11]. This motion accounts for rotating the camera along the Y-axis, but to solve the problem of moving translationally along the Z-axis, it is better to look at 4-axis gimbals, which use hydraulic suspension to counteract vertical translational motion [12]. The system to be designed will use active components such as microcontrollers and motors, rather than springs and hydraulics, but will aim to achieve a similar result. This idea is explored in the form of controlling a cameras vertical motion via tracking algorithms, and a slow and expensive device called a 'Pedelevator' [13]. This paper also shows the effective use of Kalman Filters to smooth vertical jittering motion, which may be helpful in the electronic implementation of the design at hand. While this paper explores the idea of tracking to control vertical motion, it does not go into the process of stabilizing the image.



Figure 2: The Pedelevator; this device is a motorized tripod which allows for pan-tilt motion and vertical actuation. The Pedelevator has been proven to be controllable using computer vision based tracking algorithms [13].

The task of real time vertical stabilization has been explored using methods of software and hardware [14], with the fine vertical motion being actuated by servo motors. Interestingly, this project also uses Kalman filters to aid in the digital image stabilization.

Even with the stability provided by gimbals, they lack the intelligence required by the project at hand. This is because gimbals are good at keeping a camera pointed in one general direction, but they are not designed to keep a specific object in central frame. Some ideas from gimbals may be employed in the design of the rig, but only to counteract jitters and vibrations.

The literature regarding the use of computer vision techniques of detection and tracking for image stabilization is greatly lacking. As such, new techniques must be developed to use the object detection methods discussed below to achieve some of the results seen in this section.

2.2 A brief overview of relevant computer vision concepts

Humans can process visual information effectively and robustly, performing object recognition instantaneously. Computers do not possess the advanced vision systems that humans do. The field of computer vision refers to the effort to increase the capabilities of computers in processing visual information. [15]

In the still relatively young field of computer vision, object tracking and detection refer to two large aspects of this field of study. Tracking is defined as the process where an objects motion is modelled and estimated. Detection refers to the process of defining an objects location within an image [16]. The definition of an object is determined by a classifier, which is trained for this task using various machine-learning techniques.

For the problem at hand, both tracking and detection methods must be reviewed in order to find a method to identify and track a face in the live video feed that is used by the system. While there are multiple effective and novel tools that can be used for facial detection and tracking, such as infra-red sensors [17] and depth cameras [18], the most fundamental resource available for a computer vision engineer is a camera. Acknowledging this, methods of using a video stream from a camera for facial detection and object tracking will make up the body of the literature review.

2.2.1 Image Features

In order to analyse an image for a specific problem, a computer must be able to interpret different components that make up an image. These components are called features, and can be anything from textures in an image, to edges or lines [19]. These features are used in classifying an object, and as such, the features selected for each class are problem dependent. Each feature can be expressed numerically as a matrix, in terms of properties such as colour intensity [20].

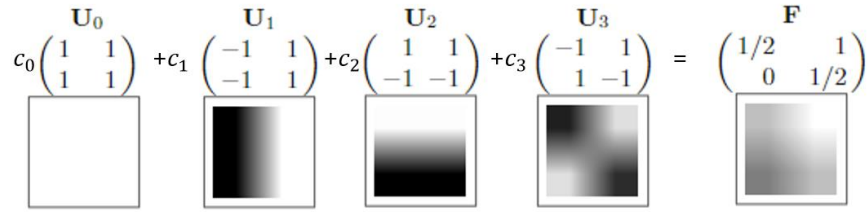
$$c_0 \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + c_1 \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix} + c_2 \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix} + c_3 \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 1/2 & 1 \\ 0 & 1/2 \end{pmatrix}$$


Figure 3: Black White intensity of a feature window; a set of basis functions (U) can be scaled by coefficients (c) and linearly added to produce a new discrete image F. Images are made up of sections like F, that each can be decomposed into a set of basis functions. These basis functions can be used tell us what kind of features are present in a region of an image [20].

Although the task of feature selection is problem specific, there are still common criteria used when selecting what features will be used to detect or track an object [21]. Features can be tracked directly, [22], [23] but these methods do not distinguish between the different object classes which may have similar features. This means, for example, if a corner is tracked, the tracker doesn't care if this feature belongs to a chair or a table.

The problem under review requires for a specific object class to be tracked (a face). In object detection, different methods are used to determine what combination of features make up an object class. Detection is used to classify and locate an object based on its inherent features. Using features within an image, it can then be attempted to spatially locate the object to be kept stable within that image. A more in depth review of object detection can be found later in this section.

2.2.2 Bounding Boxes

In object detection and tracking, the *Region of Interest* (ROI) refers to the spatial region in the image where the object being tracked or detected exists. For a detection algorithm, the ROI contains all the features which the classifier has learned to associate with a class of object. A common way for detection algorithms to output the location of a region of interest is with a *bounding box* [24]. Bounding boxes are a common output of detection algorithms, and are composed of a width value, a height value, and pixel locations.

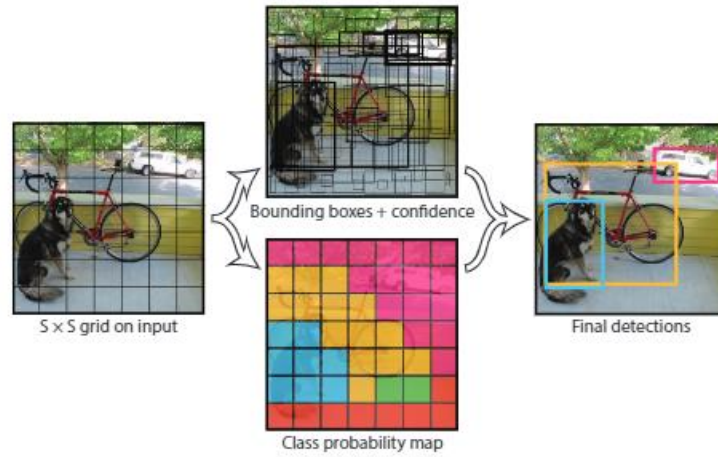


Figure 4: Bounding Boxes; some model detection algorithms divide an input image into a grid, and determine which parts of the grid are likely to contain a specific object class. Multi-class object detectors like YOLO (seen above, and discussed further in the literature review) treat detection as a regression problem, where classification is based on the probability of a set of features belonging to a specific class. This figure shows how an image is decomposed into bounding boxes, and then based on probability, a class of object is determined to be detected within a bounding box [25]. The image on the right shows the output of the detection algorithm: accurately located bounding boxes.

These pixel locations will be invaluable to this project, as stability has been defined as the task of keeping the desired object in centre frame. Bounding boxes are a good tool to spatially locate the object around which the image is to be stabilized.

Bounding boxes can also be used in the training of an artificial neural network, with the desired output of the network being manually set by drawing a bounding box around an object of a specific class in an image [26].

Apart from detection and training, bounding boxes are utilized in some tracking algorithms, where the ROI which is being tracked is defined by a bounding box [27]. This is useful in cases where an object is to be tracked rather than a single point or feature. Tracking algorithms which used bounding boxes include MOSSE trackers, Median Flow trackers, and Kernalized Correlation Filters.

2.3 Object Detection

Computer vision is often used in classification problems, where a system is implemented to both detect and identify an object. Detection without complicated object classification is the process of assigning a label $w \in \{0,1\}$ to a region of an image, indicating whether an object is detected in that region ($w=1$) or not ($w=0$) [28]. This is in its own right referred to as a classifier, as this system is classifying whether an object is present or not. To understand the fundamentals of object detection, the training of the classifier, and the detection algorithm used in conjunction with this classifier must be investigated. By detecting where in the image an object is spatially located, a system can be designed to correct for undesired motion from vertical instability. In order to create classifiers for the object in the image to be stabilized, the methods by which image feature analysis can be used to train or create classifiers, which in turn will detect the location of the object, must be examined.

2.3.1 Neural Network Approach for Classification

Classifiers can be implemented in *artificial neural networks* (ANN). These networks are trained with a set of input data and a related set of desired output data. By cycling through this data, the ANNs change parameters (called weights) in order to improve the performance of the network for regression and classification problems. From this black box situation, ANNs are used as modelling tools, which model an input-output relationship for a set of data [29].

ANNs are made up of different *layers*, which are in turn made up by *neurons*.

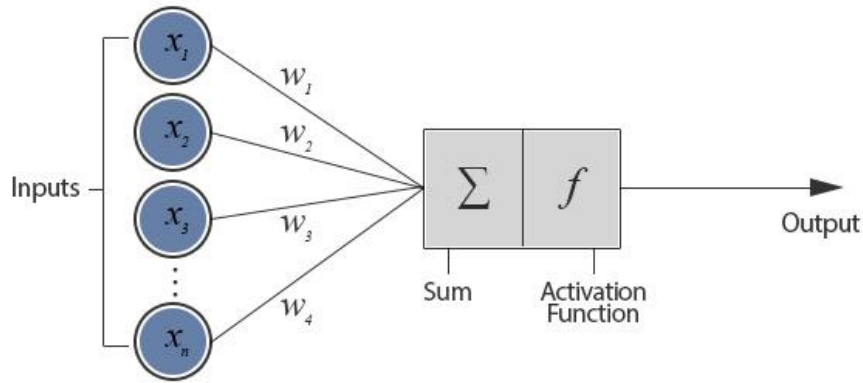


Figure 5: A single neuron or "perceptron". Neurons have the value given by the sum of the input values (X), multiplied with their respective weights (W). The activation function defines the function mapping the given inputs to the desired output [30].

ANN normally have 3 layers; the input, hidden and output layers. The inputs and outputs between adjacent layers are defined by the activation function used (with different activation functions being effective for different problems). The input layer is where the data is supplied to the network. This layer connects to a series of hidden layers (although there might be just one) which are used for processing the input data, to achieve the desired output at the output layer.

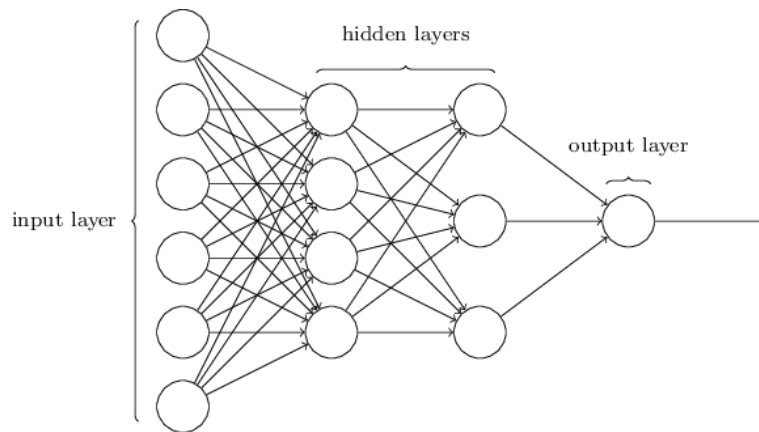


Figure 6: A regular 3-layer ANN, with fully connected layers (every neuron in the network is connected to every neuron in both adjacent layers in the network). In abstract, a classifier takes an image in at its input layer, with each pixel as an input, and once the network has been trained, will output the classification result at the output layer [31]

Different ANN architectures can be implemented for different problems. The traditional ANN architecture is useful for many problems, but this architecture is not preferable for image processing. This is because when an input image is flattened (the process whereby each pixel in the image is arbitrarily separated and sent to a node in the input layer), the spatial structure of the image is not preserved [31]. This means that for the network to take advantage of special features within the image, it must be trained to do so. Fortunately, there is a neural network architecture which takes advantage of the special features of an image inherently: *Convolutional Neural Networks* (CNN).

CNNs have fewer connections and parameters than normal ANNs, and as such are more computationally efficient to train for image classification problems, such as the problem under review. CNNs can be explained briefly by describing 3 core concepts which differentiate them from normal ANNs [31], [32]:

1. **Local Receptive Fields:** Normally the input to a neural network is a flat fully connected layer, where the inputs can be depicted as a vertical line of neurons, and each input neuron is connected to each hidden layer neuron in the proceeding layer. In CNNs an input is a small localized square of neurons (which are pixels with intensity values) taken from the larger set of

neurons in the input image. These regions are spatially localized, and are known as *local receptive fields*.

2. **Feature Maps:** For each local receptive field, there is a corresponding hidden neuron in the adjacent hidden layer. Each hidden neuron in a layer has the same set of weights assigned to it, meaning that these neurons each detect the same feature, but at different special locations in the image (each corresponding to a local receptive field). These sets of weights are known as *kernels*. This mapping of the input layer to the hidden layer through a kernel generates a *feature map*. In training the network as to what values each kernel should have, the same algorithms used in training the weights of an ANN can be referred to [33]. This means that machine learning will be used in determining what features for a class are useful in detection. The layer made up of featuremaps is known as the *convolutional layer*.

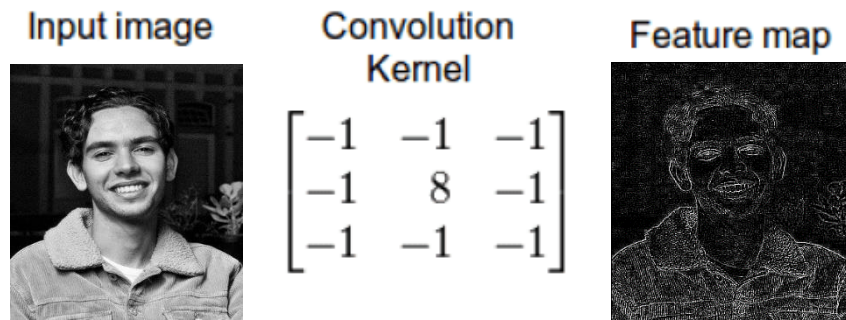


Figure 7: Effects of applying a convolutional Kernel to an input image. This kernel acts as a feature map for the edges of the input image, extracting this feature for further computation. The kernel acts as an edge detector in simple terms.

3. **Pooling Layers:** Following the convolutional layer, the pooling layer simplifies the data extracted from the previous layer, creating a *condensed feature map*. By condensing a feature map, a region of $n \times n$ neurons in the original feature map can be converted to a single neuron in the pooling layer. There are different methods of condensing this information, such as *average pooling* which takes the average pixel value from a region in a specific feature map, but the most popular pooling method is *max pooling* which takes the maximum value of a region in a feature map. Max pooling gives a general indication as to whether a feature is detected in a region of the image, while losing accuracy regarding where exactly in the special region this feature is found.

These layers and concepts can be repeated multiple times throughout a CNN, until the desired features are extracted. This leads to varied and complex architecture using the core principals discussed above [34]

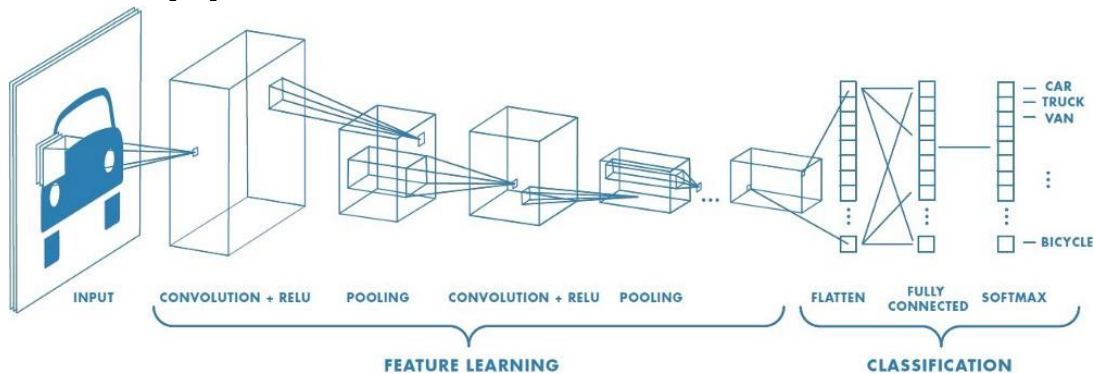


Figure 7: The general architecture of a CNN; Here the convolutional layers are referenced to with their ReLU activation functions. A CNN may be used cascaded with a regular ANN, with the CNN being used for feature extraction and learning and the ANN being used for classification purposes once the useful features have been extracted [35]

For real time purposes, like those of the requirements presented by the problem at hand, the speed of the object detector is paramount. The R-CNN method (*Regions with CNN features*) combines traditional computer vision techniques with CNNs to increase the accuracy and decrease the training time of a CNN based network by solving the CNN localization problem (the problem of localizing where in an image an object is detected). This technique has impressive accuracy, but lacks the real-time speed that the system to be designed requires, taking up to a minute to classify an image on a CPU [36]. R-CNNs are improved by combining their concepts with those of *Spatial Pyramid Pooling* (SPP-net) to create *Fast R-CNN*, which are up to 25 times the speed of R-CNNs, bringing object detection even closer to real time [37], [38]. Fast R-CNN are improved further by *Faster R-CNNs*, which introduce region proposal networks, which are used in calculating the probability of a region being background or foreground [39]. Other than this, Fast R-CNN and Faster R-CNN architecture is very similar, but the latter is up to 10 times faster than the former, making it the ideal choice of R-CNN architecture for the real time purposes required by the project at hand [38], [39].

It is worth noting for the problem at hand that although R-CNN architecture can be used for close to real time image processing, the hardware requirements for implementing these networks are high at this present time, with the standard computers used to generate *test time* results commonly utilizing high-end GPUs. The size of a CNN is limited by both the amount of time available to train it (as it can take days or even weeks to train a network) and the memory available on the processing unit used to train it [34]. GPUs are generally used for the task of training because of their highly parallelized architecture. They also are preferable over CPUs because of their faster execution time for image processing, and massive memory [40]. The reliance of these architectures of GPUs makes it difficult to implement them on a low-power, small-memory (the Raspberry Pi has only 1GB of RAM) embedded device. Though it is possible to optimize a CNN so that it can be trained and run on an embedded device, the image processing capabilities are far from real time, and even with complicated optimization algorithms, the improvement is marginal [41], [42]. A popular trend is to solve this problem by implementing the CNN model used on a server with a powerful GPU, and connect the embedded device to this server [43].

Custom hardware which makes use of FPGAs and ASIC components can also be used to accelerate the performance of Faster R-CNNs to achieve state-of-the-art real time results for vision and detection, even on underpowered embedded devices [44]. There are also software accelerators available (such as *OpenVino* [45]) which are purpose built tool kits for optimizing computer vision tasks performed by CNN models. Some software accelerators (like OpenVino) can be deployed on embedded systems, allowing for more complex machine learning tasks to be performed on edge devices [46]. Highly specialized SoCs such as the *Intel Movidius neural compute stick* [47] act as hardware accelerators built to enhance the performance of computer vision tasks by embedded systems such as the Raspberry Pi [46]. By using accelerated hardware and software such as the Movidius NCS and OpenVino, the processing of an image frame from a live video stream on the RPI can be brought down to 80 milliseconds [48], bringing the detection close to real time.

There are CNNs which have been purpose built for facial detection, like the *Deep Dense Face Detector* [49], which have minimal complexity and far fewer parameters. There are a myriad of robust facial detectors implemented using CNN architecture [50], [51], but at present time these implementations can only run with close to real time results on GPUs, and achieve image processing times of around one frame per second on high end CPUs.

YOLO or *You Only Look Once* [25] and its subsequent iterations [52] represent a relatively new and cutting edge method of multi-object detection in real-time using CNN architecture. YOLO is extremely fast compared to other object detection methods, and operates at real-time, with a loss in accuracy

(which may be acceptable for a task requiring high speed) compared to other object detection methods such as Faster R-CNN.

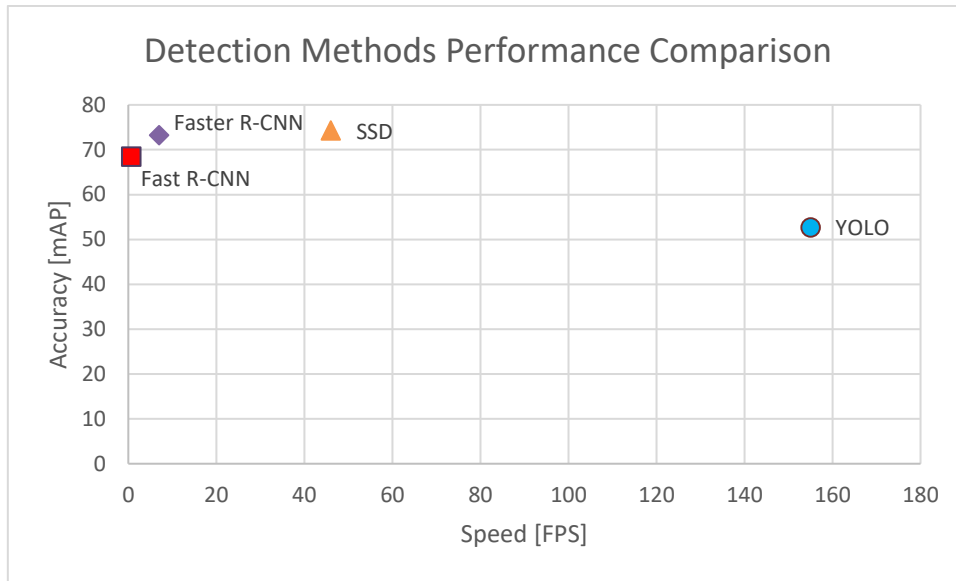


Figure 8: Detection Methods Performance Comparison; From the data seen in relevant papers [38], [53], a comparison can be made for the speed/accuracy trade-off often found in detection methods. The speed is measured in image frames analysed per second (*FPS*) and the accuracy is measured in mean Average Precision (*mAP*), a popular metric for measuring object detector accuracy. Here it can be seen that YOLO greatly outperforms other detection methods in terms of speed, but has a significant, loss in accuracy.

As with other detection methods, the optimal performance of these architectures comes from their deployment on high-end GPUs, meaning performance will dip when these frameworks are deployed on embedded systems. Additionally, YOLO has a model size of 735MB in its original iteration, and 193MB in its second iteration, meaning they are too large to deploy on an embedded device [54]. There are purpose-built frameworks like *Fast YOLO* [55] and *YOLO-LITE* [56] which optimize memory and energy usage, allowing for YOLO detection methods to be implemented in real time on non-GPU devices, such as embedded systems. These frameworks suffer from a loss in accuracy in exchange for their speed on embedded systems.

It is possible to deploy pre-trained models using the latest version of the YOLO framework [57] specifically for facial detection. These models are often trained using *Wider Face* [58], which is a face detection benchmark dataset. By training a model for a single object class, there will inherently be less parameters in the network because of the decreased number of features to learn. A model based on YOLO architecture trained to detect only faces can be deployed on a CPU with improved speed to a normal YOLO model, but the real-time capability on a CPU of this model is still lacking [59].

2.3.2 Haar cascades

The seminal paper, *Rapid Object Detection Using a Boosted Cascade of Simple Features* [60] expanded upon the idea to use *Haar* basis functions for feature detection in an image [61]. Some CNNs spend in the region of 90% of their computation time in the convolutional layer [42]. This is partly because of the computationally expensive process of classification, which requires rich, robust features provided by the convolutional kernels determined by training, [42] [62]. By using manually determined *Haar-Features*, instead of a large number of trained convolutional kernels, it is possible to implement an image detection algorithm with close to real time performance on an embedded device.

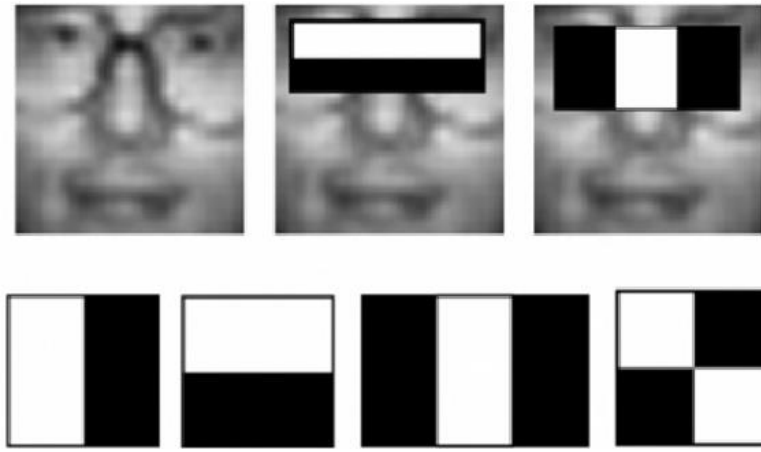


Figure 9: Haar features; The Haar basis functions (*bottom row*) are applied across an image using a sliding window. The sum of the pixels in the white area are subtracted from the sum of the pixels in the black area, giving a value as to how much a region correlates to a Haar feature, i.e if the feature is detected in that region. The face (*top row*) has Haar basis functions overlaying some of the features they would detect to show how these functions can be made to detect areas of different intensities within an image of a face [60] [63]

In the paper first mentioned, [60] a method still relevant today was put forward to achieve real time face detection. The papers contributions included the *integral image* which greatly simplifies the calculations needed in feature extraction, and the use of *AdaBoost* [64] to create highly effective classifiers. AdaBoost is used to train an *Attentional Cascade* to determine the relevance of a given detected feature, and from this training be able to determine what combination of feature detections corresponds to the positive detection of a given class, like a face. In fact, the problem was originally developed to extract and detect facial features, making it ideal for the needs of the project developed throughout this report [65].

The simplicity and relevance of this technique for the project at hand is evident in the many papers where Haar cascades are implemented on embedded devices to perform real time detection tasks ranging from eye detection on an embedded device (with detections around 60ms) [66] to complex tasks, such as controlling a gimbal on a drone to track facial features [67]. This indicates that it may be a good approach to take in trying to achieve real time face detection on an embedded device. Additionally this method is computationally inexpensive, as there is no deep learning required. This simple method can also be made more robust for a video stream, meaning real time detection actually makes it better, as the regions that is used for feature extraction can be localized to a region near the last positive detection, or by methods of motion tracking [61].

The technique is not perfect though, and has an inherent trade-off between speed and accuracy of detections (referring to the number of false positive detections) depending on the strictness of its classifier [60]. For the system under design, a balance must be found to achieve accuracy at an acceptable detection speed. Additionally, this method in its purest form is not rotation invariant, stemming from the square and rectangle feature windows. There has, however, been research into solving this rotation problem [68].

2.4 Tracking algorithms

Where detection aims to determine the location of an object, tracking aim to estimate the motion of an object model. This is represented by a trajectory of a tracked body within an image frame. There are different ways to represent an object when tracking it, such as optical flow techniques [69], and with the use of bounding boxes, seen in the popular Median Flow tracker [70]. The Median Flow tracker estimates the trajectory of a bounding box between consecutive frames by tracking an object in the forward and backwards directions in time, and then measures the difference between the two

trajectories. This difference is known as the “Forward Backward Error”. Minimizing this error allows for accurate trajectory determination in a video feed. Another well explored and computationally inexpensive object motion representation method is with point tracking [71] [21].

By training neural networks relationships between objects and motion using videos and images with manually labelled bounding boxes, the weights from these networks can be used to track objects within bounding boxes with real time performance. This implementation is called the *GOTURN* tracker [26]. This method does the bulk of its computation offline to learn the weights of its neural network, but online it uses these set weights for simple computation and tracking. This method might be ideal for an embedded device, as the real-time tracking is computationally inexpensive, and extremely fast.

Tracking is much simpler, in the computational sense, than detection, but it is common for a detection algorithm to be utilized in the initialization of a tracking algorithm in order to accurately show the tracker the desired point or bounding box to track [16]. A robust tracker which uses a correlation filter, and requires the initialization of its initial bounding box (which can be achieved with a detection algorithm) is the *MOSSE* filter (*Minimum Output Sum of Squared Error*) [27]. Correlation filters work by estimating the best filter to be applied to an input image in order to achieve a desired response. These filters are trained online. The reason that correlation filters are so fast at analysing images is because of their computational simplicity, which makes use of the Fourier domain. After initialization, the correlation filter searches for the maximum correlation between the initial ROI bounding box and a region in the current frame. This new region of maximum correlation is now set as the object to search for in the next frame. Correlation filters track objects through rotation and occlusion, making them extremely robust. Not only is this tracking method robust, but it is extremely fast. Its computational complexity is dependent on the number of pixels in the image, so by altering this parameter, it may be possible to implement this tracker on an embedded device. Another fast and accurate tracking algorithm which uses correlation filters and is initialized with a bounding box is the *KCF Tracker* (*Kernelized Correlation Filter*) [72].

In addition to initializing the tracker, feature detection can also be used in making trackers more robust against occlusion [73]. A combination of tracking and detection could be implemented on an embedded device to generate the required stabilization results, by achieving the initial accuracy needed when detecting an object to stabilize an image around, and then using tracking to model the motion of this object in real time.

3. Methodology

This section aims to detail the processes and methods that will be followed in order to solve the problem at hand. The items detailed in this section will inform all design decisions to be made throughout the project.

3.1 Research

Section 2 explored the state-of-the-art relevant to solving the problem this project poses. The ideas explored in this research section will be invaluable in the proceeding sections, as the concepts and techniques studied will be applied and considered in performing the tasks to come. This research forms the backbone of all proceeding design and implementation.

3.2 Design Process

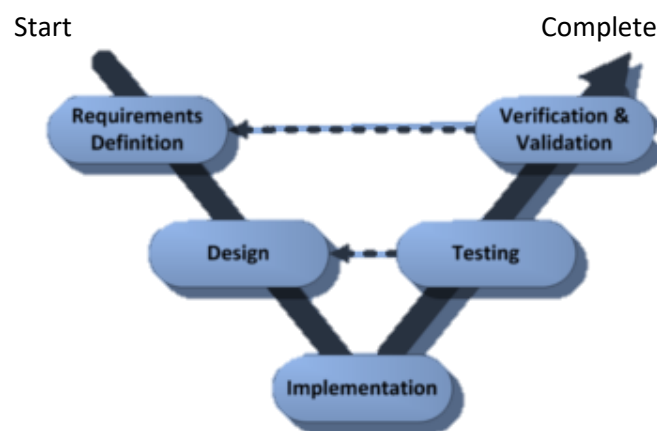


Figure 10: V-Model used in the design Process; this algorithmic approach to design ensures that the initial requirements defined are validated. This circular approach ensures that throughout the design process, the requirements for the project are referred back to.

The V-model design approach (see above) will be implemented in the design of the system as a whole as well as its smaller subsystems. This design approach requires for the system requirements and test procedures to be clearly stated, so that the project can be validated before its completion.

3.3 System level Requirements Analysis

Before proceeding with the design process, and subsequent sub-system designs, it is important to map out exactly what functional requirements the system must meet, in order to meet the broader user requirements of the system. The user requirements will be assigned codes (UR_XX) to ensure that the functional requirements of the system will satisfy all user requirements. Each functional requirement will be assigned a reference code (FR_XX) for easier in text referencing, as well as for the *Acceptance Test Procedures* (ATP_XX) seen later.

3.3.1 User Requirements

User Requirement ID	User Requirement Description
UR_00	The system must perform vertical motion stabilization.
UR_01	The system must be portable and hand-held.
UR_02	The stabilization of the system must be based on the relative position of a visual subject, which is a human face.

3.3.2 Functional Requirements

Functional requirement ID	User Requirements met (ID)	Functional Requirements
FR_00	UR_00	The system must have a method to precisely actuate the camera platform in linear vertical space.
FR_01	UR_00 UR_02	Accurate control must be exercised on the linear actuator in order to achieve vertical stability while tracking a set point determined by the object of interest detected by the camera.
FR_02	UR_01	All subsystem devices must be self-contained and compact in order to be portable and hand-held
FR_03	UR_02	The system must include a method by which an objects position can be determined from a sensor.
FR_04	UR_02	The system must be able to identify a specific object class. i.e, it is not enough that the system can track bodies through space, it must also be able to track <i>the right</i> bodies through space.
FR_05	UR_00 UR_02	The system must have functionality to determine whether or not it is vertically stable relative to a visual subject.
FR_06	UR_02	The relative position between the system and the visual subject must be processed in real time.

3.3.3 Acceptance Test Procedures (ATPs)

In order to conclude whether the design developed throughout this report was successful, a set of broad, testable criteria must be defined to confirm that the user and functional requirements mentioned above have been achieved. These tests are run at the “verification and validation” phase of the V-diagram model seen above. Each ATP will be linked to one or more functional requirements from the table above. Here the final acceptance conditions for the testing phase can be found. The procedures which confirm the acceptance conditions seen in this table are explained in the testing section of this report.

ATP ID	Functional requirement ID	Acceptance condition
ATP_00	FR_00	When running, and after a reasonable settling time, the system must produce a response with no tracking error of the desired set-point.
ATP_01	FR_01, FR_05	When tracking a stable set point, the set-point must correspond with the centre of the subject being tracked (the centre of the face).
ATP_02	FR_02	The completed system must be able to be held by a user in one hand, like a standard video camera.
ATP_03	FR_03, FR_04	The system must be able to accurately draw a bounding box around the face being tracked, without losing the face in the image for more than one second at a time.
ATP_04	FR_03, FR_04	The system must be self-initializing, and must be able to detect a face (assuming one is present in the image frame) within one second of activating the system.
ATP_05	FR_06	The system must run process and respond to information from the camera module with a frame rate close to real time. This rate is determined to be 60 frames per second.

ATP_06	FR_01	The system must take less than one second to correct for a step in output disturbance.
ATP_07	FR_01	The system must be able to change its direction of motion (from upwards to downwards or vice versa) instantaneously.

3.4 Software selection

In selecting software, a set of technical specifications were defined in order to meet the functional requirements mentioned above:

Technical Specifications:

- Software that can perform computer vision tasks such as object tracking and detection is required (FR_03, FR_04, FR_05)
- The software must be able to process images in real time (at more than 60fps) (FR_06)
- The software must be able to initialize GPIO ports for control signals (FR_00)
- The software must be able to operate on an embedded device (FR_02)

The language selected to meet these requirements was *Python version 3.7* [74]. Python is a high level coding language with many additional packages and libraries that can be installed, including packages used to generate *Pulse Width Modulation* (PWM) signals to interface with motors via a micro-computers GPIO ports. Python can run effectively on devices with only 512 MB of RAM. This shows that this language has extremely low hardware requirements, and can be deployed on embedded devices. Many embedded devices come with Python Pre-installed, such as the *Raspberry Pi*.

To fulfil the required computer vision tasks, an essential library for such applications must be imported to Python: *OpenCV* [75]. OpenCV (Version 4.0.0) comes with a selection of easy to use APIs, which can initialize tracking and detection algorithms in real time using simple function calls. These functions can implement algorithms such as *Haar Cascades* and the *Mosse Tracking Algorithm* discussed in the literature review using a single line of code, making them extremely powerful tools in achieving the required computer-tasks. OpenCV can run on as little as 1GB of RAM, meaning that it too can be compiled on an embedded device. Real time image processing performance will be subject to the hardware and algorithms used.

Another important package to the project is *Pi Camera* [76]. This package allows for easy interfacing with the Raspberry PI's camera module (elaborated upon below). The GPIO package for the RPi will also be used throughout the project to interface the selected micro-controller with the system used to achieve vertical motion.

3.5 Hardware Selection

This section will detail the deliberation in deciding what hardware will be used in the process of designing the system as hand. This includes component and subsystem selection. This section will include details on the technical specifications required for each subsystem.

3.5.1 Micro-Controller selection

An embedded device in the form of a micro-controller (μC) will be used as the computer handling the software and code of the system under design. This is because of the compact nature (fulfilling FR_02) of these devices, as well as their GPIO ports which can be used to interface with external hardware. The μC selected must meet the following technical specifications:

Technical Specifications:

- The device selected must be able to interface easily with a camera (FR_03)
- The μ C must be able to support the software required for computer vision tasks (FR_04)
- The μ C must be able to interface with the selected system of vertical actuation (FR_00, FR_01)
- The system must be compact (FR_02)

Under these considerations, the device selected was the *Raspberry PI 3 Model B v1.2 (RPI)* [77]. This device has a 64 bit CPU with 4 cores, and is clocked at 1.2 GHz. The RPi has 1GB of RAM, which is shared with the Pi's GPU. The RPi can interface with cameras either through its USB ports, or through its CSI port. The latter method can be used to connect the RPi with its purpose built *Raspberry PI Camera Module* [78].

One of the main attractions of the RPI include support of software used in computer vision applications, such as *OpenCV* and *Tensorflow*.

The RPI has multiple GPIO pins used to interface with hardware such as the motors and actuators which are being considered as potential tools to be used in achieving vertical motion. Notably, the RPI can output an easily adjusted *Pulse Width Modulation (PWM)* signal which can be used for precision motor control.

The RPI has dimensions measuring 85.60 mm \times 56.5 mm \times 17 mm , making it a compact device.

As mentioned above, most state of the art image detection frameworks are only able to run optimally on high end CPUs or GPUs. The computers used in almost all of the papers discussed so far regarding computer vision implementation were many times more powerful than the RPI (with the RPI having only 1 GB of RAM). This means this lack of power will have to be accounted for when determining what architecture is used to perform the computer vision side of the systems operations.

3.5.2 Camera selection

A camera is a versatile tool in solving computer vision tasks.

Technical Specifications

- The camera must be small and light (FR_01)
- The camera must be capable of interfacing with a RPI (FR_06)
- The camera must have a resolution and frame rate capable of processing images in real time (FR_03) (FR_06)

In selecting a camera that can be used with the RPI, an obvious choice was the *Raspberry PI Camera Module v2*, as it is designed specifically to interface with the RPI [78]. This module had physical dimensions of 25 \times 24 \times 9 mm, and weighing 5g, making it extremely small and light weight. This camera module has a *python* package which can be used to utilize the various functions of the module [76], which include the ability to downscale the image resolution (with a maximum image resolution of 3280 \times 2464) which improves the accuracy of tracking methods which use correlation filters, such as the MOSSE and KCF tracker [27]. There are also functions available to the camera module which activate *Electronic Image Stabilization*, producing a less jittery image as an input to the software used.

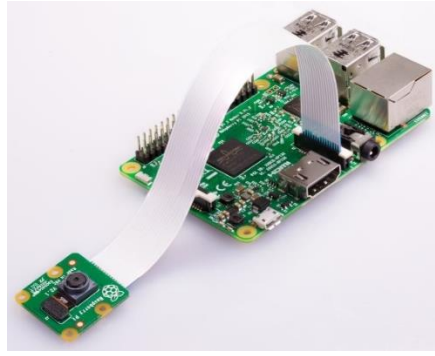


Figure 11: The Raspberry Pi 3 connected to the Raspberry Pi Camera Module; The Pi Camera is designed to connect directly to the computer using the Camera Serial Interface (CSI) [8].

3.5.3 Vertical actuator selection

The selection of the method to produce vertical motion to satisfy the functional requirements (FR_00, FR_01, and FR_02) proved to be the sub-system with the most viable options available to it. In determining which system will be used, the following technical specifications were to be met:

Technical Specifications

- The system must be capable of pure linear vertical motion (FR_00)
- The actuation provided must satisfy the calculated control action for the system (FR_01). In other words, the system must move quickly enough and with enough force (0.049N being the force required to vertically accelerate the Pi Cam).
- The actuator must be accurate enough to perform the desired level of object stabilization, with no tracking error when stable. (FR_00, FR_01).
- The system must be able to interface with and receive a control signal (a PWM signal) from a Raspberry Pi (FR_01)
- The system must be compact enough to be integrated into a hand-held physical system (FR_02). Because the actuator is the largest component no matter which exact method was to be decided upon, the rig would be built around this sub-system, making its size the least important of its parameters, as long as the system was still hand held.

The system of actuation decided upon utilized a *Stepper Motor*. These motors benefit from accurate position control, which is a useful characteristic when meeting setpoint related control requirements. Additionally, stepper motors benefit from a *holding torque*, which is the torque required to rotate the motors rotor even when the rotor is not moving by way of an external force. This holding torque is useful for the problem at hand, as if the camera position does not need to be adjusted, no additional control signal will be required, as the holding torque will keep it in place. Steppers can be controlled with the RPI by interfacing with its GPIO ports. The RPI has the ability to output a *Pulse Width Modulation* (PWM) signal, which can be used to accurately control the speed and direction of a stepper motors rotations. One downside of using a stepper motor is that there is no feedback from the motor itself as to the orientation of its rotor. This positional information can be compensated by using the RPI camera and the computer vision techniques selected to generate this data.

The model of stepper motor chosen was a *NEMA 17 Stepper Motor*, manufactured by *Wantai* [79].

A breakdown of the important specifications for this device includes:

- 48N.cm holding torque
- Step Angle (degrees): 0.9 Deg (400 steps/rev)
- 2-Phase (Rated Current: 1.7A/phase)
- Rated voltage: 3V

The holding torque should be more than sufficient to actuate the RPI camera, evident in that the gear radius used in converting the rotational motion to linear motion will be in the range of centimetres, and the extremely low weight of the module. The motor with a small step angle was chosen to ensure that the maximum accuracy in controlling the motors position can be achieved. The rated voltage and current of the motor will be important in selecting a motor driver.



Figure 12: The NEMA 17 Stepper Motor; The NEMA code 17 refers to a set of specifications that apply to the physical dimensions, torque, and power levels of motors in this class. It allows for the designing of systems that work with an entire class of motor, rather than a single model [79]

With the decision to use a stepper motor, more questions get brought to light regarding how to integrate the motor into the larger system. Stepper motors are able to provide rotational motion, not vertical linear motion. This means a method of translating the motion provided by the stepper into vertical linear motion is needed. There were three options considered: using a pulley system, using a lead screw and integrated nut, and the method chosen, using a rack and pinion. The rack and pinion was chosen because of its mechanical simplicity, which will theoretically make the system easy to control. The other two options would be better if the camera represented a heavy load, but because of the light weight nature of the RPI camera module, the rack and pinion is an ideal choice.

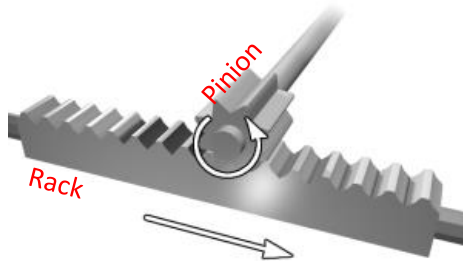


Figure 13: A Rack and Pinion; This gear system convert's rotational motion to linear motion through the interlocking of the gear teeth between the rack and pinion [80]. The relationship between the rotational speed of the pinion and the linear speed of the rack is dependent on the radius of the pinion.

Additionally, the stepper motor will need a motor driver in order to interface with the RPI. The RPI generates a pulse signal through its PWM pin output, which has a pulse frequency (Hz) which determines the speed of the stepper motor. This signal cannot be directly applied to the motor because of its 2-phase nature, and as such, the motor driver acts as an intermediary by taking in the control signal from the RPI, and using this signal to apply the correct power to the motor in order to achieve precise position and speed control.

The driver chosen was the *DRV8825 Stepper Motor Driver Carrier, High Current* [81].

This device was attractive for the following technical reasons:

- It has six different step resolutions (number of steps per input pulse), going as high as 1/32-step. This means that there are different levels of accuracy in the rotational increments taken, which is useful in our high accuracy task. This higher resolution also results in smoother rotational motion, which could allow the system to move seamlessly in conjunction with the body it is tracking.
- It can interface with 3.3V logic inputs, which is the logic output of the RPI
- It can output current up to 2.2A per phase, with the max current output being adjustable using an on-board potentiometer. This means it can be used with the stepper motor, requiring 1.7A per phase.
- It is notable that the driver operates from 8.2V-45V. The motor is rated at 3V, but the two should work fine together, as long as the current is limited on the driver to 1.7A per phase. This is because the driver will ensure that the motor does not draw too much current, which is far more dangerous in causing the motor to dissipate too much power than supplying it with a higher voltage.

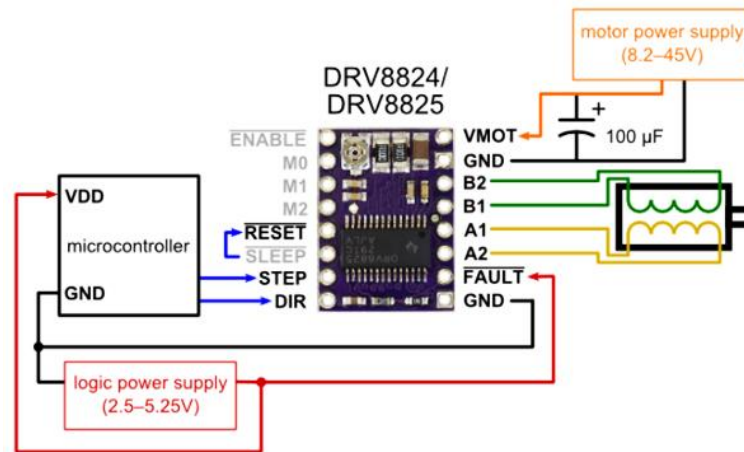


Figure 14: Interfacing a Microcontroller and Stepper motor with the DRV8825; The driver acts as an interfacing tool between the microcontroller (in this case, the RPI) and the stepper motor (the NEMA 17). The DIR pin is used in selecting the motor's rotational direction, the M0-2 pins are used in setting the microstep resolution, and the STEP pin is used in applying a speed controlling PWM signal. These pins provide easy direction, speed, and accuracy control of the motor [81].

Through this method of component selection, the vertical actuator will be implemented using a NEMA 17 stepper motor to generate the desired control unit, a rack and pinion gear system to convert the rotational motion from the motor into linear motion, and a DRV8825 to interface the sub-system that generates linear motion with the RPI. These separate components are used to make up the black box sub-system of the vertical actuator, which satisfies the technical specifications mentioned above.

3.6 Computer vision algorithm selection

The system will use a combination of detection and tracking algorithms, with this relationship elaborated upon in the design section. The final selection of these algorithms will be decided upon after the results from testing the whole system have been analysed. Here, a shortlist of algorithms that theoretically would work with this project is developed and qualitatively justified. The technical specifications that each algorithm to be tested include:

Technical specifications:

- Must be able to run in real time on a Raspberry Pi (FR_06)

- The algorithms must be able to provide information regarding the location of the ROI within a frame (FR_03, FR_04, FR_05)
- The algorithms must be self-initializing, and be able to detect the region of interest to be tracked without any external inputs (FR_04)
- It must be possible to implement these algorithms in Python 3.7

3.6.1 Object Detection algorithm selection

The detection algorithm to be used is the *Haar-Cascade* method, specifically for facial detection [65]. This method was selected because of its comparatively low computational expense compared to the other object detection methods discussed in the literature review. Deploying cascade based detection algorithms on embedded devices is a well solved problem, and has been achieved on devices far less powerful than the RPi [82], implying that it is a plausible solution to the problem at hand. Additionally, there are many open source pre-trained facial classifiers available online which will simplify the task of implementing a detection algorithm [83].

Haar-cascades suffer from high false detection rates, and suffer from poor robustness to occlusion and rotation. This means that compare to tracking algorithms, Haar-cascades do not perform as well in tracking an object between frames. An object tracker will be deployed in conjunction with the detection algorithm to supplement these shortcomings.

3.6.2 Object tracking algorithm selection

There are three tracking algorithms that are going to be compared and tested once the design of the system has been finalized: *Median Flow Tracker* (MFT) [70], *MOSSE Tracker* [27] and *KCF Tracker* [72]

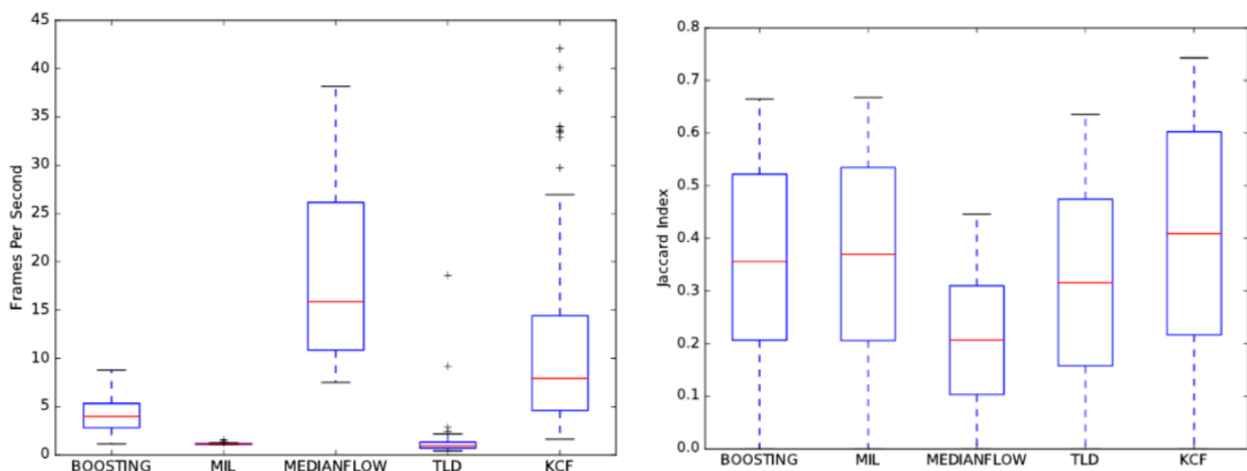


Figure 15: Box plot revealing averages and distributions of image processing speeds (left) and accuracy (right) for different trackers; these comparisons were made between trackers provided by the OpenCV tracker API, deployed on an RPi 3. From this data, it is seen that only the MedianFlow and KCF trackers perform at speeds close to real time, with the MFT achieving the best results in speed, and the KCF in accuracy [84]. The Jaccard index is a commonly used tool to compare two sets of data, and it was used here to compare the bounding boxes assigned by the algorithms to a set of ground-truth bounding boxes manually defined by the administrator of the test. In this way, accuracy is measured. Notably, this test was performed before the MOSSE tracker was added to OpenCV's tracker API.

The MFT [70] can perform with real-time speed on an embedded device, but tracking performance dips severely in the presence of occlusion or fast motion in the object being tracked. Additionally, the MFT has been found to be the least accurate (by the Jaccard index) of the tracking algorithms available on OpenCV (at the time of the referred papers publication) [84]. The tracker benefits from accurate tracking

failure reporting. This could potentially allow for the robustness of the tracker to be improved, if the correct response to a tracking failure is implemented.

The MOSSE tracker [27] can run at extremely high speeds on an embedded device, achieving results even higher than the MFT. It is robust to variations in light, scale and rotation. It also benefits from excellent occlusion detection, which allows for the tracker to continue tracking an object once an occlusion had been removed.

The KCF tracker [72] is both fast and accurate, as seen in the graphs above, the KCF excels in both speed and accuracy when deployed on an embedded device. A notable problem that arises from using the KCF tracker is that it is not capable of recovering from full occlusion. This negative aspect of the tracker could be amended by using a detection algorithm to reinitialize the tracker after full object occlusion has occurred.

All tracking algorithms mentioned can be implemented using OpenCV's tracker API. Using this API, these trackers require an initial ROI to be identified within an image, as an input, in order for the tracker to be initialized. It is possible to manually select this ROI in an image, but for the needs of this project, the Haar-cascade detection algorithm will be used to initialize the ROI used by the decided upon tracking algorithm. Because all three of the tracking algorithms discussed present viable solutions to the problem at hand, their differences in performance will be compared in the testing section of this report in order to determine which algorithm is empirically the best option for the final design of the system.

3.7 Workstation Setup

Following the hardware and software selection process, an environment must be set up in which these subsystems can be integrated into a coherent workstation where future designs can be tested and implemented.

The first step in this process is to install the most recent supported operating system for the Raspberry Pi, *Raspbian Buster* [85]. This version includes a GUI based desktop environment as well as some of the other software required for the project pre-installed, such as Python 3.7. Once the operating system is downloaded, *Win32 Disk Imager* [86] is used to flash the operating system (in the form of an .img file) to a 16GB micro SD card, as is required for the Raspberry Pi. To access the RPi, throughout the project, an SSH terminal will be used.

Once the Raspberry Pi is set up with a new operating system, the next step is to install the *picamera* [76] package for easy interfacing with the Raspberry Pi Camera Module [78]. This is achieved using the python package installer (*pip*), directly from the terminal using the following command:

```
Python -m pip install picamera
```

The task of installing the python computer vision package *OpenCv* [75] is the undertaking in the setup process most prone to failure. Packages are usually simple to install on python using either *pip* or the *apt-get* repositories. Because of the non-standard architecture of the Raspberry Pi, the most recent version of OpenCV is currently not available for installation on either of these repositories. Because of this, it is necessary to compile *OpenCV4* (the latest version as of the time of writing) from source. To do this, a set of dependencies must be installed, followed by a series of steps to complete the process of building and compiling this software. This process was guided by Adrian Rosebrock's excellent tutorial [87].

Once it is confirmed that OpenCV is successfully installed on the RPi, the RPi camera module can be set up and some basic code should be written to test that OpenCV, picamera, and the camera module have been configured correctly, and can work in unison.

The camera module can be setup by plugging it directly into the Raspberry Pi's camera port with a ribbon cable.

At this point, it can be determined that the software side of the project workstation has been set up and configured correctly. Next comes the task to set up an environment for testing and integrating the hardware. A breadboard is used to connect the RPi to the DRV8825 motor driver, by wiring the appropriate GPIO pinouts from the RPi to their corresponding pins on the motor driver.

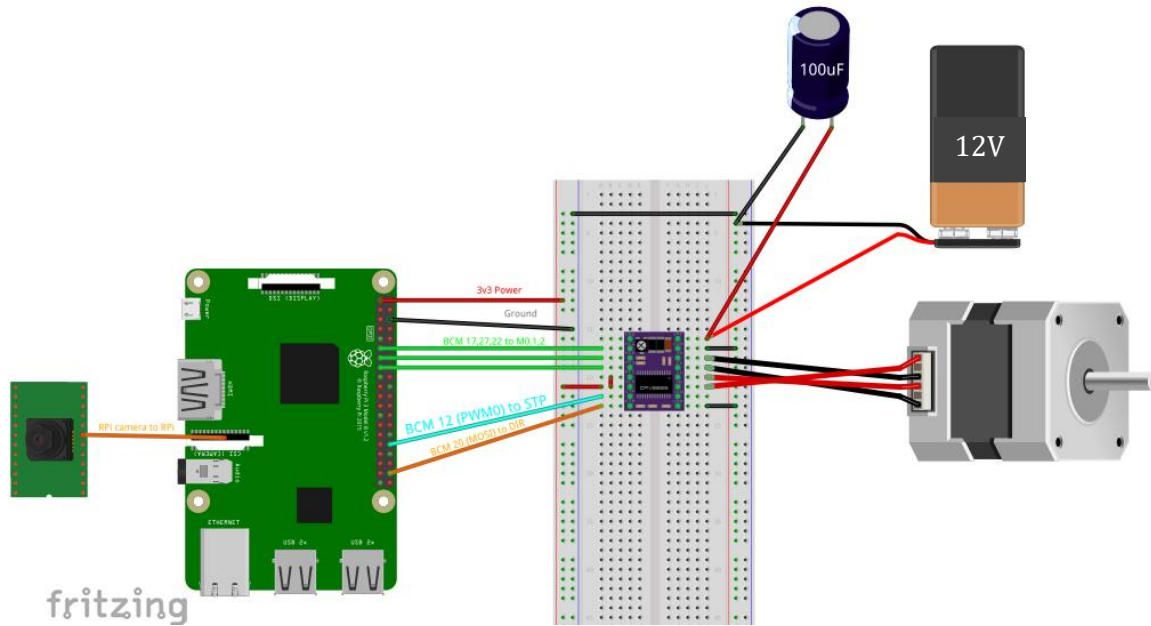


Figure 16: The hardware workstation; Appropriate GPIO ports from the RPi are connected to corresponding pins on the DRV8825. The step mode (M0-2) and direction (DIR) pins on the driver can be connected to any GPIO pin on the RPi, as their functionality is selected with a high or low logic signal. The step (STP) pin on the driver has to be connected to a PWM pin on the RPi, so that the step frequency can be controlled. The 12V power supply depicted here is actually a variable DC power supply, which is set to output 12V. Because of the nature of the wiring required to correctly connect the driver to a NEMA 17 motor, this connection is hard to interpret from this image. To see this connection in greater detail, refer to figure 14.

Before connecting the motor to the breadboard, it must be ensured that the DRV8825 is configured with the correct current limit for the system by adjusting the trimmer potentiometer found on the device. To achieve the correct current limit, the following equation (specifically used for the DRV8825) must be used:

$$CurrentLimit = VREF \times 2$$

The motor is rated at 1.7A per phase, and as such, the potentiometer should be adjusted until 0.85V is read by a multimeter connected to the VREF pin on the driver. The driver is not recommended to provide more than 1.5A without cooling, so a simple heatsink must be installed on the driver chip.

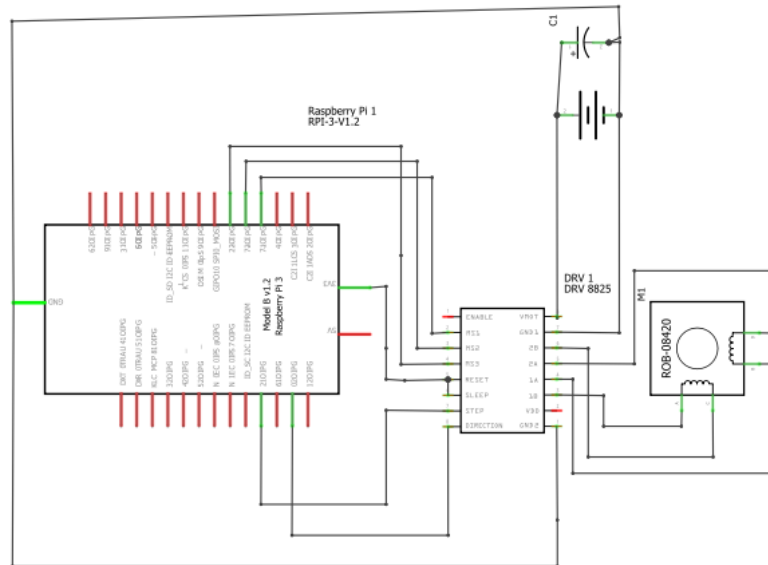


Figure 17: Circuit diagram of RPi interfacing with the DRV8825 and NEMA 17

Once the DRV8825 has been seen to be properly connected to the stepper motor, some basic tests are to be run to see if the motor direction and step mode can be controlled as required from the RPi.

At this point, Python 3.7 has been configured on the Raspberry Pi with all the packages needed, and the RPi has successfully interfaced with both the RPi camera module and the stepper motor. It can be determined that the workstation is set up for the project's needs, and the next step of developing more detailed sub-system designs can be initiated.

4. Design

With the requirements defined and the software and hardware components selected, a design process must be followed to integrate these requirements and sub-systems. The design process begins with an overall system design, which is then broken down into sub-system designs. A link to a video of the final system design being showcased can be found in the appendix.

4.1 System Design

At its core, the problem of stabilizing a body (a faces location within a camera frame) around a reference (the centre of the image) is a control problem. Therefore, the system will be modelled using classic closed loop control.

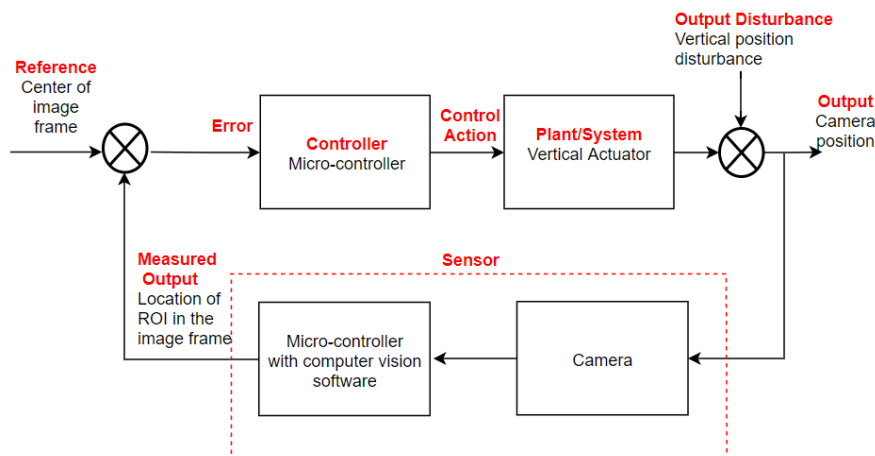


Figure 18: The classic control feedback loop for the system; the red writing represents the black-box sub-systems in terms of the classic control loop, and the black writing represent these sub-systems implementation within the system under design. The stabilization of the systems when vertical output disturbance is present is the true nature of the control problem at hand.

The control loop will have a static reference point which is to be tracked, being the centre of the image frame captured by the camera. In this way, vertical stabilization will be achieved by aiming to keep the ROI in the centre of the image frame using control methods.

The subtraction of the negative feedback (and the subsequent error), the pre-processing of the camera image, and the control action will all be computed using the Raspberry Pi. This means that the majority of the implementation of the control loop seen above will be achieved by developing Python code on the RPi.

Although the feedback loop is a useful tool in understanding how the overall system works, in implementing control of this system, specifically the stepper motor, a simplistic approach will be taken in the design of the actual controller algorithm, due to some non-linear characteristics of the motor system (discussed more below).

4.2 Hardware Design and Implementation

The design part of the hardware used in the system involves developing hardware that allows for the interfacing between the stepper motor, the camera module and the rack and pinion system. The V-model design process mentioned above must be implemented throughout.

4.2.1 Hardware design process

The first step in designing the hardware required for achieving vertical stabilization is to create a sub-system which can generate vertical motion. As described above, the system decided upon will make use of a NEMA 17 stepper motor interfacing with a gear system, namely a rack and pinion. To calculate the maximum radius that the pinion can have, it is necessary to do a simple static equilibrium equation using the force required to hold the RPi camera in vertical space:

$$F \times r_{pinion} = T_{motor}$$
$$m_{camera}g \times r_{pinion} = T_{motor}$$

With symbols m , g , r and T representing the mass of the camera, acceleration due to gravity, the radius of the pinion, and the torque provided by the motor respectively. The masses of the rack and pinion are considered to be negligible in these workings for illustrative purposes. If the equation is converted to an equality, where the motor torque is given the value of the maximum holding torque that can be provided by the NEMA 17, and the weight of the camera is subbed in, the following can be derived:

$$0.04905N \times r_{pinion} \leq 0.48N.m$$
$$r_{pinion} \leq 9.786m$$

This calculation reveals that because of the relatively low weight of the camera module, any pinion gear radius less than 9.768m would work for the problem at hand. This means that pinion selection is almost trivial, as the requirements to move the RPi camera will certainly be met.

With this calculation in mind, a design must be developed to create a rack and pinion system that can interface with the selected NEMA 17 stepper motor. Because of the small load on the rack and pinion, the strength and integrity of the materials used to manufacture the gear system are not of critical importance. In fact, because of the relatively low torque available from the stepper motor, using a low weight rack is the best option. From these realizations, a 3D printed rack and pinion system printed in PLA is developed.

The process of designing the rack and pinion gear system is a matter of combining and modifying two different open source 3D models. The first of these models is a rack and pinion gear system designed specifically to attach to a NEMA 17 stepper motor [88]. The standardization of all NEMA 17 motors ensures that this model will fit to the specific motor that is used in this project.



Figure 19: 3D printed NEMA 17 rack and pinion design, showing each component (left) and the system assembled (right); the print is made up of 6 components, and requires two M3 nuts and bolts to fully assemble [88] .

This system requires some modifications to meet the requirements of this project, all of which involve making alterations to the rack of the system. One of these alterations is to lengthen the rack. In its

original form, the rack is 7cm. With this length, there can be, at most, 5.5cm of the rack protruding from its enclosure. With the camera module on the racks end (discussed more below) there would be only 2.5cm available to adjust the vertical position of the camera module. It is therefore imperative to change the length of the rack. The rack was doubled in length, from 7cm to 14cm, resulting in a system with 9.5cm available to move the camera in vertical space.

The next modification to be made to the rack involves the task of adding a container for the RPi camera module to the rack of the gear system. In this way, when the rack moves in linear space, it actuates the camera module along its line of action. As mentioned before, the model used for the camera case is also an open source 3D model [89].



Figure 20: 3D printed Raspberry pi camera case, showing the printed components with the camera (left) and the assembled system (right) [89].

To implement this new rack design, *SolidWorks* is used lengthen the rack, and combine the rack 3D model with that of the camera module.

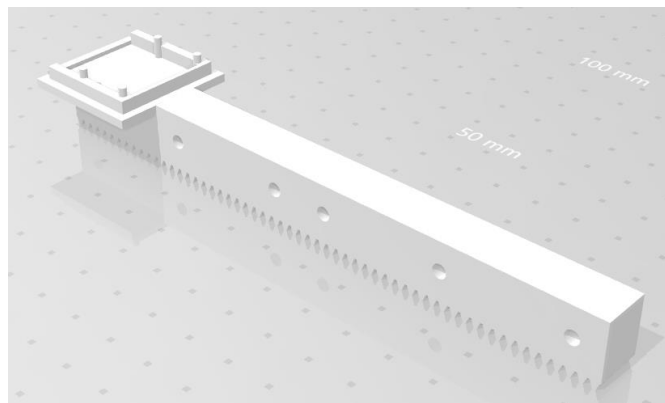


Figure 21: The newly designed rack; the new rack is longer, and is designed to mount the Raspberry Pi camera module.

Once printed, it is a simple matter of attaching the gear system and its container to the stepper motor, mounting the camera module on the rack, and sliding the rack into place.

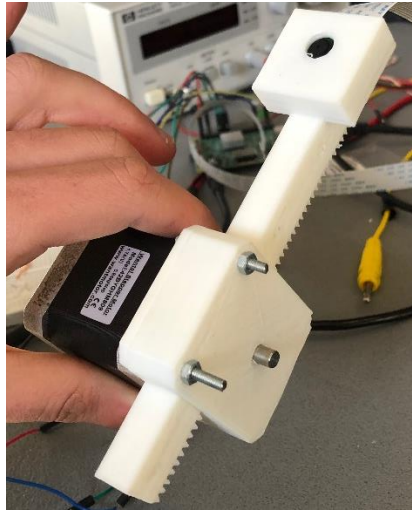


Figure 22: The fully assembled system

4.2.2 Hardware performance testing and design iterations

In order to confirm that the hardware is working as required, and to determine what iterations to the hardware design are still required, some functionality tests must be run on the relevant hardware subsystems. From these tests, some of the limitations of the hardware can be identified, and some errors corrected, resulting in iterations in the hardware design.

When running performance measurements on the hardware, the basic elements to be used in the control of the system have to be confirmed to be working as required, starting with the motor driver. The DRV8825 motor driver is designed to operate with a fixed 50% duty cycle. This means that the rotational speed of the motor is not controlled by adjusting the duty cycle, but rather by adjusting the frequency of the PWM signal into the motor driver, through the STEP pin of the driver.

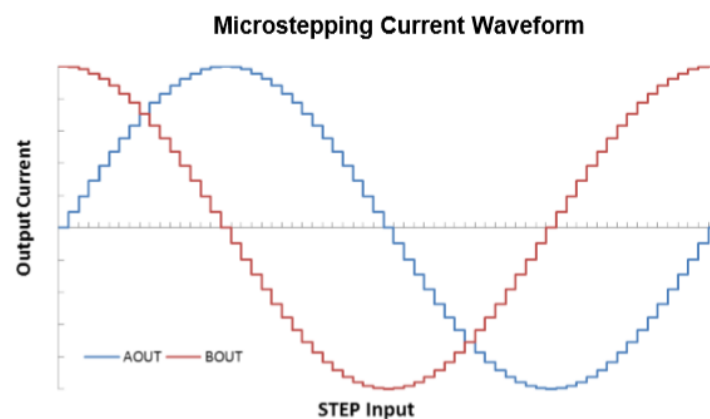


Figure 23: Microstepping Current Waveform of the DRV8825 motor driver; the driver uses an internal indexer which is able to accurately control the current level in each output winding according to its current index, resulting in a specific electrical angle for each index. Each step mode of the driver corresponds to a different indexer, selected using the MODE pins. For each rising edge from the STEP input causes the indexer to proceed to its next state. Therefore, by increasing the frequency of each rising edge, the indexer is cycled through faster, and the motor rotates at a higher speed. By using the driver's microstepping functionality, the rotational motion of the motor will be smoother because of the more precise excitation of the windings [81].

The maximum linear speed of the system is measured by applying the highest frequency PWM that causes the motor to rotate from rest. Stepper motors require a speed build up to be able to run at top speed. Because the system will have to change directions rapidly, this speed build up is not always present. Therefore, when testing the linear speed of the system, it must be done from rest. From a series of tests to see what stepper mode configuration and pulse frequency can get the top speed, it is found that this top linear speed is in full step mode, with a pulse frequency of 2000Hz. The linear speed from

this configuration, starting from rest, is found to be 10.58cm/s. This value is derived by timing the full extension of the rack from its housing using a high speed camera. This speed cannot be improved, as it is limited by the capabilities of the motor, and therefore no iterations to design can be performed to improve this value.

In addition to performance tests, the system is tested by observing the camera jitters present when actuating the rack. From this movement, a design error can be observed. The rack can scrape along a set of screws, causing jitters. The screws must be filed down to fix this problem. The camera is tested for image blur by recording a video when the rack is moving at top speed. The blur is noticeable on lower camera shutter speeds (below 20fps) but with higher speeds (above 30fps) the blur is unnoticeable.

When testing the different step modes used by the motor driver, it is noted that each mode requires different pulse frequencies to achieve the same rotational speed in the motor. The higher the resolution of the step mode (with microstepping being the highest and full stepping being the lowest), the slower the revolution of the motor. This trade off means that when controlling the motor, there may be appropriate and inappropriate instances to use (and potentially switch between) the different step modes.

When referring to the V-diagram based design process mentioned above, the iterations made to the hardware design, and the issue discovered in testing which called for that iteration, can be summarized as follows:

1. The original rack was too short for the purposes of the project, so SolidWorks was used to lengthen it
2. The original design did not have a container for the camera module used, so one was added
3. The printed design had dimensions which were slightly incorrect for fitting the camera module into the container. The container was sanded down.
4. The assembled system could not be tested freely because of the limited length of the ribbon cable connecting the RPi to the camera module. A 60cm cable was ordered and attached, fixing this problem.
5. Two screws were pressing into the rack, causing jittering and unwanted vibrations in the racks motion. These screws were filed down to allow for smooth linear motion.

4.3 Software Design and Implementation

As mentioned, the control of the system is to be achieved digitally using a Raspberry Pi micro-controller to adjust the rotational speed of the stepper motor using a PWM signal. This rotational motion is converted into linear motion using a rack and pinion gear system. Because at this point the hardware has been designed, tested, and confirmed to be working for the tasks at hand, the workstation configuration discussed earlier must be used throughout the iterative process of designing and testing the code used to control the system. The complete code designed in this section can be found by following a link provided in the appendix.

4.3.1 High level code design

Before any actual code was written, a black box design to implement the control feedback loop shown above in Python code on the Raspberry Pi must be developed. This design therefore is generated with the camera modules ability to take in image data, and the RPi's ability to output a control signal in the form of a PWM signal both in mind.

The flowchart diagram below is used to show the complete path that the code will take from receiving an input image to outputting a control signal through the RPi GPIO ports. This algorithmic sequence can be repeated continually in order for it to be applied to a constant video feed. The diagram will briefly be elaborated upon to explain the exact functioning of each row in the flowchart, by way of explaining the system as an algorithmic sequence, which can be continuously repeated:

1. The processing loop is entered into
2. An image is read from the camera module, and processed by the Raspberry Pi
3. If a tracking algorithm is already initialized to track a face, the program moves to the next row. If not, the program will use a detection algorithm to locate a face in the image. Only when a face is detected, the system will move to the next row
4. If a new face was detected in the last row, a tracking algorithm is initialized. Alternatively, one has already been initialized. The centre pixel of the ROI from the tracker is then used to compare to the centre of the image frame to calculate the error.
5. If the error is determined to be 0, the algorithm skips to the last step, meaning no control action is applied.
6. Depending on the sign of the error (positive or negative) the direction that the motor will turn is decided. This is achieved by applying a logic signal to the DIR pin of the DRV8825 motor driver.
7. Depending on the magnitude of the error between the centre of the ROI and the centre of the image frame, the resolution of the stepper motor is decided upon, between three levels of accuracy.
8. The step mode is activated based on the previous row. This is achieved by exciting the STEP MODE pins of the driver using logic signals from the RPi GPIO ports. If the error is large, the step mode is set to quarter step, in order to achieve fast, but relatively inaccurate error correction. This idea is used for three different error levels, where the greater the error magnitude, the greater the motor speed and the smaller the accuracy.
9. The control signal is outputted by a PWM pin from the RPi, applying a PWM signal to the STEP pin of the motor driver.
10. The loop is ended. When the code is implemented, it will be at this point that the algorithm returns back to step 1.

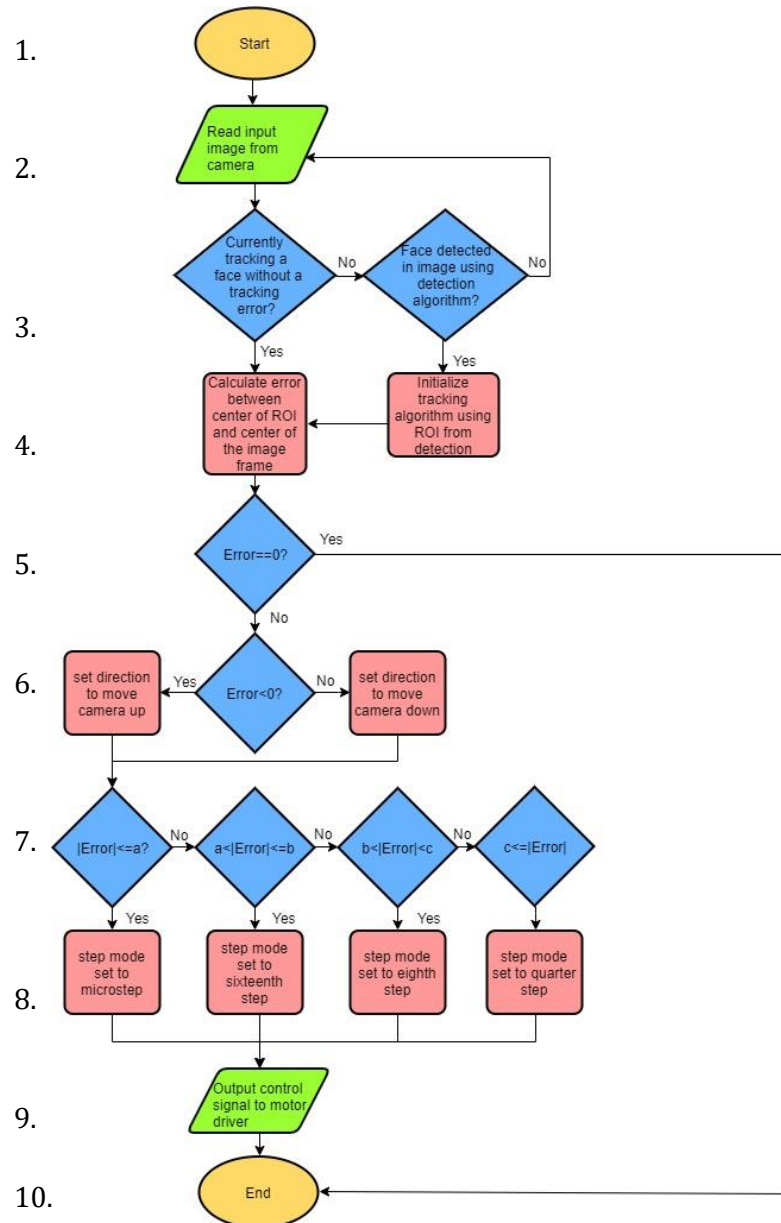


Figure 24: Flowchart explaining black-box design of code; the values 'a', 'b' and 'c' seen in the diagram are parameters to be tuned through iterative testing to achieve the best results. The corresponding algorithmic steps can be seen on the left of the flowcharts rows.

Once the overall system has been designed as specified above, a myriad of sub-system designs are to be finalized.

4.3.2 Computer vision algorithm design and integration

The use of OpenCV ensure that the implementation and initialization of computer vision tracking and detection algorithms is made relatively simple. The more complicated design task is then to select tracking and detection algorithms which can work efficiently on the under-powered RPi, and to implement code which is optimized to run these algorithms. The combination of a detection and a tracking algorithm will be used to increase the robustness of the tracker. Earlier in this paper, it was explained why Haar-cascades for facial feature detection would represent the detection algorithm used, and why the MedianFlow, KCF and MOSSE trackers are to be considered for the tracking algorithm used by the final system.

From the flow diagram above, it is notable how little the detection algorithm is to be used in the actual implementation of the system. For this project, the detection algorithm acts more as a supplement to

the tracking algorithm decided upon. The detection algorithm will be used for two tasks: To initialize the tracker, and to improve the robustness of the tracker to occlusion and lighting changes present in the image, which may cause tracking failures.

OpenCVs *Tracker* class can be used to initialize the desired tracking algorithm. Once an object of this class has been created, the tracker is initialized using the following line of python script:

```
retval=tracker.init(image,boundingbox)
```

Here, 'tracker' represents an object from the tracker class which has already been created. The returned value from this line of code is a Boolean value, being true if initialization was successful and false if not. The parameter 'image' represents the initial image frame from the video feed the tracker is using. The second parameter 'boundingbox' is extremely useful for the task at hand. This parameter gives the initial bounding box, or ROI, within the image, effectively telling the algorithm what to track. Because Haar-cascades return a bounding box revealing where a face is detected in an image, the output from this detection algorithm can be used as an input to the tracking algorithm used. This makes the automatic initialization of the system relatively simple, as all this task requires is to run the detection algorithm continually until a face is detected, at which time the location of this ROI is passed as an input to the tracking algorithm, initializing it in the process.

This method of initialization is also used to make the tracker more robust to tracking failures. For every image frame from the continuous video feed provided by the camera module, the following line of code is run, once the tracker has been initialized:

```
retval, boundingbox=tracker.update(image)
```

At this point, the tracker is updated by taking in the most recent image frame as its input parameter. This action outputs the current ROI in the form of a bounding box, and a Boolean which is true in cases where the tracked object can be located in the current frame, and False otherwise. If false, it does not mean that the tracker has failed in terms of its basic operations, but rather that the subject being tracked is not present in the image. This Boolean object can be used to make the tracker more robust, by using the detection algorithm when the tracker has lost the subject to reinitialize the tracker. Because the trackers being used have means of reinitializing themselves when the subject being tracked re-enters the image frame, the detection algorithm should not be deployed as soon as the subject is lost. Rather, there must be a delay before switching from the tracker algorithm to the detection algorithm. The length of this delay is tuned through iterative testing, and is finalised as 0.5 seconds. This delay is long enough for brief tracking errors to occur, providing enough time for the tracker to reinitialize itself, but also short enough for the complete initialization of the tracker using the detection algorithm to be unnoticeable to an observer of the system.

The integration and functioning of the detection and tracking algorithms can be seen explicitly in lines three and four of the system level algorithm seen above and in figure 24.

4.3.3 Computer Vision Algorithm Optimization

In order to optimize the speed of the tracking algorithms (specifically the MOSSE and KCF trackers, which use correlation filters), the resolution of the video feed is set as low as possible, to 160×128 (being the lowest resolution allowed by the Pi camera). This speeds up the programs analysis of each image frame whilst tracking for a number of reasons. One such reason is that there are less pixels for the algorithms to search through when tracking. Another reason for this implementation is that the initial bounding box will be set around a low resolution image of a face. Because of the more abstract

nature of this image, the tracker is searching for a less detailed version of a face in each image. This makes the system more robust to rotation and distortion in the facial image. The video feed is also set to capture images in black and white. This makes the system more robust to the effects of lighting changes within an image, and slightly increases the speed of the tracker used.



Figure 25: Analysis of a typical input image, with relevant tracker information; this image format is used as an output when testing the system, and is an accurate representation of the image and information used by the system. As mentioned, the image has a resolution of 160×128 , and is analysed in black and white. Displayed in the top right corner are the tracker type used, the image frames analysed per second by the tracking algorithm, and the error in pixels between the centre of the ROI and the centre of the image frame (with this calculation elaborated upon in the proceeding sections). The Bounding box describing the ROI is represented by the blue box in the centre of the image.

4.3.4 Motor Control Design

The abstract description of how the stepper motor is controlled is seen in the flow chart and system algorithm described above, from row four to row nine.

The process begins by calculating the distance, measured in pixels, between the centre of the image frame and the centre of the bounding box defined by the tracking algorithm used. This distance describes the error, when describing the system as a control feedback loop. The sign of the error determines the direction of the motors rotation, which in turn effects the motion of the rack and camera. This directional information is applied to the motor with a logic signal from the RPi to the DIR pin of the DRV8825. The error is determined in the code as follows:

$$\text{Error} = \text{bbox}[1] + (\text{bbox}[3]/2) - 64$$

This equation gives the error of the control feedback loop. ' $\text{bbox}[1] + (\text{bbox}[3]/2)$ ' represents the vertical centre of the ROI (which has been returned by the tracking algorithm used, as discussed above), and is equivalent to adding the vertically lowest pixels location, with the value half of the bounding boxes height. Because the resolution of the image is 160×128 , the '64' seen in the equation represents the halfway point (half of 128) in the vertical axis. The error here is therefore identical to that of a control loop, as it is equal to the output (the vertical centre of the ROI as determined by the tracker used) minus the setpoint or reference (the vertical centre of the image). This error value will be used multiple times throughout the code, as seen bellow.

It was mentioned in the 'hardware testing' section that a PWM signal with a specific frequency will achieve different rotational speeds in the motor depending on the active step mode. From this observation a problem arises, in that if a control algorithm were to be used to adjust the systems speed by controlling the pulse frequency, there would be non-linear regions of control when changing the step

mode. Changing the step mode is important for the task at hand, because the system would benefit from precise, slow movements when adjusting for a small error (microstepping) but fast movement when adjusting for larger errors (half or quarter stepping). This non-linear region means that classical control techniques will not be applicable in the design of the controller.

The way in which control is achieved is by adjusting the step mode depending on the magnitude of the current error. Each step mode has a set PWM frequency applied to it, which in turn achieves a set vertical speed in the rack. Because the stepper motor's rotation can be stopped instantly (by setting the duty cycle of the PWM signal to 0), to achieve control with no overshoot, the motor simply needs to be stopped when the tracking error is equal to zero. The error correction is then similar to that of a high gain controller, where the error is corrected in the shortest time allowed by the system. From iterative testing of the system, it is noted that the lower resolution step modes are prone to camera jitters, so they are only applied when there is a large error. The system uses a 'gearing down' method to correct the error, where a large error is corrected by a step mode which is low resolution in its rotational precision, but fast in its actuation of the rack. Once the error is smaller, the step mode resolution is increased, and in turn the speed is decreased. Finally, when the error is close to zero, the motor implements microstepping to smoothly and slowly correct the small errors.

The four step modes and their respective applied PWM signal frequencies used are determined by iterative testing and tuning. The step modes decided upon are *microstep* (1/32 step), *sixteenthstep* (1/16 step), *eighthstep* (1/8 step) and *quarterstep* (1/4 step). For every frame where an error is non zero, and the tracker is detecting a face, the direction and step mode are set for that unique error.

4.3.5 Function design

i. Computer vision function design

Because the Haar-cascade method of detection needs to be applied at multiple points in the system, this algorithm must be implemented in the form of a Python function which is created and named *look*. This function is designed to be called in the following way:

`(x,y,w,h)=look(image)`

The function takes the current image frame as its input parameter. Within the function, a Haar-cascade face classifier is implemented, and once a face is detected, the function returns the coordinates (in terms of a pixel location using 'x' and 'y' coordinates) of the bottom left vertex of the rectangle defining the bounding box around the face. The width, 'w', and height, 'h', of the bounding box are also returned, allowing for a complete description of the ROI within the current image frame. This returned bounding box is used to either initialize the tracker for the first time, or reinitialize the tracker if the tracked subject has been absent from the image frame after a certain amount of time.

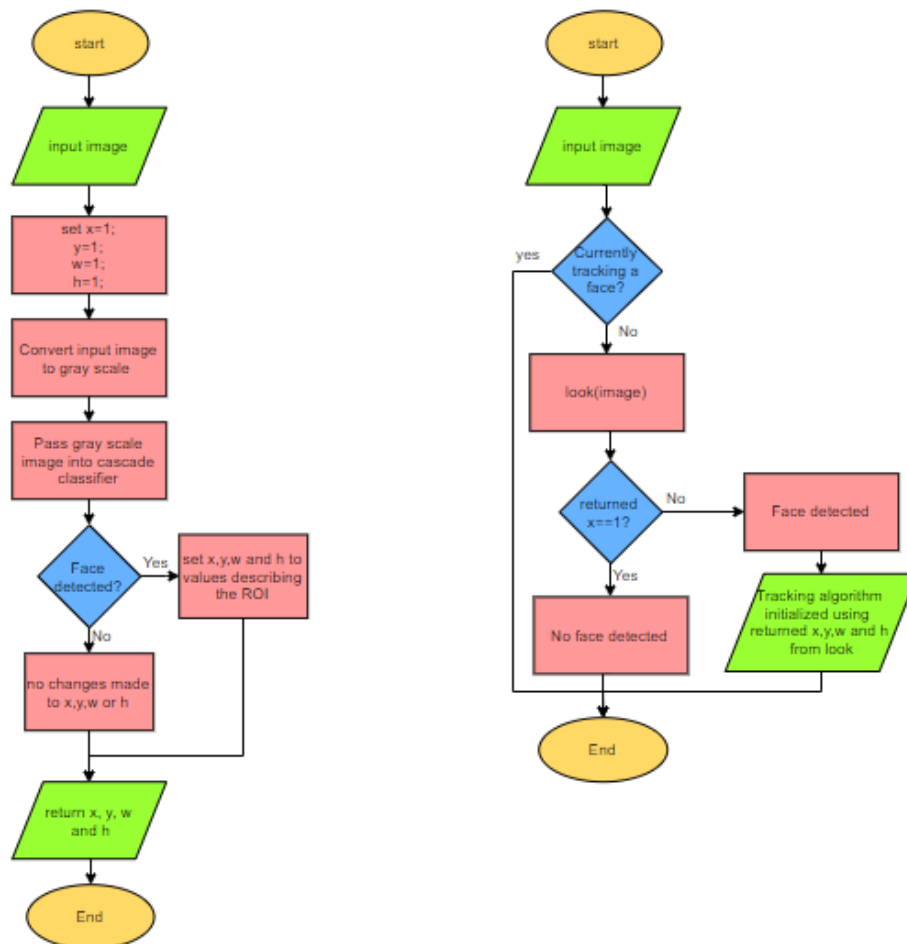


Figure 26: Flowchart describing look (left) and its usage (right); Haar cascades require a black and white input image, which is why the input image must be converted to grayscale. The flowchart on the right describes lines 3 and 4 of the flowchart (figure 24) seen above in a more technical sense, by describing the use of variables within the code.

ii. Motor control function design

Because the changing between step modes needs to happen at multiple points in the code, and requires repetitive lines of code to instantiate the required GPIO ports for each step mode, Python functions must be created to handle each step mode change, and are called as follows:

```
frequency=fullstep()
```

Here the calling of the full step function can be seen. The step mode is set by outputting the respective logic signal corresponding to each mode to the M0-2 pins on the DRV8825. These functions perform two tasks: set the relevant step mode using GPIO ports, and return the PWM frequency used for that step mode. These specific PWM frequencies can be determined through the manual tuning of the frequency parameter until the greatest speed for that step mode is found. Tuning is necessary because if the frequency were to be too high, there would be too much current drawn from the power supply used, causing the supply to trip and interrupt the control of the system, and if it were to be too low, the rotational speed of the motor would be suboptimal.

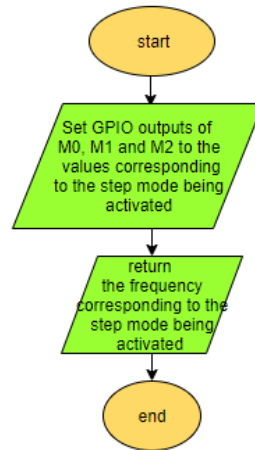


Figure 27: Flow chart for a stepmode function; these functions take in no input, and return the frequency required for that specific step mode.

Another function must be created to choose which step mode is to be applied for a given error, and then activate that step mode. Therefore, a step mode function, as described above, will act as operators to be called from within this new function. The function is called choosestep, and it is called as follow:

`choosestep(e)`

Here, the input parameter 'e' is the absolute value of the current error. This function does not return a value, as its task is to set the required step mode for a given error, which is done through the instantiation of GPIO ports, rather than the outputting of values. When called, this function sets the PWM frequency depending on the current error. The frequency used to set the PWM signal is returned from the step mode function called. The corresponding step mode to the frequency has already been activated by this point, as the step mode GPIO ports are activated when their related step mode function is called from within choosestep.

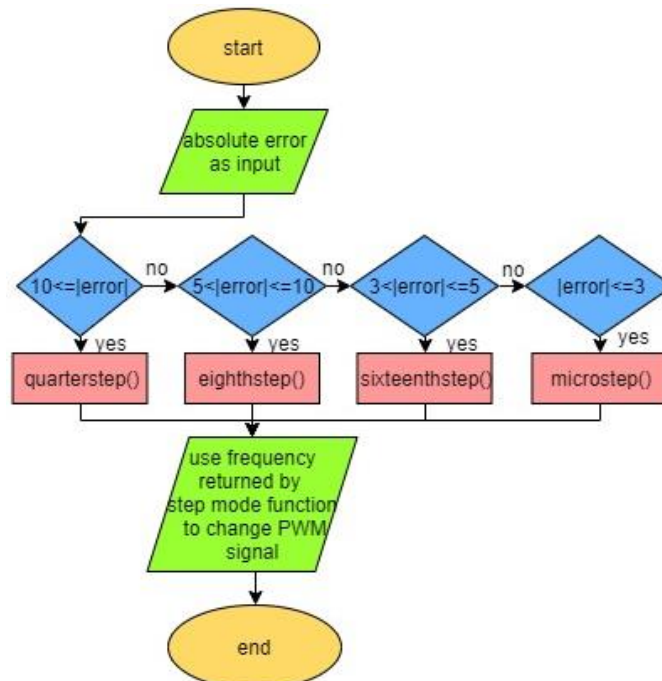


Figure 28: Flow diagram of the choostep function; In the flow diagram above (Figure 24), the parameters 'a', 'b' and 'c' are tuned to the bounds around errors with magnitudes of 3, 5 and 10 respectively.

In the Python code itself, the entire process of controlling the stepper motor is achieved in the following three lines, which make use of the functions described above:

```
elif((error)<0):  
    GPIO.output(DIR,UP)  
    choosestep(abs(error))
```

The first line here uses the sign of the current error to determine the required vertical direction of motion, which is then set in the second line using the DIR pin of the motor driver. The final line of code uses the choosestep function described above to set the PWM signal, and determine and set the step mode required for the current error. By running these lines, the motion of the motor has been completely defined.

From the system algorithm described above, it is seen in line 5 that when the error is 0, no control action is taken. In code, this is achieved by setting the duty cycle of the PWM signal to 0% when the error is between -0.5 and 0.5 pixels.

5. Testing

The process of testing and comparing the tracking algorithms deliberated between is described here. Additionally, the methods by which the system is tested in order to meet the ATPs described above are defined in this section.

5.1 Testing environment setup

An environment in which repeatable system testing can occur is essential to the testing process. When comparing the tracking algorithms performance, it is important that the performance of each algorithm is compared under identical conditions. A testing environment must be designed to allow for repeatable simulations to be run on the system. The results of these simulations can then be compared between the different algorithms in order to determine which tracker achieves the best performance. Additionally, the performance of the system when the motor is running and when it is off can be compared in order to see the improvements to camera stability that the designed system can achieve. This testing environment will also be used in confirming the ATPs defined in the methodology section of this report.

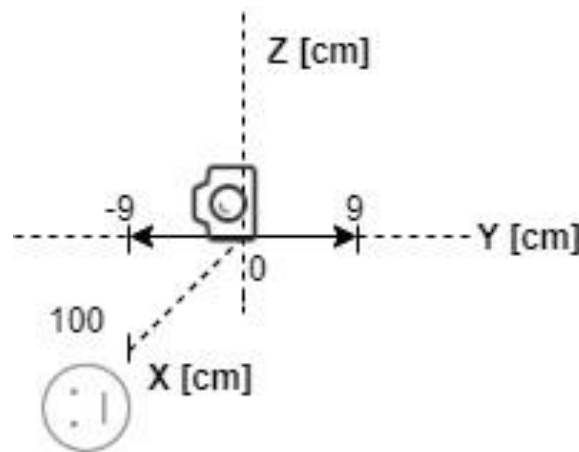


Figure 29: The testing environment; the system is placed on its side, with a printed image of a face placed adjacent to it, one meter away. In this way, when the system moves in the horizontal 'Y' axis, linear vertical motion is simulated.

The system is placed on a camera slider which allows for linear motion in horizontal space. For the testing of the system, the slider can be moved left and right by 9cm (a limit on the system as a result of the rack length) to simulate vertical motion. There is a constant light source on the face, allowing for repeatable testing.

This environment does not provide perfectly repeatable testing, as the moving of the camera slider providing the linear motion is induced manually by hand, which results in movement speeds which are not perfectly repeatable. Because of this, each test must be run multiple times, which results in a data set to be analysed. The proceeding tests must be performed ten times in order to generate valid results to be analysed in proceeding sections.

5.2 Test design

The tests to be performed using the environment described above must be designed to represent the kind of real world instability that a camera might suffer from. The following two tests are performed using the testing environment described above in order to determine by what margin the system improves vertical stability. The tests will be run on each tracking algorithm considered, and with the system turned off. In this way, the tracking algorithm with the best performance under these test conditions can be determined, and its improvement to vertical stability can be empirically shown by comparing the results from the final system configuration to those results when the system is turned off.

5.2.1 Stepped output disturbance test

The first test run will simulate a stepped output disturbance. This will be achieved by sharply pushing the camera slider from its centre position (where the face is in centre frame) to the 9cm mark on the horizontal axis. This motion simulates a step disturbance in the output of the system. The broad task at hand is related to output disturbance rejection, so by seeing how the system corrects for a step in this input allows us to analyse the broad performance of the system. This motion could correspond to real world instances including a scenario where a camera man trips holding a camera, or goes over a speedbump while filming from a car window.

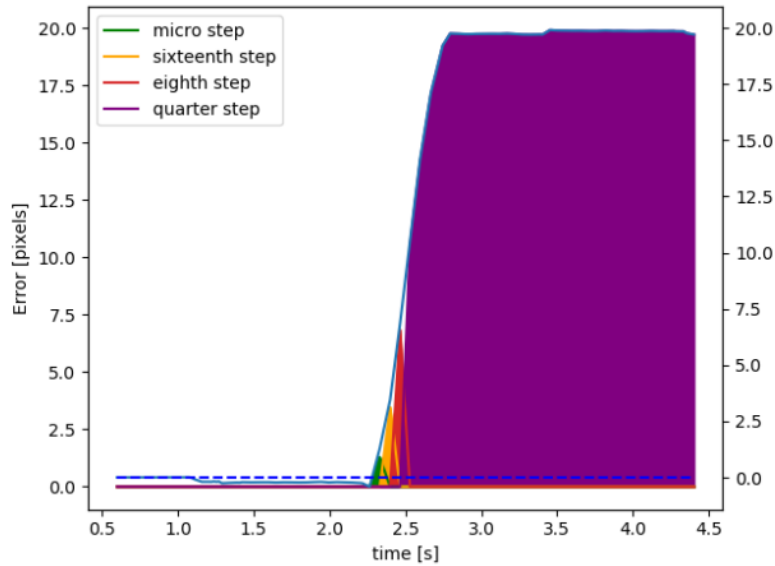


Figure 30: Error vs. Time for the simulation of a step in output disturbance; Seen here, the error is stepped to its maximum value as allowed by the testing environment. The coloured spaces in the graph refer to the step mode which would be activated if the system was running.

5.2.2 Sinusoidal output disturbance test

The next test run will simulate a sinusoidal output disturbance by moving the slider between the -9cm and 9cm mark at a constant speed for a set period of time, in order to induce simple harmonic motion. This tests the system's ability to counteract smooth, sustained motion, as well as its ability to switch the direction in which the rack is actuated. This simulation could test for real world disturbances such as motion from a walking camera man, or other slow sustained vertical disturbances.

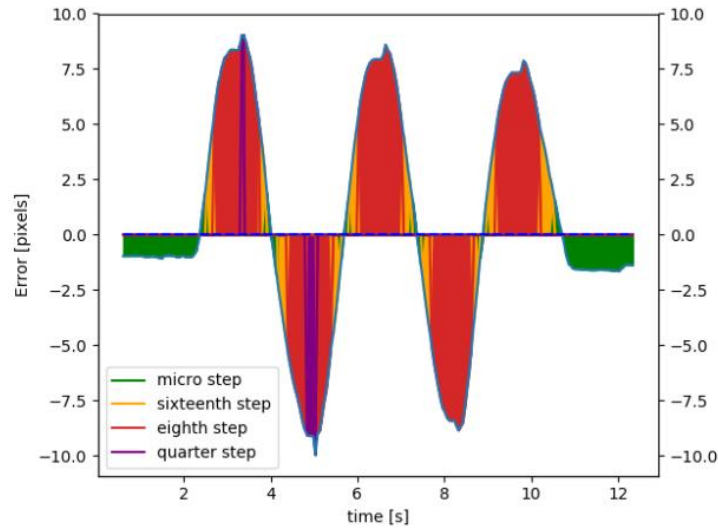


Figure 31: Error vs. Time for the simulation of a sinusoidal output disturbance; the slider is moved left and right (horizontal) to simulate up and down (vertical) motion. The coloured spaces in the graph refer to the step mode which would be activated if the system was running.

5.3 Performance measures

The performance of the system under the tests described above will be measured using ideas of system optimization from control. In this way, the best tracking algorithm for the system can be decided upon by comparing certain error characteristics. While these performance measures are not being directly optimized through testing, they can be used to compare system configurations.

5.3.1 The Integral Squared Error (ISE)

This value increases with larger errors rather than smaller ones. By optimizing this value, the system will achieve faster elimination of large errors, while accepting small errors. The optimization of this performance characteristic results in faster responses, but allows for small oscillatory errors. The optimization of this performance characteristic would result in a real world system that would prejudice fast camera movements, like in the scenario where a camera man would trip holding the camera, or go over a speed bump while filming from a car, but allow for smaller oscillatory errors from disturbances present while a camera man is walking and filming at the same time. These small oscillatory errors could also come in the form of hand shake. The ISE is then an appropriate metric by which to measure the step test mentioned above, and when optimized, will produce a system robust to large, instantaneous errors.

$$ISE = \int \varepsilon^2 dt$$

5.3.2 The Integral Absolute Error (IAE)

Found by integrating the absolute error of the system over time, the optimization of this performance characteristic achieves systems with less sustained oscillations. This is essential to the problem at hand, as sustained oscillations in the error would result in jitters in the camera image. This metric is also appropriate for the comparison of the systems performance under oscillatory motion from real world disturbances present in scenarios where a camera man is walking while filming. This metric is therefore appropriate for determining the systems performance under the sinusoid test.

$$IAE = \int |\varepsilon| dt$$

5.3.3 Integral Time-weighted Absolute Error (ITAE)

This performance characteristic produces optimized systems with fast settling times by adding a time dependant weight to the IAE. This metric weights errors which exist over a larger time period more than errors occurring at the start of the test. Because the sinusoidal actuation of the camera slider is applied manually, the ITAE is not an applicable performance metric for that test. This is because the final ITAE value could drastically change depending on the inconsistent and non-repeatable frequency and amplitude of the sinusoid, especially as time progresses throughout a test run. When the system is optimized for this system, it will have fast error correction, making it ideal in determining the different configurations respective performances under the step test. This is essential in finding if the system has met the ATPs set out in the methodology section.

$$ITAE = \int t|\epsilon| dt$$

5.3.4 Tracking Failures

As mentioned above, the tests are to be run using a printed image of a face to contribute to the repeatability of the experiment. The centre of this image is to be marked with a small cross. For each test being run, a video recording of the results is generated. Within this recording, the centre of the face as it is perceived by the tracking algorithm used, is enclosed in by a red circle generated using OpenCV. Every time the cross physically marked on the image moves out of the circle determined by the tracker, it is considered a tracking failure. Additionally, every time the tracking algorithm fails to successfully track the face (meaning the tracker fails to locate the face within the image), it is also considered a tracking failure. In this way, the accuracy of the different algorithms can be compared, as well as the robustness of each algorithm to the disturbance generated by the motion of the rack, which could potentially lead to camera movements too fast for the tracking algorithms to continuously track the face.

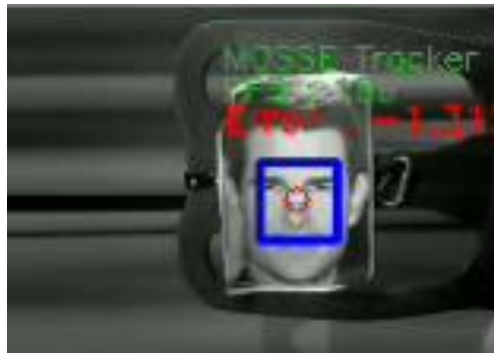


Figure 32: A screenshot from a video recording used to determine the number of tracking failures in a test; this low resolution image provides an example of successful tracking. The small cross on a white piece of tape (seen on the nose of the face) represents the centre of the face, and the fact that it exists within the red circle reveals that the system is tracking the face accurately.

5.4 ATPs

Each acceptance condition from the ATP acceptance condition discussed in the table found in the methodology section is to be assigned a specific test procedure which will confirm whether that requirement has been met.

ATP ID	ATP description	Test procedure
ATP_00	A stable response with no tracking error about the set-point	This condition can be confirmed by observing the error vs. time graphs from the stepped output disturbance tests, as the

		correction to the step must result in the stable tracking of the set point (with zero error).
ATP_01	The tracking error must be zero when the face is in the centre of the image	This acceptance condition can be judged by observing the videos recorded from each test to see if the cross drawn at the centre of the face is in the middle of the image frame when the system detects a tracking error of zero.
ATP_02	The system must be hand held	This is judged by running seeing if the system can operate as required when held by a user.
ATP_03	The system must not fail to generate a bounding box around the face in the image frame for more than one second at a time	This ATP is determined to be passed if no failures occurred for more than one second throughout the testing process
ATP_04	The system must be self-initializing	This test is passed if a bounding box can be set around a face in the image frame within one second
ATP_05	The system must run at real time	This test is passed if an image can be processed at a speed above 60 frames per second.
ATP_06	Step Output disturbances must be corrected quickly	Using the step test, this ATP is passed if a step output disturbance can be corrected to produce an error of zero with a settling time less than one second
ATP_07	The systems direction of motion must be able to change instantly	This ATP is judged using the sinusoid test to determine if there are any undesired responses when the systems direction of motion is switched

6. Results

Using the testing methods and parameters described in the proceeding section, the following sets of results were generated. A complete set of the data generated during the testing process can be found by following a link provided in the appendix.

6.1 Stepped output disturbance test results

The stepped output disturbance test described above was run ten times on the system, and the following sets of data regarding the relevant performance parameters were generated.



Figure 33: Comparison of the results from the step test between when the system on and when it is disabled; these graphs show that when the system is on, the maximum error reached is smaller than when it is off. The comparison also reveals the improvement to the error correcting capabilities of the system when it is on.

6.1.1 ISE data for stepped output disturbance test

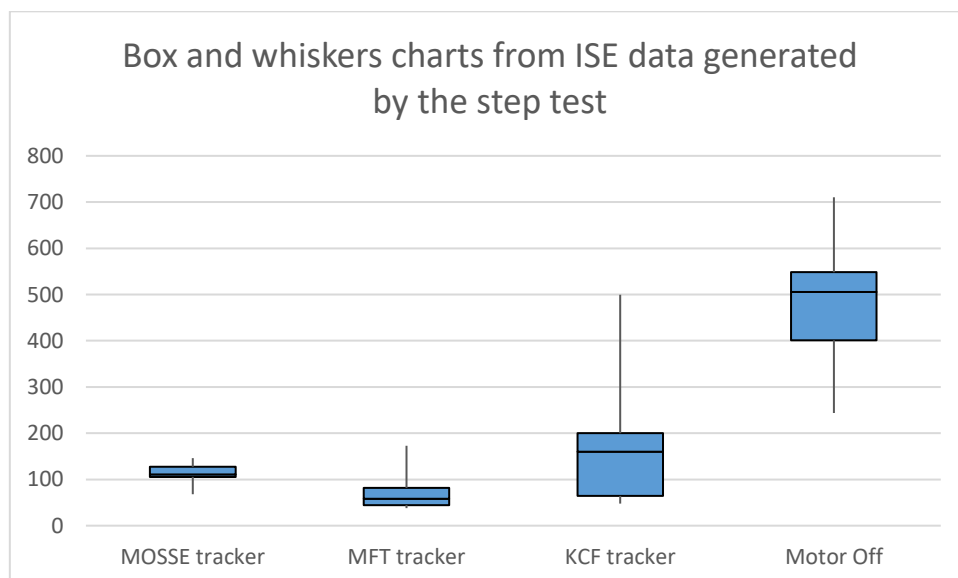


Figure 34: Box and whiskers charts from ISE data generated by the step test

Figure 34 depicts the integral squared error data from each tracking algorithm when the step test is applied to the system, as well as the special case where the system is turned off. The Median Flow tracker has the lowest median value out of the configurations, the lowest minimum value, and the second lowest maximum value after the MOSSE tracker. The MFT also has the second smallest inter-quartile range, after the MOSSE tracker. The KCF tracker has the largest ISE median value out of the tracking algorithms, as well as a significantly high maximum value and inter-quartile range. It is notable that all three tracking algorithms have generate data sets of smaller numerical value than the case where the system is turned off.

6.1.2 IAE data for stepped output disturbance test

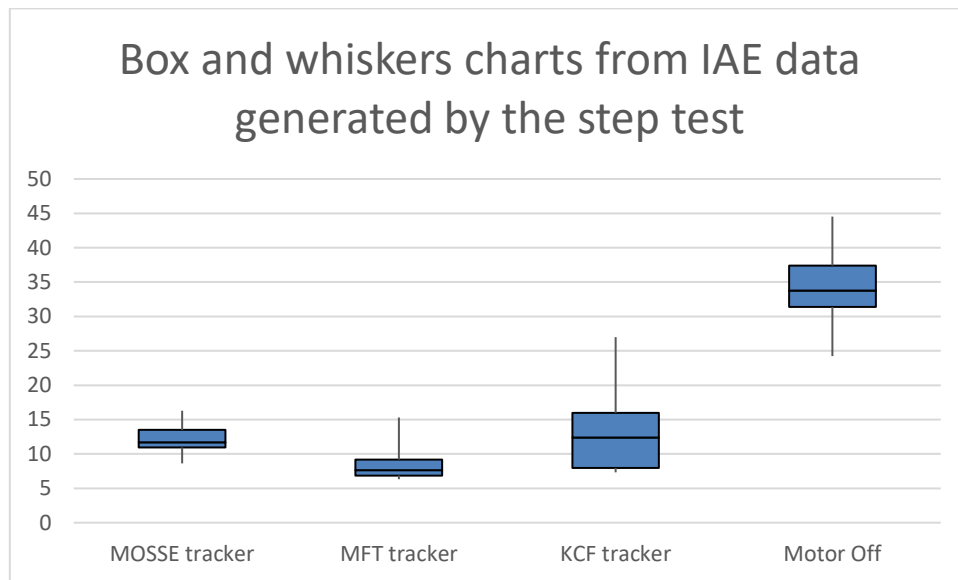


Figure 35: Box and whiskers charts from IAE data generated by the step test

This figure shows the performance of the system under the step test when using the integral absolute error performance characteristic. Like for the ISE, the MFT tracker has the lowest median value, followed by the MOSSE tracker, and then the KCF tracker. This order is consistent for the maximum value observed, and the tightness of the data spread within the interquartile range across tests. Again, much higher values are generated in the case where the system is turned off.

6.1.3 ITAE data for stepped output disturbance

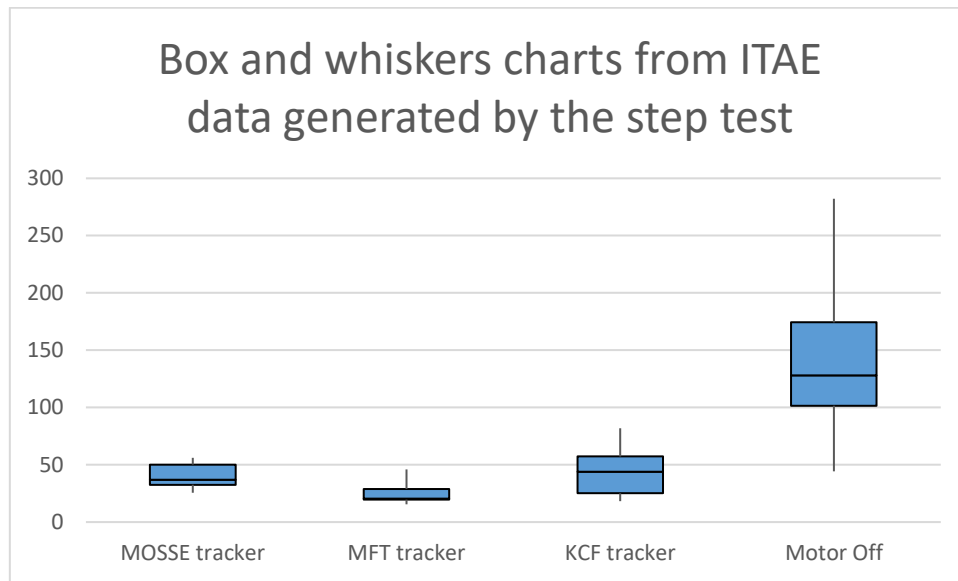


Figure 36: Box and whiskers charts from ITAE data generated by the step test

Once again, when observing the data generated between the trackers, the MFT has the lowest median value, lowest minimum and maximum values, and the most compact interquartile range out of the tracking algorithms. Additionally, these values are all significantly lower when the system is turned on, regardless of which tracking algorithm is being considered.

6.1.4 Tracking failure data for stepped output disturbance

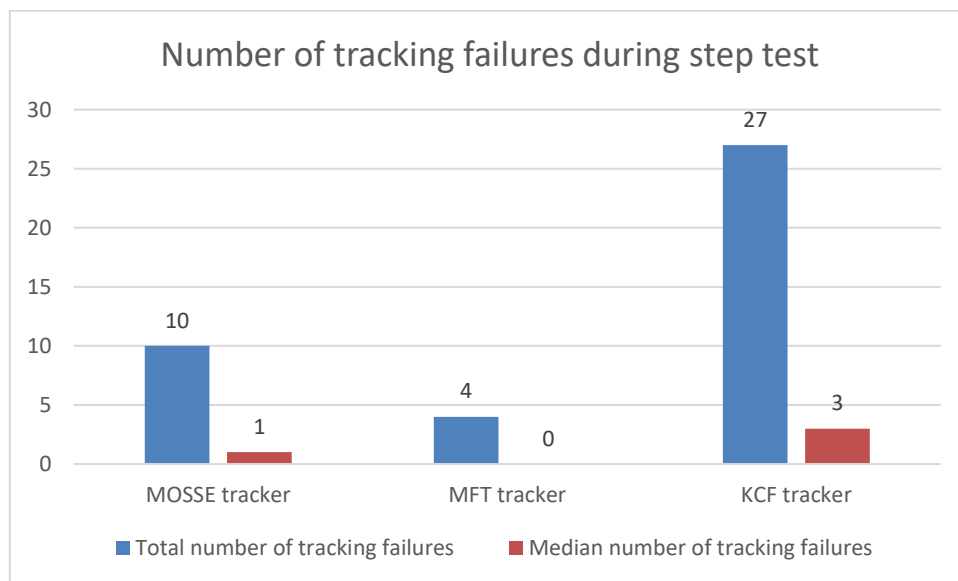


Figure 37: Number of tracking failures during step test

Figure 37 reveals how many tracking failures occurred throughout the testing of the systems response to a stepped output disturbance, as well as the median number of failures per test. The KCF tracker was prone to significantly more tracking failures than the other two tracking algorithms. The MFT tracker generated the least failures across runs, followed by the MOSSE tracker.

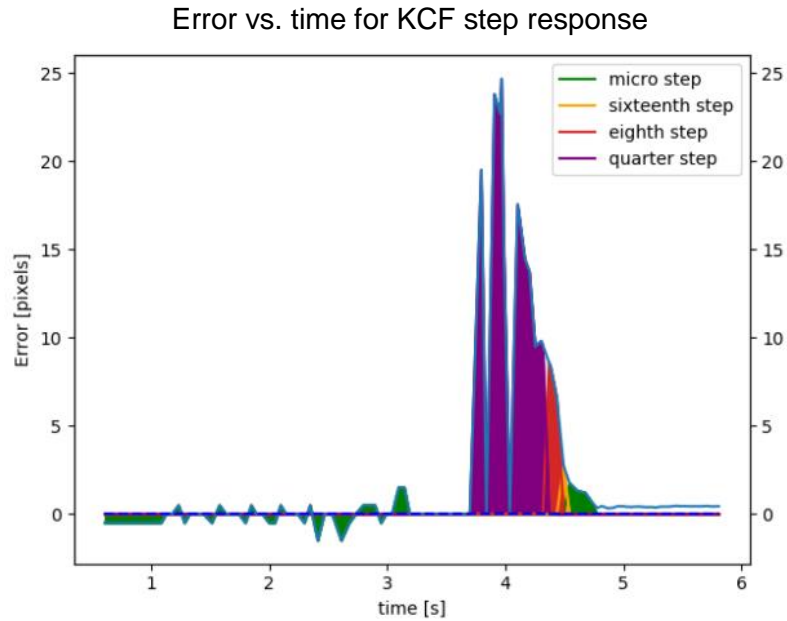


Figure 38: Error vs. time for KCF step response; tracking failures can be seen in the error graphs generated by each test when the error jumps to zero. This shows that no motor action was taken, as the face is not being successfully tracked at these points in time. In this test, there were two tracking failures.

6.2 Sinusoidal output disturbance test results

Once the stepped output disturbance test were run, the sinusoidal output disturbance test described above was run ten times on the system, and the following sets of data regarding the relevant performance parameters were generated. As explained above, the ITAE is not an appropriate performance measure for the sinusoidal test.

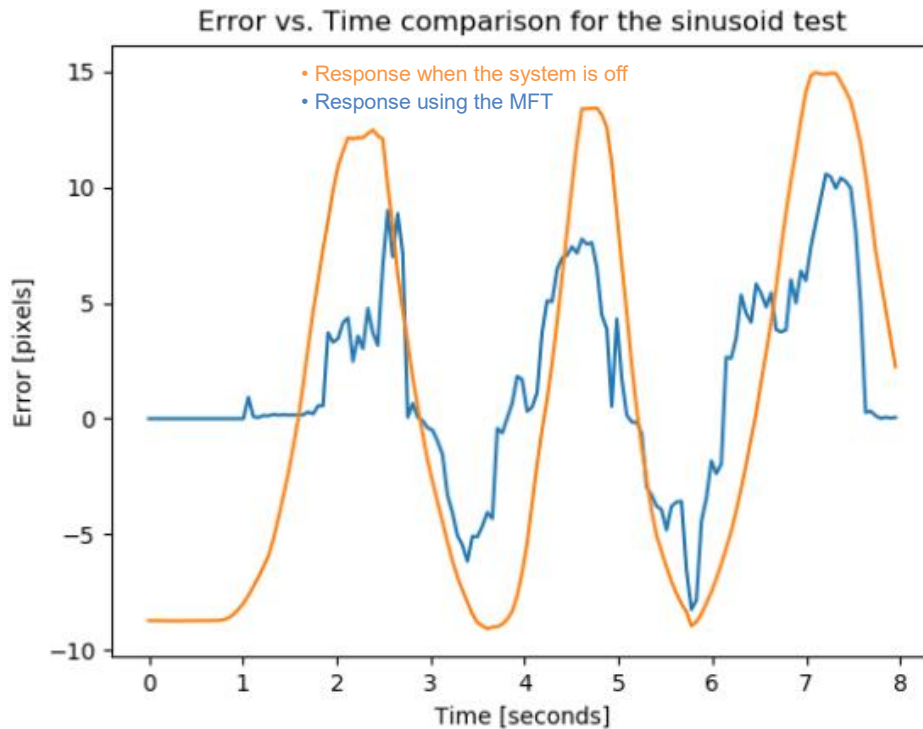


Figure 39: Comparison of the results from the sinusoid test between when the system on and when it is disabled; these graphs show that when the system is on, the maximum error reached is smaller than when it is off. It also shows faster error correction when the system is on.

6.2.1 ISE data for sinusoidal output disturbance test

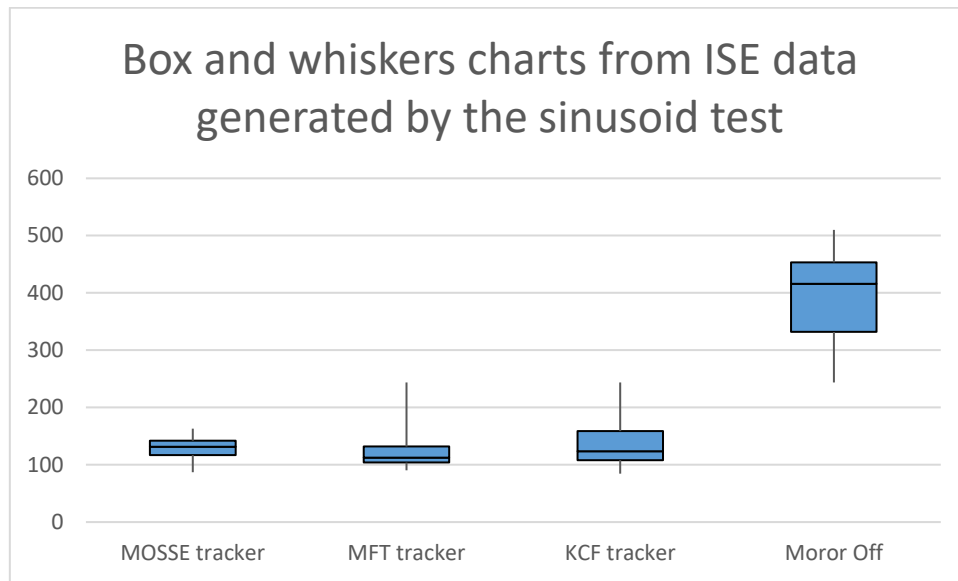


Figure 40: Box and whiskers charts from ISE data generated by the sinusoid test

All the trackers generate similar minimum values and medians when generating ISE data under the sinusoid test. The median of the MFT is the lowest, followed closely by the KCF tracker and then the MOSSE. Maximum values of the ISE data is very similar for the KCF and MFT trackers, with the MOSSE tracker achieving the lowest value. The values from the tracking algorithms were all significantly lower than those from the case where the system is turned off.

6.2.2 IAE data for stepped output disturbance test

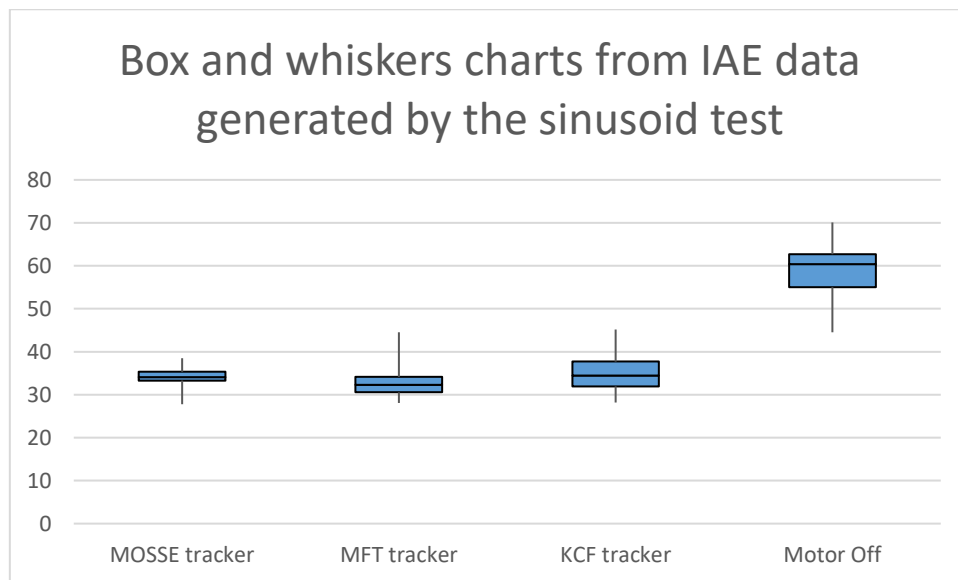


Figure 41: Box and whiskers charts from IAE data generated by the sinusoid test

As with the previous case, the IAE data is relatively similar across the different trackers when tested using a sinusoidal output disturbance. The minimum values and median values of the trackers are very similar, although the MFT does achieve the smallest IAE median value. The MOSSE tracker achieves a significantly compact interquartile range. All three of the tracking algorithms achieve data sets with lower values than when the test is run with the motor turned off.

6.2.3 Tracking failure data for sinusoidal output disturbance

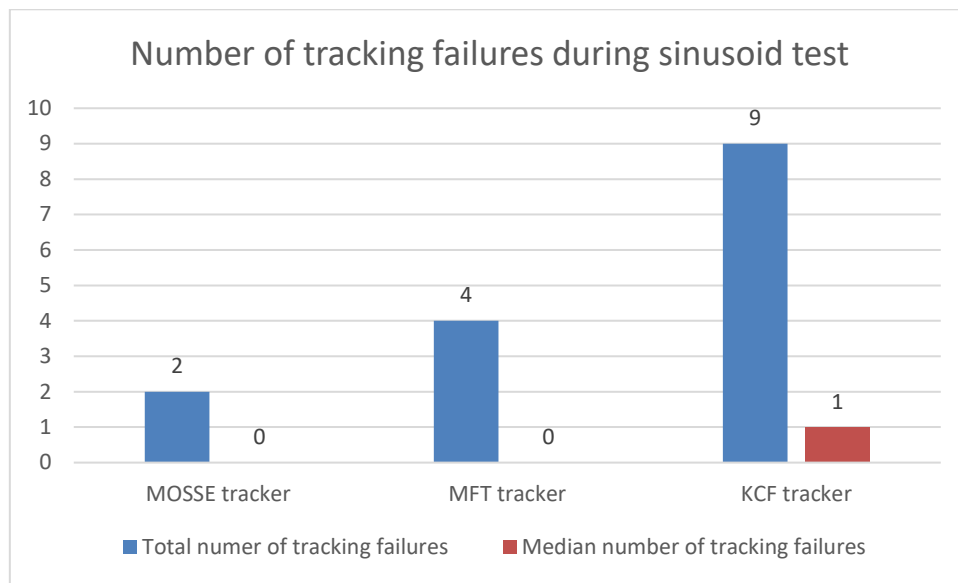


Figure 42: Number of tracking failures during sinusoid test

Figure 42 shows that both the MOSSE and MFT trackers achieve a median value of 0 tracking failures under the sinusoidal output disturbance test, with the MOSSE tracker scoring the fewest total tracking failures. Like with the stepped output disturbance test, the KCF tracker had significantly more tracking failures across the testing process.

6.3 ATP test results

ATP_00 is tested by observing the error vs. time graph for the step test. In figure 43 it can be seen that after a step, the system returns to tracking the setpoint with a stable error value of 0.

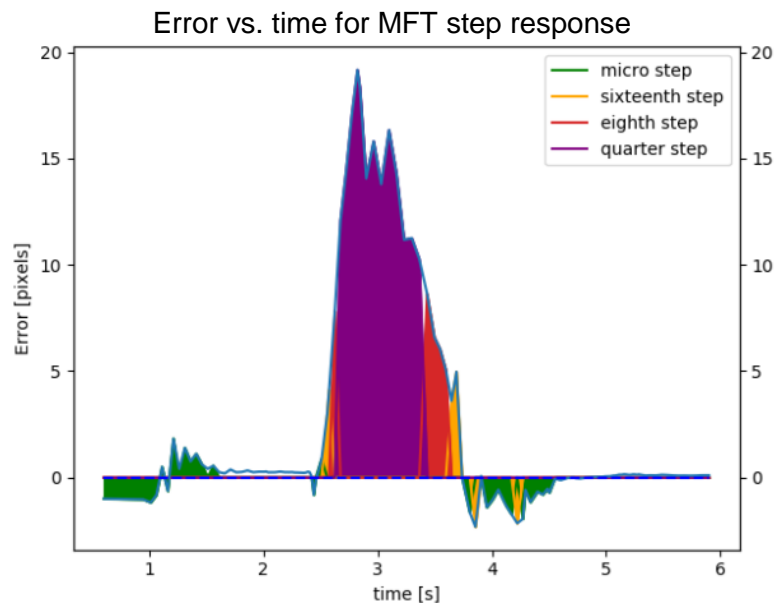


Figure 43: Error vs. time for the step test using the MFT tracker

ATP_01 is judged by viewing the video recordings from various tests. These recordings confirmed that the error the system detects is zero (or acceptably close to zero) when the face is at the centre of the image frame, as can be seen from the screenshot in figure 44.



Figure 44: A screenshot from a video recording; This image reveals that when the face is in the centre of the image frame, the error is close to zero (here it is 0.193)

ATP_02 is judged simply by observing the systems performance when it is run whilst being held in a user's hand. The system performed acceptably when initialized and held facing a user, while being held in hand.

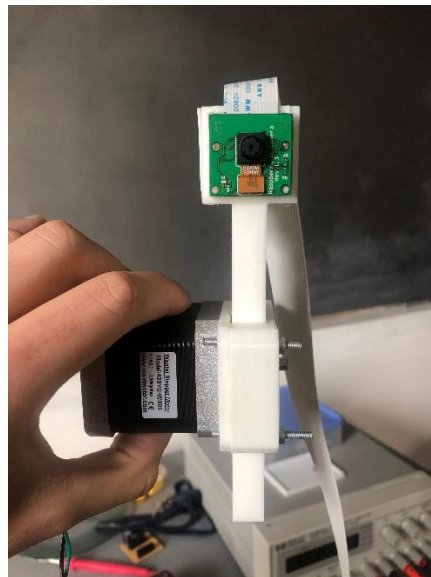


Figure 45: The system being run as a hand-held device

ATP_03 was performed throughout the testing process. Throughout the process, a tracking failure never occurred for more than 0.5 seconds.

ATP_04 was tested by powering up the system and timing how long it takes for the system to self-initialize an initial bounding box. This took no more than 0.4 seconds after running the software and powering up the system.

ATP_05 was tested by viewing the video recordings from the tests. Throughout all the tracking process, the slowest image processing time observed was 100fps by the MFT.

ATP_06 was tested using the step test. From this test, the settling time was measured from the peak of the output disturbance until when the error returns to a value of zero. From the various tests run, the fastest recovery was within 0.4 seconds by the MFT tracker (seen in figure 46), and the slowest recovery was 1.1 seconds by the KCF tracker.

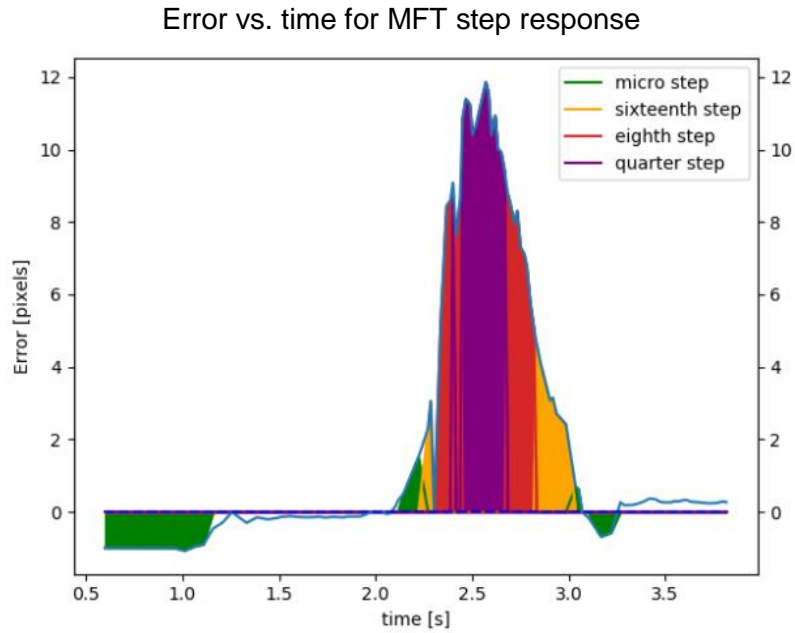


Figure 46: Error vs. time for a MFT step test; here the settling time, as taken from the peak to when the error is zero, is 0.4 seconds.

Finally, ATP_07 was tested with the sinusoid test. From this test, it was seen that smooth transitions occurred throughout the testing process when the direction of motion changed, as is seen in figure 47.

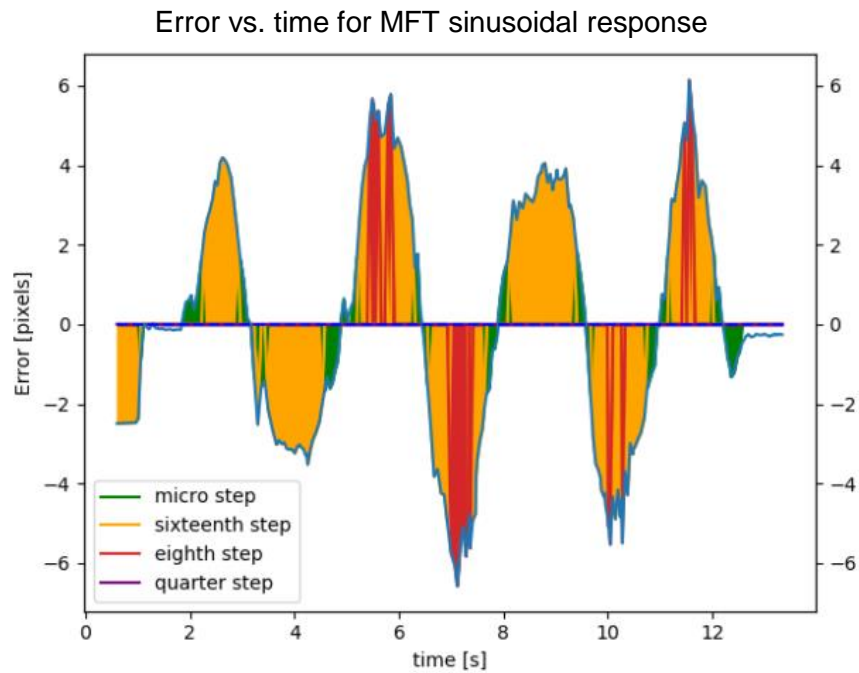


Figure 47: Error vs. time for a MFT sinusoidal test; Here, like in all the sinusoidal tests, it can be seen that changing the direction of the racks motion (it is moving up when the error is negative and down when the error is positive) occurs seamlessly with no undesired behaviours.

7. Discussion

By referring to the direct comparison of the system response when the system is off and when it is on, it is seen that the maximum error reached is lower in both the step and sinusoid tests. Additionally, both reveal faster error correction. This implies broadly that the system successfully improves the stability of the camera under both tests. This implication is expanded upon by analysing the more detailed aspects of the data provided in the results section.

For the error metrics used to evaluate the testing procedures described above, the system configuration which has the lowest values for these performance measures is considered to be the optimal one. This is because these error values are higher if there are more errors present, and as such, the optimal configuration is the one with the fewest errors, as judged by each metric. From the step test, in terms of the IAE, the ITAE and the ISE, the Median Flow tracker performed the best, as it achieved the lowest median value per run by these three metrics, followed by the MOSSE tracker, and then the KCF tracker. These rankings were consistent for all three of these error metrics. The MFT also achieved the tightest interquartile range for the IAE and the ITAE, and was narrowly beaten by the MOSSE tracker in terms of the tightness of this range for the ISE. Having a tight interquartile range is a sign of consistency, so for the step test the MFT performed with the most consistent results in terms of the IAE and the ITAE. The MFT also achieved the lowest median number of tracking failures per test from the step test, meaning it is also the most robust of the trackers to this kind of test. With the MFT performing the best under all four of the error metrics used to judge the performance of the system under the step test, it is determined that this tracking algorithm is the optimal choice under these test conditions.

For the sinusoidal test, the MFT performed best in terms of the IAE and ISE once again, followed by the MOSSE tracker and then the KCF tracker. The MOSSE and MFT both achieved a median tracking failure count of zero, although the MOSSE did suffer from more total failures. The KCF performed the worst by this metric. Unlike the step test, the results for the three tracking algorithms in terms of the IAE and ISE were very similar under this test, with each algorithm achieving similar median values and interquartile ranges. This implies that the testing environment of the step test is a harsher one on the system, as the sub-optimal algorithms performed significantly worse than the MFT, whereas for the sinusoid test the results were widely similar. To add to this point, there were significantly more tracking failures during the step test than the sinusoidal test for each system configuration. This reveals that the system that has been designed performs better under slower movement than fast steps in disturbance. Similar as the results may be, the Median Flow tracker still performed the best out of the three algorithms, and since it is the optimal algorithm under both test environments, the system configuration which uses the MFT tracker is the final and optimal system design.

The ATP test results confirm that every ATP set for the system was met when using the MFT tracker. If a successful design is measured purely by these metrics, it can be said that the system designed in this report successfully achieves the goal of achieving vertical image stabilization.

A notable aspect of the results generated by running the sinusoidal and step tests is that each of the tracking algorithms performs significantly better by every performance metric than the case when the motor is turned off. This means that the system improves the camera stability no matter which tracking algorithm is used. This points towards success in achieving the goal of vertical stabilization, as the system designed in this report produces results with smaller error values than an un-stabilized camera recording. The MFT achieves a median value no more than half the value of the median when the system is turned off across all the error metrics. This is a significant improvement.

The control loop and algorithmic approach taken in using a stepper motor to actuate a camera in order to keep that camera in line with a body could be used as a template for future research in the field of camera stability. This design blueprint of using computer vision and actuators to keep a subject at the centre of an image frame as a method of achieving stability could be applied to achieve other types of stability, such as in the horizontal plane or along rotational axis. To achieve this, the only adjustment required would be to the line of motion provided by the actuation testing, and would require subsequent tuning and testing. This report has proven that this broad method can work, by applying it to one kind of directional instability. The ideas seen here can therefore be applied to other projects aiming to achieve stability in a specific dimension, using computer vision techniques.

The novel method of using computer vision and an actuator to achieve directional stability could also be applied as a supplement to other camera stability approaches. A system like the one seen in this report could be integrated with a camera gimbal in order to glean the advantages and benefits of both devices. This contribution could lead to wide research and application.

The design presented here is therefore more of a proof of an idea than a marketable invention. For a product to be developed using the device presented here, the rig of the system and materials used would have to be reworked in order to create a self-contained system, which could also house a higher quality video camera, as the Pi camera is configured with an extremely low resolution. The design produced here could therefore be up-scaled and made portable in order to make it more widely applicable for consumer use.

8. Conclusions

This report has detailed the design and implementation of a system conceived to achieve vertical stability for a video camera using computer vision techniques. A detailed description of the methodologies and design procedures followed to create this device was included so that every design decision made throughout this project can be accounted for.

With the system configured as described throughout this report, and using the Median Flow tracker which was determined to be the optimum tracker for the needs of this project, a device has been designed which successfully produces improvements to the vertical stability of a camera in terms of every metric used in the testing of the system. This has been proven in the discussion and analysis sections of this report.

Although this report has generated a novel solution to solve the problem of vertical stability, the design produced is not one which is widely applicable, as it works only for the Raspberry Pi camera module. This means that it would not work with most other cameras, as the camera used must be able to interface with the Raspberry Pi. Additionally, although the system is hand held, the wires connecting the camera to the Raspberry Pi and the stepper motor to the motor driver limit the mobility of the device. This means that the system could not actually be used to correct many of the real life vertical stability errors that a camera man might face.

If the portability problems of the system from the wires could be solved, the system seen here could be integrated into a larger rig, which uses the Raspberry Pi camera as a secondary, lower quality camera to a high quality video recording camera housed in the same rig. In this way, the Raspberry Pi camera could be used in achieving vertical stability for the rig as a whole, while the other camera could be used purely for filming. This implementation would greatly enhance the utility of the system designed in this report.

Although the system does not have wide use, its novelty and improvements to vertical stability imply that the project developed here was a success within the scope it was designed for, and could provide a blueprint for future work in the field of camera stability.

9. Recommendations

There were some design and implementation options which were ignored throughout the development of this project due to the scope and limitations present in its development. Some potential design alternatives and recommendations can be seen here:

1. By designing a rig which can house both the raspberry pi itself along with a power supply for the motor and raspberry pi, the system could be made completely portable, as the range limit from wiring would be removed. This would make the system designed here more widely applicable.
2. Design a camera rig which also houses a high quality video camera in order to determine the improvements this system offers to a high quality video camera.
3. It would be interesting to see how the system would perform if it was run using YOLO, as this detection algorithm can offer excellent real time performance and accuracy.
4. Only pre-trained models were used throughout this report. It would be interesting to compare the results if the detection algorithm was deployed using a classifier trained by the author.
5. It would be interesting to compare the results if classical control techniques were employed to control the stepper motor.
6. Use a voltage source with a high current output. The voltage source used in this report limited the current available to the stepper motor, which potentially disadvantaged the final results of the project.
7. Implement the system designed here with a normal camera gimbal. In this way, the system seen here would account for z-axis stability while the gimbal accounted for rotational stability.
8. See how the system performed if made to stabilize the image around an object other than a face. The broadening of this system characteristic would make the system designed here more widely applicable.

10. References

- [1] Y.-Y. Shih, S.-F. Su and I. Rudas, "Fuzzy based hand-shake compensation for image stabilization," in *2012 International Conference on System Science and Engineering (ICSSE)*, Dalian, 2012.
- [2] K. Sato, S. Ishizuka, A. Nikami and M. Sato, "Control techniques for optical image stabilizing system," *IEEE Transactions on Consumer Electronics*, vol. 39, no. 3, pp. 461-466, 1993.
- [3] D. Sachs, S. Nasiri and D. Goehl, "Image stabilization technology overview," *InvenSense Whitepaper*, 2006.
- [4] N. A. Tsofigkas, S. Xalkiadis, D. Xu and I. French, "A guide to digital image stabilization procedure — An overview," in *2011 18th International Conference on Systems, Signals and Image Processing*, Sarajevo, 2011.
- [5] R. Chereau and T. P. Breckon, "Robust motion filtering as an enabler to video stabilization for a tele-operated mobile robot.," *Electro-Optical Remote Sensing, Photonic Technologies, and Applications VII; and Military Applications in Hyperspectral Imaging and High Spatial Resolution Sensing*, vol. 8897, p. 88970I, 2013.
- [6] J.-Y. Chang, W.-F. Hu, M.-H. Cheng and B.-S. Chang, "Digital image translational and rotational motion stabilization using optical flow technique," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 1, pp. 108-115, 2002.
- [7] J. Dong and H. Liu, "Video Stabilization for Strict Real-Time Applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 4, pp. 716-724, 2017.
- [8] Raspberry Pi , "Raspberry Pi Camera Module," 28 November 2018. [Online]. Available: <https://www.raspberrypi.org/documentation/raspbian/applications/camera.md>. [Accessed 1 August 2019].
- [9] Get More Shots, "How Does Gimbal Work in a Camera Stabilizer?," [Online]. Available: <http://getmoreshots.com/how-does-gimbal-work-in-a-camera-stabilizer/>. [Accessed 1 August 2019].
- [10] L. Zhang, "Quaternion defines 3-dimensional rotation," 17 February 2017. [Online]. Available: <https://blog.csdn.net/linolzhang/article/details/55549136>. [Accessed 1 August 2019].
- [11] Evo Gimbals, "How Does a 3 Axis GoPro or DSLR Gimbal Work?," 20 October 2017. [Online]. Available: <https://www.evogimbals.com/blogs/evo-blog/how-does-a-3-axis-gopro-or-dslr-gimbal-work>. [Accessed 1 August 2019].
- [12] V. Renee, "Jockey Motion is a 4-Axis Gimbal That Can Also Upgrade Your 3-Axis Ronin & AllSteady," 26 September 2015. [Online]. Available: <https://nofilmschool.com/2015/09/jockey-motion-4-axis-gimbal-also-upgrade-3-axis-ronin-allsteady>. [Accessed 1 August 2019].
- [13] R. Y. D. Xu, J. M. Brown, J. M. Traish and D. Dezwa, "A computer vision based camera pedestal's vertical motion control," in *2008 19th International Conference on Pattern Recognition*, Tampa, 2008.
- [14] A. Amanatiadis, A. Gasteratos, S. Papadakis and V. Kaburlasos, *Robot Vision*, IntechOpen, 2010.
- [15] B. Zhang, "Computer vision vs. human vision," in *9th IEEE International Conference on Cognitive Informatics (ICCI'10)*, Beijing, 2010.
- [16] Z. Kalal, K. Mikolajczyk and J. Matas, "Tracking-Learning-Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1409-1422, 2012.

- [17] J. Kang, A. Borkar, A. Yeung, N. Nong, M. Smith and M. Hayes, "Short Wavelength Infrared Face Recognition for Personalization," in *International Conference on Image Processing*, Atlanta, GA, 2006.
- [18] T. Li, W. Hour, F. Lyu, Y. Lei and C. Xiao, "Face Detection based on Depth Information Using Hog-LBP," in *2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, Harbin, 2016.
- [19] J. Brownlee, "Discover Feature Engineering, How to Engineer Features and How to Get Good at It," September 26 2014. [Online]. Available: <https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/>. [Accessed 3 August 2019].
- [20] J. Bigun, *Vision with direction*, Springer, 2006.
- [21] J. Shi and C. Tomasi, "Good Features to Track," Cornell University, 1993.
- [22] F. Tomiyasu, T. Hirayama and K. Mase, "Wide-Range Feature Point Tracking with Corresponding Point Search and Accurate Feature Point Tracking with Mean-Shift," in *2013 2nd IAPR Asian Conference on Pattern Recognition*, Naha, 2013.
- [23] Y. Tsuduki, H. Fujiyoshi and T. Kanade, "Mean Shift-based Point Feature Tracking Using SIFT," *IPSJ Journal*, vol. 49, pp. 35-45, 2007.
- [24] A. Zhang, Z. Lipton, L. Mu and A. Smola, *Dive into Deep Learning*, 2019.
- [25] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, 2016.
- [26] H. D., T. S. and S. S., "Learning to Track at 100 FPS with Deep Regression Networks," *Lecture Notes in Computer Science*, vol. 9905, 2016.
- [27] D. S. Bolme, J. R. Beveridge, B. A. Draper and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Francisco, 2010.
- [28] S. J. D. Prince, "Applications, Face and Pedestrian detection," in *Computer Vision: Models, Learning and Inference*, London, University College London, 2012, pp. 161-167.
- [29] V. Kecman, *Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*, The MIT Press, 2001.
- [30] L. Jacobson, "Introduction to Artificial Neural Networks- Part 1," 5 December 2013. [Online]. Available: <http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7>. [Accessed 28 July 2019].
- [31] M. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015.
- [32] A. Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: <https://cs231n.github.io/convolutional-networks/>. [Accessed 28 July 2019].
- [33] V. Bushaev, "How do we 'train' neural networks?," 27 November 2017. [Online]. Available: <https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73>. [Accessed 28 July 2019].
- [34] A. Krizhevsky, I. Sutskever and G. E. Hilton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, pp. 1097-1105, 2012.
- [35] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks-the ELI5 way," 15 December 2018. [Online]. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. [Accessed 28 July 2019].

- [36] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, 2014.
- [37] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015.
- [38] A. Sachan, "Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN, YOLO, SSD," 2017. [Online]. Available: <https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/>. [Accessed 29 July 2019].
- [39] S. Ren, K. He, R. B. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *CoRR*, 2015.
- [40] D. Strigl, K. Kofler and S. Podlipnig, "Performance and Scalability of GPU-Based Convolutional Neural Networks," in *18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, Pisa, 2010.
- [41] A. Monteiro, M. d. Oliveira, R. d. Oliveira and T. d. Silva, "Embedded application of convolutional neural networks on Raspberry Pi for SHM," *Electronic Letters*, vol. 54, no. 11, pp. 608-682, 2018.
- [42] Koustubh, "Accelerating Convolutional Neural Networks on Raspberry Pi," 2018. [Online]. Available: <https://cv-tricks.com/artificial-intelligence/deep-learning/accelerating-convolutional-neural-networks-on-raspberry-pi/>. [Accessed 29 July 2019].
- [43] D. C. d. Andrade, "A Simple Way to Deploy Any Machine Learning Model," 9 January 2019. [Online]. Available: <https://towardsdatascience.com/a-simple-way-to-deploy-any-machine-learning-model-106d463e9a4b>. [Accessed 29 July 2019].
- [44] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, Paris, 2010.
- [45] Intel, "Intel Distribution of OpenVino Toolkit," [Online]. Available: https://software.intel.com/en-us/openvino-toolkit?source=post_page-----. [Accessed 29 July 2019].
- [46] P. Deka, "AI Accelerator Products," 5 February 2019. [Online]. Available: <https://towardsdatascience.com/ai-accelerator-products-e02cbb698d93>. [Accessed 29 July 2019].
- [47] Intel, "Movidius," [Online]. Available: https://www.movidius.com/?source=post_page-----. [Accessed 29 July 2019].
- [48] M. West, "Adding AI to the Raspberry Pi with the Movidius Neural Compute Stick," 7 June 2019. [Online]. Available: <https://www.bouvet.no/bouvet-deler/adding-ai-to-edge-devices-with-the-movidius-neural-compute-stick>. [Accessed 29 July 2019].
- [49] S. S. Farfade, M. J. Saberian and L.-J. Li, "Multi-view Face Detection Using Deep Convolutional Neural Networks," in *ICMR '15 Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, Shanghai, 2015.
- [50] H. Li, Z. Lin, X. Shen, J. Brandt and G. Hua, "A convolutional neural network cascade for face detection," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, 2015.
- [51] K. Wang, Y. Dong, H. Bai, Y. Zhao and K. Hu, "Use fast R-CNN and cascade structure for face detection," in *2016 Visual Communications and Image Processing (VCIP)*, Chengdu, 2016.
- [52] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, 2017.
- [53] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single Shot Multibox Detector," in *European conference on computer vision*, Springer, 2016.

- [54] A. Wong, M. J. Shafiee, F. Li and B. Chwyl, "Tiny SSD: A Tiny Single-Shot Detection Deep Convolutional Neural Network for Real-Time Embedded Object Detection," in *2018 15th Conference on Computer and Robot Vision (CRV)*, Toronto, 2018.
- [55] M. J. Shafiee, B. Chwyl, F. Li and A. Wong, "Fast YOLO: a fast you only look once system for real-time embedded object detection in video," *arXiv preprint arXiv:1709.05943*, 2017.
- [56] R. Huang, J. Pedoeem and C. Chen, "YOLO-LITE: A Real-Time Object Detection Algorithm Optimized for Non-GPU Computers," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018.
- [57] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
- [58] S. Yang, P. Luo, C. C. Loy and X. Tang, "WIDER FACE: A Face Detection Benchmark," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [59] I. Itzcovich, "Faced: CPU Real Time Face detection using Deep Learning," 26 September 2018. [Online]. Available: <https://towardsdatascience.com/faced-cpu-real-time-face-detection-using-deep-learning-1488681c1602>. [Accessed 31 July 2019].
- [60] P. Viola and M. J. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001.
- [61] C. P. Papageorgiou, M. Oren and T. Poggio, "A General Framework for Object Detection," in *International Conference on Computer Vision*, 1998.
- [62] C.-F. Wang, "What's the Difference Between Haar-Feature Classifiers and Convolutional Neural Networks?," 4 August 2018. [Online]. Available: <https://towardsdatascience.com/whats-the-difference-between-haar-feature-classifiers-and-convolutional-neural-networks-ce6828343aeb>. [Accessed 30 July 2019].
- [63] H. T. e. a. Ngo, "An FPGA-based design of a modular approach for integral images in a real-time face detection system," *Mobile Multimedia/Image Processing, Security, and Applications*, vol. 7351, p. 73510B, 2009.
- [64] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119-139, 1997.
- [65] P. Viola and M. J. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, vol. 2, no. 57, pp. 137-154, 2004.
- [66] M. Cerny and M. Dobrovolny, "Eye tracking system on embedded platform," in *012 International Conference on Applied Electronics*, Pilsen, 2012.
- [67] J. I. Flores-Delgado, L. G. Martínez-Santos, R. Lozano, I. Gonzalez-Hernandez and D. A. Mercado, "Embedded control using monocular vision: Face tracking," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, Miami, 2017.
- [68] u. S., Z. N., Y. Q., W. Y., Y. M. and W. J., "Rotated Haar-Like Features for Face Detection with In-Plane Rotation," *Lecture Notes in Computer Science*, vol. 4270, 2006.
- [69] B. Horn and B. Schunck, "Determining Optical Flow," *Artificial Intelligence*, vol. 17, no. 1-3, pp. 185-203, 1981.
- [70] Z. Kalal, K. Mikolajczyk and J. Matas, "'Forward-Backward Error: Automatic Detection of Tracking Failures," in *Proc. 20th Int'l Conf. Pattern Recognition*, Istanbul, 210.
- [71] B. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proc. Seventh Int'l Joint Conf. Artificial Intelligence*, 1981.
- [72] J. F. Henriques and e. al., "High-speed tracking with kernelized correlation filters," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 3, pp. 583-596, 2014.

- [73] L. Zhang, T. Luo, Y. Sun and L. Yang, "A fast filter tracker against serious occlusion," in *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Shenzhen, 2016.
- [74] Python, "Python 3.7.0," 2018 June 27. [Online]. Available: <https://www.python.org/downloads/release/python-370/>. [Accessed 14 August 2019].
- [75] OpenCV, "About," [Online]. Available: <https://opencv.org/about/>. [Accessed 14 August 2019].
- [76] D. Jones, "picamera," [Online]. Available: <https://picamera.readthedocs.io/en/>. [Accessed 7 August 2019].
- [77] Raspberry Pi, "Raspberry Pi 3 Model B," [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. [Accessed 6 August 2019].
- [78] Raspberry PI, "Camera Module V2," [Online]. Available: <https://www.raspberrypi.org/products/camera-module-v2/>. [Accessed 6 August 2019].
- [79] Micro Robotics, "Stepper Motor 0.5Nm NEMA17 0.9 deg/step," [Online]. Available: <https://www.robotics.org.za/42BYGHM809?search=nema%2017>. [Accessed 13 August 2019].
- [80] J. L. Gibson, "Rack and Pinion Steering, What in the World is it?," 20 November 2014. [Online]. Available: <http://blog.autostarusa.com/rack-and-pinion-steering/>. [Accessed 13 August 2019].
- [81] Pololu, "DRV8825 Stepper Motor Driver Carrier, High Current," [Online]. Available: <https://www.pololu.com/product/2133>. [Accessed 13 August 2019].
- [82] S. Guennouni, A. Ahaitouf and A. Mansouri, "A Comparative Study of Multiple Object Detection Using Haar-Like Feature Selection and Local Binary Patterns in Several Platforms," *Modelling and Simulation in Engineering*, vol. 2015, p. 8, 2015.
- [83] R. Lienhart, "adaboost frontal face detector," 2000. [Online]. Available: https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml. [Accessed 7 September 2019].
- [84] V. Lehtola, H. Huttunen, F. Christophe and T. Mikkonen, "Evaluation of Visual Tracking Algorithms for Embedded Devices," pp. 88-97, 2017.
- [85] Raspberry Pi Foundation, "Raspbian," 10 July 2019. [Online]. Available: <https://www.raspberrypi.org/downloads/raspbian/>. [Accessed 29 August 2019].
- [86] "Win32 Disk Imager," 6 July 2018. [Online]. Available: <https://sourceforge.net/projects/win32diskimager/>. [Accessed 29 August 2019].
- [87] A. Rosebrock, "Install OpenCV 4 on your Raspberry Pi," 26 September 2018. [Online]. Available: <https://www.pyimagesearch.com/2018/09/26/install-opencv-4-on-your-raspberry-pi/>. [Accessed 29 August 2019].
- [88] sennan2d, "New rack actuator for NEMA17," 27 March 2019. [Online]. Available: <https://www.thingiverse.com/thing:3520982>. [Accessed 3 September 2019].
- [89] VGer, "Raspberry pi camera case/enclosure," 21 May 2013. [Online]. Available: <https://www.thingiverse.com/thing:92208>. [Accessed 3 September 2019].
- [90] J. D. Kramer and C. Jacky, "How to write biblos," vol. 1, no. 1, 2006.
- [91] P. Skalski, "Let's code a Neural Netork in Plain NumPy," 12 October 2018. [Online]. Available: <https://towardsdatascience.com/lets-code-a-neural-network-in-plain-numpy-ae7e74410795>. [Accessed 28 July 2019].
- [92] Google Research, "Google AI Open Images- Object Detection Track," 2018. [Online]. Available: <https://www.kaggle.com/c/google-ai-open-images-object-detection-track>. [Accessed 28 July 2019].

- [93] bang good, "DC 12V 50-500mm 900N Stroke Tubular Motor 2/4/8/12/16/20 Inch Linear Actuator Motor - 50mm," [Online]. Available: https://www.banggood.com/DC-12V-50-500mm-900N-Stroke-Tubular-Motor-248121620-Inch-Linear-Actuator-Motor-p-1526731.html?gmcCountry=ZA¤cy=ZAR&createTmp=1&utm_source=googleshopping&utm_medium=cpc_bgs&utm_content=xibei&utm_campaign=pla-mix-za-pc-0617&ad. [Accessed 12 August 2019].

11. Appendix

11.1 The GitHub repository containing the final code design of the project

https://github.com/shaikadish/Vertical_stabilization_project/blob/d4dd47ee1599b6cd96d0f701057a11cd2b165540/test_videov4.py

11.2 The complete data set generated in the testing process, saved to GitHub

https://github.com/shaikadish/Vertical_stabilization_project/blob/d4dd47ee1599b6cd96d0f701057a11cd2b165540/results_thesis.xlsx

11.3 A video showcasing the final system design

<https://www.youtube.com/watch?v=53na2Fg-nFM&feature=youtu.be>