UNIVERSITY OF CAPE TOWN

Department of Mechanical Engineering

RONDEBOSCH, CAPE TOWN
SOUTH AFRICA

PROJECT No:
55

FINAL REPORT 2020

Refine the Digital Shearography system based on a Raspberry Pi PC, a camera and a miniature shearing device

Miguel Allan Garcia Naude, GRCMIG002

Project Brief

Digital Shearography is an optical non-destructive testing technique used to inspect objects for defects. The technique relies on a shearing device, a laser to illuminate the object being tested, a camera to view the object through the shearing device and a PC to store the images captured by the camera. Purpose written software is used to control the system and subtract images from each other in order to produce the resultant fringe patterns.

A prototype low cost inspection system using the credit card sized Raspberry PI PC, a USB based camera and a miniature shearing device has been built last year. This project has two aims:

1. To review the existing optical system and improve the optical shearing geometry in order to maximise the camera field of view.
2. Refine the existing software (either in Python or C) by developing a user-friendly GUI.
3. The GUI will need to:
    - Provides access to and control of the camera variables.
    - Allow the user to set a range of inspection parameters which will be determined between the appointed student and supervisor.
    - Provide a real-time camera view window.
    - Manage the shearographic inspection process and display the fringe pattern results obtained in a suitable visual format.
    - Provide the ability to save the image results obtained.

You will be required to demonstrate your final solution using a set of manufactured samples with introduced defects in order to determine the improvements.

SUPERVISOR: *Mr D Findeis*

Word Count: 16998

Plagiarism 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 significant contribution to, and quotation in, this report / project from the work(s) of other people has been attributed and has been cited and referenced.

3. This report/project 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 his or her own work.

Signature: 

1 Table of Contents

2 List of Figures

3 List of Tables

4 List of Listings

5 List of Equations

6 Nomenclature

$\boldsymbol{I_s}$	Intensity of the Shearogram
$\boldsymbol{I}$	Intensity of reference speckle pattern
$\boldsymbol{I'}$	Intensity of the speckle pattern after loading
$\boldsymbol{B}$	A constant dependent on the laser light amplitude
∅	Phase difference between light waves from two points
Δ∅	Relative phase change resulting from a relative deformation between points
$\frac{\boldsymbol{\partial d}}{\boldsymbol{\partial x}}$	Deformation derivative with respect to shearing direction
$\boldsymbol{\lambda}$	Laser light wavelength
$\boldsymbol{S}$	Shear distance
$\boldsymbol{n}$	Number of fringes
$\boldsymbol{h}$	Width of the camera sensor [mm]
$\boldsymbol{f}$	Focal length of a lens [mm]
$\boldsymbol{FOV}$	Field of View [mm]
$\boldsymbol{WD}$	Working distance [mm]
$\boldsymbol{AFOV}$	Angular Field of View [◦]
$\boldsymbol{a}$	Side length of the cube beam splitter [mm]
$\boldsymbol{d_1}$	Distance between the cube beam splitter and the imaging lens [mm]
$\boldsymbol{d_2}$	Distance between the cube beam splitter and mirror one [mm]
$\boldsymbol{d_3}$	Distance between the cube beam splitter and mirror two [mm]

7 Abstract

Digital Shearography is an optical non-destructive testing technique that is used to inspect objects for defects. The benefits of Non-destructive testing and in particular, Digital Shearography, has resulted in it being useful in maintenance and quality conformance in automotive and aeronautical industries. Most Digital Shearography devices make use of Michelson Interferometers as their shearing device, however devices making use of this shearing device are limited to having a small angular field of view. By incorporating a 4f system into a Michelson Interferometer based inspection device, the angular field of view of the system is no longer limited by the Michelson Interferometer, thus maximising the camera field of view. By developing a user-friendly GUI to interact with the developed inspection device, the Shearographic Inspection process is managed and fringe patterns resulting from real-time subtraction of images using Digital Shearography are displayed. This enables non-destructive tests to be effectively carried out in order to gather object defect information.

8 Acknowledgments

First and foremost, I would like to thank my supervisor, Dirk Findeis, for his contribution to this project. I am extremely grateful for Dirk's knowledge and guidance in the subject of Digital Shearography in particular. Dirk has also been very understanding in what has been an extremely complicated and stressful year throughout the Covid-19 pandemic, which I am also very grateful for.

Secondly, I would like to thank Pierre, Heinrich and all the other staff in the workshop for putting long hours into my project to produce the parts that I needed.

Thirdly, I would like to thank Beverley Glass, who has been ever willing to help me with acquiring parts for my project and sorting out the finances that go along with it.

I would also like to thank my friends, in particular Matthew Lock who, using his experience in software development, has helped me gain a good understanding of many difficult concepts.

Lastly, I would like to thank my girlfriend and my family, my parents and my brother, for their endless support during the last four years of University.

9 Introduction

As complex modern materials that are susceptible to defects, specifically composites, are used increasingly in industries such as the automotive and aeronautical, the need for non-destructive testing (NDT) techniques has enlarged [1]. These techniques can be used to detect for defects, such as cracks and delaminations within objects [2] without causing permanent damage to the original specimen [3]. This is not only important for detecting defects in components before they are put into use but is also suitable for use in the maintenance inspections of parts [4]. One of these techniques, Digital Shearography, involves the analysing of images that exhibit fringe patterns to highlight the presence of defects [2].

9.1 Subject

This report covers the research, detailed design phase and implementation of project 55, Refine the Digital Shearography system based on a Raspberry Pi PC, a camera and a miniature shearing device. The relevant literature pertaining to Digital Shearography and GUI development is covered in the report, as well as the design process with final mechanical designs of the selected concept. The report also includes a description of the Python GUI development process. The commissioning and testing of the device is covered, followed by a discussion and recommendations for future projects. The report is then concluded.

9.2 Problem definition and background

The project is a review of an existing low-cost inspection prototype that was built in 2019. There are areas for improvement in the existing system, specifically the prototype's field of view (FOV) and the existing user interface.

The problem can be defined as designing and implementing improvements of the optical shearing geometry of the system to maximise the camera's FOV and developing a suitable GUI to improve user experience in order to be able to accurately inspect objects for defects by producing fringe patterns.

9.3 Project scope and constraints

Digital shearography may be used to acquire qualitative and quantitative defect information. However, based on the allocated budget and time, and the effects of the pandemic, it will not be possible to implement a system that provides quantitative strain data. Therefore, the only physical changes to be made to the existing prototype are to improve and maximise the camera field of view. The inspection system will also be limited to only detecting defects due to the out-of-plane deformation gradient of the object being inspected.

The GUI being developed will be limited to providing access to and control of system variables, allowing for a range of inspection parameters to be set, providing a real-time camera view window, displaying the fringe pattern results and providing the ability to save image results. This also includes the development of the backend code that the frontend will interact with to bring about changes to the inspection camera.

The brief originally stated the inspection system would be based on a Raspberry Pi. However, the scope has been changed such that the program will be run on a Windows machine, as specified by the supervisor, Mr D Findeis.

9.4 Plan of development

The report begins by introducing the existing low-cost inspection system, which is followed by a detailed literature review. The report then outlines the Product Requirement Specifications (PRS) before focusing on the design process of the physical inspection prototype. This begins with the preliminary conceptual designs from which the final design is developed and presented. The report then details the GUI development process, beginning with a proposed GUI preliminary design before detailing the final design and documenting the programming implementation process. The report then presents a section on the commissioning and testing of the products, after which a discussion of the results and conformance of the products to the PRS is presented. The report then details recommendations for future versions of the project, followed by a conclusion.

10 The Existing Low-Cost Inspection System Prototype

The existing prototype is a Michelson Interferometer-based shearography system comprising of a CCD camera, imaging lens, cube beam splitter, shearing and reference mirrors and a base with a surrounding casing.

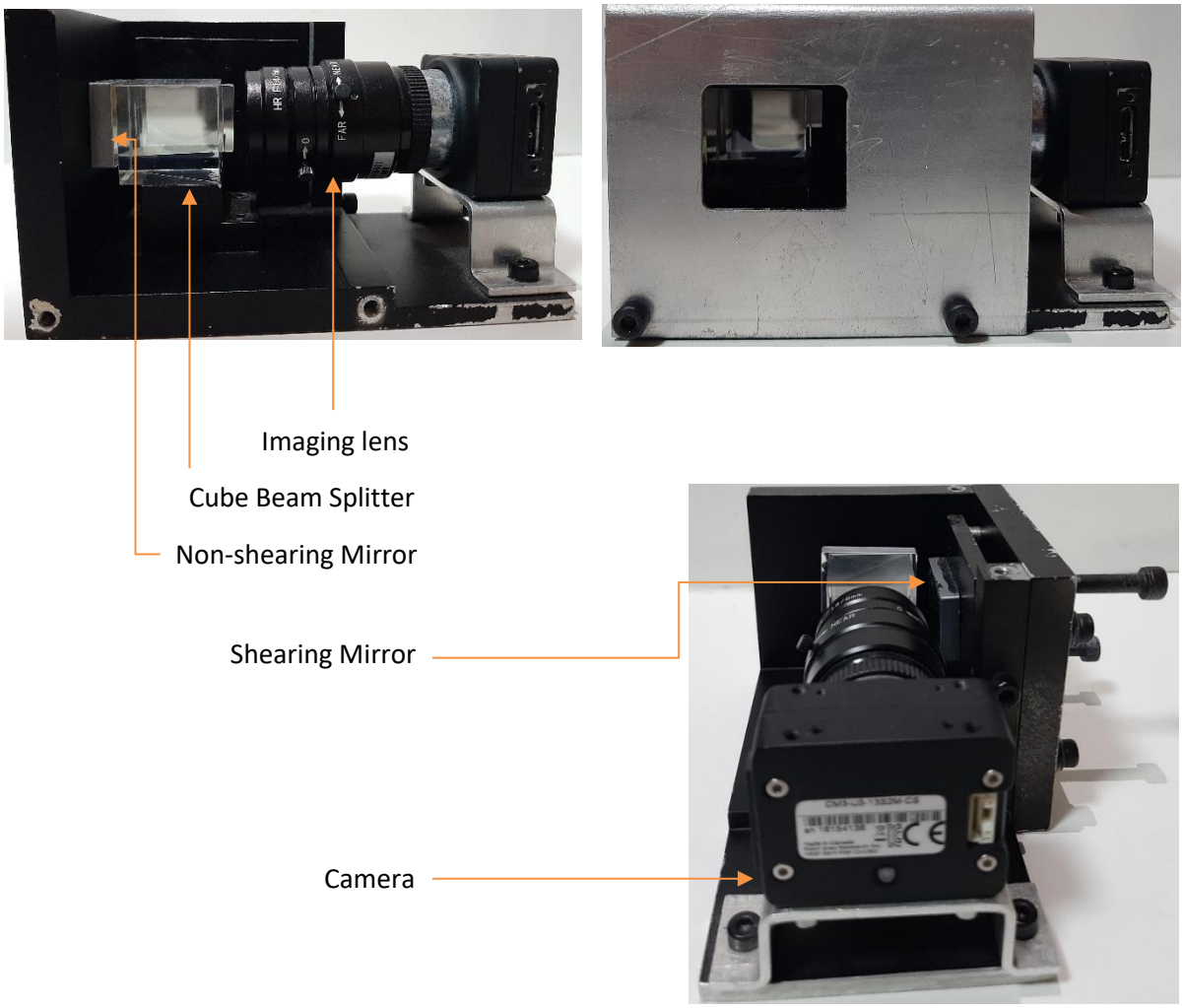


Figure 1: The Existing Inspection Device

Although the current system can be used to inspect objects for defects, it fails to maximise the field of view of the camera and imaging lens, leading to inspections of long duration for large objects. The nature of this design means that the field of view of the existing system is limited by the Michelson Interferometer (cube beam splitter and two mirrors) rather than the imaging lens of the camera. The camera is the Chameleon3 CCD camera equipped with an 8 mm focal length imaging lens. As a result, the camera has an approximate horizontal angular field of view (AFOV) of $33.4°$ and vertical AFOV of $25.4°$, while the AFOV of the Michelson Interferometer is only approximately $21.1°$ (refer to Appendix H).

If the AFOV of the camera and imaging lens is greater than that of the Michelson Interferometer, the image captured by the CCD camera contains both useful inspection data, captured through the Michelson Interferometer, and non-useful data that is as a result of the AFOV of the camera overlapping that of the Interferometer. The regions of useful (denoted by the blue rectangle) and non-useful data (outside of the blue rectangle) are pointed out in the figure below.

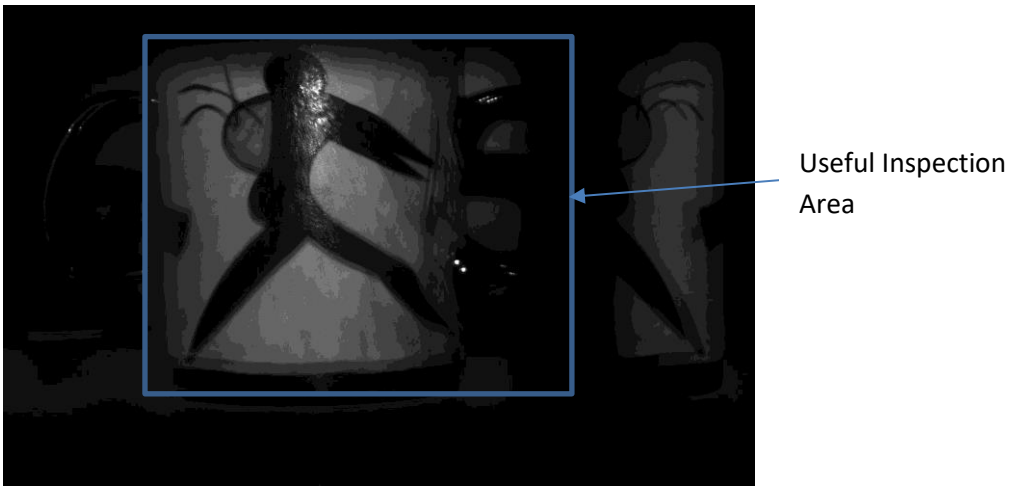


Figure 2: Camera View of a mug through the Existing Inspection Device

The existing software interface is a command line environment which requires a series of input commands. A user that is unacquainted with the software would struggle to operate it as it is complicated and non-user friendly. This existing interface also provides no real-time feedback about the inspection and one is unable to alter any camera properties or variables once the inspection procedure has started.

11 Literature Review

11.1 Digital Shearography

11.1.1 Introduction

The Shearography Non-Destructive Testing (NDT) technique was developed to address the limitations of holography [5]. Unlike Electronic Speckle Pattern Interferometry (ESPI) and holography that directly measure surface displacements, Digital Shearography measures derivatives of surface displacements. Consequently, one does not need to differentiate displacement data to yield strain information [5]. Compared to ESPI and holography, Digital Shearography requires: a simpler optical setup, reduced laser coherent length and reduced vibration isolation, as it is a 'self-referencing' technique that does not require a second reference laser beam [5][6]. It is also insensitive to small rigid body motion, as the motion does not produce strain [1]. These characteristics of Digital Shearography systems make them suitable for use as a portable system in an industrial environment.

Due to their high strength and low-density properties, composite materials such as Carbon Fibre are being used increasingly in aerospace and automotive industry applications. However, due to the complexity of their structure, composite materials have an increased possibility of having defects, particularly delamination, a separation between layers [1]. Composite components, particularly in the aircraft industry, are exposed to high stress in-service conditions, making NDT important in identifying defects before the failure of components to ensure a high safety operation [7]. Other than being used for the inspection of aircraft structures, Shearography is also used for the evaluation of tires in the rubber industry, and the measurement of strains and residual stresses [5].

While Digital Shearography was developed to directly measure quantitative strain values, in industry it has been majorly used for defect detection [8]. If quantitative data is desired, there needs to be direct interpretation of the fringe pattern and calculation of the phase distribution, which would require phase-stepping techniques.

11.1.2 Description of the Digital Shearography Setup

Shearography is a laser-based optical interferometry inspection technique that makes use of a shearing device to introduce image shearing. A modified Michelson Interferometer offers a simple setup and ease in changing shearing direction and magnitude, making it the most commonly used shearing device [1]. It typically consists of a reference mirror, a shearing mirror and a beam splitter. Incorporating this with a monochromatic light source of sufficient coherent length [2], a beam expander, a CCD/CMOS camera with an imaging lens, a computer and a display monitor, a typical shearography setup can be created.

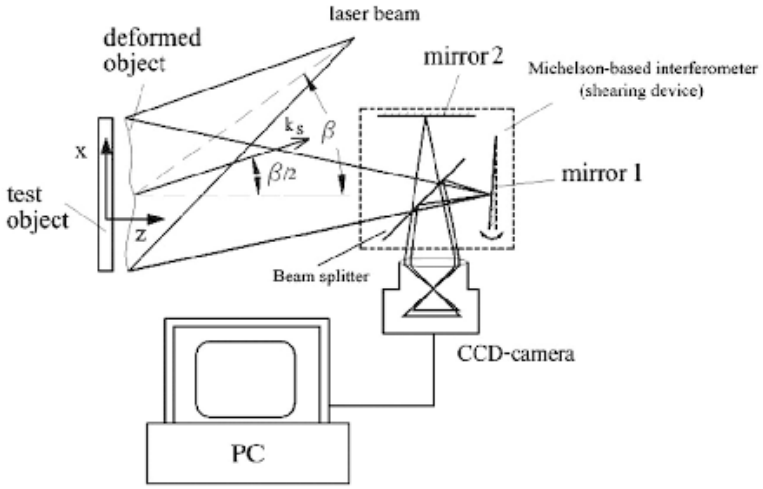


Figure 3: A traditional Michelson Interferometer based Shearography inspection system [9]

Another method to introduce image shear is the use of an optical wedge, although this is uncommonly used as it is more difficult to vary the magnitude of the image shear. The magnitude of the image shear produced is dependent on the distance of the wedge from the object, the angle of the wedge and its refraction rate [7].

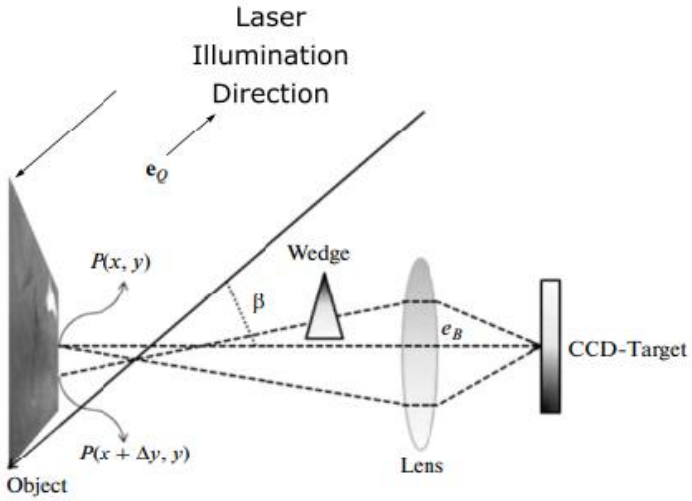


Figure 4: Wedge Prism based Digital Shearography inspection system [10]

Attention needs to be paid to the laser being used for the application. A small, lightweight and well-priced laser module is desired for industrial applications. As a result, lasers that are usually used for such applications, such as He-Ne lasers, are unfavourable as they are large and expensive [11]. Digital Shearography relies on the interference of light waves reflected from two points on an object. Typically, the optical path length difference between these points is very small. As a result, the required laser coherence length is not large and lasers with a coherence length of a few millimetres are often suitable. This means that laser diodes are often used even though they do not have a long coherent length. The small size of these diodes also makes them suitable for portable Digital Shearography prototypes [11]. It is also necessary to consider the heat generated in the laser diode, as it can cause the wavelength of the light emitted to vary over time. The increasing temperature results in the light wavelength changing, making it difficult to achieve the desired fringe patterns. The wavelength therefore needs to be regulated such that it remains consistent over time. Current and/or temperature stabilization of the laser diode power source can be used to achieve this. Normally only one or the other is necessary to regulate the wavelength, as Shearography does not require a long coherent length. However, for applications that require a high performance laser with a longer coherent length, such as ESPI, both current and temperature stabilisation are required [11].

11.1.3 Operation and Fringe Formation

As Figure 3 Illustrates, the laser illuminates the object's surface that is to be inspected. The camera is then used to view the object through the shearing device, which results in the superposition of two images of the object being received. One of these images is displaced horizontally or vertically relative to the other image by tilting the shearing mirror, such that two images of an object are viewed on the display monitor [6]. This results in a region of overlapping in the sheared images, where light reflected from two separate points interfere. This interference results in an intensity based speckle pattern which can be viewed on the computer display monitor after they are digitised [2]. This original speckle pattern is then saved to be used as a reference.

The object is then subjected to a load that results in deformation of the object. Any deformation of the object's surface will result in a relative phase change between the two points mentioned previously [1]. Speckle distribution is given as a function of the phase and intensity of the laser light's reflection off the object's surface and consequently, any deformation of the object's surface will result in the change of the speckle pattern produced in the image plane of the CCD camera [2]. These new speckle patterns are then digitised, saved and compared to the reference image.

If there is direct digital subtraction of the reference image from the image obtained after deformation, a fringe pattern displaying areas of pixel intensity correlation and decorrelation can be produced [1]. These fringe patterns depict the displacement derivative of the object with respect to the direction of applied shear. A real-time fringe pattern can be produced if the reference image is subtracted from the video frames in real-time [5].

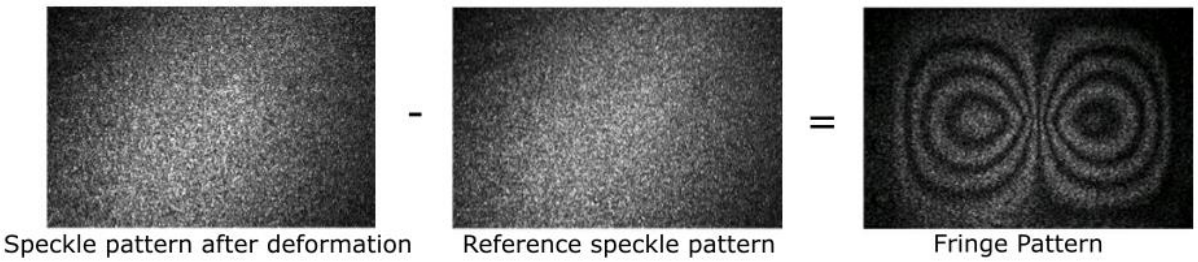


Figure 5: Formation of a fringe pattern [1]

The fringe pattern consists of alternating dark and light fringes. This can be represented by the following equation [1]:

$$|I_s| = |I^{'} - I| = B|\cos(\emptyset + \Delta\emptyset) - \cos\emptyset)|$$

I_s : Intensity of the Shearogram
I : Intenisty of the reference speckle pattern
$I^{'}$: Intensity of the speckle pattern after loading
B : A constant term dependent on the laser light amplitude
$\emptyset$: Phase difference between light waves from two points
$\Delta\emptyset$: Relative phase change resulting from a relative deformation between points

Equation 1: Intensity of a Shearogram

Dark fringes occur in the fringe pattern when $|I_s| = 0$. This occurs when $\Delta\emptyset = 2\pi n\ (n\ \epsilon\ Z)$. Therefore, the relative phase difference between adjacent fringes is 2π.

The relationship between relative phase change and relative deformation between two points can be represented by the following equation [2]:

$$\Delta\emptyset = \frac{4\pi}{\lambda}\frac{\partial d}{\partial x}S$$

$\Delta\emptyset$: Relative phase change resulting from a relative deformation between points

λ : Laser light wavelength

S : Shear distance

$\frac{\partial d}{\partial x}$: deformation derivative with respect to the shearing direction

Equation 2: Relationship between relative phase change and relative deformation

This above equation highlights that the phase change due to the deformation of the object is directly proportional to the shear distance, S. This implies that the measurement sensitivity of Digital Shearography is dependent on the magnitude of the shear induced and can be adjusted by varying it [6]. This equation also highlights that fringes of the same phase represent regions of equal deformation gradient [2].

Knowing that the difference between adjacent fringes is 2π, the magnitude of deformation derivative in the direction of shear is given by:

$$\frac{\partial d}{\partial x} = \frac{n\lambda}{2S}$$

n : Number of fringes

Equation 3: Magnitude of the deformation derivative in the direction of shear

The above equation highlights that an increase in the concentration of fringes corresponds to an increase in displacement gradient for a given region on the object's surface. It also means that digital shearography can directly measure strain data, if the shear distance, laser light wavelength and number of fringes is known.

11.1.4 Fringe Interpretation

As shown above, the number of fringes in an area directly corresponds to the magnitude of the displacement gradient. If a flaw were to exist in a sample being inspected, after being subjected to the load, significantly more deformation would take place around this defect. The strain anomalies that result are detected by the shearographic system. These areas are identified in the shearogram by concentrations of alternating dark and light fringes in a 'butterfly' pattern, this is illustrated in figure 5, demonstrating Digital Shearography's ability to identify locations of defects present in an object.

11.2 Optics

11.2.1 Introduction to Field of View (FOV)

In the context of optical devices, FOV refers to the angle through which electromagnetic radiation enters the device [12].

The field of view of a camera and lens is directly dependent on the Angular Field of View (AFOV) which is the angle of observation associated with the width/height of the image sensor of the camera and the focal length of the imaging lens [13].

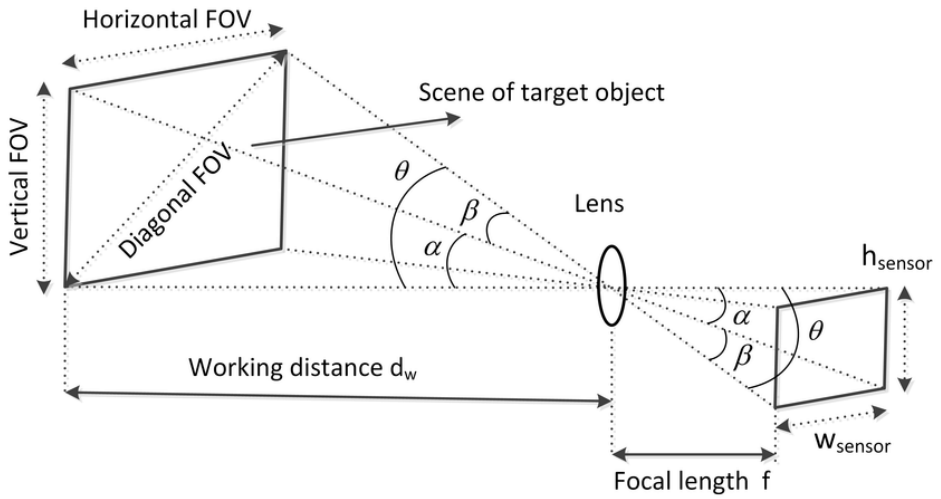


Figure 6: Illustration of Field of View and Angular Field of View

The AFOV can be represented by the following equation [13]:

$$AFOV\ [°] = 2\tan^{-1}\frac{h}{2f}$$

$AFOV$: Angular Field of View [°]

h : Width of the camera sensor [mm]

f : Focal length of the imaging lens [mm]

Equation 4: AFOV of a camera and imaging lens

The FOV of the camera can further be determined by considering the working distance, the distance between the front principal plane of the lens and object plane, as depicted in figure 4. The following equation provides the approximate FOV [13].

$$FOV = 2(WD)\tan\frac{AFOV}{2}$$

FOV : $Field\ of\ View$ $[mm]$

WD : Working Distance [mm]

$AFOV$: Angular Field of View

Equation 5: Field of View of a camera

If a camera were to be used with a fixed focal length lens, the FOV of this system could be increased by increasing the working distance, changing the imaging lens for one with a smaller focal length or the camera can be changed out for one with a larger sensor [13]. Although using a lens of shorter focal length results in a greater FOV, the amount of distortion associated with some short focal length lenses can result in changes to the AFOV with respect to working distance. Shorter focal length lenses also struggle to provide the same high level of performance of lenses with longer focal lengths.

A camera may also be operated with a zoom or vari-focal lens. These lenses allow for their focal length to be altered, allowing them to have a variable angular field of view. However, vari-focal/zoom lenses are more expensive, are often larger in size and cannot offer the same level of performance as fixed focal length lenses [13].

11.2.2 Field of View and Digital Shearography

Given that the existing prototype is Michelson Interferometer-based which is the most widely used shearing device, this section will focus on the field of view of a Michelson Interferometer-based Digital Shearography system.

In traditional Michelson Interferometer-based shearography, the Field of View of the system is dependent on both the Michelson Interferometer and the camera/imaging lens combination, but the AFOV of the Camera is independent of the AFOV of the Interferometer. Hence it is beneficial to have the AFOV of the camera equal to the AFOV of the interferometer such that the AFOV of one does not limit the AFOV of the entire system. The AFOV of the entire system is limited by the AFOV of the Interferometer [14] which is given by the equation below. The Interferometer variables in the equation are illustrated in figure 7 [14]:

$$AFOV = 2\tan^{-1}\frac{a}{2(2a + d_1 + d_2 + d_3)}$$

a : Side length of the cube beam splitter [mm]

d_1 : distance between the cube beam splitter and the imaging the imaging lens [mm]

d_2 : distance between the cube beam splitter and mirror one [mm]

d_3 : distance between the cube beam splitter and mirror two [mm]

Equation 6: AFOV of a Michelson Interferometer

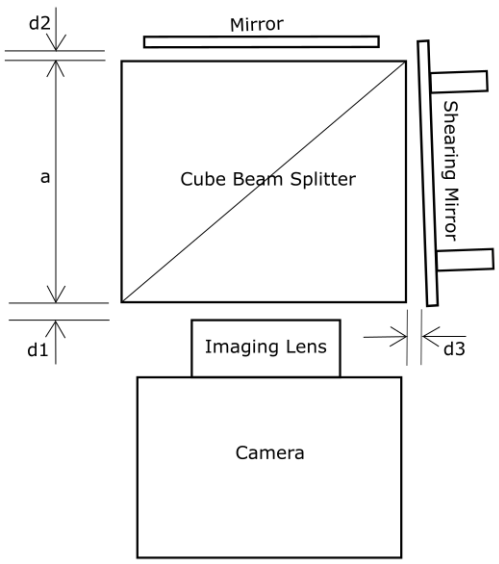


Figure 7: Illustration of the physical parameters influencing the FOV of a traditional Michelson Interferometer[14]

The equation above shows that the maximum obtainable AFOV of the interferometer is 28 degrees if the spacing between the cube and the mirrors/imaging lens is reduced to zero. Consequently, unless an imaging lens with a sufficiently long focal length is used, the AFOV of the interferometer and the camera will be unequal. This limits the efficiency of the shearography if the object being inspected is large and the available working distance is short [14].

The maximum AFOV of the shearography system may be increased by making use of a 4f System. Unlike a traditional Michelson Interferometer-based shearography system, a 4f System separates the imaging lens from the camera and the image-shearing Michelson Interferometer is placed in the optical path between the two. Consequently, the AFOV of the shearography system is only dependent on the imaging lens/camera combination and can be represented by the traditional AFOV equation for a camera and imaging lens [14]:

$$AFOV_{4f}\ [°] = 2\tan^{-1}\frac{h}{2f}$$

$AFOV_{4f}$: AFOV of a 4f system

h : Width of the camera sensor

f : Focal Length of the imaging lens

Equation 7: AFOV of a 4f system

As illustrated in Figure 8 below, the 4f System incorporates two additional lenses with the same focus length. The one lens is placed between the imaging lens and the Interferometer and the other between the Interferometer and the Camera, with the total optical path length between the imaging lens and the camera being equal to four times the focus length of the lenses [14].

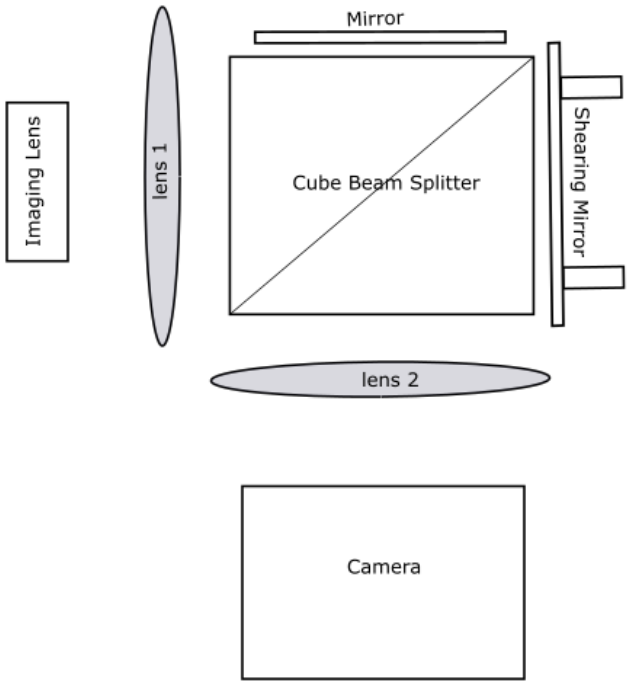


Figure 8: The optical arrangement of a Michelson Interferometer incorporating a 4f System [14]

The lenses are placed such that the mirrors in the Interferometer are at the focus plane of the two lenses, this is referred to as the aperture or Fourier plane. The imaging plane of the imaging lens, the input plane, and the sensor of the camera, the detector/image plane, are at the other focus planes of lens one and lens two respectively [14]. The first lens performs the Fourier transform to the image on

the input plane, which is obtained at the aperture plane. The second lens then applies the inverse Fourier transform to the image on the aperture plane, resulting in the image on the detector plane. The result is a 180◦ rotation of the image from the input plane with the image size remaining the same.

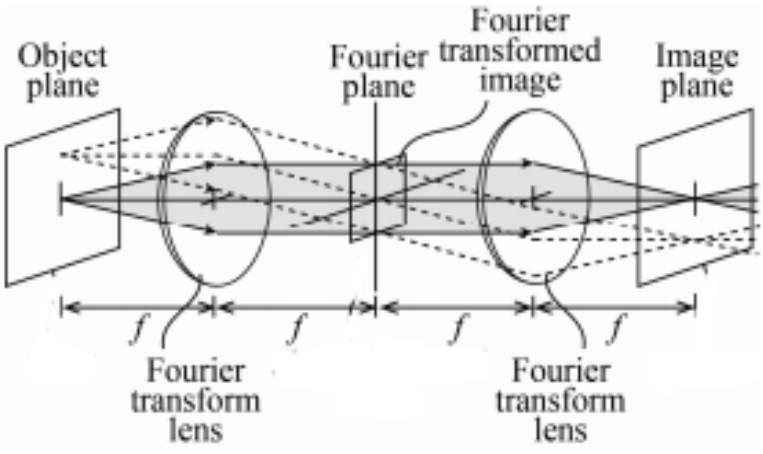


Figure 9: 4f Optical System and relevant planes [15]

While using a 4f System ensures that the Interferometer no longer limits the AFOV of the system, it also has some disadvantages. It results in the image being flipped both horizontally and vertically, decreases image quality and changes the image intensity distribution. The image direction can be rectified with software, while the quality of the image may be improved by using high-quality Fourier lenses [14]. The change in the image intensity distribution results from the mirrors acting as special filters in the 4f system, causing in an uneven intensity distribution at the detector plane, while the image at the input is uniform. This, in turn, results in a decrease in brightness of the image detected by the camera sensor [14]. More even intensity distribution can be acquired by using mirrors that are larger than the input image and a camera with a larger dynamic range [14].

11.3 Programming and Graphical User Interface (GUI) Design

11.3.1 GUI Design

The GUI is an important component of any software application. Most developers adhere to user interface (UI) design principles to ensure that a successful interface design is a more likely outcome [16].

Consistency should be maintained throughout the UI. Hence the visual style and the ways in which users interact with the interface and the use of terminology should remain consistent to avoid confusion [16].

The UI should aim to reduce the short-term cognitive load by minimising the number of actions required to complete a task and by avoiding too much information being presented at one time. Headings and the grouping of similar items should also be used to help users find what they are looking for [16].

The user should be made to feel in control of the application by implementing features such as permitting the easy reversal of actions, providing informative feedback about the status of the system and prompts that inform a user of a completed process [17].

The UI should cater to a diverse range of users. It should not sacrifice the appeal to experts with an overly simplistic interface, but it should provide new users with explanations, examples and tutorials that help them familiarise themselves with product [16][17]. Potential users with disabilities should also be considered and hence the way in which information is conveyed needs to be addressed. Errors should be managed by accurately informing the user as to why it occurred and how the problem may be solved [16].

11.3.2 Application Development and GUI Design Patterns

The following section focuses on UI design patterns, specifically the model-view-controller and its associated patterns which are widely used in application development.

The model-view-controller (MVC) pattern functions by developing the UI frontend (view and controller) independently from the application backend (model). The model subsystem is responsible for maintaining application-specific data and providing access to the data and consists of a set of public functions that are used to achieve the application functionality. The view is responsible for displaying application data to the user, while the controller handles interactions with the user and notifies view of changes in the model [18].

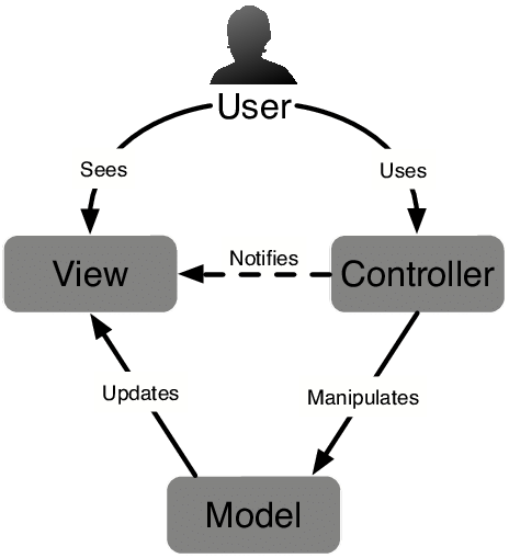


Figure 10: A traditional MVC pattern [19]

Benefits of using an MVC pattern include: The separation of the responsibilities of the system as each subsystem is responsible for a single feature. The view and the model can be changed independently as they are separated from one another, which also means that multiple views can simultaneously share the same model and models [20].

However, in practice the MVC pattern has often has problems with separating the view from the controller. It is unclear whether some components of the system belong in the view or the controller as some of these components may need to be displayed by the view as well as manipulated/modified by the controller [20]. Because of input and output being interconnected in GUIs, the MVC pattern has often been modified.

The model-view pattern is a pattern in which the view and controller are combined into a single class. In GUI development, these classes are referred to as widgets. Most GUI toolkits contain widgets that use the view and controller as a single class [20].

Another adaption of the traditional MVC pattern uses the controller as a mediator between the model and view rather than an input handler. In this pattern, the view is responsible for both output and low-level input, while the controller sits between the model and view passing changes between the two [20]. This is illustrated in the figure below.

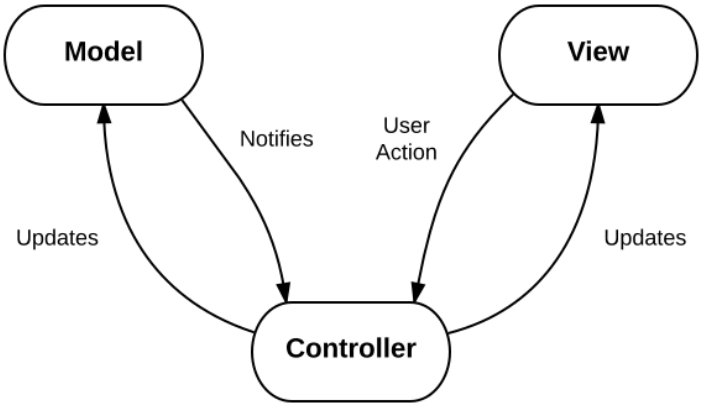


Figure 11: A modified MVC pattern [21]

11.3.3 Languages and Associated GUI Development Toolkits

The existing software is to be refined in Python or C/C++ by developing a GUI. Each of these languages offer various GUI frameworks for both cross-platform and platform specific interfaces. The most popular and established toolkits are explored below.

Table 1 exploring popular cross-platform GUI frameworks available for Python and C/C++

Toolkit	Language	Notes
Qt	C++ (built-in) Python (PyQt)	This toolkit includes a GUI library that uses native style application programming interfaces to render widgets that are consistent with the appearance of the platform [22]. Qt also has an IDE (QtCreator) that generates code for the layout of applications [23].
Tk	C (built-in) Python (TkInter)	A relatively simple GUI toolkit [22]. Has a large number of resources as well as a large developer community available to assist with a GUI development [23].
GTK+	C (built-in) Python (PyGObject)	Provides python bindings to GTK view objects such as windows and widgets. GTK+ 3 can make use of an interface designer called Glade that allows for easy creation of dynamic and responsive GUIs [23].

11.3.4 Camera and Image Processing Python Libraries

11.3.4.1 Image Subtraction

As mentioned in a previous section, in order to obtain the fringe patterns, there needs to be direct subtraction of the intensity images before and after loading of the specimen [1], which is to be completed using external image arithmetic operation software. There are many libraries available for image processing and manipulation in Python, two of the most commonly used libraries being Pillow and OpenCV-Python [24]. Pillow (PIL) is a python specific library that contains basic image processing functionality, including image enhancements and filtering [25], while supporting a wide range of image formats [24]. OpenCV-Python is a Python wrapper for the C/C++ coded library, OpenCV (Open Source Computer Vision). As a result, OpenCV-Python is fast, making it popular for computationally intensive computer vision programs [25]. OpenCV is also compatible with other python libraries such as NumPy and SciPy [24].

As noted by equation 1, the subtraction of two intensity images needs to result in non-negative intensity values at all points, in other words, the intensity value of each pixel needs to be equal to- or greater than zero after the subtraction of the two images. Therefore, the absolute value of the subtraction is taken. To perform this operation, the following method from the OpenCV library may be used [26].

$$\text{dst} = \text{cv.absdiff}(\text{src1}, \text{src2})\ [26]$$

This calculates the per-element absolute difference between two arrays or between an array and a scalar.

Parameters:

- src1 – First input array (representing the intensity of pixels of an image) or a scalar.
- src2 – Second input array or a scalar.

Returns:

- dst – output array (representing the absolute difference in intensity between the pixels of the two images). This array is the same size and type as the input arrays.

How it would be used:

$$\text{Shearogram Intensity array} = \text{cv.absdiff}(\text{image array after loading}, \text{reference image array})$$

Listing 1: Absolute Subtraction of Arrays using OpenCV

The real-time subtraction of the reference image from an image of the object after it has been loaded can be equated to background image subtraction that is used for tracking and analysis of objects, since the reference image is essentially a stationary image. The cv.absdiff() function is typically used for background subtraction and as explained before, determines the absolute difference between the pixels of the two inputted image arrays. This allows for the pixels of the object that are moving, or have moved, to be extracted and displayed [27].

The following method from PIL (Pillow) may also be used [28].

$$\text{image3} = \text{PIL.ImageChops.difference}(\text{image1}, \text{image2})$$ [28]

This calculates the absolute value of the difference between each pixel of the two images.

Parameters:

- image1 – First image object (PIL.Image.Image object type).
- image2 – Second image object.

Returns:

- image3 – Output image object. Pixel intensity is the absolute value of the difference of the pixels of image1 and image2.

How it would be used:

$$\text{Shearogram image} = \text{PIL.ImageChops.difference}(\text{image after loading}, \text{reference image})$$

Listing 2: Absolute Subtraction of Images using Pillow

The obvious difference between the two methods is the parameter data type. The cv.absdiff() method passes in arrays (matrices), while the PIL.ImageChops.difference() method requires two image objects of an image format, for example JPG or PNG.

It should be noted that methods cv.subtract() and PIL.ImageChops.subtract() may not be used, as after the subtraction of the images (or arrays), resultant points with negative intensity values are rounded up to zero [29][30]. These points, when displayed as an image, would result in a black pixel, which would prevent the accurate display of fringe patterns.

11.3.4.2 FlyCapture SDK Programming Basics and Usage

The Point Grey Chameleon3 USB 3.0 Digital Camera is the CCD camera available for this application. A Software Development Kit (SDK), FlyCapture SDK, has been developed to allow for the development of applications making use of the camera. The FlyCapture SDK has a Python Wrapper, PyCapture2, that allows for the development of applications in Python [31]. The relevant documentation for use of the Application Programming Interface (API) classes and methods is provided with references for both Python and C++. This allows for access to and control of camera variables from the relevant developer environments [31]. Appendix E provides examples of basic use of the PyCapture2 library to connect to and configure the Chameleon3 camera for use in a Python environment. It should be noted that the PyCapture2 library is only supported for Python versions up to Python 3.6.

11.3.4.3 Image enhancements

Once the camera is setup and data is being captured in the image buffers from the camera, the image data may be enhanced or modified to present a more desirable outcome. This may be achieved by altering some of the on-board camera image processing variables, or by using software libraries to modify images received from the camera. Some of the image variables that one might be interested in altering include, but are not limited to image brightness, contrast, exposure and sharpness. The ability to alter some of these variables may be particularly useful in Digital Shearography applications as it could lead to fringe patterns being more easily identified.

The Python Pillow library provides a module called ImageEnhance that contains a variety of classes that can be used for image enhancement. Of these, it provides the ability to alter image colour balance, contrast, brightness and sharpness [32]. An example of how to alter the contrast of an image is shown below [33]. The other image properties may be altered in a similar fashion.

```
from PIL import Image, ImageEnhance
image = Image.open("example.png")
enhancer = ImageEnhance.Contrast(image)
enhanced_image = enhancer.enhance(2.0)
```

1. Create an image object for contrast enhancement:

   ```
   enhancer = ImageEnhance.Contrast(image)
   ```

   Parameter:

- Image – pass in the image variable (of some image format) whose contrast is to be altered

  Returns:

- enhancer – image object to which an enhancement can be applied.

2. Alter the contrast of the image:

   ```
   enhanced_image = enhancer.enhance(2.0)
   ```

   Parameter:

- Float number that controls the enhancement. Passing in a value of 1.0 will return a copy of the original image. The greater the value, the higher the contrast of the image will be. A value of 0.0 returns a grey image.

  Returns:

- enhanced_image – an image with altered contrast

Listing 3: Image Enhancement Using Pillow

The PyCapture2 library available for the Chameleon3 camera also provides classes and methods to alter some of the on-board image processing features. Some of the camera properties that can be accessed using the software include image brightness, gain, auto exposure and sharpness [34]. For Digital Shearography applications, it is advantageous to disable functions such as auto exposure as it ensures that the inspection procedure remains somewhat consistent and reduces the effect any light change in the testing environment would have on the results. Similarly, the gain of the camera, the amplification of the signal from the camera sensor [35], can be auto adjusted or set manually. It should be noted that increasing the gain amplifies the entire signal, including background noise, which could lead to an image of relatively low quality.

12 Product Requirement Specifications (PRS)

PRS are defined to highlight the goals of the completed project. The criteria highlighted in the PRS should be referred to throughout the project to ensure that the goals set are being met by the product. Upon completion of the project, the final product can be compared with the PRS list to assess the success of the product. If this project were to be repeated, a comparison between the PRS list and the finished product of this project will outline how future projects can be improved.

The project can be broken down into two parts, the review of the existing optical system and improvement of the optical shearing geometry to maximise the camera field of view, and the development of a user-friendly GUI. Each of these parts have independent PRS as listed below in order of importance:

Review and Improvement of the existing optical shearing geometry:

1. The product must maximise the camera field of view.
2. The device must be able to alter magnitude and direction of image shear.
3. The device must be portable.

Development of the GUI application:

1. The program must manage the shearographic inspection process and display the fringe pattern results obtained.
2. The GUI must provide a real-time camera view window.
3. The program must allow for the image results obtained to be saved.
4. The GUI must provide access to and control of camera variables.
5. The user must be able to set a range of inspection parameters.
6. The GUI is to be user friendly and hence it should:
    a. Place the user in control of the UI.
    b. Be comfortable for the user to interact with.
    c. Require as little effort/input from the user as possible.
    d. Be consistent with regards to visual style and interactions.
    e. Be visually appealing.

13 Review and Improvement of the Existing Optical System and Optical Shearing Geometry

13.1 Concept A

This concept focuses primarily on maximising the AFOV of the Michelson Interferometer to decrease the difference between the AFOV of the Interferometer and the AFOV of the camera. This may be achieved by decreasing the distance between the imaging lens and the cube beam splitter and decreasing the distance between the mirrors and the cube beam splitter. As covered previously, the theoretical maximum AFOV of the Interferometer is 28 degrees. However, it is impossible to achieve in practice as the spacing between the beam splitter and the shearing mirror and between the beam splitter and the imaging lens cannot be reduced to zero, due to the manner in which the image shearing must be achieved and the geometry of the imaging lens. The proposed concept below attempts to increase the AFOV of the existing Interferometer from 21 degrees to approximately 25 degrees (refer to Appendix H).

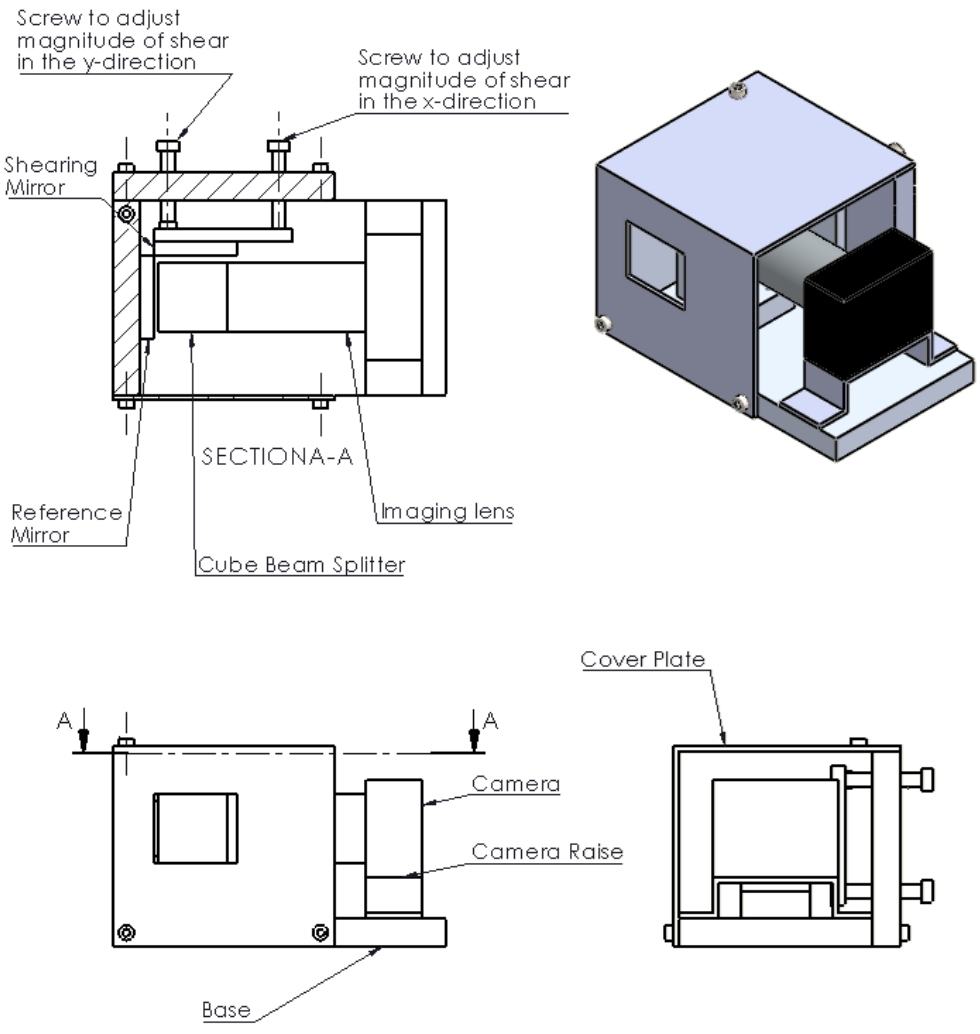


Figure 12: Concept A

13.2 Concept B

Concept B focuses on overcoming the AFOV limitation of the Michelson Interferometer. The proposed concept improves on the existing system by incorporating a 4f system that ensures that the AFOV of the system is independent of the Interferometer. The two plano-convex lenses would need to be acquired along with their respective mounts and careful attention would need to be paid to the positioning of these lenses. Although this concept is complex, it would allow for the AFOV of the system to be changed based on the focal length of the imaging lens placed in front of the Interferometer.

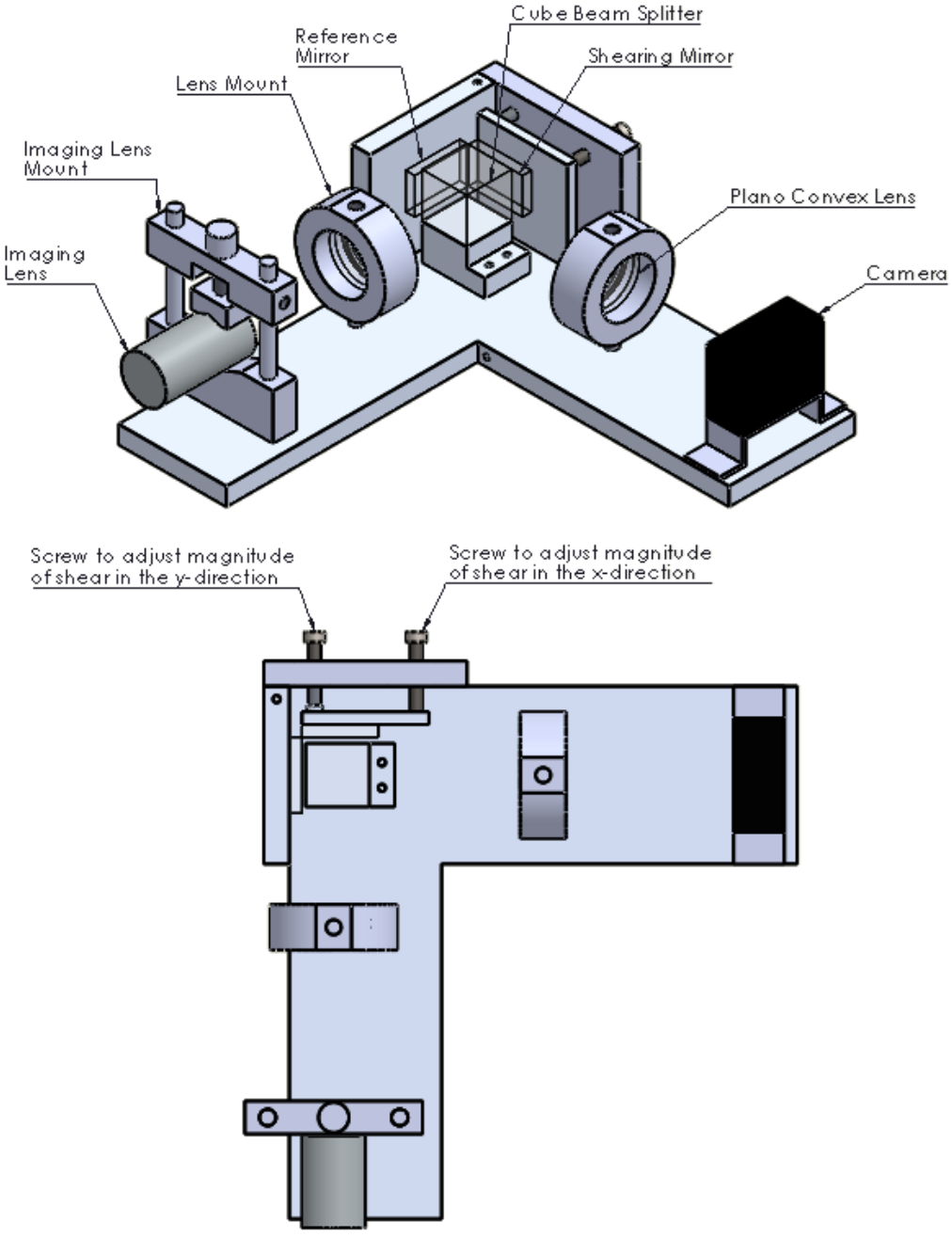


Figure 13: Concept B

The imaging lens mount would also need to be acquired from an external supplier if the capabilities of the workshop mean that it cannot be designed and made. The disadvantages of using a 4f system, such as the decrease in image quality and changes to the image intensity distribution should also be considered.

13.3 Concept Subjective Scoring Table and Final Concept Selection

The proposed conceptual designs are evaluated according to the criteria below and the displayed rating scheme.

Table 2: Rating scheme for the concepts

Inadequate	0
Weak	1
Satisfactory	2
Good	3
Excellent	4

Table 3: Optical Review and Improvement Concept Scoring Matrix

		Concepts			
		Concept A		Concept B	
Selection Criteria	**Weight**	**Rating**	**Weighted Score**	**Rating**	**Weighted Score**
Complexity	15%	4	0,6	2	0,3
Estimated Cost	30%	3	0,9	2	0,6
Achieved Angular Field of View	40%	2	0,8	4	1,6
Time to build/acquire and assemble components	15%	3	0,45	2	0,3
	Total		2,75		2,8

The concept scoring matrix indicates that Concept B would be a slightly better design for the improvement of the existing optical system and is the most likely concept to proceed with. Although it would be more complex and expensive, it offers to ability to change the field of view of the system by exchanging the imaging lens for another or by using a vari-focal lens. If executed correctly, it is guaranteed to maximise the field of view of the inspection prototype and has the potential to increase the field of view by using an imaging lens with a smaller focal length. While Concept A would increase the field of view of the system, the overall field of view is still limited by the Interferometer.

The current Covid-19 situation should be considered, and if it is found that they acquisition of components would be a too lengthy or costly process, then the concepts should be re-evaluated or the development of a theoretical design of concept B should be considered.

13.4 Final Design

The final design of the inspection device to improve the optical shearing geometry in order to maximise the camera field of view is derived from the Concept B as described in section 13.2. The final design is fundamentally very similar to the concept, with a few differences, namely the way in which the imaging lens is mounted to the base and the manner in which the lens mounts are secured.

Figure 14 below shows the complete final detailed design assembly with the cover attached. The figures that follow label the internal components of the inspection system. All the part drawings relating to the detailed design can be viewed in Appendix D.

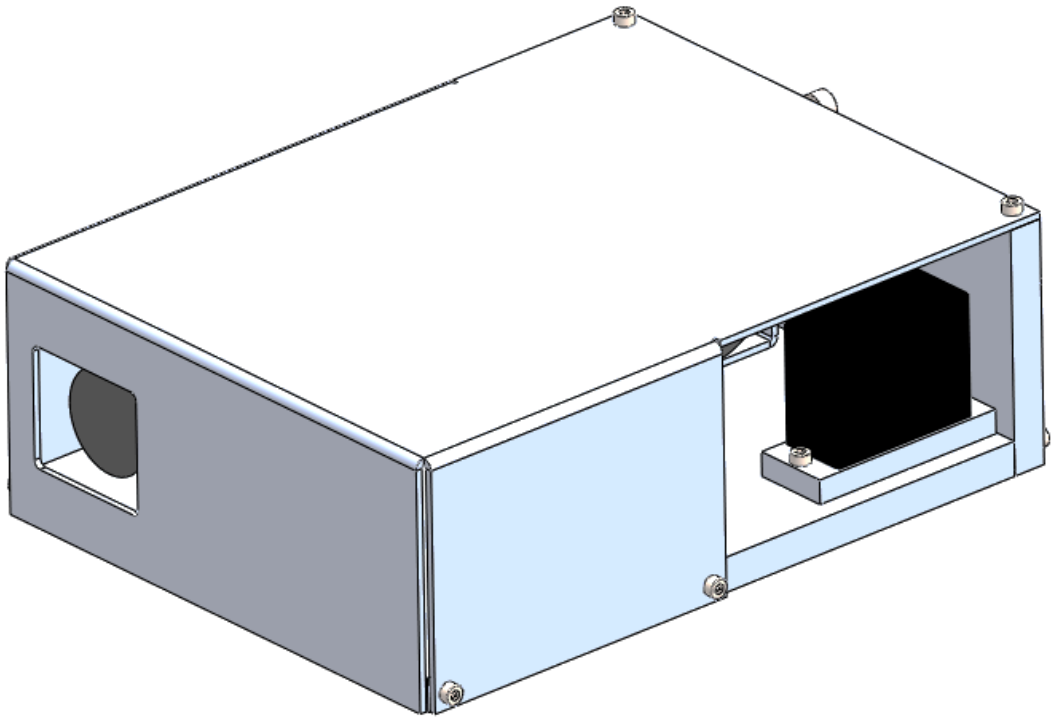


Figure 14: The Inspection Device Final Design Assembly

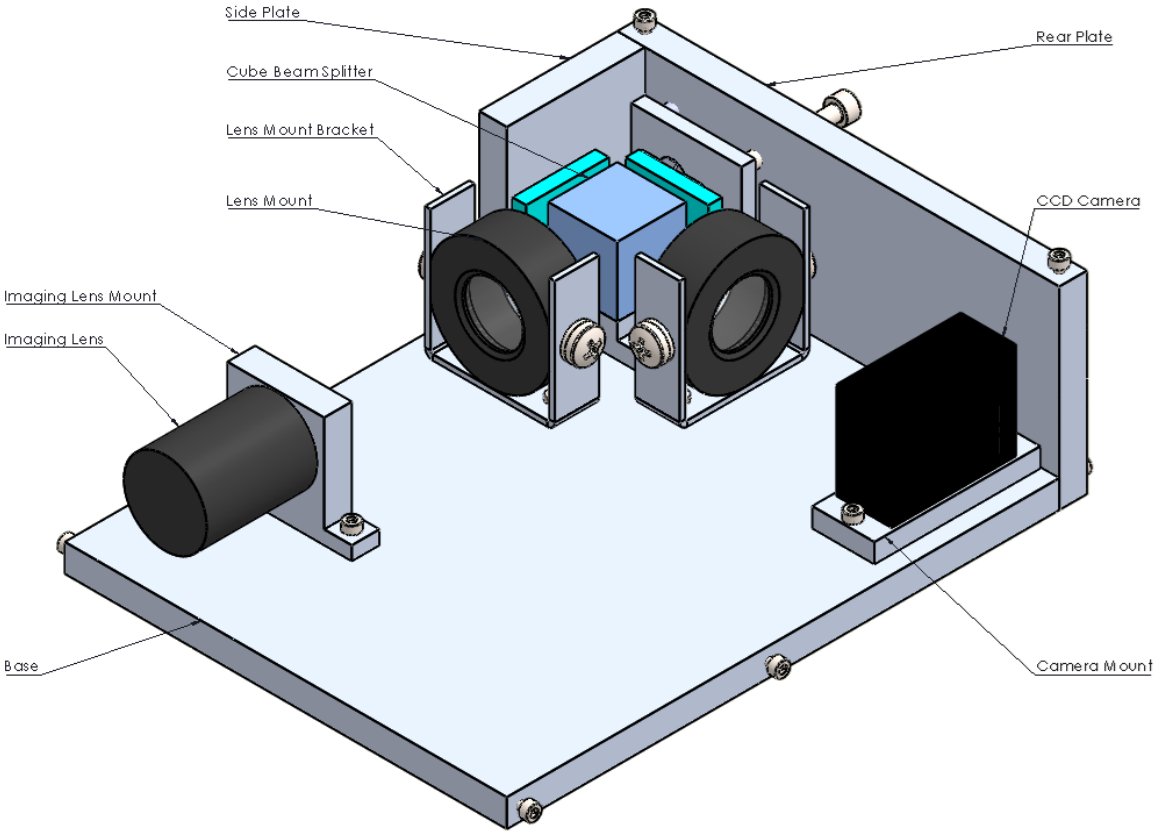


Figure 15: The Inspection Deice Final Design Assembly Internals

It should be noted that the imaging lens, camera, lens mounts, mirrors and the cube beam splitter depicted in the figure above are only representing the external dimensions of these components. This is only to assist with conveying the layout of the physical product. These parts were either already obtained or were to be ordered from an external supplier.

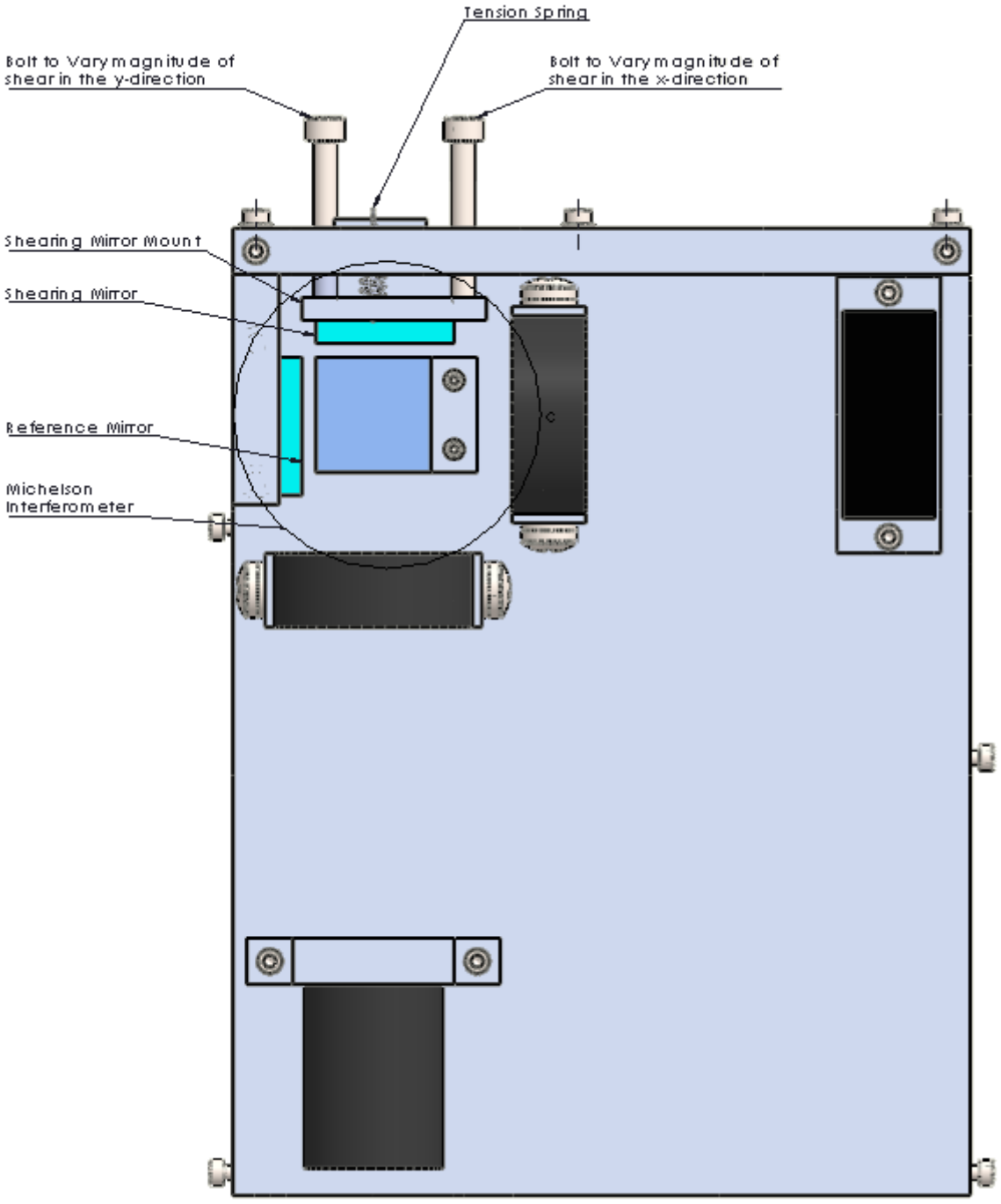


Figure 16: Top View of the Inspection Device Final Design Assembly

13.4.1 Detailed Design Description

13.4.1.1 Optical Shearing Geometry

As mentioned in section 11.1 the Digital Shearography technique requires a shearing device to introduce image shearing of identical images by displacing one of these images horizontally or vertically relative to the other. The shearing device that is used in this design is the commonly used modified Michelson Interferometer. This device consists of a cube beam splitter, a reference and shearing mirror and shearing mirror mount, as viewed in the figure below. The mirrors and beam splitter used in this design were available and provided by the supervisor. The beam splitter is a 25 mm standard cube beam splitter and the mirrors are 21 x 30 mm aluminium protected mirrors, both from Edmund Optics.

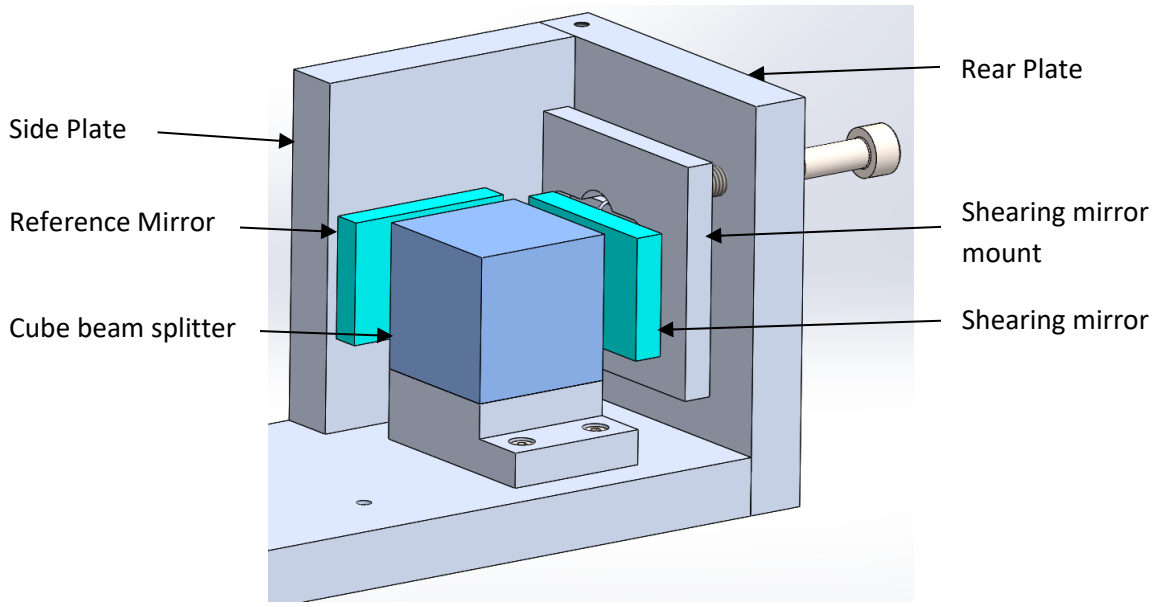


Figure 17: Modified Michelson Interferometer Section View

The Interferometer is designed such that the magnitude of the image shear can be altered using the two M5 hex socket bolts at the back of the shearing mirror mount. The direction of image shear that these bolts alter can be seen in figure 18. Turning either bolt causes the shearing mirror mount to rotate around the supporting bolt, bringing about image shear in the x- or y-direction. The support bolt not only provides a point for the shearing mirror mount to rotate about, but it also, with the tension spring, keeps the mirror mount firmly against the bolts.

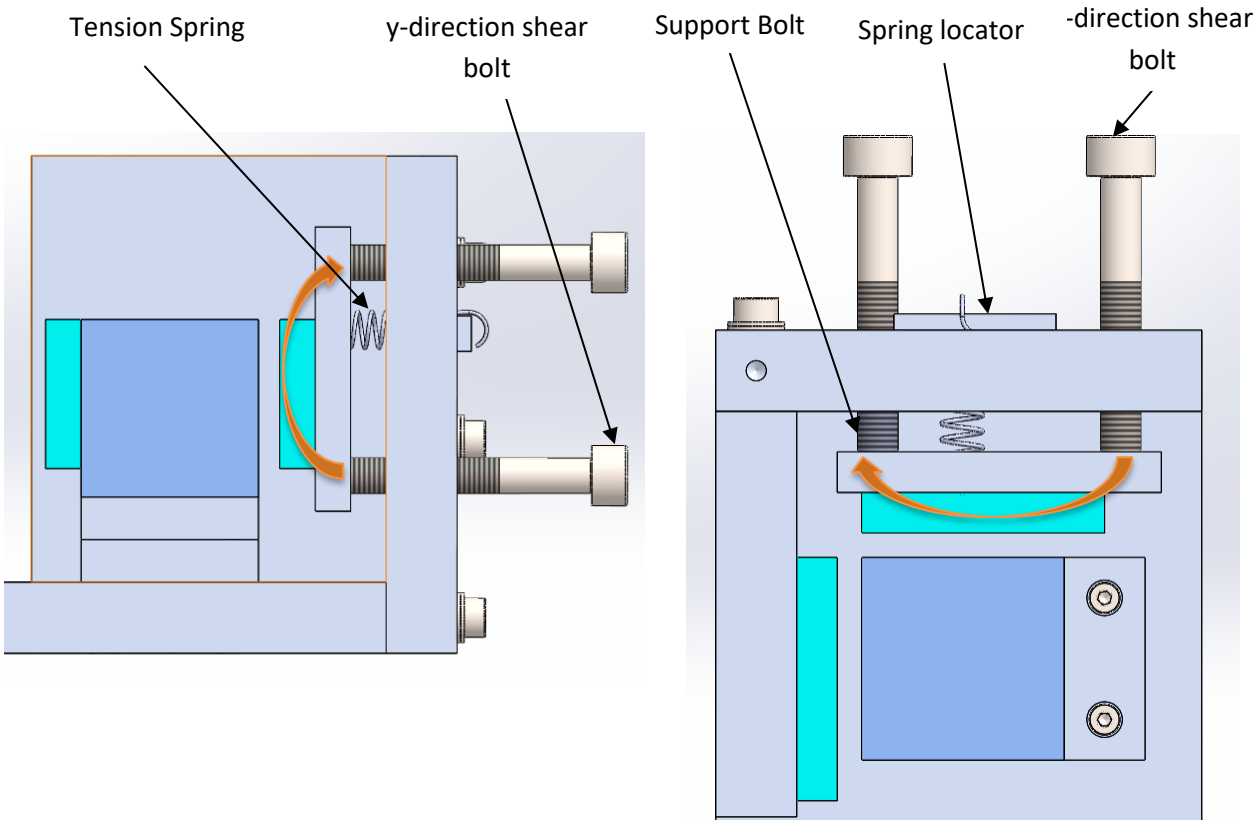


Figure 18: The Modified Michelson Interferometer Side (left image) and Top View

The figure below displays the front and back views of the shearing mirror mount, the front side is designed to contain the spring locator, to which the hooked tension spring attaches to, while the back side indicates the points of contact with the two shear altering bolts and the support bolt. The other side of the spring is attached to the other spring locator which is held in tension against the back of the rear plate.

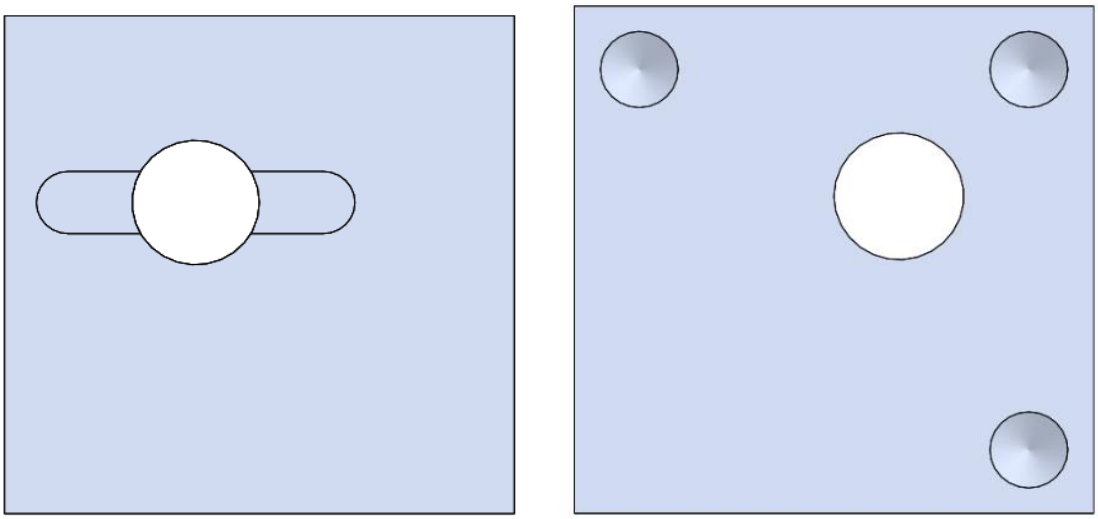


Figure 19: Front (left image) and Back Views of the Shearing Mirror Mount

The shearing and reference mirrors are joined to the shearing mirror mount and side plate respectively using 0.8 mm thick double-sided tape. This is only temporary to ensure that the mirrors are placed correctly. Once satisfied, the mirrors may be joined using a more permanent adhesive.

The cube beam splitter (CBS) sits on the CBS mount, which is designed to raise the beam splitter to the appropriate height, which is defined by the height of the two plano-convex lenses. The beam splitter is joined to its mount using the 0.8 mm thick double-sided tape, which like the mirrors, can be replaced with a permanent adhesive once proven satisfactory. The mount joins to the base of the design using two M2 × 12 mm hex socket bolts and is shown in the figure below.

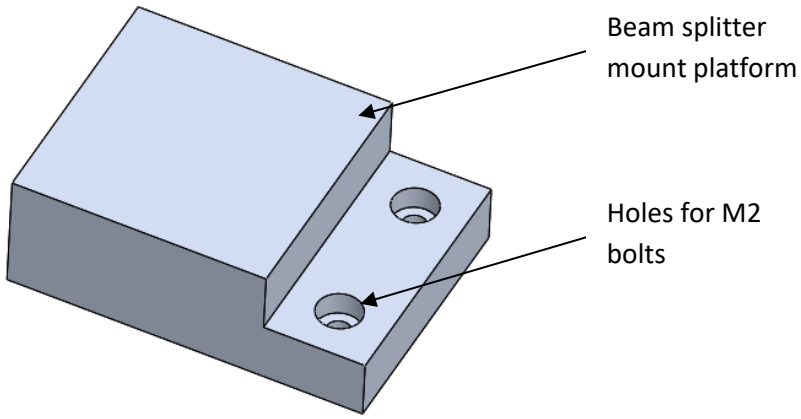


Figure 20: Cube Beam Splitter Mount

13.4.1.2 Field of View, Lenses and Lens Mounts

Together with the traditional Michelson Interferometer shearing device, the final design incorporates a 4f system, which aims to maximise the field of view of the inspection device by separating the camera and imaging lens. As explained in section 11.2.2, this is performed by placing one plano-convex lens between the Interferometer and the imaging lens and between the Interferometer and the camera. Theoretically, the resulting angular field of view is dependent on only the imaging lens focal length and the dimensions of the camera sensor, as represented by equation 7:

$$AFOV_{4f}\ [°] = 2\tan^{-1}\frac{h}{2f}$$

Careful design attention needs to be paid to the positioning of all the optical components such that their focal planes line up like in the diagram below.

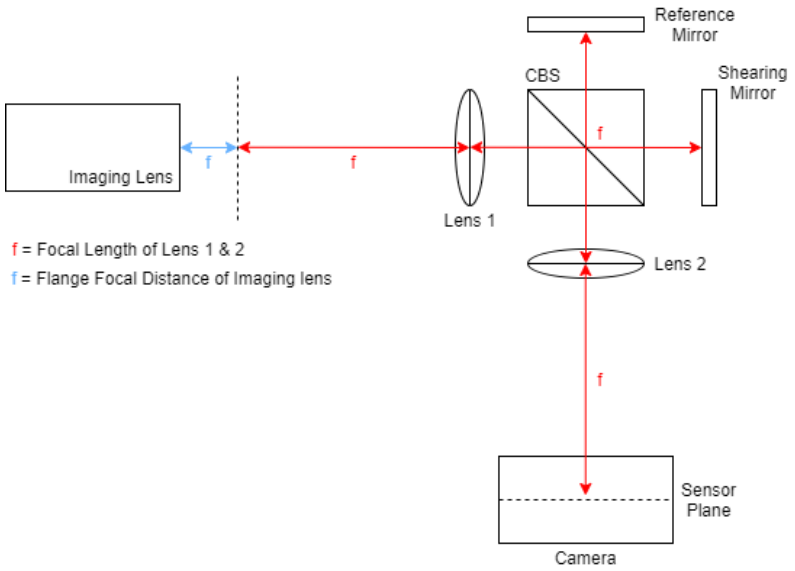


Figure 21: 4f System based Shearography design Component Layout

As a 25 mm CBS and 21 × 30 mm mirrors are available for use, it was decided that lenses of similar size should be used for Lenses 1 and 2. Edmund Optics offers lenses in 20 mm, 25 mm, 25.40 mm and 30 mm diameter, so it was decided that lenses of 25 mm diameter would be used. Knowing the size of the lenses that would be used, a lens mount for lenses 1 and 2 could be selected. It was decided that buying the lens mounts would be more beneficial than designing and having them made locally, as the lenses are delicate, and Edmund Optics offers lens mounts made specifically for certain types of lenses. The lens mount chosen for this application was the 25.0 mm Optic Diameter, Optic Mount from Edmund Optics [36]. The mount has two points for fastening on either side, a hole for an M6 bolt and another hole for a 1/4-20 UNC bolt. A diagram of this mount can be seen below, and the full drawing can be found in Appendix D.

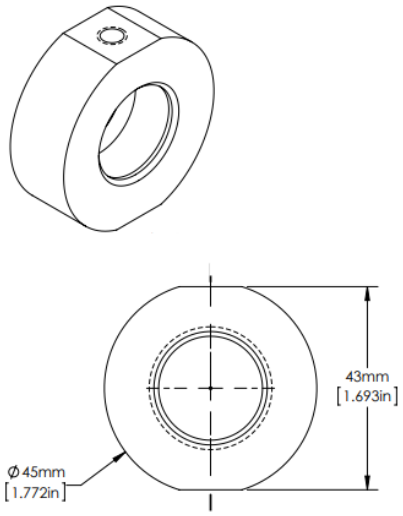


Figure 22: Lens Mount Diagram [36]

Another design consideration is the magnitude of the lens focal length. If the focal length of a lens is too small, the two lens mounts would interfere. It was found that the minimum lens focal length required is 60 mm. Therefore, the lens selected was the 25.0 mm Diameter, 60 mm focal length, uncoated, Plano-Convex lens from Edmund Optics [37]. The back focal length of this lens is given as 56.90 mm, however the front focal length was not given and had to be approximated using the Edmund Optics focal length calculator and the lens specifications given [38]. This approximated the front focal length of the lens to be 60.03 mm. A simple diagram of the lens is provided below.

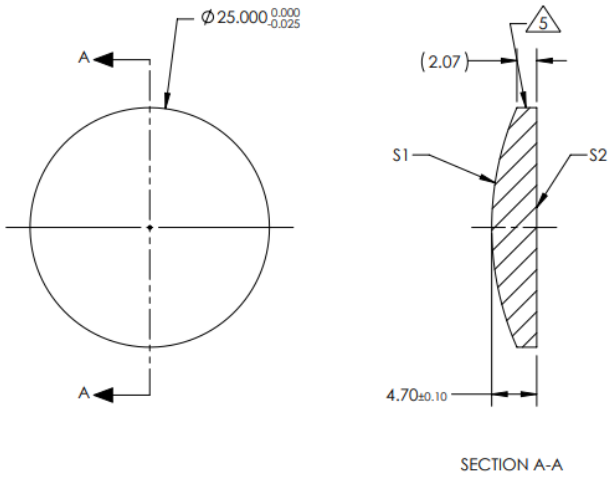


Figure 23: 25 mm Diameter, 60 mm Focal Length Plano-Convex Lens [37]

Using the back and front focal lengths of the lens, figure 21 could be revised to look like the following:

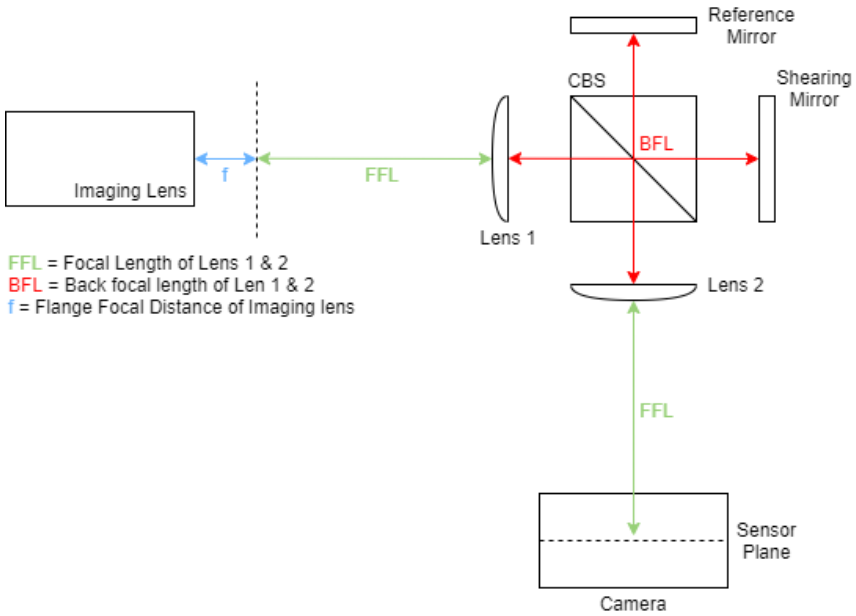


Figure 24: A revised 4f system based Shearography design component layout

Using the available dimensions of the lens and lens mount from their respective drawings, the following diagram was drawn to assist with determining the correct placement of the lens and lens mounts.

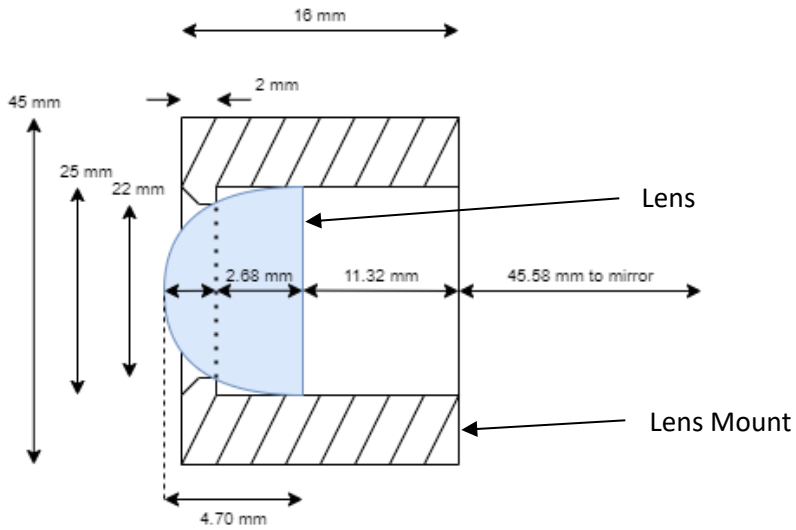


Figure 25: Rough Cross Section Diagram of the Len Mount with the Lens inside (not to scale)

The diagram above is drawn assuming that the lens will be pushed through to the edge of the lens mount. With the back focal length and the front focal length of the lens, this diagram was then used to determine the distance between the back of the lens mount and the back focal plane (45.58 mm) and the distance between the front of the lens mount and the front focal plane (60.05 mm) respectively.

13.4.1.3 Lens Mount Bracket

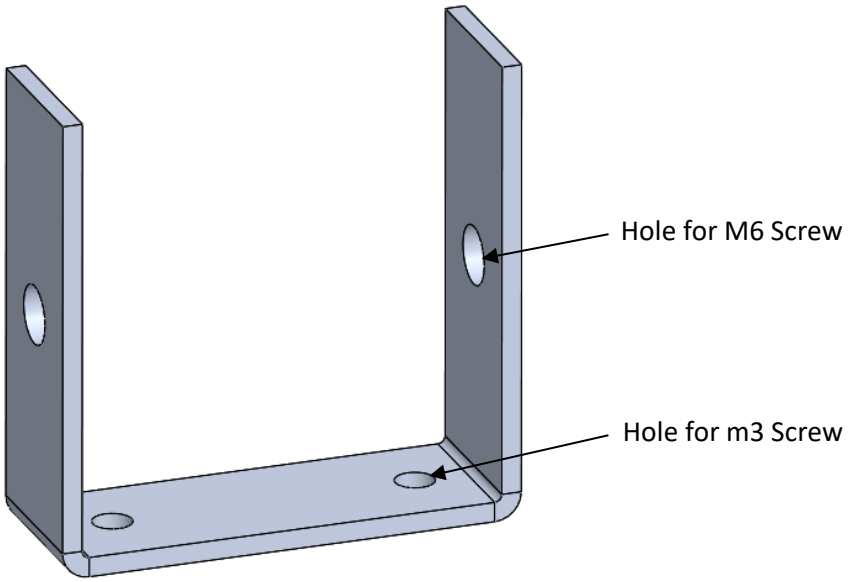


Figure 26: Lens Mount Bracket

Brackets are to be used to attach the two lens mounts to the base of the design. The flat sides of the lens mount would sit flush against the sides of the bracket and would be located axially using a M6 × 8 mm cross pan head screw. The bottom edge of the mount would sit on the bottom side of the bracket, such that it is prevented from rotating about the screw. The bracket and lens mount together define the height of the centre of the lens. This height then defines the required height of the imaging lens mount, CBS mount, the camera mount and the mirror positioning.

13.4.1.4 Imaging Lens Mount

In order to correctly support and position the imaging lens at the correct height, a mount needs to be designed. A simple clamp around the imaging lens would have been insufficient as the effectiveness of the 4f system is highly sensitive to the placement of the optical components in relation to one another. The imaging lens should not be able to move along its optical axis as the focal plane of the imaging lens and the front focal plane of the first plano-convex lens need to coincide. Knowing this, the imaging lens mount was designed by taking advantage of the imaging lens thread. The available imaging lens for this application is the Kowa LM8JCM 2/3" 8 mm F1.4 Manual IRIS C-Mount Lens [39]. C-mount lenses make use of a C-Mount thread which virtually corresponds to a metric thread of M25.5 × 0.75 mm [40]. The design of the imaging lens mount is displayed below.

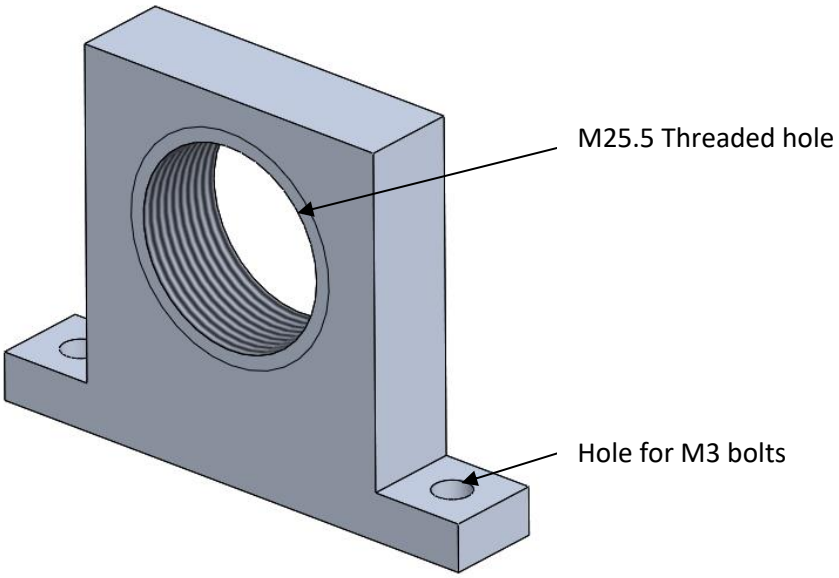


Figure 27: Imaging Lens Mount

The use of a threaded mount instead of a clamp ensures that the imaging lens is positioned at the correct height and cannot be moved axially after being tightening. The mount is fastened to the base of the design using two M3 × 10 mm hex socket bolts. Another design consideration in this case is the

Flange Focal Distance (FFD) of the imaging lens, which is specific to the type of lens. For C-Mount lenses, the FFD is 17.526 mm, meaning that the front side of the imaging lens mount needs to be placed 17.526 mm from the front focal plane of the first plano-convex lens.

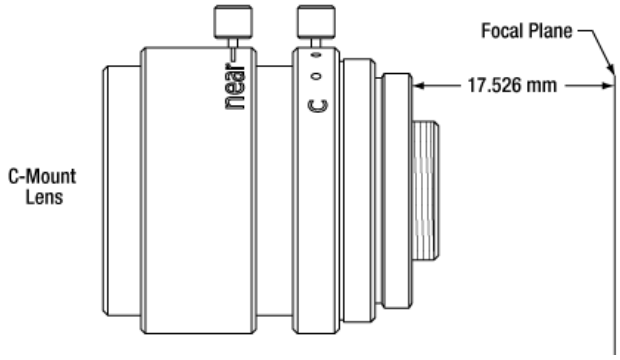


Figure 28: Front Focal Flange Distance of C-Mount lens [41]

13.4.1.5 Camera Mount

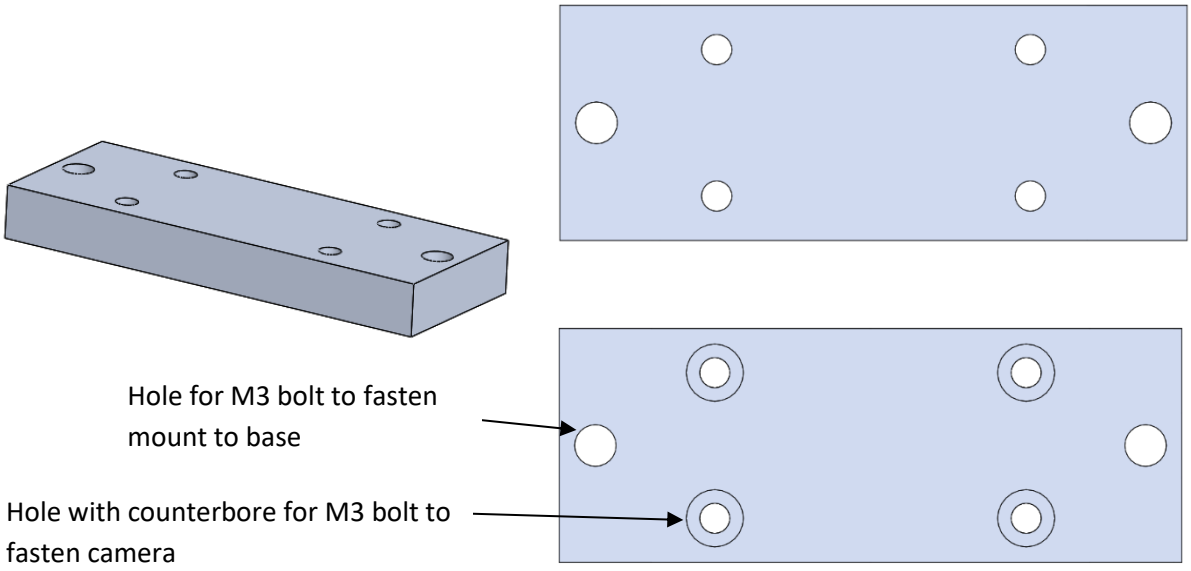


Figure 29: Camera Mount and Top and Bottom Views of Camera Mount

The camera mount, as seen above, is required to fix the camera to the base and raise the camera to the required height such that the front focal point of the second plano-convex lens falls on the camera CCD sensor. The camera is fastened to the mount using four M3 hex socket bolt which screw in from the bottom side of the mount. The mount is attached to the base using two M3 × 12 mm hex socket bolts on either side of the mount.

13.4.1.6 Cover

The cover needed to be designed to prevent as little stray light from entering the inspection system as possible. Hence it is designed to enclose as much of the design as possible, only leaving an opening at the input to the imaging lens and some space on the left side of the camera to allow for connection of the USB 3.0 cable. The cover attaches to the rest of the design body using six M3 × 6 mm hex socket bolts as indicated in the diagram below. Any other light on the back side of the prototype is prevented from entering the system by the rear plate.

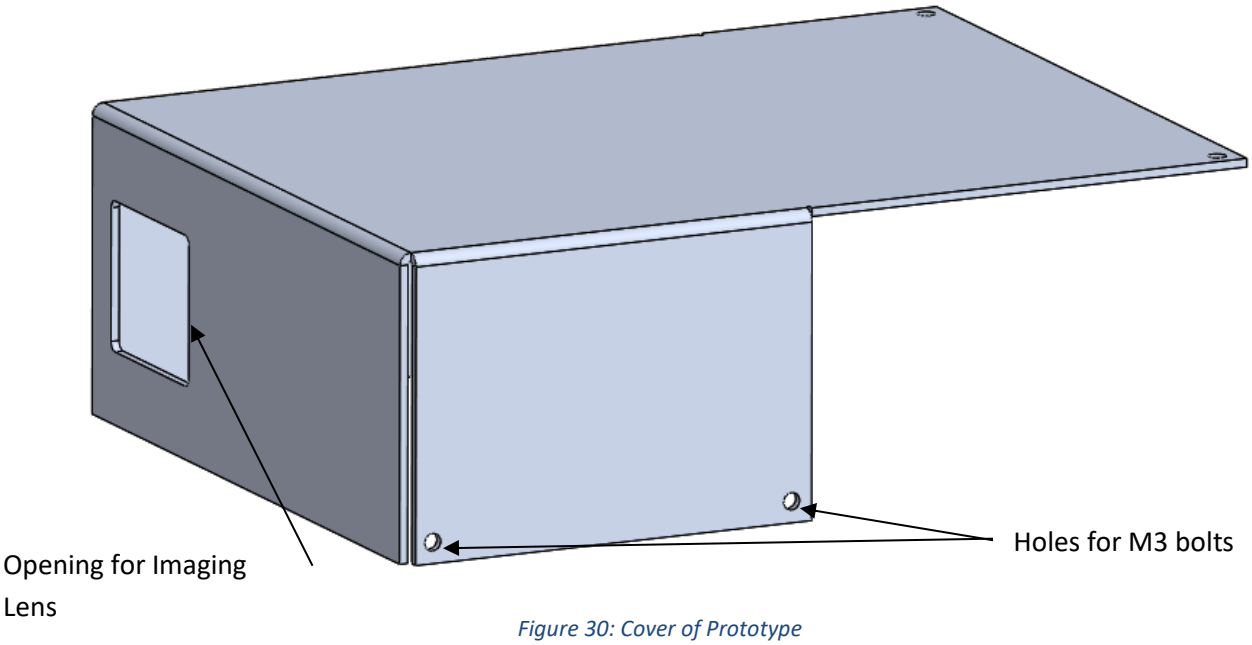


Figure 30: Cover of Prototype

13.4.1.7 Base, Rear Plate and Side Plate

The Base, Rear Plate and Side Plate support all the other components in this design. The location of the holes for attachment of components such as the imaging lens mount and the lens mount bracket have been carefully designed according to the requirements illustrated in figures 24 and 25. The Rear Plate fastens to the base at the back using three M3 × 16 mm hex socket bolts and the Side Plate fastens using two M4 × 12 mm hex socket bolts which screw in from the underside of the base.

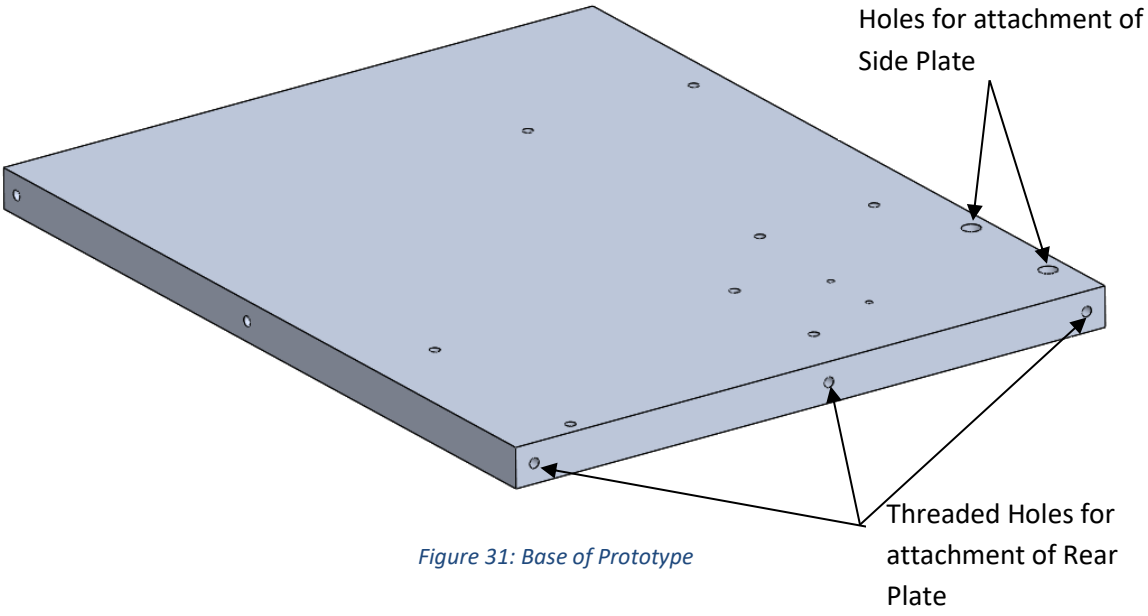


Figure 31: Base of Prototype

The Rear Plate is responsible for supporting the two bolts for altering the magnitude of the image shear as well as the support bolt. To ensure structural rigidity, the Rear Plate and Side Plate are fastened to each other using two more M3 × 16 mm hex socket bolts.

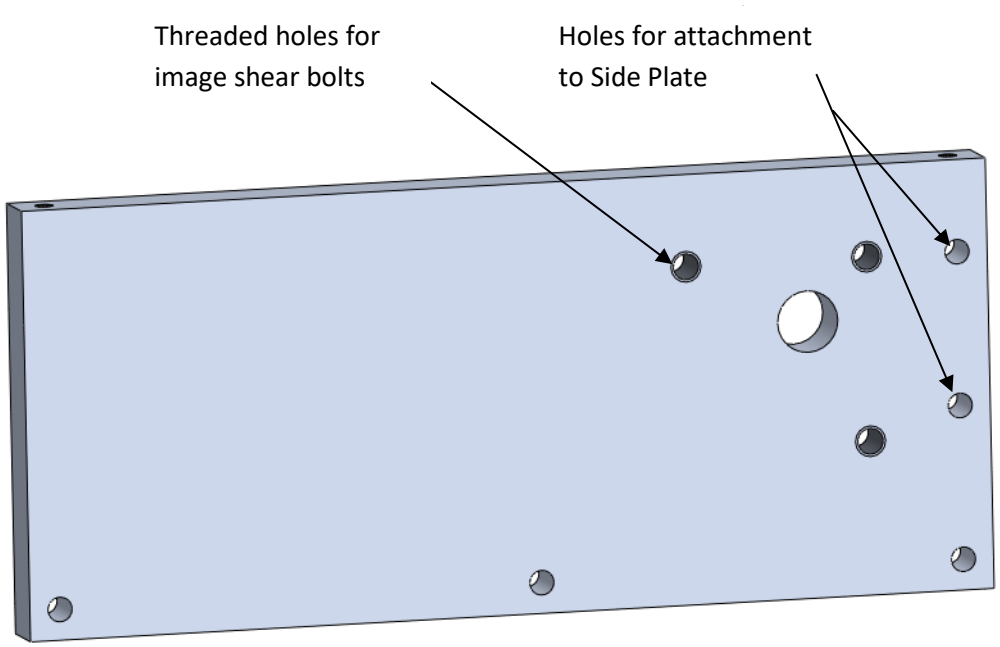


Figure 32: Rear Plate of Prototype

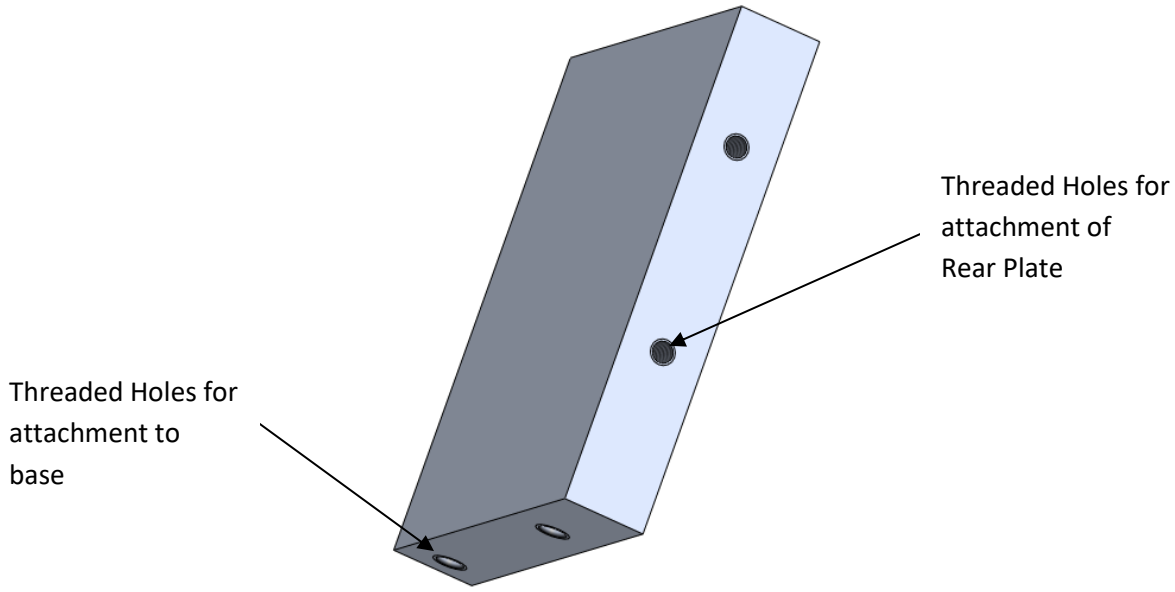


Figure 33: Side Plate of Prototype

13.4.2 Primary Material Selection and Manufacturing Considerations

The primary material selected for the components mentioned above was aluminium sheeting ranging in thickness from 2 mm to 10 mm. An important consideration for material selection is the stresses that the material of the components may be subjected to during operation. However, as the operation of this prototype involves no moving parts and none of the components are subject to large static loads, a detailed stress analysis did not need to be performed. Aluminium was selected as the material to be used as it has a relatively high strength to weight ratio, is resistant to corrosion and easy for the workshop/external venders to work with (ductile and malleable) [42]. Selecting all the components to be made from a single material is also convenient for both time and money purposes as the material may be bought in bulk and from a single supplier.

The base, Rear Plate, Side Plate, Camera Mount, Shearing mirror mount and Beam splitter mount can all be manufactured using workshop cutting tools. The Imaging lens mount would also have to be cut to size, but the C-mount thread would need to be tapped.

The Cover and the two Lens Mount Brackets were sourced from an external vendor as they require sheet metal bending. These components are to be made from 2 mm thick aluminium sheeting which is cut and bent into the desired shape. For this purpose, Vulcan Steel was used as the external vendor.

14 Review of the Existing Software Interface

As noted in section 10, the existing software interface is a command line environment, a difficult interface to use if one is unfamiliar with it. It also does not include various features that would be beneficial when conducting a shearographic inspection process, such as a real time display. This has created the demand for a user-friendly Graphical User Interface (GUI) to conduct the inspections and acquire the inspection data.

14.1 Proposed GUI Design

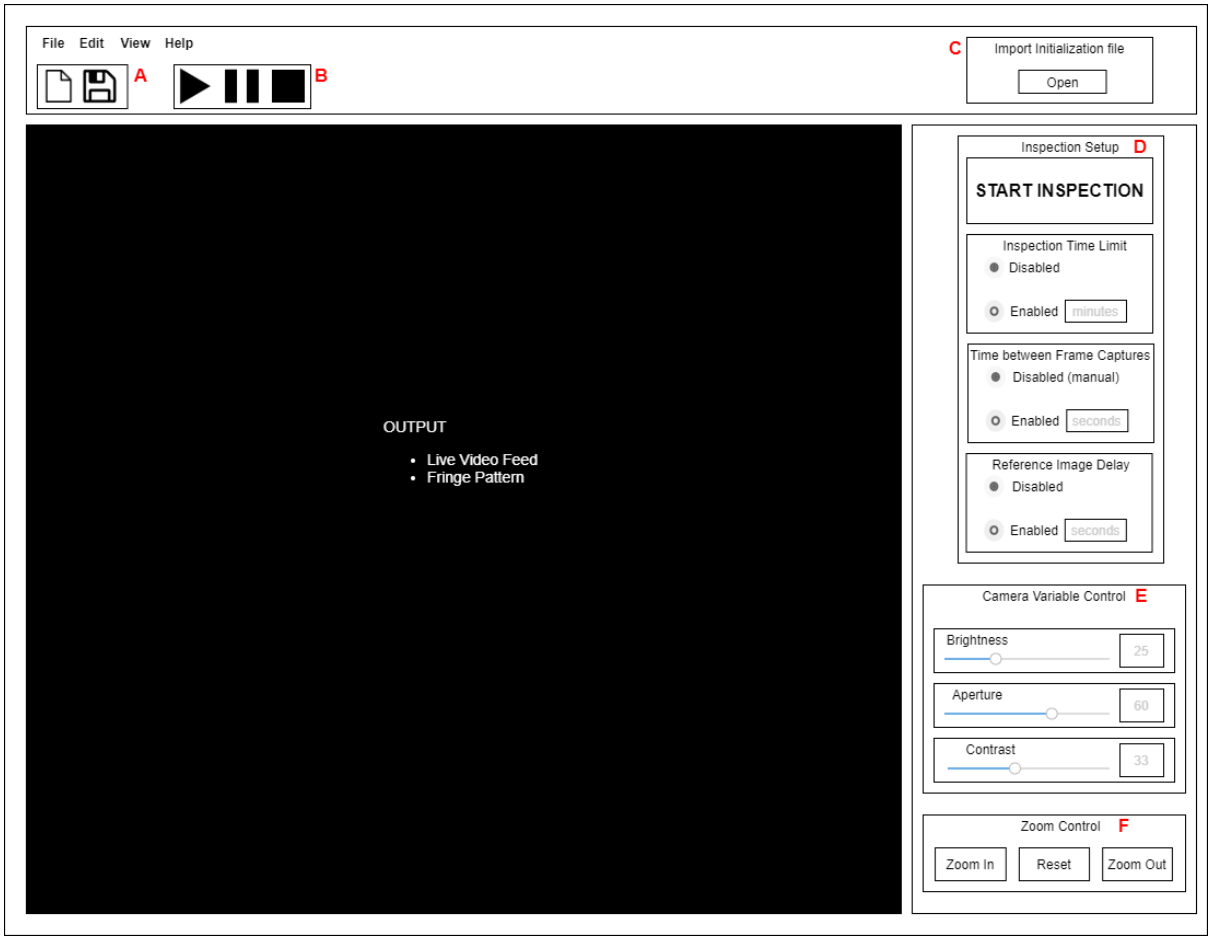


Figure 34: The proposed GUI design

This GUI could be developed in Python or C/C++, however developing a GUI in Python would take less time and be easier to implement than creating a GUI in C as it is easier to debug problems. Furthermore, Python code would be easier to understand for future academics who wish to make improvements to the application. Even though the GUI application is to be developed to run on a Windows machine, the use of Python as the development language could allow the application to be configured to be run on a Raspberry Pi as originally intended so long as the relevant external libraries are installed. Python, because of its power, versatility and friendly syntax, is the main programming

language of use on Raspberry Pi and Raspian, the Pi operating system, comes with Python preinstalled [43]. For these reasons, Python is GUI application is to be developed in Python.

Table 4: The functions of each GUI feature

Label	Feature	Function
A	Save/Fresh Inspection	The blank document icon allows for a new inspection process to be set up. The inspection settings should remain the same as the previous inspection and begin upon pressing the 'start inspection" button. The save icon will allow users to save the frame that is present on the output display to a directory of their choice.
B	Video Controls	These allow for the inspection procedure to be paused if the user wants to save a frame present on the display. The live video feed can then be resumed or stopped completely.
C	Initialisation file import	This would allow the user to import a text file that sets the inspection parameters and camera variables to their liking without the need for manual setup. This would also allow for the creation of default inspection setups.
D	Inspection Setup	The inspection parameters can be set manually here. The following may be set or disabled: • The inspection time limit in minutes. • The time between frame captures of the output display. • The delay between the beginning of the inspection process and the time at which the reference image is taken. The "start inspection" button would be pressed to begin the inspection process once the inspection parameters and camera variables have been set.
E	Camera Variable Control	Allows for the camera variables brightness, aperture and contrast to be set manually with a slider or by inputting an integer in the input box.
F	Zoom Control	Allows for areas of the display to be zoomed in or out on. If there is no zoom lens available, this will have to be implemented in software.

14.2 Implemented GUI Application

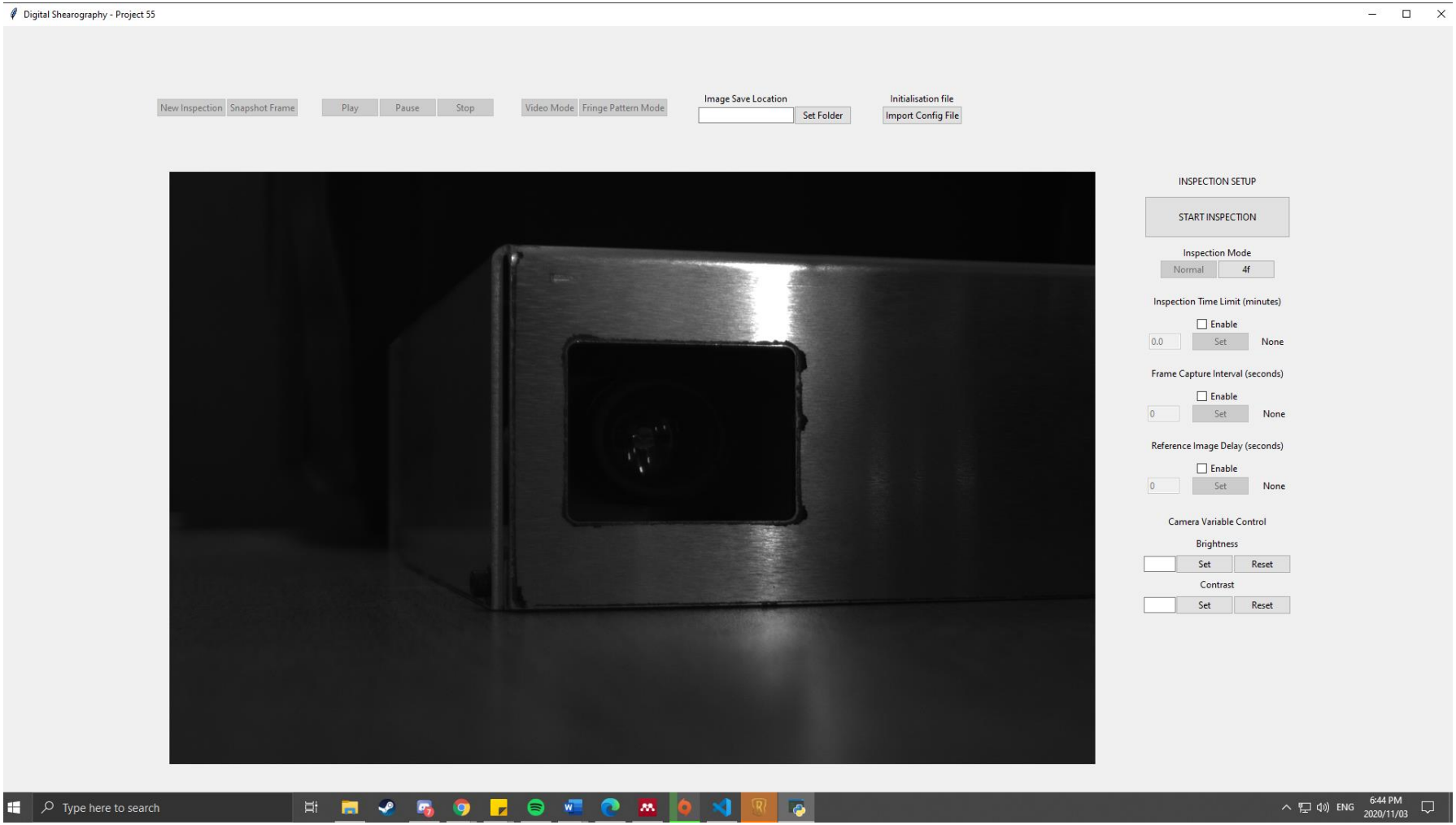


Figure 35: Designed GUI Application

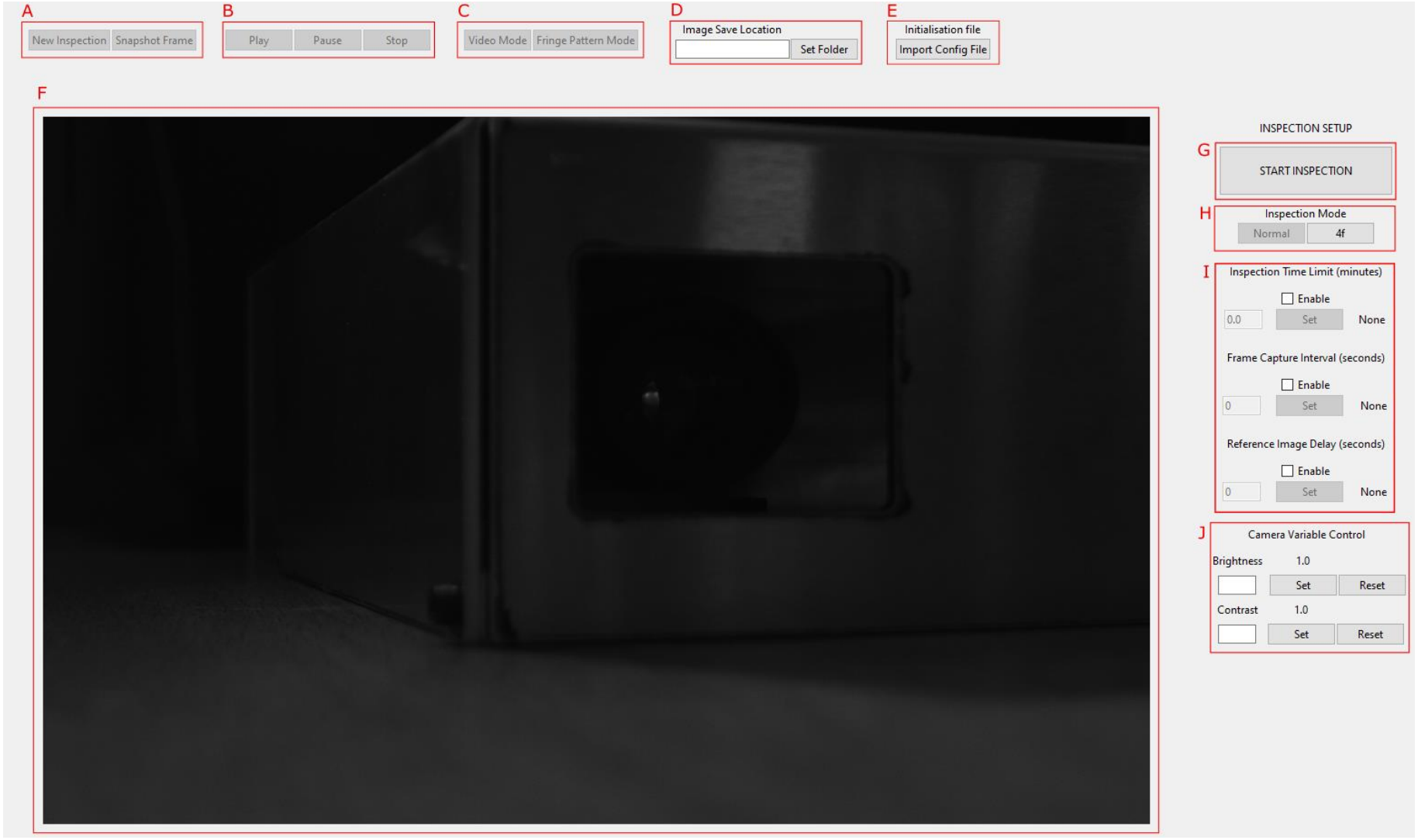


Figure 36: Designed GUI Closeup with Labels

14.2.1 GUI Functions

The two above figures display the GUI application that has been developed to assist with the inspection process. Figure 35 shows a full screen view of the GUI, while Figure 36 provides a more closeup view of the GUI and its features, which are labelled.

Table 5: The functions of each developed GUI feature

Label	Feature	Function
A	New Inspection and Snapshot (save) Frame	The *New Inspection* button allows for the setup of a fresh inspection. The inspection parameters are reset to default, while the Camera Variable values (brightness and contrast) remain the values they were set for the previous inspection. The new inspection process begins upon hitting the *START INSPECTION* button.
B	Video Controls	• The *Pause* button allows for the inspection process to be paused, freezing the frame currently shown on the display. • The *Play* button allows for the inspection process to be resumed, updating the video display. • The *Stop* button allows for the inspection process to be ended after being paused. The camera view to the display is ended.
C	Display Modes	• The *Video Mode* button activates Video Mode which results in the Display (F) displaying the real-time view of the camera. • The *Fringe Pattern Mode* button activates Fringe Pattern Mode which results in the Display window showing the real-time subtraction of the reference image from the current image captured by the camera.
D	Save Folder Location	This allows for the save location of images to be set. This is set by pasting the directory of the desired save folder into the Entry provided. By default, the save location of images is set to the location in which the application is installed on the PC.
E	Configuration File	The *Import Config File* button will import the inspection parameters (I) and camera variable values (J) set in the configuration file available (config.ini), allowing for inspection settings to be automatically set.
F	Display	Depending on the display mode active, the Display window will either display a real-time video of the camera view or a real-time direct subtraction of the reference image from the current camera image.
G	Inspection Start	Upon hitting the *START INSPECTION* button, the inspection process will begin. The Inspection process should be begun after having set the desired inspection parameters and camera variable values, the desired inspection mode, as well as the desired save location of images.
H	Inspection Mode	• The *Normal* button activates Normal Inspection Mode, which is an inspection carried out using a traditional Shearography device. • The *4f* button activates 4f Inspection Mode, which is an inspection carried out using a 4f-based Shearography inspection

		device. This will rotate the images by 180◦ which is required if a 4f system is utilised.
I	Inspection Parameter Setup	The Inspection Parameters may be manually set or disabled. The following may be set: • The Inspection Time Limit in minutes. • The time interval between automatic saving of frames displayed on the Display window (F). • The delay between the beginning of the inspection process and the time at which the reference image is taken and saved. These inspection parameters cannot be changed once the inspection procedure has begun.
J	Camera Variable/Image Enhancement Control	The Camera Variables brightness and contrast may be set. These are set using a float value whereby a value greater than one increases the image enhancement and a value between zero and one decreases the image enhancement. These variables can be reset using the reset buttons that result in the image returning to its original state prior to enhancement. These values may be changed at any time during the inspection procedure.

It was decided that zoom control would not be implemented as the Chameleon3 camera being used does not support the zoom properties/controls provided in the related PyCapture2 library.

14.2.2 Process of Operation

A Unified Modelling Language State Chart has been used to represent the process of operation of the application. Different states of the application are represented by blocks, while actions between states are represented by arrows between the blocks. Prerequisites required for actions to take place are denoted by text in square brackets.

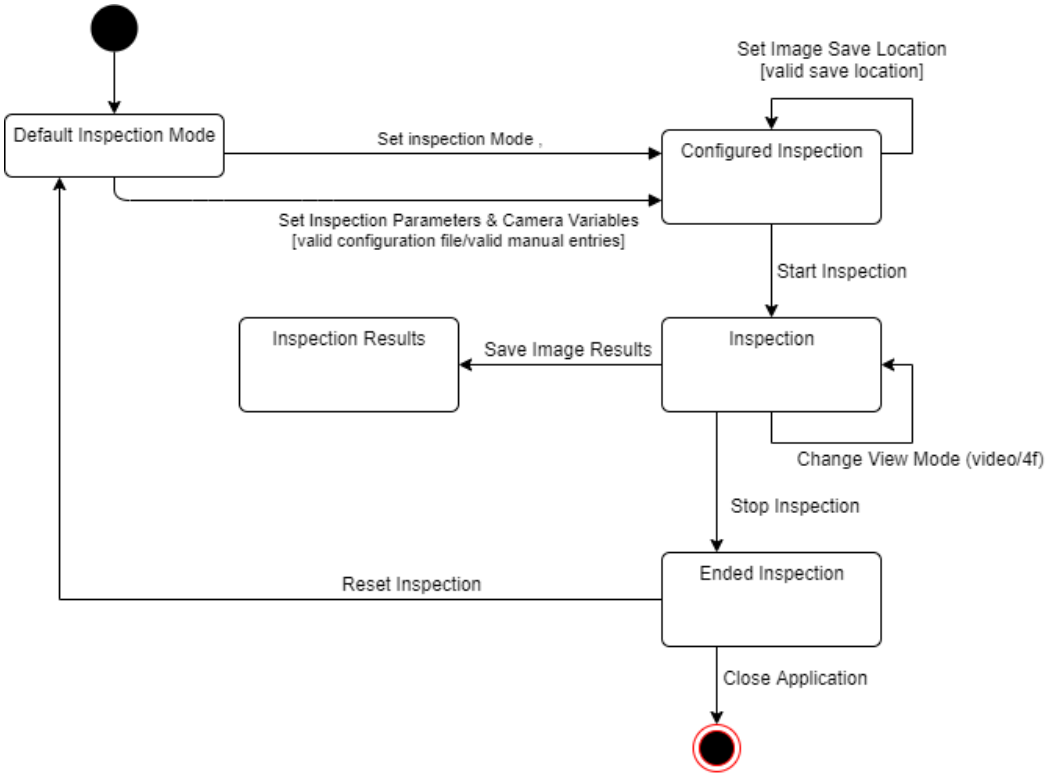


Figure 37: UML State Diagram for GUI Process of Operation

14.2.3 Application Development and GUI Design Pattern

It was decided that Tkinter would be used as the GUI development toolkit for this application. Tkinter is a native Python library that is often used for Python scripts. Tkinter was chosen as it is highly accessible and easy to learn, making complicated and detailed calls easy to use with simple methods [44]. Python scripts using Tkinter can be run on different machines without having to manually install libraries or making modifications to any Tkinter code. Tkinter is also available on all platforms that can utilise Python, including Mac and Linux, and it will adapt to the native appearance of these operating systems [44]. This means that a Tkinter application can easily be shared between Windows and Mac or Linux machines. A course on Udemy was used to learn the necessary skills for GUI development with Python and Tkinter.

14.2.3.1 Tkinter Basics

This section focuses on the basic functionality of Tkinter, including the development of the application window and the use of widgets such as label, buttons and frames. Firstly, the tkinter library needs to be imported into the Python script, as show in listing 4.

```
import tkinter as tk
from tkinter import ttk
```

Listing 4: Importing Tkinter

The ttk subclass of tkinter is similar to the traditional tk subclass, but it provides some widgets with more functionality and customization. However not all widgets fall under the ttk class.

The main application window into which the widgets are placed is responsible for running the main loop of the program. It acts as the parent (master) for all the other widgets and is responsible for the application geometry and title. The main window is defined as follows:

```
root = tk.Tk()              #create an application window object called root
root.title("Hello World")   #assign the window a title

#Insert Application Widgets

root.mainloop()             #run the application
```

Listing 5: Creating an Application Window using Tkinter

Tkinter offers various widgets, however only those required for this application will be explained. Widgets are a subset of classes in Tkinter that inherit from the class Widget. When instantiated, widgets create a window component that can be styled and customised using options defined as the widget's parameters (not methods) [44]. Typical stylings include:

Table 6: Common Styling Options of Widgets in Tkinter [44]

Option	Value	Effect
foreground	Colour (specified using keywords)	Determines the foreground colour of the widget.
background	Colour	Determines the background colour.
bd (border width)	Integer	Determines the width of a widget border.
padx, pady	Integer	Specifies the width (in pixels) between a widget and its adjacent widgets.
font	Tuple (font family, size height)	Specifies the font characteristics of the widget.

Options are assigned using the option name, followed by '=' and then the option value. For example: text = 'Example Label'.

For a widget to appear visible on the screen, a Geometric manager (grid, pack or place) must be called after the widget is configured. These geometric managers allow for widgets to be laid out appropriately in an application [44]. An example is given below:

```
root = tk.Tk()              #create an application window object called root
root.title("Hello World")   #assign the window a title

Hello_World_Label = ttk.Label(root, text="Hello World")
#Widget will not appear in the window until geometry manager is applied
Hello_World_Label.grid(row=0, column=0)
#Widget will now appear in the window in row 0 and column 0 of its parent

root.mainloop()             #run the application
```

Listing 6: Creating and Managing the Placement of Widgets in an Application Window

Types and explanations of widgets used in the development of the GUI application are listed in the tables below along with the options specific to each widget [44]:

Table 7: Frame Widget Description and Functions

Widget	ttk.Frame(*parent, options…*)	
Description	A rectangular region into which other widgets are placed (acts as a parent). It can also be used to separate different regions of widgets with padding.	
Options	width, height	Specifies the dimensions of the Frame (pixels).
Methods	after(*milliseconds,* lambda: *function)*	Calls a lambda function after a period (milliseconds) as an interrupt.

Table 8: Label Widget Description and Functions

Widget	ttk.Label(*parent, options…*)	
Description	A rectangular widget used to display text or an image.	
Options	text	The text displayed on the label.
	textvariable	The Tkinter variable (StringVar, IntVar, DoubleVar) associated with the label. Changing the Tkinter variable changes the text displayed on the label.
	image	The image displayed on the label.
Methods	after(*milliseconds,* lambda: *function)*	Calls a lambda function after a period (milliseconds) as an interrupt.

Table 9: Button Widget Description and Functions

Widget	ttk.Button(*parent, options…*)	
Description	The button is used to carry out Python functions or methods when it is pressed.	
Options	text	The text displayed on the button.
	command	The function or method that is called when the button is pressed. It is essential that a function or method us called without parenthesis following the function name.
	state	Button state can be changed from enabled to disabled.
Methods	after(*milliseconds,* lambda: *function)*	Calls a lambda function after a period (milliseconds) as an interrupt.

Table 10: Entry Widget Description and Functions

Widget	ttk.Entry(*parent, options…*)	
Description	A rectangular widget used to enter a single line of text or number.	
Options	width	Specifies the width of the Entry (pixels).
	state	Entry state can be changed from enabled to disabled.
	textvariable	The Tkinter variable (StringVar, IntVar, DoubleVar) associated with the entry. Changing the entry text results in the textvariable changing value.
Methods	after(*milliseconds,* lambda: *function)*	Calls a lambda function after a period (milliseconds) as an interrupt.
	get()	Get the current contents of the entry field and assign it to a variable.
	delete(from, to)	Delete the characters in the entry field between the index specified. The delete(0, “end”) method deletes all the contents of the entry field.

Table 11: Check Button Description and Functions

Widget	ttk.Checkbutton(*parent, options…*)	
Description	A button used to implement on-off selections. A check button requires and associated variable.	
Options	text	The text displayed next to the check button.
	command	The function or method that is called when the check button is pressed.
	onvalue, offvalue	The value of a variable when the button is checked (onvalue) and the value of a variable when the button is non-checked (offvalue).
	variable	The Tkinter variable (StringVar, IntVar, DoubleVar) associated with the check button. When the button is pressed, the variable is set to the onvalue of the check button.
	state	Check button state can be changed from enabled to disabled.
Methods	after(*milliseconds,* lambda: *function)*	Calls a lambda function after a period (milliseconds) as an interrupt.

14.2.3.2 GUI Design Layout

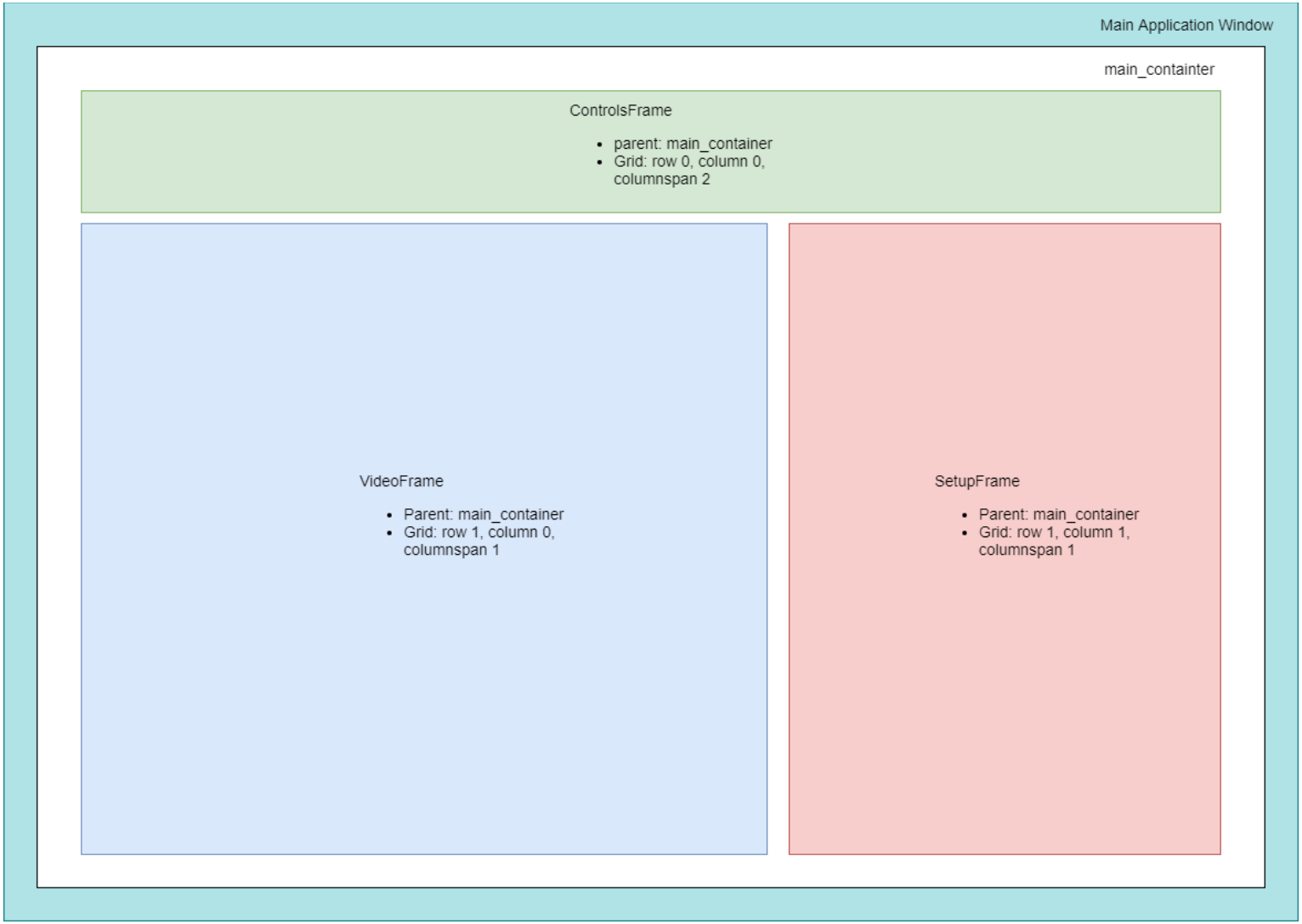


Figure 38: High-Level GUI Design Layout

The figure above presents a high-level layout of the GUI design. To summarise, inside the application window is the main_container frame, which is the parent of three other significant frames, the ControlsFrame, which contains the video controls amongst other features, the VideoFrame, which contains the video display, and the SetupFrame, which contains the setup of inspection parameters and camera variables. The rows and columns of these frames are also indicated in the above diagram. Although the ControlsFrame is placed in row 0 and column 0, it is given a columnspan value of 2 meaning it will span across columns 0 and 1. It should be noted that when widgets are gridded, their row and column placement is relative to their parent (master) and not the main application window.

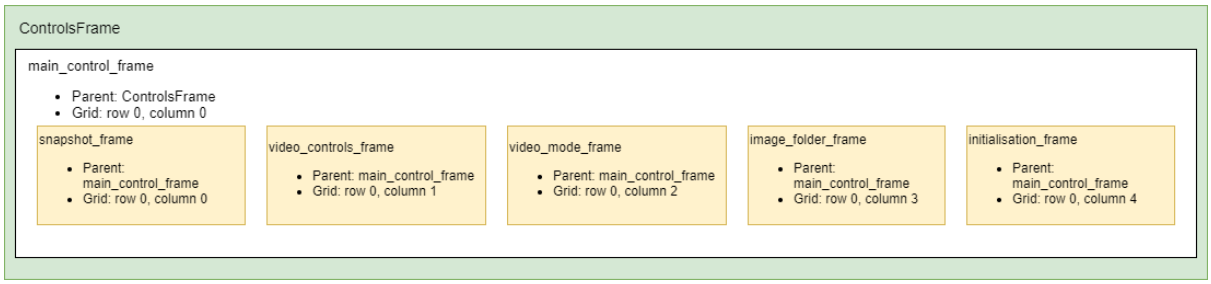


Figure 39: Low-Level ControlsFrame Frame Layout

A low-level frame layout of the ControlsFrame is shown above. This is parent to the main_control_frame, which is the parent of several lower level frames. The snapshot_frame contains the *Snapshot Frame* and *New Inspection* buttons, while the video_controls_frame contains the buttons relating to the video control, such as *Play* and *Pause*. The video_mode frame is the parent of buttons that control video mode and fringe pattern mode respectively and the image_folder_frame is the parent of widgets relating to the setting of the image save location. Lastly, initialisation_frame contains widgets relating to the configuration file.

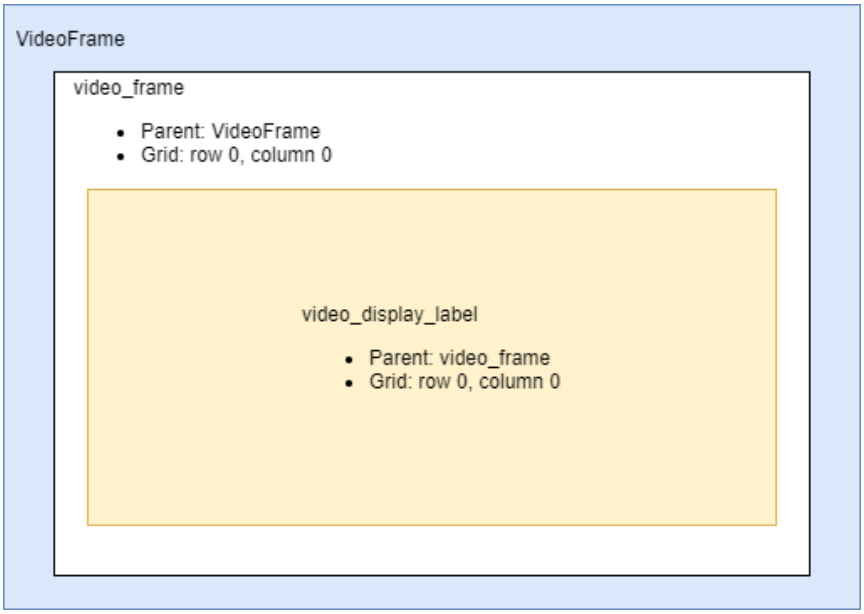


Figure 40: Low-Level VideoFrame Frame Layout

The low-level VideoFrame layout in the figure above shows that the VideoFrame is the parent of the frame, video_frame, which is the parent of the video_display_label. This label is responsible for displaying real-time video feedback and fringe patterns, which is explained in section 14.2.4.

Figure 41: Low-Level SetupFrame Frame Layout

Lastly, the low-level design of the SetupFrame is illustrated above. The SetupFrame is the parent of two other frames, the inspection_setup_frame and the camera_variables_frame. The inspection_setup_frame is the parent of four other frames, the inspection_mode_frame, inspection_time_limit_frame, Frame_capture_interval_frame and the reference_image_delay_frame, which contain widgets related to the selected inspection mode, the inspection time limit, the time between automatically saved frames and the reference image delay respectively. The camera_variables_frame is the parent of the brightness_frame and the contrast_frame, which are responsible for containing widgets controlling image enhancement.

14.2.3.3 GUI Design Pattern

The GUI design pattern that has been used is the modified Model-View-Controller pattern that uses the controller as the mediator between the model and the view, as mentioned in section 11.3.2. The GUI application consists of five different Python files. The camera and camera related objects in the app.py file acts as the model and maintain application specific data. The controls_frame.py, video_frame.py and setup_frame.py files that are contained within the 'frames' folder are three views associated with the model and are responsible for displaying the application data to the user. The controller.py file is responsible for handling interactions with the user, updating or notifying the views and model appropriately.

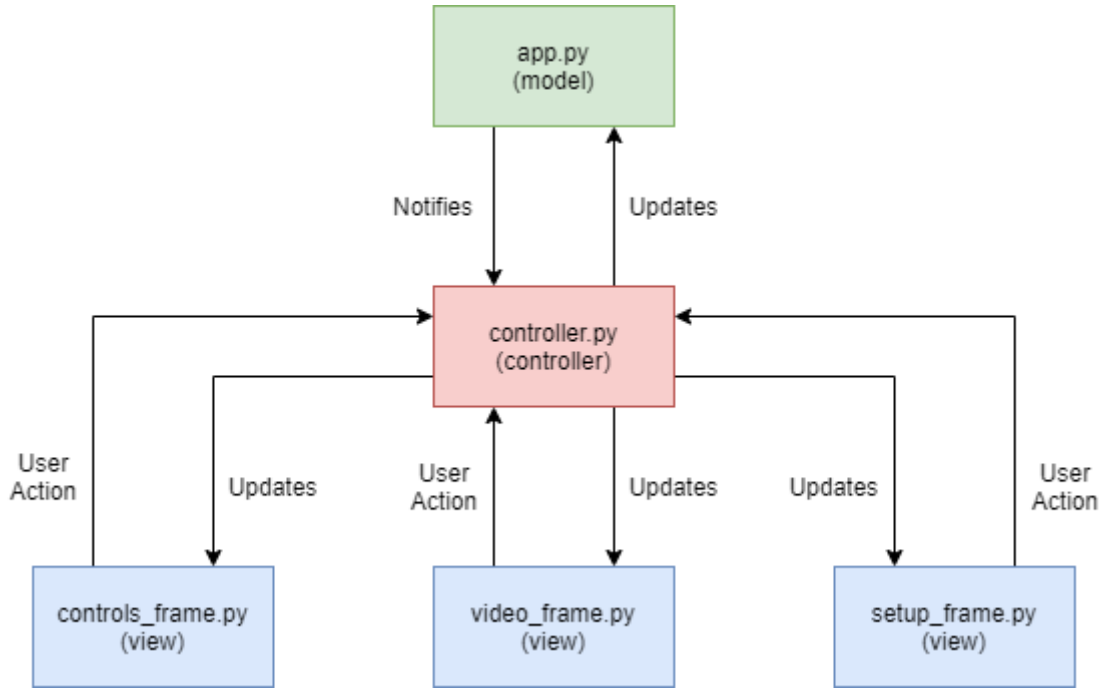


Figure 42: modified MVC pattern of developed GUI application

14.2.4 GUI Application Flow Diagrams and Explanation of Implemented Code

This section focuses on the implementation of the application code. A UML Class Diagram has been used to describe the applications classes, attributes and methods, as well as the relationships between them. UML diagrams are typically used to model and document software. A class UML diagram contains three fields, the name of the class on top, followed by the class attributes, and lastly the class methods/operations at the bottom, with related classes being connected with a line [45]. The complete Python code is available in Appendix F.

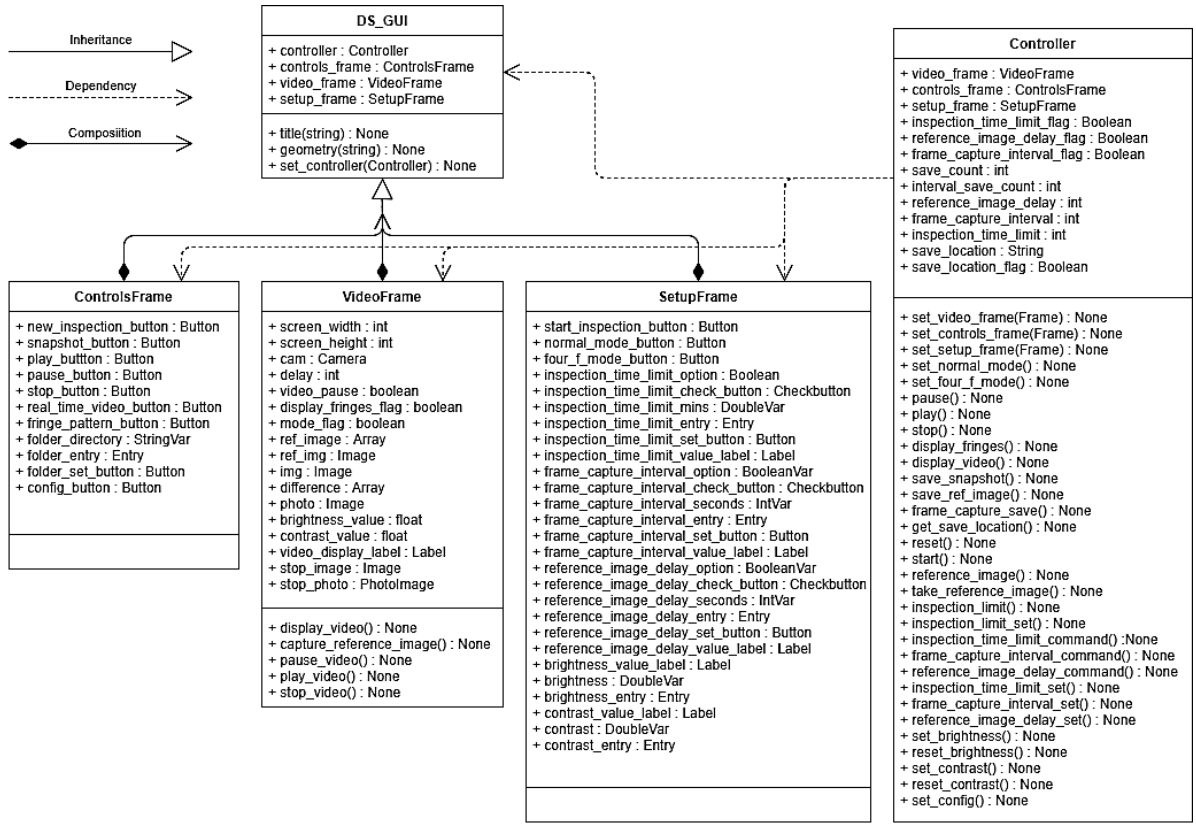


Figure 43: UML Class Diagram of Classes used to develop GUI application

14.2.4.1 Explanation of Classes

The UML class diagram provides a high-level view of the application classes and the attributes and methods related to these classes. A brief explanation of these classes is provided below, followed by explanations and flow charts of the most significant class methods.

The DS_GUI class in the app.py file is a class that creates the main application window. It inherits properties from the Tk class in the Tkinter library, making the DS_GUI class the main window of the application. This is done as follows:

```
class DS_GUI(tk.Tk):                          #create a class of type Tk class
    def __init__(self, *args, **kwargs):      #initialise the class
        super().__init__(*args, **kwargs)     #call the constructor of the Tk
                                              #class to inherit properties
```

Listing 7: Creating and Initialising the DS_GUI Class

Inside the DS_GUI class definition, the geometry and title of the application window are configured followed by the placement of frame widgets.

The ControlsFrame class in the controls_frame.py file is a frame that inherits properties from the ttk.Frame class. This frame is the parent of other frames as illustrated in figure 39, and the widgets contained within this frame relate mainly to the inspection controls. The parent of the ControlsFrame, from which properties are inherited, is passed into the class as a parameter as well as the controller which is used to call methods from the Controller class in the controller.py file.

```
class ControlsFrame(ttk.Frame):      #create a class of type ttk.Frame
    def __init__(self, parent, controller): #initialise the class
        super().__init__(parent)     #call the constructor of the ttk.Frame
                                     #class to inherit its and the parent's
                                     #properties
```

Listing 8: Creating and Initialising the ControlsFrame Class

The VideoFrame class in the video_frame.py file is a frame that inherits properties from the ttk.Frame class. The VideoFrame class can therefore be placed inside the DS_GUI class as a frame and can be a parent to other widgets. This class is responsible for containing the video_display_label widget that displays the real-time video or fringe pattern results.

```
class VideoFrame(ttk.Frame):    #create a class of type ttk.Frame
    def __init__(self, parent, screen_width, screen_height, cam, delay):
                                    #initialise the class
        super().__init__(parent)    #call the constructor of the ttk.Frame
                                    #class to inherit its and the parent's
                                    # properties
```

Listing 9: Creating and Initialising the VideoFrame Class

Parameters relating to the video display are also passed into the VideoFrame class, such as the camera object, cam, and the camera refresh rate, delay, as well as the VideoFrame's parent from which properties are inherited.

The SetupFrame class in the setup_frame.py file is a frame that inherits properties from the ttk.Frame class. This frame contains widgets relating to the setup of the inspection process and the camera variables. Properties are also inherited from the frame's parent which is passed in as a parameter. The controller, from which methods are called, is also passed in as parameter.

```
class SetupFrame(ttk.Frame):                #create a class of type ttk.Frame
    def __init__(self, parent, controller): #initialise the class
        super().__init__(parent)            #call the constructor of the ttk.Frame
                                            #class to inherit its and the parent's
                                            #properties
```

Listing 10: Creating and Initialising the SetupFrame Class

The fifth class is the Controller class from the controller.py file. This class is responsible for acting as a mediator between the model and the three files containing the frame classes (views). This allows for the model and views to be changed independently of one another without effecting the entire system, allowing for code to be easily scaled. The methods called in the 'command' option of the buttons in the ControlsFrame and SetupFrame classes are all methods from the Controller class. Hence, the Controller class is responsible for managing changes to the view and the application data. The controller is able to interact with other classes by means of a reference to instantiated class objects in the app.py file. This allows the controller to access and make changes to attribute data from these frame classes.

```
class Controller():                 #create the Controller class
    def __init__(self):             #initialise the class
```

Listing 11: Creating and Initialising the Controller Class

14.2.4.2 Explanation of Significant Methods and Code

Chameleon3 Camera Setup and Interaction

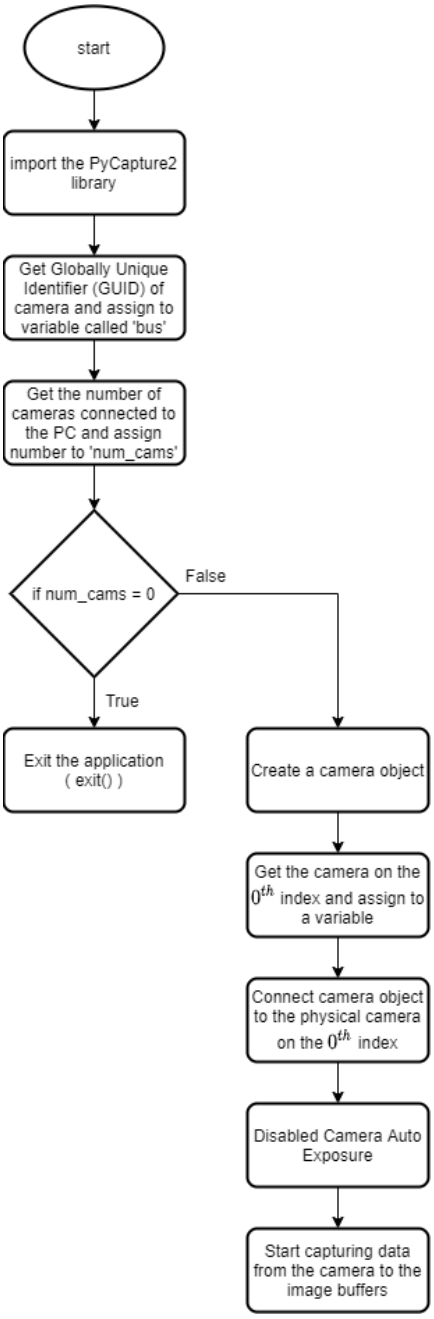


Figure 44: Flow diagram of the setup of the Chameleon3 camera

The above figure illustrates the code process to successfully setup and utilise the Chameleon3 Camera in a Python script. The code it represents is shown in listing 12 and relates to Appendix E. The process begins by identifying the GUID of the camera, which is then used to determine the number of cameras connected to the PC (lines 1 and 2). If no cameras are connected to the PC, the application is closed (lines 4-6). Next, a camera object is created (line 9) and the camera on the 0^{th} index is selected and connected to by assigning the physical camera to the camera object (lines 10-11). The autoexposure of the camera is disabled (line 14) before starting to capture data from the camera to the image buffers (line 17).

```
1:  bus = PyCapture2.BusManager()                 #identify the Globally Unique Identifier (GUID) of the camera
2:  num_cams = bus.getNumOfCameras()              #determine the number of cameras connected to the PC
3:  print('Number of cameras detected:', num_cams) #print the number of cameras detected
4:  if not num_cams:                              #if no cameras are detected exit the application
5:      print('Insufficient number of cameras. Exiting...')
6:      exit()
7:
8:  # Select camera on 0th index
9:  c = PyCapture2.Camera()             #Assign Camera object to C
10: uid = bus.getCameraFromIndex(0)     #select camera on 0th index
11: c.connect(uid)                      #connect camera object to physical camera
12:
13: # Disable On-board Image Processing
14: c.setProperty(type = PyCapture2.PROPERTY_TYPE.AUTO_EXPOSURE, onOff = False)  #Disable Auto Exposure
15:
16: print('Starting image capture...')
17: c.startCapture()                    #start capturing data from the camera to the image buffers
```

Listing 12: Chameleon3 Camera Setup

Display of real-time video and Fringe Patterns

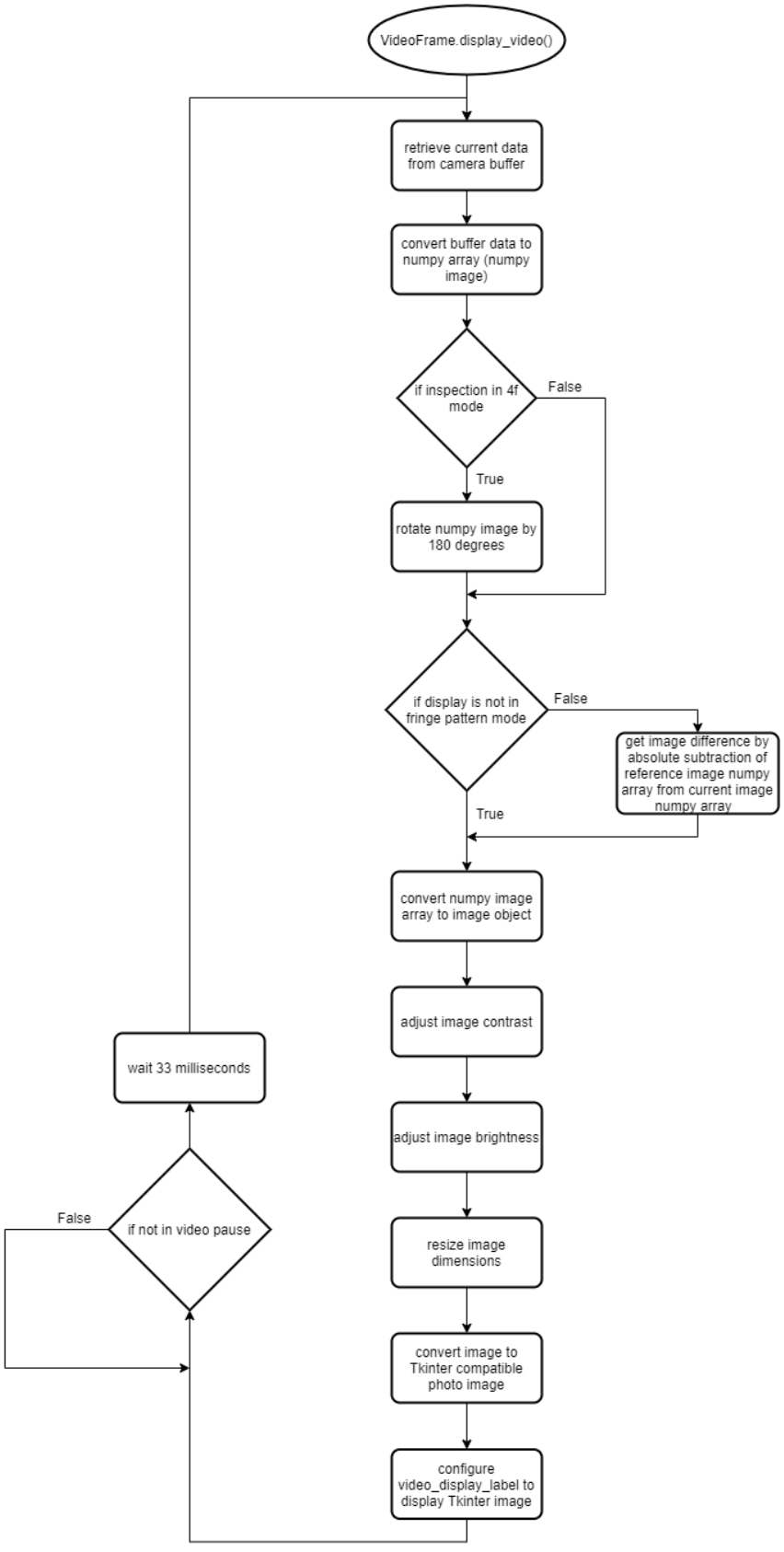


Figure 45: Flow diagram of VideoFrame.video_display() method

The video_display() method in the VideoFrame class is responsible for displaying real-time video feedback to the video_display_label. The relevant code is shown in listing 13 below. It begins by retrieving data from the image buffer (line 2) and then converts the buffer data to a NumPy array (line 3) [46]. If the inspection mode being used is 4f mode, the image is rotated by 180 degrees (lines 6-8). If the active display mode is fringe pattern mode, the reference image array is subtracted from the current image array (lines 33-44) using the OpenCV absolute subtraction method covered in section 11.3.4, if not, the image array remains unchanged. The image array is then converted to an image object (line 12/37) using a method from the Pillow library. The brightness and contrast of the image object are then adjusted (lines 15-21 and 40-46) using image enhancement methods investigated in section 11.3.4. The resulting image is then resized (line 23/48) and converted to a Tkinter compatible image (line 26/51), which is then displayed on the label (lines 29-31 and lines 54-56). If the video display has not been paused at this point, the method will be recalled after a set number of milliseconds, which is set to the refresh time of the camera, 33 milliseconds (lines 58-59).

```
1:      def display_video(self):
2:          image = self.cam.retrieveBuffer()   #retrieve data from the camera buffer
3:          cv_image1 = np.array(image.getData(), dtype="uint8").reshape((image.getRows(), image.getCols()))
4:          # Above: convert image to numpy array
5:
6:          if self.mode_flag == True:  #If 4f inspection mode is enabled, rotate numpy image 180 degrees
7:              cv_image1 = np.flip(cv_image1, axis=0)
8:              cv_image1= np.flip(cv_image1, axis=1)
9:
10:     #Display real time video or fringe patterns depending on the state of the flag
11:         if not self.display_fringes_flag:   #if fringe pattern display mode is disabled
12:             self.img = pl.Image.fromarray(cv_image1)
13:             # Above: convert numpy image array to image object
14:
15:             contrastImg = ImageEnhance.Contrast(self.img)
16:             contrastedImage = contrastImg.enhance(self.contrast_value)
17:             #Above: adjusting image to desired contrast
18:
19:             brightnessImg = ImageEnhance.Brightness(contrastedImage)
20:             Enhanced_img = brightnessImg.enhance(self.brightness_value)
21:             #Above: adjusting image to desired brightness
22:
23:             self.photo = Enhanced_img.resize((self.screen_width-700,self.screen_height-300))
24:             # Above: resize image
25:
26:             display_image = ImageTk.PhotoImage(self.photo)
27:             # Above: convert image to tkinter compatible photo image
28:
29:             self.video_display_label.config(image=display_image)
30:             self.video_display_label.image = display_image
31:             # Above: display image on tkinter label
32:
33:         elif self.display_fringes_flag==True:   #if fringe pattern display mode is enabled
34:             self.difference = cv2.absdiff(cv_image1,self.ref_image)
35:             #Above: subtract the current image from the reference image
```

```
36:
37:             self.img = pl.Image.fromarray(self.difference)
38:             # Above: convert numpy image array to image object
39:
40:             contrastImg = ImageEnhance.Contrast(self.img)
41:             contrastedImage = contrastImg.enhance(self.contrast_value)
42:             #Above: adjusting image to desired contrast
43:
44:             brightnessImg = ImageEnhance.Brightness(contrastedImage)
45:             Enhanced_img = brightnessImg.enhance(self.brightness_value)
46:             #Above: adjusting image to desired brightness
47:
48:             self.photo = Enhanced_img.resize((self.screen_width-700,self.screen_height-300))
49:             # Above: resize image
50:
51:             display_image = ImageTk.PhotoImage(self.photo)
52:             # Above: convert image to tkinter compatible photo image
53:
54:             self.video_display_label.config(image=display_image)
55:             self.video_display_label.image = display_image
56:             # Above: display image on tkinter label
57:
58:         if not self.video_pause:    #if the display is not paused
59:             self.video_display_label.after(self.delay, lambda: self.display_video())
60:             # Above: call the display_video() method after delay number of milliseconds to refresh display
61:             # to refresh display
```

Listing 13: VideoFrame.video_display() method

Reference Image Capturing

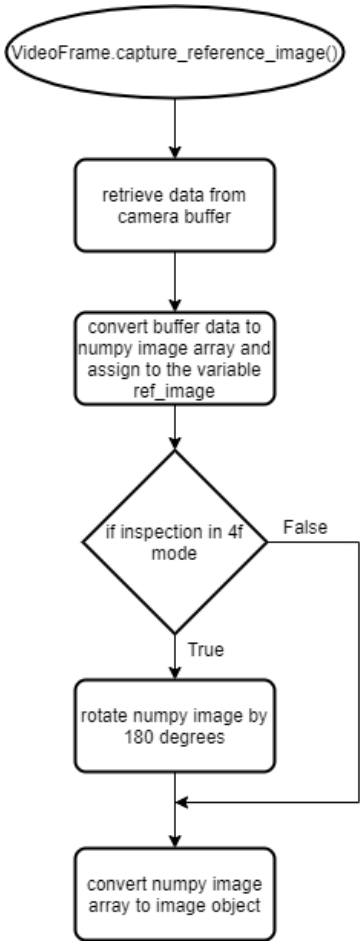


Figure 46: Flow diagram of VideoFrame.capture_reference_image() method

The capture_reference_image() method in the VideoFrame class is responsible for capturing and storing the reference image in a variable as an NumPy image array. The relevant code is shown below in listing 14. When called, it begins by retrieving data from the camera buffer (line 2). The buffer data is then converted to a NumPy array (line 3), which is then rotated 180 degrees if 4f mode is the inspection mode active (lines 6-8). Lastly, the reference NumPy image array is converted to an image object that can be saved (line 9).

```
1:      def capture_reference_image(self):      #only to be called once
2:          ref_buffer = self.cam.retrieveBuffer()  #retrieve data from the camera buffer
3:          self.ref_image = np.array(ref_buffer.getData(), dtype="uint8").reshape((ref_buffer.getRows(), ref_buffer.getCols()))
4:          # Above: convert ref_buffer to numpy array to be used in subtraction of images
5:
6:          if self.mode_flag == True:  #If 4f inspection mode is enabled, rotate numpy image 180 degrees
7:              self.ref_image = np.flip(self.ref_image, axis=0)
8:              self.ref_image = np.flip(self.ref_image, axis=1)
9:          self.ref_img = pl.Image.fromarray(self.ref_image)
10:         # Above: convert numpy image array to image object
```

Listing 14: VideoFrame.capture_reference_image() method

Saving of Image Results

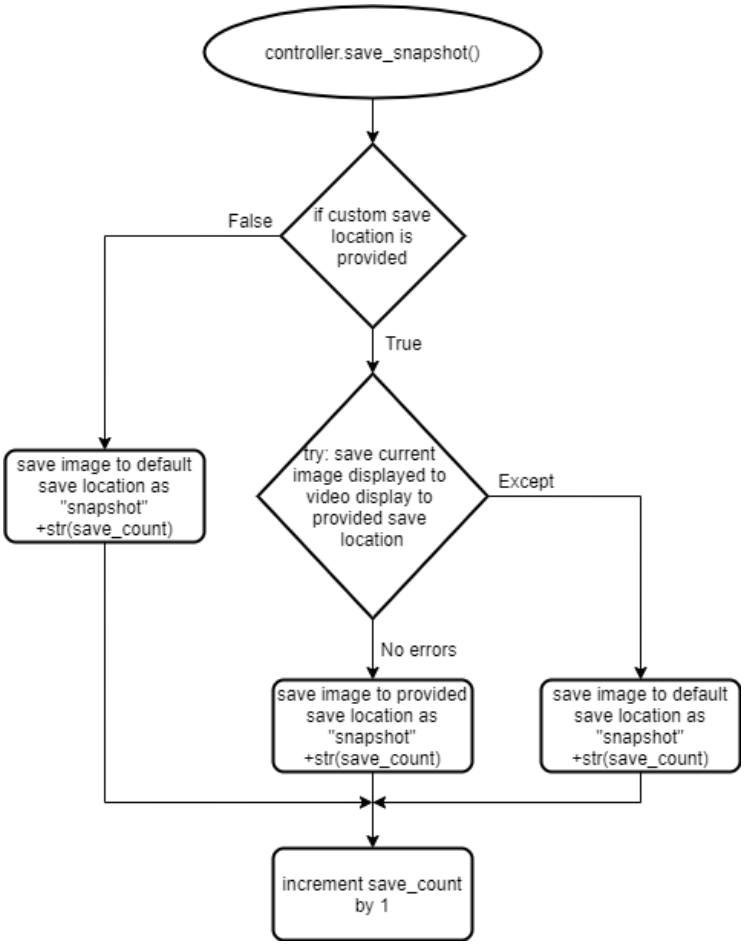


Figure 47: Flow diagram of Controller.save_snapshot() method

The flow diagram above illustrates the save_snapshot() method of the Controller class. This method is responsible for saving an image to a desired or default location when the *Snapshot Frame* button is pressed. The relevant code is given in listing 15 below. It begins by checking if the user has provided a desired save location for images (line 2). If a save location has been provided, the method tries to save the image currently shown on the display to the desired location (lines 4-5). If any error occurs, the method will save the image to the default save location (lines 7-9) and if a save location is not provided, the image is saved to the default location (lines 11-13). Lastly the image save count is incremented by one, which is used to differentiate between image saves (line 14). Other methods in the controller that involve the saving of images utilise a similar framework.

```
1:      def save_snapshot(self):    #method to save images when snapshot button is hit
2:          if self.save_location_flag:     #if a custom save location has been provided
3:              try:                        #try save the image to the provided location
4:                  snapshot_name = self.save_location+"\\snapshot"+str(self.save_count)+".jpeg"
5:                  self.video_frame.photo.save(snapshot_name)
6:
7:              except:                     #if an error occurs, save the image to the default save location
8:                  snapshot_name = "snapshot"+str(self.save_count)+".jpeg"
9:                  self.video_frame.photo.save(snapshot_name)
10:
11:         else:                           #else, save the image to the default save location
12:             snapshot_name = "snapshot"+str(self.save_count)+".jpeg"
13:             self.video_frame.photo.save(snapshot_name)
14:         self.save_count = self.save_count+1     #increment the save count by 1
```

Listing 15: Controller.save_snapshot() method

The code and methods explained above are the most significant for the success of the GUI application. However, there are other methods that contribute to the functionality of the developed application, as shown in appendix F.

15 Planning

The effects of the COVID-19 pandemic have made project planning exceptionally important, as access to resources and logistics that would normally be taken for granted have been limited. To assist with project planning and to ensure that the project is completed in time, a Gantt chart and Work Breakdown Structure were developed, which can be seen in appendix A. The pandemic also prevented supervisors and students from meeting in person, however alternatives ways to communicate were used, such as email and WhatsApp. The important project milestones are listed below.

Table 12: Project Milestones and Dealines

Milestone or Deadline	Date
Project topic allocation	24/02/2020
Weekly meetings begin	28/02/2020
Interim Report Submission	19/06/2020
Online drawing checks	03/08/2020 – 14/08/2020
Drawings submission	21/08/2020
Lab become accessible	07/09/2020
Final Report Submission	13/11/2020
Online oral presentation submission	01/12/2020

16 Experimental Details and Testing Procedure

There are several different criteria that the final product needs to fulfil, as defined in the PRS. Namely, the ability of the product to maximise the camera field of view, the ability of the product to manage the shearographic inspection process and display the fringe pattern results and the functionality of the GUI application need to be tested. Different testing procedures are to be used to check for the improvement of the field of view and the ability of the product to manage the shearographic inspection process and display the fringe pattern results obtained. The GUI application is utilised in both testing procedures and the performance of the GUI will assist in identifying its conformance to the PRS. Since the objective of this project is to improve upon an existing inspection system, it is only natural that the developed inspection device is compared to the existing one.

16.1 Methodology for determining the Field of View capability of the Inspection Device

The following details the method undertaken to determine the field of view of the developed inspection prototype (and the existing prototype). For this process, a grid of known dimensions is to be used to physically measure the field of view of each of the prototypes at different working distances. A section of this grid, which consists of 10 mm by 10 mm blocks is shown below:

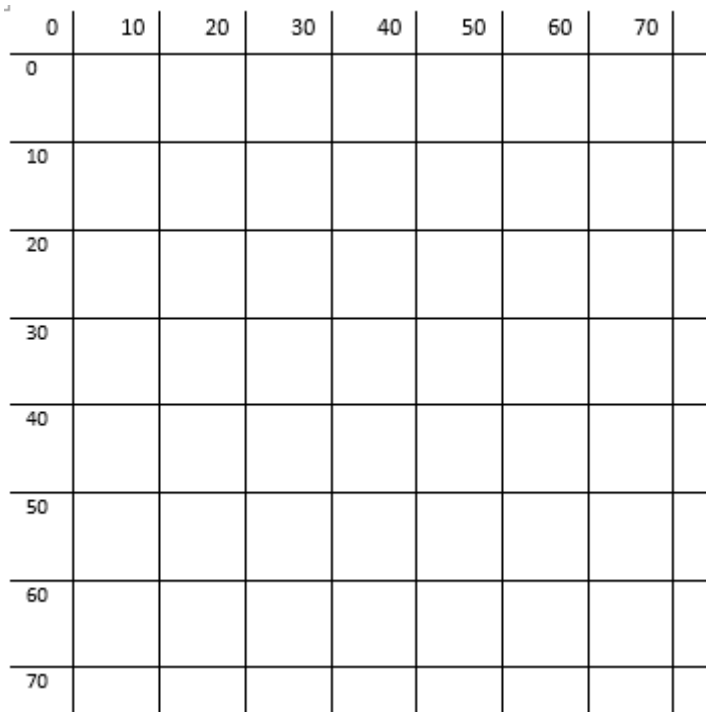


Figure 48: Grid used for determining Field of View of Inspection Prototypes

The method is as follows:

1. The inspection prototype is placed a known working distance away from the gridded sheet such that the light entry point of the prototype is pointing at the grid.

2. Connect the CCD camera to the PC via the USB 3.0 cable and launch the inspection GUI application (app.py).

3. Ensure that the video display of the GUI shows the grid sheet.

4. Click the *Snapshot Frame* button to save the frame displaying the grid sheet.

5. Count the number of grid blocks that can be seen in the saved image.

6. Using the number of blocks determine the magnitude of the horizontal and vertical field of view (in mm).

7. Knowing the working distance (in mm) and the magnitude of the field of view, equation 5 is then used to determine the angular field of view of the inspection device:

$$FOV = 2(WD)\tan\frac{AFOV}{2}$$

$$FOV - Field\ of\ View\ (mm)$$
$$WD - Working\ Distance\ (mm)$$
$$AFOV - Angular\ Field\ of\ View\ (°)$$

8. Repeat this process for both the newly developed and the existing inspection prototypes and compare their respective achieved angular field of views.

16.2 Methodology for Investigating the ability of the Inspection System to manage the shearographic inspection process and display fringe pattern result

The following details the method necessary to prepare the Shearographic system and perform Non-destructive tests. This method was used to test for the presence of known defects in a given specimen to confirm whether the designed prototype and the developed GUI application could meet their Product Requirement Specifications. The test specimens used for the testing are cross sections of helicopter rotor blades consisting of a carbon fibre skin filled with a Nomex honeycomb that has been resin dipped. The defects that have been introduced are circular cut-outs, as shown in figures 49 and 50.

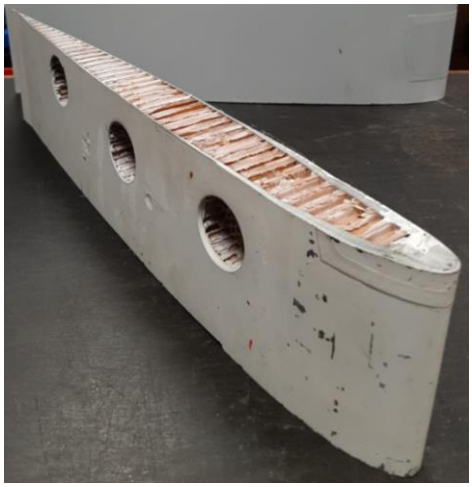


Figure 49: Test Specimen 1 with circular cut-outs shown

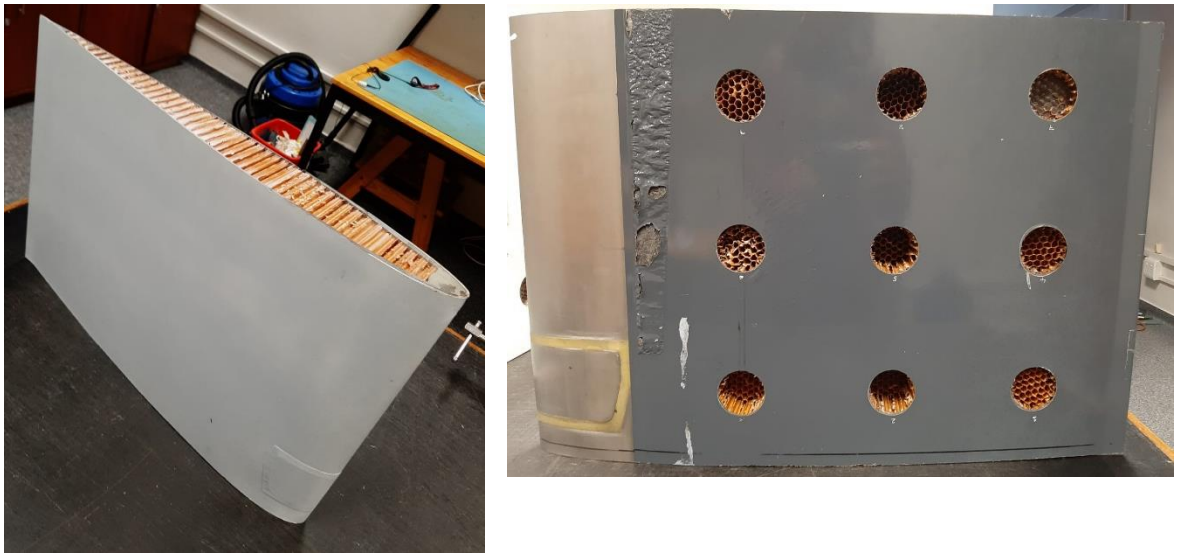


Figure 50: Test Specimen 2 with circular cut-outs shown (right image)

It should be noted that several factors will affects the inspection conditions. These include the size, shape and colour of the object, as well as the presence of any vibrations and the type of stressing applied to the object [47]. It is also extremely important that the operator does not ever look directly into the laser light source.

1. The inspection prototype is placed such that the light entry point of the prototype is facing the given inspection specimen.

2. Connect the CCD camera to the PC vis the USB 3.0 cable and run the inspection GUI application (app.py).

3. Upon application launch, use the app's real-time/live video display to ensure that there is a view of the desired inspection area.

4. Point the laser light source at the inspection specimen and turn it on. Wait a couple of minutes for the laser to heat up to operating temperature and to stabilize with environmental conditions. Ensure that the laser is adequately illuminating the specimen.

5. On the right side of the GUI, under Inspection Setup, select the type of inspection mode that is to take place. If a 4f based device is being used, select '4f' otherwise keep inspection mode on 'normal'.

6. Set the save location of images by pasting the directory of the desired save location folder into the entry provided.

7. Configure the inspection setup in the app to optimise the inspection process and set the camera variables/image enhancements (brightness, contrast) to display the desired image, or make use of the configuration file 'config.ini' and the initialisation import feature to automatically configure set parameters. The image enhancement values may be changed throughout the inspection procedure.

8. Introduce some image shear on the inspection prototype by turning either of the adjustment bolts at the back of the prototype. It is always best to shear the image in either the horizontal or vertical direction only. The amount that these bolts are turned corresponds to the magnitude of the image shear and the real-time video display can be used to ensure that the desired image shear magnitude is set.

9. After setting the image shear and the software-controlled inspection parameters, the inspection procedure is started by clicking the *START INSPECTION* button.

10. Once the reference image is taken (either immediately after starting the inspection or at a set time specified during the inspection setup), the *Fringe Pattern Mode* button at the top the application can be pressed to display the shearogram.

11. At this point the object should be stressed using thermal, mechanical, pressure or vibration stressing, depending on the composition and size of the object being tested. In this case, heat is applied to the back of the test specimen using a heating lamp.

12. As the specimen is stressed, resulting fringe patterns displayed on the computer monitor indicate the presence of defects. The frames displaying these fringe patterns can then be saved by using the *Snapshot Frame* button in the top left of the GUI or by setting an interval between which images are automatically saved, prior to beginning the inspection.

13. The inspection procedure is ended by pressing the *Pause* button followed by the *Stop* button and the inspection procedure can be reset by clicking the *New Inspection* button.

17 Results

A shearographic inspection device based on the final design section 13.4 has been constructed to overcome the field of view limitations of the existing inspection prototype. The device incorporates a 4f system with a traditional Michelson Interferometer based shearography device to create a system whose field of view is only dependent on the imaging lens and camera being used. The device features an 8 mm focal length Kowa LM8JCM imaging lens and the Chameleon3 camera with a $\frac{1}{3}$ inch CCD inside. Theoretically, the angular field of view of the device is approximately $33.4°$ in the horizontal direction and $25.4°$ in the vertical direction. However, this can be increased by making use of a C-mount imaging lens with a shorter focal length.

The constructed device has an approximate weight of 2 kg and has dimensions (Length × Width × Height) of 212 mm × 164 mm × 72 mm.

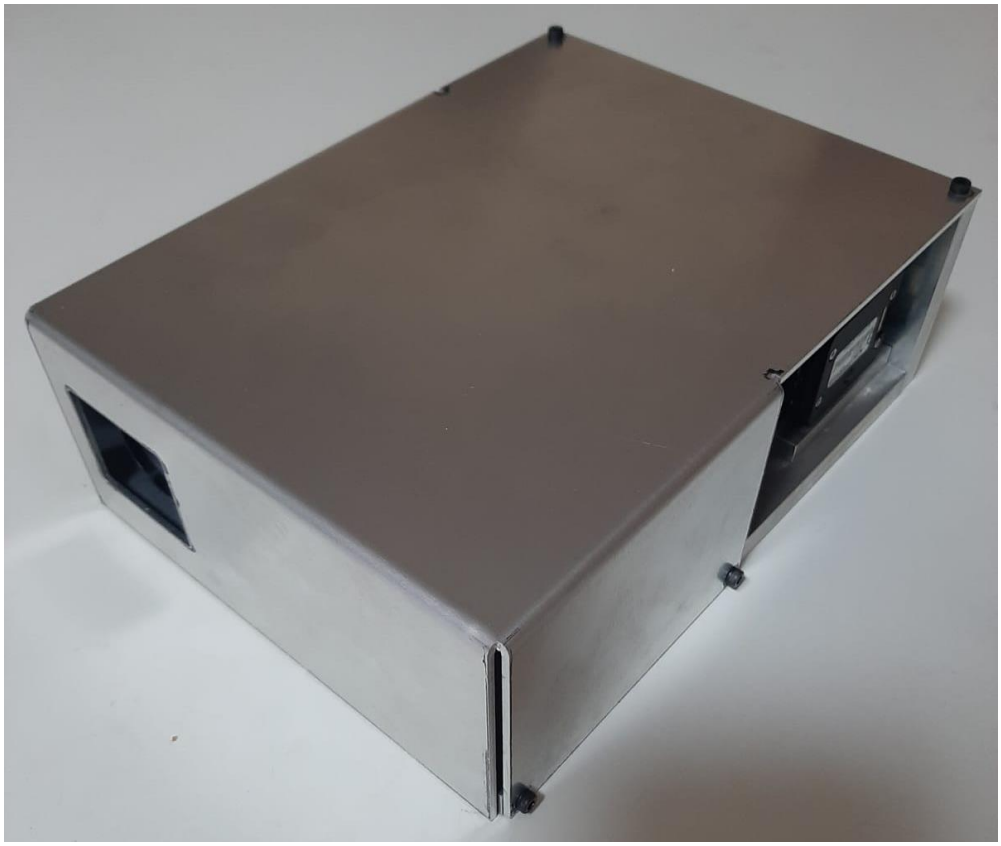


Figure 51: The constructed inspection device with the cover attached

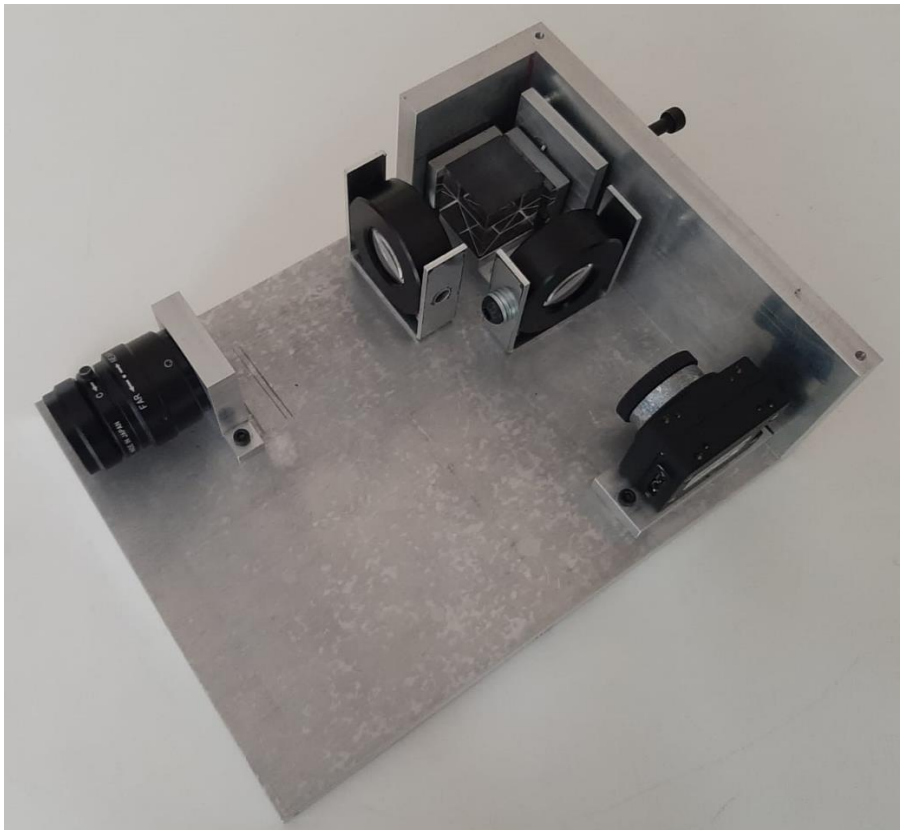


Figure 52: The constructed inspection device without the cover

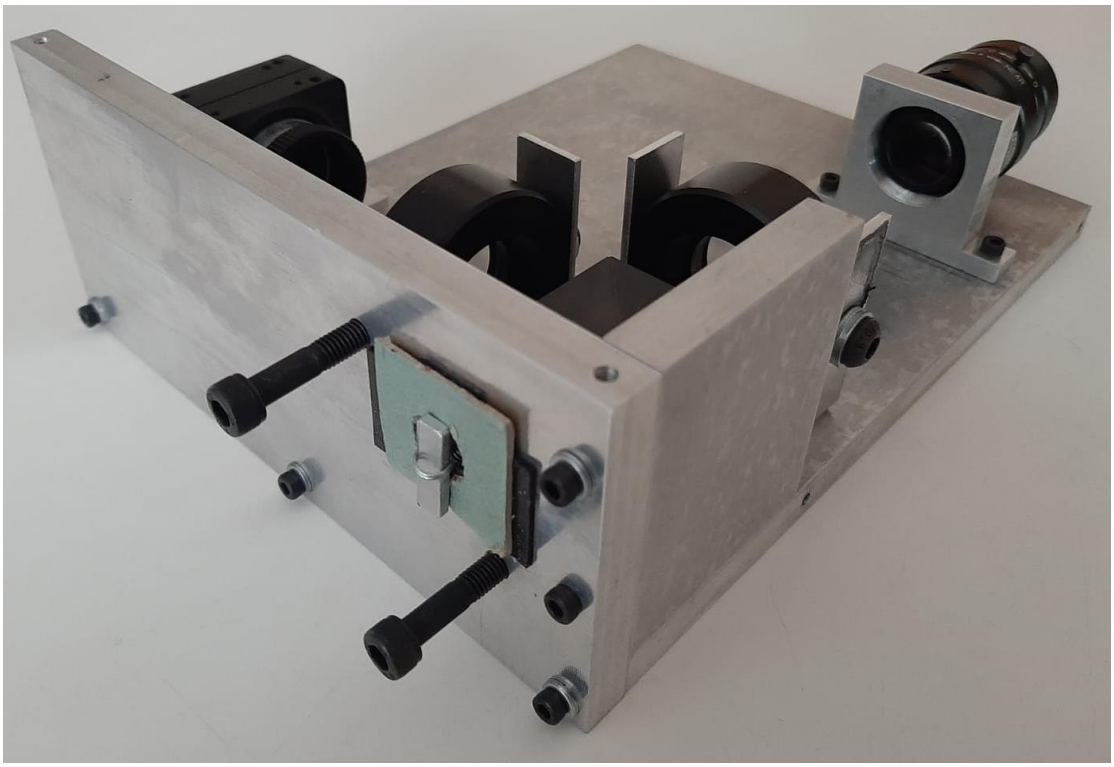


Figure 53: The constructed inspection device rear view showing shearing bolts

17.1 Overall System Results

It is quickly apparent that there is a decrease in quality and brightness of the resulting image displayed in the GUI when the newly constructed inspection device is used over the already existing inspection device. However, this was to be expected and reasons for it are explained in the discussion. This means that higher brightness and contrast values need to be set in the GUI to more easily identify fringe patterns and defects when conducting inspections with the newly constructed inspection device.

17.2 Field of View Results

The field of view of the inspection devices was determined using a working distance of approximately 200 mm and the testing procedure explained in section 16.1. It should be noted that the testing conditions were consistent and both devices were utilising the same imaging lens and camera.

At this working distance, the existing inspection device provided an approximate field of view of eight whole blocks (80 mm) in both the horizontal and vertical direction, as shown below. As per equation 5, this corresponds to an approximate angular field of view of $22.6°$.

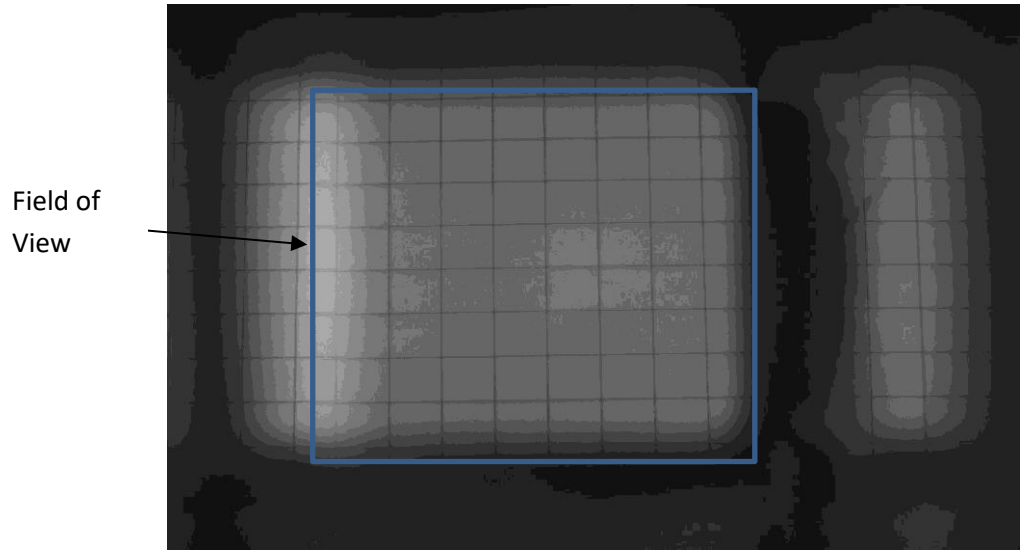


Figure 54: Field of view of existing inspection device at 200 mm working distance

At the same working distance, the new inspection device provided an approximate field of view of 12 blocks (120 mm) in the horizontal direction and nine whole blocks (90 mm) in the vertical direction, as shown below. As per equation 5, this corresponds to an approximate angular field of view of $33.40°$ and $25.36°$ in the horizontal and vertical directions respectively.

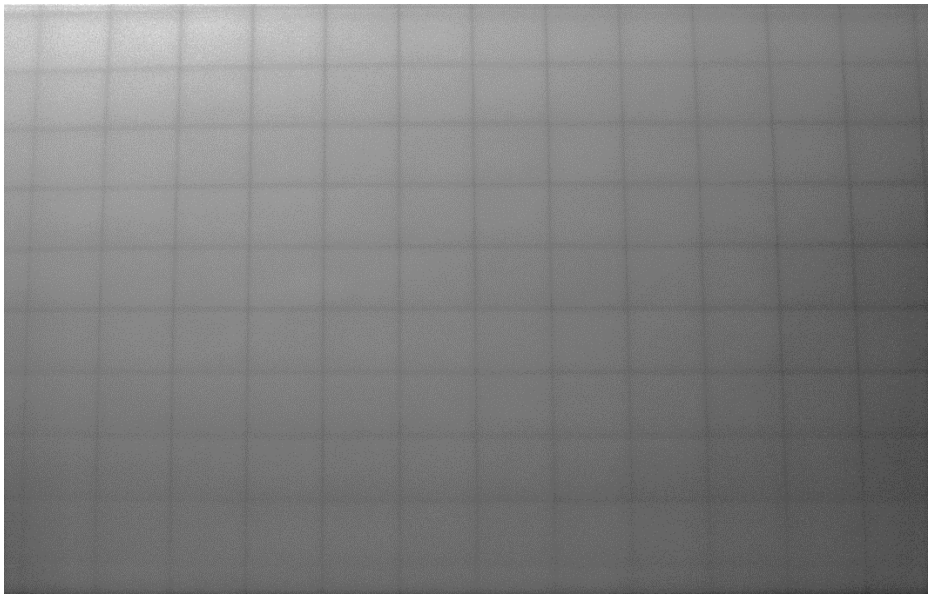


Figure 55: Field of view of inspection device with 4f system at 200 mm working distance

17.3 Shearographic inspection process and resulting fringe patterns

A series of comparison experiments using the testing methodology laid out in section 16.2 have been conducted on the provided test specimens using the developed and existing inspection devices. These experiments compare the ability of the two inspection devices to produce the fringe patterns resulting from out-of-plane deformation of the test objects. The results of these comparison experiments, along with their set inspection parameters and camera variable values are explored in the following section.

The first time this inspection procedure was run using the newly developed inspection device, it was unable to locate the object defects by producing and displaying the fringe patterns that would have resulted. The reason for this was identified to be the slight incorrect positioning of the imaging lens, 4f lenses and the camera, due to the play in the points that fasten these components to the base as a result of missing tolerances. This resulted in the focal planes of these components not coinciding, critically decreasing the quality of the image on the detection (camera sensor) plane, to the point where the testing object could not be accurately viewed on the GUI display. This was rectified to a degree by incrementally adjusting the position of each optical component until the image quality was satisfactory.

Inspections conducted on test specimen 1:

The first set of comparison experiments were conducted on test specimen 1. Both inspection devices were placed at a working distance of approximately 300 mm from the desired inspection area. The used inspection parameters and camera variables of each inspection system are listed below

Table 13: The used inspection parameters and camera variable values for experiment 1

Inspection Parameter	New Inspection Device (with 4f system)	Existing Inspection Device (without 4f system)
Inspection time limit (min)	None	None
Frame Capture Interval (s)	1	1
Reference Image Delay (s)	0	0
Brightness	2.0	1.0
Contrast	2.0	1.0

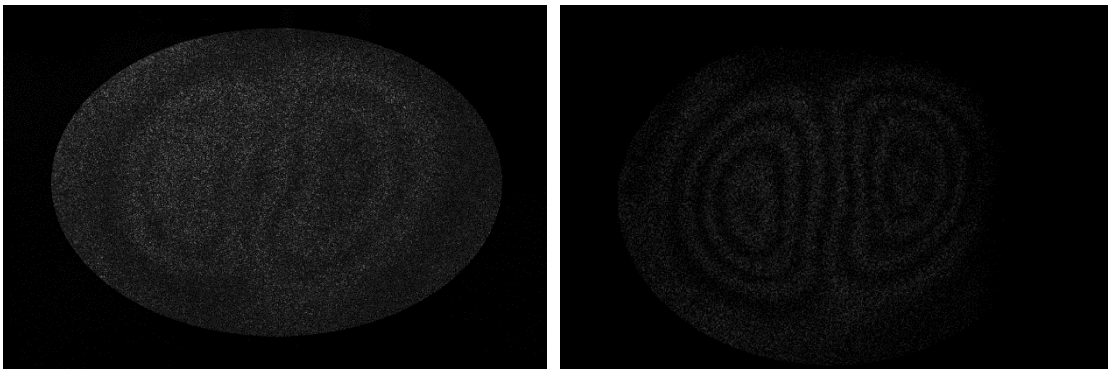


Figure 56: Comparison experiment 1 showing the real-time subtraction using Digital Shearography with a 4f system (left) and real-time subtraction using digital shearography without a 4f system (right)

While both systems can identify the defect, the lower image quality of the 4f system impacts the quality of the resulting fringe patterns, as it is unable to produce the fringe patterns with the same quality as the existing inspection system.

Inspections conducted on test specimen 2:

The next comparison experiment was conducted on test specimen 2, placing both inspection devices at a working distance of approximately 220 mm away from the desired inspection area.

Table 14: The used inspection parameters and camera variable values for experiment 2

Inspection Parameter	New Inspection Device	Existing Inspection Device
Inspection time limit (min)	None	None
Frame Capture Interval (s)	1	1
Reference Image Delay (s)	0	0
Brightness	3.0	1.0
Contrast	1.5	1.0

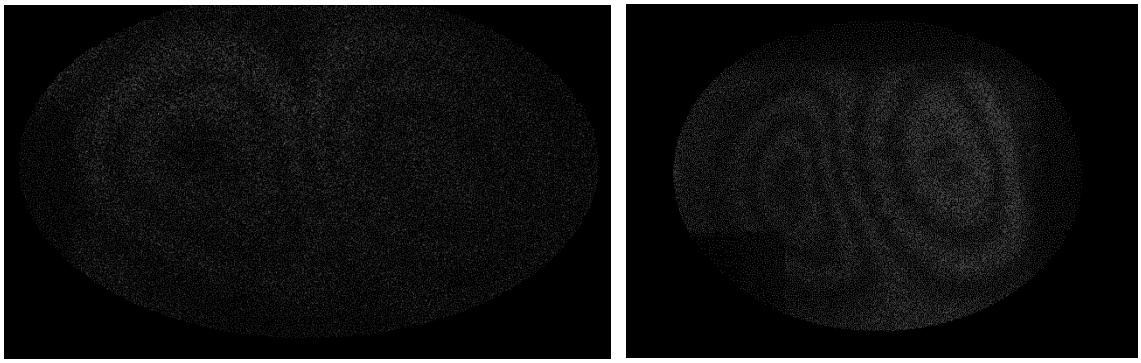


Figure 57: Comparison experiment 2 showing the real-time subtraction using Digital Shearography with a 4f system (left) and real-time subtraction using digital shearography without a 4f system (right)

As in experiment 1, both systems can identify the defect. However, the new inspection system is unable to produce fringe patterns with the same clarity as the existing inspection system.

Lastly, a comparison experiment was conducted on test specimen 2 where both inspection devices were placed at a working distance of approximately 450 mm away from the specimen.

Table 15: The used inspection parameters and camera variable values for experiment 3

Inspection Parameter	New Inspection Device	Existing Inspection Device
Inspection time limit (min)	None	None
Frame Capture Interval (s)	1	1
Reference Image Delay (s)	0	0
Brightness	4.0	1.0
Contrast	2.0	1.0

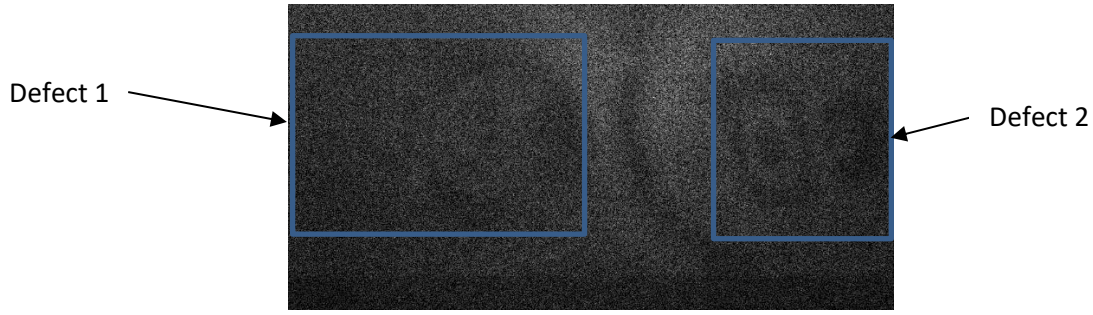


Figure 58: Comparison experiment 3 showing the real-time subtraction using Digital Shearography with a 4f system

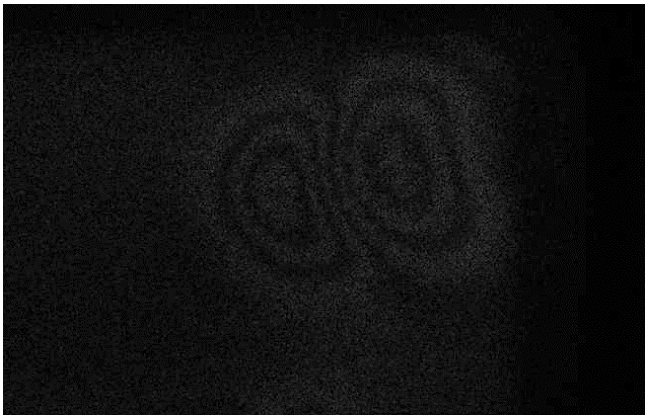


Figure 59: Comparison experiment 3 showing the real-time subtraction using Digital Shearography without a 4f system

At a working distance of 450 mm the new inspection device can identify two of the defects present in the test specimen, however the existing inspection device is only able to identify one. This highlights the larger field of view capabilities of the 4f system. The existing inspection device was only able to view both defect areas when placed at a working distance of 750 mm away from the test specimen.

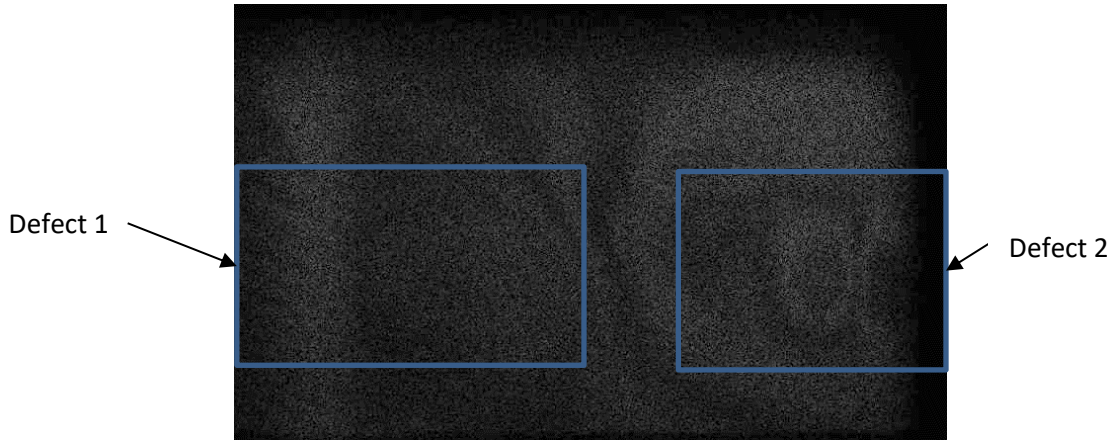


Figure 60: Real-time subtraction using Digital Shearography without the 4f system at a working distance of 750 mm

When trying to detect both defects, it could be argued that the quality of the resulting fringe patterns is superior in the device incorporating the 4f system by comparing figures 58 and 60.

17.4 Graphical User Interface

As can be seen from the results above, the GUI is able to display the fringe pattern results, however the image quality is dependent on the inspection device and working distance being used. The GUI is also able to save the image results obtained, as shown in the example below. In this case the images were saved to the desired save location at one second intervals during the inspection.

Name	Date	Type	Size	Tags
interval_capture0	2020/10/30 11:34 AM	JPEG File	134 KB	
interval_capture1	2020/10/30 11:34 AM	JPEG File	135 KB	
interval_capture2	2020/10/30 11:34 AM	JPEG File	37 KB	
interval_capture3	2020/10/30 11:34 AM	JPEG File	88 KB	
interval_capture4	2020/10/30 11:34 AM	JPEG File	96 KB	
interval_capture5	2020/10/30 11:34 AM	JPEG File	105 KB	
interval_capture6	2020/10/30 11:34 AM	JPEG File	110 KB	
interval_capture7	2020/10/30 11:35 AM	JPEG File	112 KB	
interval_capture8	2020/10/30 11:35 AM	JPEG File	111 KB	
interval_capture9	2020/10/30 11:35 AM	JPEG File	103 KB	
reference_image	2020/10/30 11:34 AM	JPEG File	57 KB	

Figure 61: GUI Saved Inspection Images Example

The GUI application can also successfully control the camera variables, brightness and contrast. The figure below shows two images of an identical view. The one image is unaltered, whereas the other image has increased contrast (a contrast input value of 3).

Figure 62: Example of GUI Contrast altering capabilities

18 Discussion

18.1 Overall Comments

Relating back to the PRS, the aim of this project was, firstly, to produce a portable shearography inspection device that is capable of maximising its camera field of view and performing inspections to acquire qualitative defect information of an object, and secondly, to develop a user-friendly GUI application that is able to manage the inspection process and display fringe patterns resulting from the real-time subtraction of images using Digital Shearography. The results of the project will now be discussed in relation to these aims. At the end of this section, the conformance of two deliverables with their product requirement specification is discussed to determine the overall success of the project.

The first difference noticed between the images obtained using the inspection device with the 4f system and the existing inspection system is the decrease in brightness of the images obtained using the device incorporating the 4f system. This result was expected as one of the disadvantages of using a 4f system is the decreased brightness of the image detected at the camera sensor, as the mirrors act as spatial filters in a 4f system which causes an uneven intensity distribution, as explained in section 11.2.2. This effect can be limited by using larger mirrors and the ability of the GUI to alter the brightness of the image may be used to overcome the issue.

18.2 System Field of View

It can be said that the main objective of this project was to modify the optical shearing geometry of the existing shearography device to maximise its camera field of view, which prompted the development of a device using the 4f system to overcome the limitation of the Interferometer. The table below summarises and compares the approximate angular field of view results of the two inspection devices and compares these results to calculated theoretical values.

Table 16: Summary comparing field of view results of the Shearography devices with and without a 4f system

	Inspection device with 4f system		**Inspection device without 4f system**	
	Theoretical	Actual	Theoretical	Actual
Horizontal AFOV (◦)	33.4	33.4	21.1	22.6
Vertical AFOV (◦)	25.4	25.36	21.1	22.6

These results obtained were expected and they closely resembled the calculated theoretical results. This confirms the ability of the 4f system to maximise the system field of view by overcoming the limitations of the Michelson Interferometer and ensure that the AFOV of the system is only dependent

on the imaging lens and camera being used. Using the same imaging lens and camera, the device utilizing the 4f system provides AFOV increases of almost a 48 percent in the horizontal direction and over 12 percent in the vertical direction. This highlights the superiority of the system with the 4f system over a system without it, with regards to their achieved field of view. It should be reemphasized that the AFOV of the device with the 4f system may be altered by simply swapping out the imaging lens for another with a different focal length, which may be beneficial in some applications.

18.3 Shearographic inspection process and resulting fringe patterns

In order to be able to detect defects/flaws within an object, the inspection system needs to be able to conduct the shearographic inspection process and produce and display the resulting fringe patterns resulting from the real-time subtraction of images using Digital Shearography. As can be seen in the results section, both inspection devices, with and without the 4f system, are able to produce the resulting fringe patterns. However, the 4f-based inspection system was unable to produce the fringe patterns with the same quality and clarity as the existing inspection system. This can be attributed to the increased complexity of the 4f-based inspection system, along with the time and monetary limitations of the project, as explained in the following. The reason for the lower image quality in the 4f-based inspection system is likely a combination of three factors. The first is the imperfect placement of the optical components due to the lack of tolerances at the points where these components fasten to the base of the design. Due to the high sensitivity of optical components, it was necessary that these components were placed exactly according to the design, such that their respective focal planes coincided exactly. For every component that is mispositioned, another source of error is introduced to the system, further decreasing the quality of the image at the detector plane. However, it would not have been possible to tolerance all these dimensions as they would have greatly increased the manufacturing cost of the inspection system and an external supplier would have been needed to manufacture most of the designed components. Even after having identified the issue, there was inadequate time available to resolve the issue by redesigning and manufacturing components. Secondly, although bracket was intended to prevent tilting of the lens mounts, due to a slight deviation from the design in the manufacturing of the bracket, the edge of the lens mount did not sit flush against the bottom bracket, allowing for slight tilt. If one of the lenses is tilted, the focal point of the lens may be altered, having a negative effect on the quality of the image produced. Due to time limitations, a new part to hold the mounts could not be redesigned and manufactured in time to perform additional testing. Thirdly, the presence of stray light entering the system could also have had

negative effects on the results, as it decreases the signal to noise ratio of the system, decreasing the resultant image quality. This can be rectified by coating the surface of to reduce reflections of stray light within the device. There was insufficient time available to complete this, however it should be applied in future as highlighted in the recommendations.

Regardless of the decreased image quality, the 4f-based inspection system is still able to perform the Digital Shearography process and produce fringe patterns to locate object defects and ultimately it is a success in that regard as it can perform the primary function of a Digital Shearography inspection system, even though it can be improved upon.

18.4 GUI Performance

The GUI performed as expected. It is not only able to perform all its necessary functions, but it performs them efficiently with no delay or latency between operations. It requires little cognitive load and is very simple to operate, while providing several functions that are likely to be useful during future NDT inspections, regardless of the inspection device being used. Being developed in Python means that this programme is highly accessible, as it can be used on any device with Python installed, including a Raspberry Pi, so long as the libraries and drivers required for the Chameleon3 camera are installed. The process for the installation of these requirements as well as how to use the programme is explained in the README provided in the appendix and the application folder. Time limitations meant that some desirable functions that were unrelated to the scope could not be implemented but are mentioned in the recommendations.

18.5 Conformance with the PRS

To determine the overall success of the project, the products developed are compared with the PRS list that was drawn up to ensure that the goals set are met. With regards to the review and improvement of the existing optical shearing geometry, this new inspection device meets each of the PRS. The product is successful in maximising the camera field of view, it is able to alter the magnitude and direction of the image shear using the two adjustment bolts, and being approximately 2 kg, the device is portable and small enough to hold in a hand or store in a small bag. After performing a budget analysis, the requirement of the inspection device needing to remain within the allocated budget was removed, which was approved by the supervisor.

With regards to the development of the GUI application, all the PRS are met. The developed GUI can manage the shearographic inspection process and display the fringe pattern results obtained, as well as a display of the real-time camera view window. Image results can be easily saved at random points or at set intervals to a desired location set by the user. The application also provides the ability to alter the brightness and contrast of the image displayed, as well as the ability to set desired inspection parameters. The GUI is simplistic and easy to use, yet it performs many desirable functions that someone well-acquainted with the Digital Shearography inspection process would want. Visually, the GUI is simplistic, yet appealing. It remains consistent with the colours and fonts used in styling the application, making it a comfortable and user-friendly application to utilise in a NDT inspection process.

Although the final product is not perfect, it fulfils the aims and objectives set in the PRS. Therefore, the overall project is named a success.

19 Recommendations

Due to the time and monetary limitations of the project there are many recommendations that could be considered to improve the results of this product or to conduct a future iteration of this project, building upon what has been developed.

19.1 Overall System Recommendations

As mentioned in section 9.3, the developed inspection system is to be limited to gathering qualitative data, namely identifying the location of defects or flaws in an object due to the out-of-plane deformation gradient. However, Digital Shearography can be used to directly measure strain quantities. While traditional real-time subtraction Shearography (which has been utilised for this project) is rarely used to determine strain quantities in industry due to its low measurement sensitivity of a single fringe (equal to a phase change of 2π radians), phase-shift Digital Shearography techniques can be used to accurately determine strain data quantities as the measurement sensitivity of these techniques is at least 10 times higher than when using traditional real-time subtraction [1]. There are two types of phase-shift techniques used for Digital Shearography, namely temporal phase-shift Digital Shearography and spatial-phase Digital Shearography. Temporal phase-shift is used in applications where high accuracy measurements of static or quasi-static objects is required, while spatial phase-shift is used in applications that require dynamic measurements [1]. It is recommended that a future iteration of this project incorporates one of these techniques such that accurate quantitative strain data may be acquired.

19.2 Design Recommendations

To reduce stray light that enters the inspection device from effecting the resulting images, a black coating should be applied to the surface of the components that support the optical components of the system. A black coating, which is typically anodised onto the aluminium, would reduce the signal to noise ratio of the inspection optical system by adsorbing much of the stray light. This could improve the resultant image received at the detection plane [48].

The slight misplacement of the optical components due to the lack of tolerancing was identified as one of the issues that resulted in the poor image quality when using the inspection device. Rectifying this by tolerancing the various mounts containing the optical components, and the base, would be highly expensive, therefore another solution is required. It is recommended that the base be redesigned to allow for component positioning to be adjusted. A potential solution could be the inclusion of a slot in the base and the redesign of the imaging lens, brackets and camera mount to fit into the slot, as slider mechanisms. This would allow for the positioning of the optical components to

be altered until their respective focal planes coincide, at which point they would be fastened to the base, resulting in high quality image.

A new part should be designed to fasten the 4f lens mounts to the base, as the designed brackets allowed slight tilting of the lenses. The new part should ideally enclose the lens mounts, ensuring no rotation or translation of the mounts can occur.

The requirement of a laser light source with a relatively small coherence length means that relatively small laser diodes can be used for Digital Shearography applications, as mentioned in section 11.1.2. The small size of these diodes makes them suitable for using in portable Digital Shearography prototype. It is recommended that a laser diode, along with its necessary heat regulation components, should be incorporated into the Shearography inspection device such that all equipment necessary for a NDT inspection process is packed into a single portable device.

19.3 Software Recommendations

In recent times, Machine learning methods and algorithms have been applied in quality control and predictive maintenance applications. As per *Expert System*, "Machine learning is an application of artificial intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed [49]." The ability of Machine learning algorithms to learn from training data and locate patterns in unseen or new data makes it suitable for Non-Destructive Testing [50]. This can be used to automatically process images and detect patterns in different NDT methods, such as optical, ultrasonic or X-ray. In the case of Digital Shearography inspections, it would require the use of deep learning methods, such as Region Based Convolutional Neural Networks (R-CNN) models, to automate the recognition of typical speckle fringe patterns, increasing the efficiency of the overall inspection process for defects [51]. This requires these models to be trained on existing data/samples of images containing fringe patterns. The number of different samples that are used in training the model directly correlates to its effectiveness in automatically identifying the fringe patterns when deployed. Training the model with too few samples could result in reduced effectiveness and false recognition of fringe patterns [51]. If Shearograms are obtained for various parts of the object being inspected using a successfully trained Machine learning model, a full-scale image map of the defect distribution in the object can be obtained by using image stitching. An example is shown in the figure below.

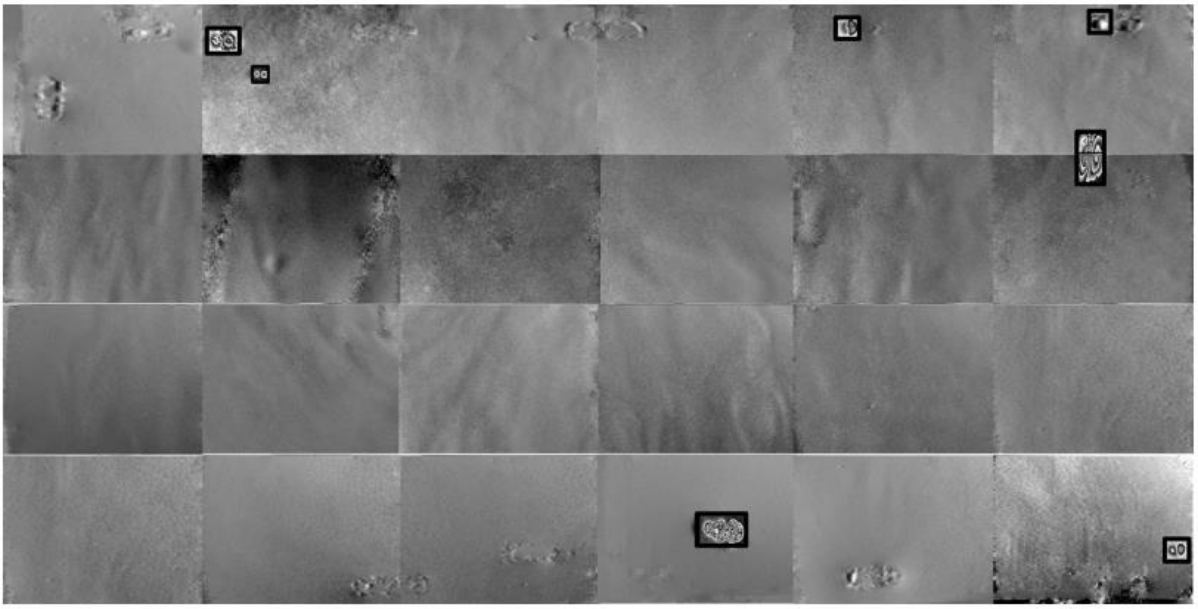


Figure 63: The defect distribution over the entire surface of an object that was inspected using automatic recognition of defects [51]

It is recommended that a machine learning algorithm be developed and be incorporated into the GUI application to assist with and improve the efficiency of the detection of the defects in specimens.

20 Conclusion

The aims and objectives of this project to maximise the camera field of view of a Digital Shearography system and to develop a user-friendly GUI to manage the inspection process and provide various desirable functions were accomplished, making the project a success. This report details how the camera field of view of a Digital Shearography system was maximised by incorporating a 4f system that overcomes the limitations of the Michelson Interferometer shearing device, increasing the horizontal and vertical angular field of view of the inspection system by approximately 48 and 12 percent respectively. It was found that while the 4f system maximises the camera field of view, its complex design makes it difficult to achieve a high-quality image, effecting the quality of the fringe patterns produced during the inspections. The process of successfully designing and implementing a user-friendly GUI to assist with identifying defects in object due to out-of-plane deformation gradients was also covered. Lastly, recommendations to improve the developed inspection system were made as well as recommendations for future development of the Digital Shearography prototype.

21 References

[1] Q. Zhao, X. Dan, F. Sun, Y. Wang, S. Wu, and L. Yang, “Digital shearography for NDT: Phase measurement technique and recent developments,” *Appl. Sci.*, vol. 8, no. 12, 2018, doi: 10.3390/app8122662.

[2] D. Findeis and J. Gryzagoridis, “Inspection of Aircraft Components With the Aid of Portable Digital Shearography,” 1995.

[3] “What is Non-Destructive Testing (NDT)? Methods and Definition - TWI.” https://www.twi-global.com/technical-knowledge/faqs/what-is-non-destructive-testing (accessed Apr. 09, 2020).

[4] D. Findeis and J. Gryzagoridis, “<title>A comparison of the capabilities of portable shearography and portable electronic speckle pattern interferometry</title>,” *Nondestruct. Eval. Heal. Monit. Aerosp. Mater. Compos. III*, vol. 5393, pp. 41–49, 2004, doi: 10.1117/12.539731.

[5] Y. Y. Hung, “Applications of digital shearography for testing of composite structures,” *Compos. Part B Eng.*, vol. 30, no. 7, pp. 765–773, 1999, doi: 10.1016/S1359-8368(99)00027-X.

[6] J. B. Deaton, Jr. and R. S. Rogowski, “<title>Electronic shearography for nondestructive evaluation: the influence of the field of view and the shearing angle</title>,” in *Industrial Optical Sensing and Metrology: Applications and Integration*, 1993, vol. 2066, pp. 2–13, doi: 10.1117/12.162099.

[7] H. Feng, J. Zhang, and X. Liu, “Studies on digital shearography for testing of aircraft composite structures and honeycomb-based specimen,” *Appl. Mech. Mater.*, vol. 121–126, no. c, pp. 1264–1268, 2012, doi: 10.4028/www.scientific.net/AMM.121-126.1264.

[8] B. A. Bard, S. Wu, and B. R. Tittmann, “A Compact Digital Phase-Stepping Shearography System,” *Rev. Prog. Quant. Nondestruct. Eval.*, vol. 15, pp. 673–677, 1996, doi: 10.1007/978-1-4613-0383-1_87.

[9] D. Akbari, N. Soltani, and M. Farahani, “Numerical and experimental investigation of defect detection in polymer materials by means of digital shearography with thermal loading,” *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.*, vol. 227, no. 3, pp. 430–442, 2013, doi: 10.1177/0954405412473054.

[10] S. Sfarra, P. Theodorakeas, N. P. Avdelidis, and M. Koui, “Thermographic, ultrasonic and optical methods: A new dimension in veneered wood diagnostics,” *Russ. J. Nondestruct. Test.*, vol. 49, no. 4, pp. 234–250, Apr. 2013, doi: 10.1134/S1061830913040062.

[11] L. X. Y. and Y. Y. Hung, “Digital Shearography for Nondestructive Evaluation and Application in Automotive and Aerospace Industries,” no. 248.

[12] M. Rouse and M. Haughn, “What is field of view (FOV)? - Definition from WhatIs.com,” May 2017. https://whatis.techtarget.com/definition/field-of-view-FOV (accessed Apr. 19, 2020).

[13] “Understanding Focal Length and Field of View | Edmund Optics.” https://www.edmundoptics.com/knowledge-center/application-notes/imaging/understanding-focal-length-and-field-of-view/ (accessed Apr. 19, 2020).

[14] S. Wu, X. He, and L. Yang, “Enlarging the angle of view in Michelsoninterferometer-based shearography by embedding a 4f system,” *Appl. Opt.*, vol. 50, no. 21, pp. 3789–3794, 2011,

doi: 10.1364/AO.50.003789.

[15] T. Kurihara and Y. Takaki, “Improving viewing region of 4f optical system for holographic displays,” *Opt. Express*, vol. 19, no. 18, p. 17621, 2011, doi: 10.1364/oe.19.017621.

[16] “The 4 Golden Rules of UI Design | Adobe XD Ideas.” https://xd.adobe.com/ideas/process/ui-design/4-golden-rules-ui-design/ (accessed May 13, 2020).

[17] M. Densmore, “Mobile Architectures,” no. July, 2018.

[18] M. Densmore, “Views, Widgets, Layouts, Activities,” 2018.

[19] “Model-view-controller architecture | Download Scientific Diagram.” https://www.researchgate.net/figure/Model-view-controller-architecture_fig7_320249584 (accessed Jun. 04, 2020).

[20] “Reading 5: UI Software Architecture.” http://web.mit.edu/6.813/www/sp17/classes/05-ui-sw-arch/ (accessed Jun. 02, 2020).

[21] “MVC explained!” https://www.educative.io/edpresso/mvc-explained (accessed Jun. 04, 2020).

[22] “GUI Toolkits — Mastering EOS.” https://cis.gvsu.edu/~meos/gui-toolkits.html (accessed Jun. 05, 2020).

[23] “How to create a GUI application with Python | Opensource.com.” https://opensource.com/resources/python/gui-frameworks (accessed Jun. 05, 2020).

[24] R. Yadav, “Top 7 Image Processing Libraries In Python,” 2019. https://analyticsindiamag.com/top-8-image-processing-libraries-in-python/ (accessed Oct. 18, 2020).

[25] P. Pandey, “10 Python image manipulation tools | Opensource.com,” Mar. 18, 2019. https://opensource.com/article/19/3/python-image-manipulation-tools (accessed Oct. 18, 2020).

[26] Open Source Computer Vision, “OpenCV: Operations on arrays.” https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga6fef31bc8c4071cbc114a758a2b79c14 (accessed Oct. 18, 2020).

[27] Authentise, “How to track objects with stationary background?,” Jun. 07, 2016. https://www.authentise.com/post/how-to-track-objects-with-stationary-background (accessed Oct. 19, 2020).

[28] Pillow (PIL Fork), “ImageChops (‘Channel Operations’) Module — Pillow (PIL Fork) 3.1.2 documentation.” https://pillow.readthedocs.io/en/3.1.x/reference/ImageChops.html#PIL.ImageChops.subtract (accessed Oct. 18, 2020).

[29] “opencv - How to subtract two images using python opencv2 to get the foreground object - Stack Overflow.” https://stackoverflow.com/questions/21425992/how-to-subtract-two-images-using-python-opencv2-to-get-the-foreground-object/34585872#34585872 (accessed Oct. 19, 2020).

[30] “Pixelwise subtract, with negative numbers - OpenCV Q&A Forum.” https://answers.opencv.org/question/46235/pixelwise-subtract-with-negative-numbers/

(accessed Oct. 19, 2020).

[31] “FLIR Machine Vision Cameras REGISTER REFERENCE,” 2017. https://flir.app.box.com/v/Flycapture2SDK/file/576348676828 (accessed May 20, 2020).

[32] Pillow (PIL Fork), “ImageEnhance Module — Pillow (PIL Fork) 3.0.0 documentation,” 2015. https://pillow.readthedocs.io/en/3.0.x/reference/ImageEnhance.html (accessed Oct. 20, 2020).

[33] Kite, “pil - Increase the contrast of an image - Python code example - Kite.” https://www.kite.com/python/examples/3184/pil-increase-the-contrast-of-an-image (accessed Oct. 20, 2020).

[34] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, “Chameleon3 Technical Reference,” pp. 707–721, 2018, doi: 10.1145/3196494.3196522.

[35] “Imaging Electronics 101: Basics of Digital Camera Settings for Improved Imaging Results.” https://www.edmundoptics.com/knowledge-center/application-notes/imaging/basics-of-digital-camera-settings-for-improved-imaging-results/ (accessed Oct. 21, 2020).

[36] Edmund Optics, “25.0mm Optic Diameter, Optic Mount | Edmund Optics.” https://www.edmundoptics.com/p/250mm-optic-diameter-optic-mount/19272/ (accessed Nov. 02, 2020).

[37] Edmund Optics, “25.0mm Dia. x 60.0mm FL, Uncoated, Plano-Convex Lens.” https://www.edmundoptics.com/p/250mm-dia-x-600mm-fl-uncoated-plano-convex-lens/5616/ (accessed Nov. 02, 2020).

[38] Edmund Optics, “Focal Length Calculator | Edmund Optics.” https://www.edmundoptics.com/knowledge-center/tech-tools/focal-length/ (accessed Nov. 02, 2020).

[39] Kowa, “Kowa 2/3" LM8JCM Lens.” https://lenses.kowa-usa.com/jcm-series/411-lm8jcm.html (accessed Nov. 01, 2020).

[40] Stemmer Imaging, “Optics: Lens mounts | STEMMER IMAGING.” https://www.stemmer-imaging.com/en/knowledge-base/lens-mounts/ (accessed Nov. 01, 2020).

[41] Thorlabs, “Optical Component Threading Adapters with C-Mount (1.00"-32) Threads.” https://www.thorlabs.com/newgrouppage9.cfm?objectgroup_id=8210 (accessed Nov. 01, 2020).

[42] Metal Supermarkets, “Most Common Uses of Aluminum | Metal Supermarkets - Steel, Aluminum, Stainless, Hot-Rolled, Cold-Rolled, Alloy, Carbon, Galvanized, Brass, Bronze, Copper,” May 09, 2016. https://www.metalsupermarkets.com/common-uses-aluminum/ (accessed Nov. 03, 2020).

[43] J. Van Schooneveld, “Build Physical Projects With Python on the Raspberry Pi – Real Python,” Jun. 01, 2020. https://realpython.com/python-raspberry-pi/ (accessed Oct. 27, 2020).

[44] B. Dufour and W. H. Chang, “An introduction to Tkinter.” https://www.cs.mcgill.ca/~hv/classes/MS/TkinterPres/#Advantages (accessed Nov. 05, 2020).

[45] J. Johnson, “All You Need to Know About UML Diagrams: Types and 5+ Examples.” https://tallyfy.com/uml-diagram/#class-diagram (accessed Nov. 06, 2020).

[46] “python - Get image from Point Grey camera using PyCapture2 and OpenCV - Stack Overflow.” https://stackoverflow.com/questions/44012780/get-image-from-point-grey-camera-using-pycapture2-and-opencv (accessed Nov. 13, 2020).

[47] University of Cape Town, “UCT-NDT Digital Shearography User Manual Ver 2.0A,” 2008.

[48] T. B. Outram, “Black Coatings to Reduce Stray Light,” 2009. Accessed: Nov. 09, 2020. [Online]. Available: http://www.tak2000.com/data/finish.htm.

[49] Expert System, “What is Machine Learning? A definition - Expert System,” May 2017. https://expertsystem.com/machine-learning-definition/ (accessed Oct. 31, 2020).

[50] C. Wunderlich, C. Tschöpe, and F. Duckhorn, “Advanced methods in NDE using machine learning approaches,” *AIP Conf. Proc.*, vol. 1949, no. April 2018, 2018, doi: 10.1063/1.5031519.

[51] Y. Ye, K. Ma, H. Zhou, D. Arola, and D. Zhang, “An automated shearography system for cylindrical surface inspection,” *Meas. J. Int. Meas. Confed.*, vol. 135, pp. 400–405, 2019, doi: 10.1016/j.measurement.2018.11.085.

[52] Edmund Optics, “21 x 30mm Protected Aluminum, λ/4 Mirror | Edmund Optics.” https://www.edmundoptics.com/p/21-x-30mm-protected-aluminum-lambda4-mirror/2350/ (accessed Nov. 11, 2020).

[53] Edmund Optics, “25mm VIS, Polarizing Cube Beamsplitter | Edmund Optics.” https://www.edmundoptics.com/p/25mm-vis-polarizing-cube-beamsplitter/9356/ (accessed Nov. 11, 2020).

[54] F. Integrated and I. Solutions, “PyCapture2 API Reference,” 2017.

[55] I. FLIR Integrated Imaging Solutions, “FlyCapture2Test.” 2017.

[56] “Chameleon3 USB3 | FLIR Systems.” https://www.flir.com/products/chameleon3-usb3/ (accessed May 20, 2020).

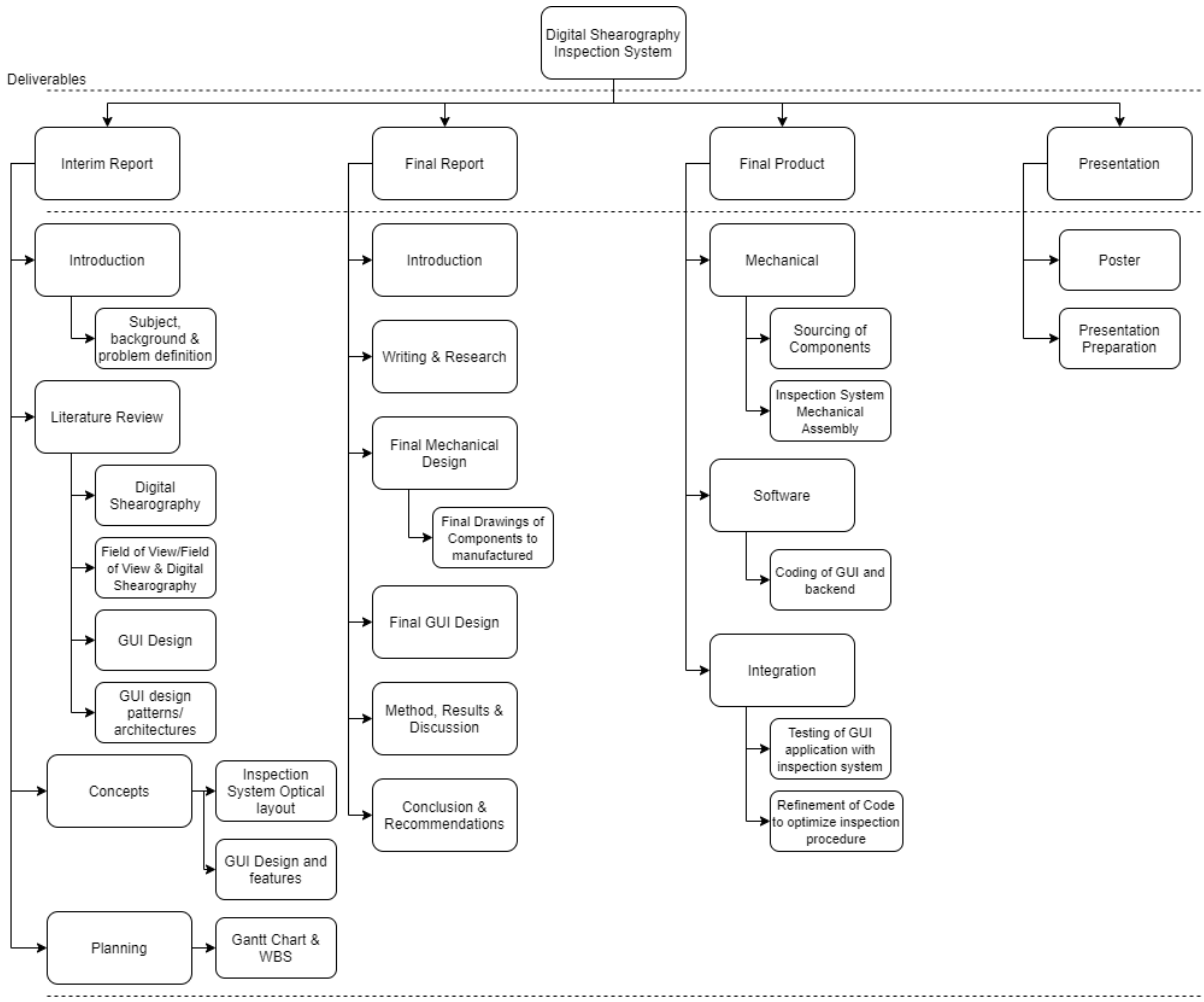
[57] “UI-1241LE - CAMERA FINDER - Products.” https://en.ids-imaging.com/store/ui-1241le.html (accessed Jun. 14, 2020).

[58] IDS, “Start_uEye_Manual.” .

22 Appendices

22.1 Appendix A - Planning

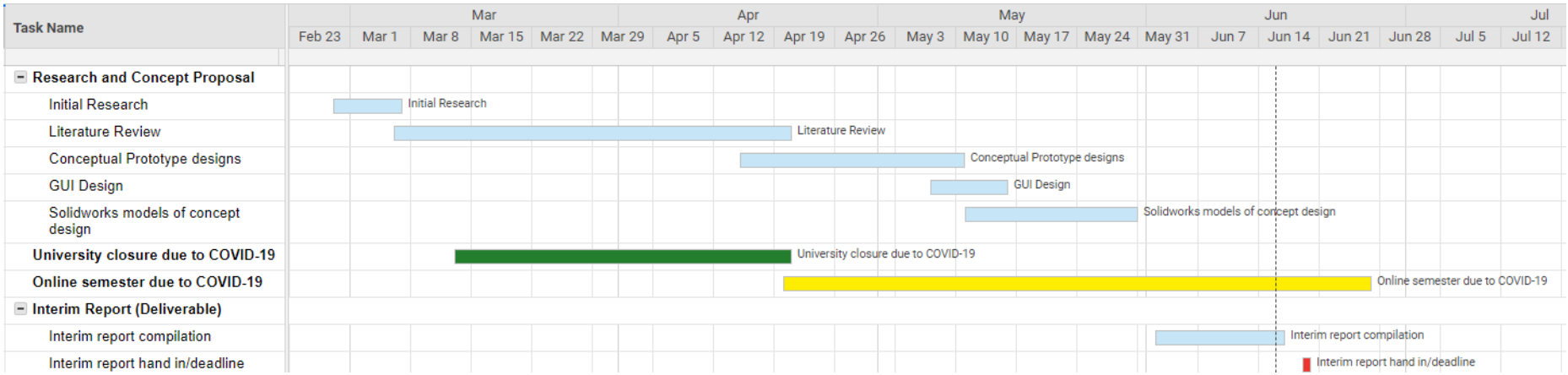
A Gantt chart and Work Breakdown Structure (WBS) were developed earlier in the project. The Gantt chart was used to allocate time to project tasks and highlight the deadlines of deliverables. If these tasks/deadlines are completed within their allocated time the project should be completed in time. The WBS was used to break down deliverables more manageable tasks. It allows for time and focus to be allocated to smaller tasks that when combined, produce the final deliverables. It also acts as a checklist, ensuring that no deliverable tasks are disregarded. The work breakdown structure is shown below, followed by the Gantt chart.

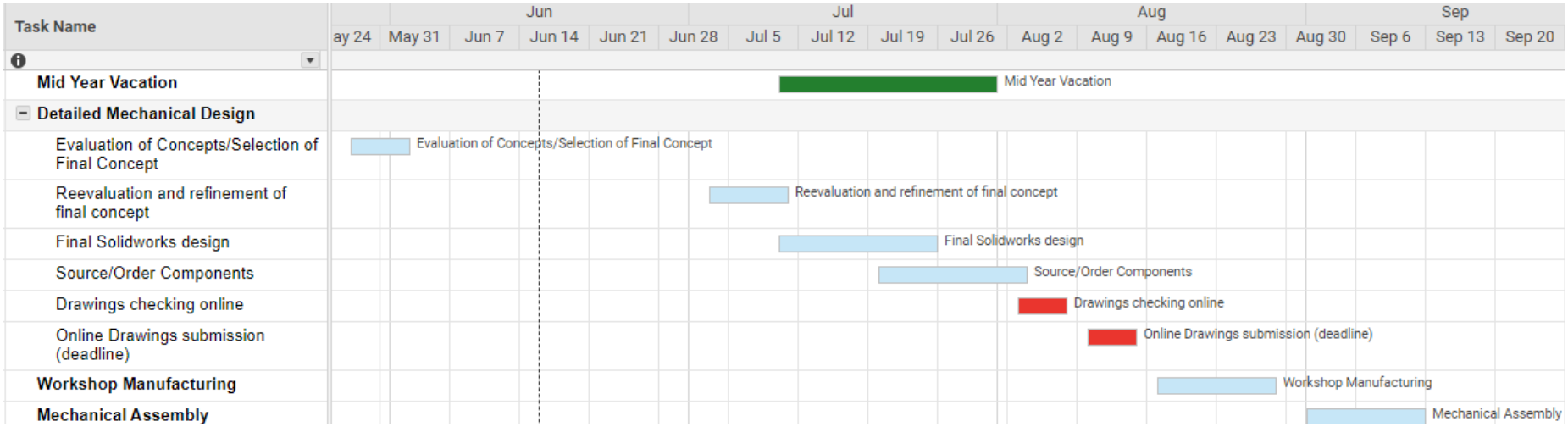


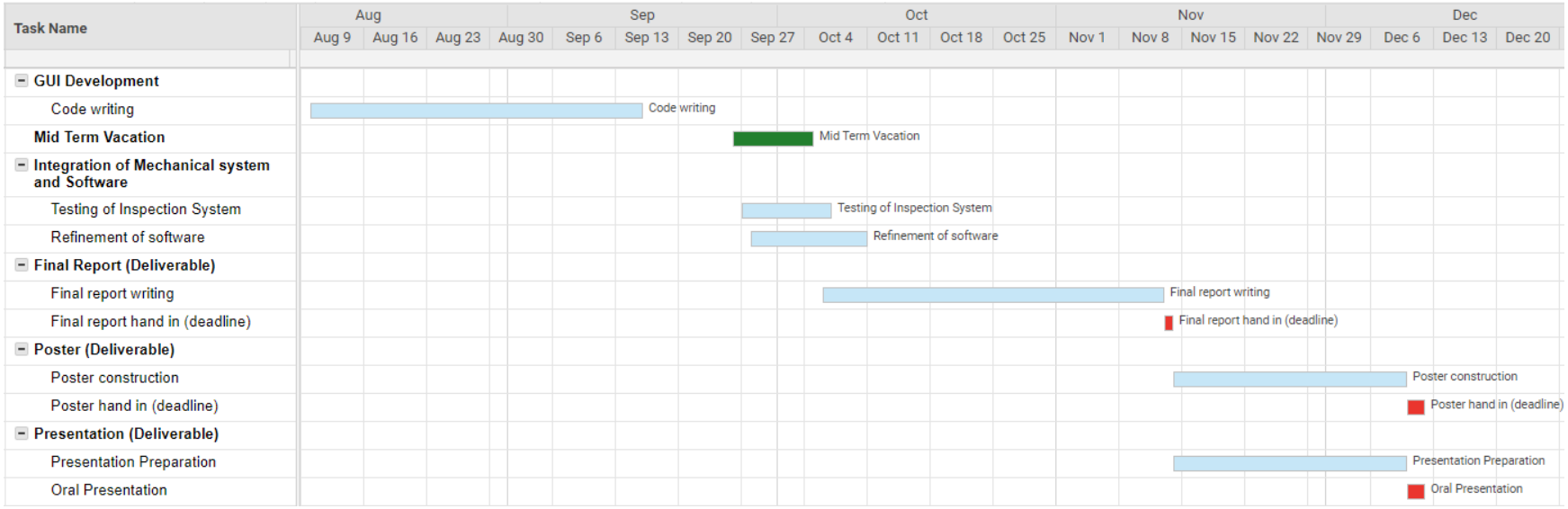
Since the WBS was developed, the Poster has been removed as one of the final deliverables and thus no longer falls part of the WBS.

The first Gantt chart below shows the planned allocation of time to different tasks, while the second Gantt chart below represents the actual amount of time spent completing these tasks. It can be seen that both of these charts account for the changes that the pandemic has brought about to the university calendar.

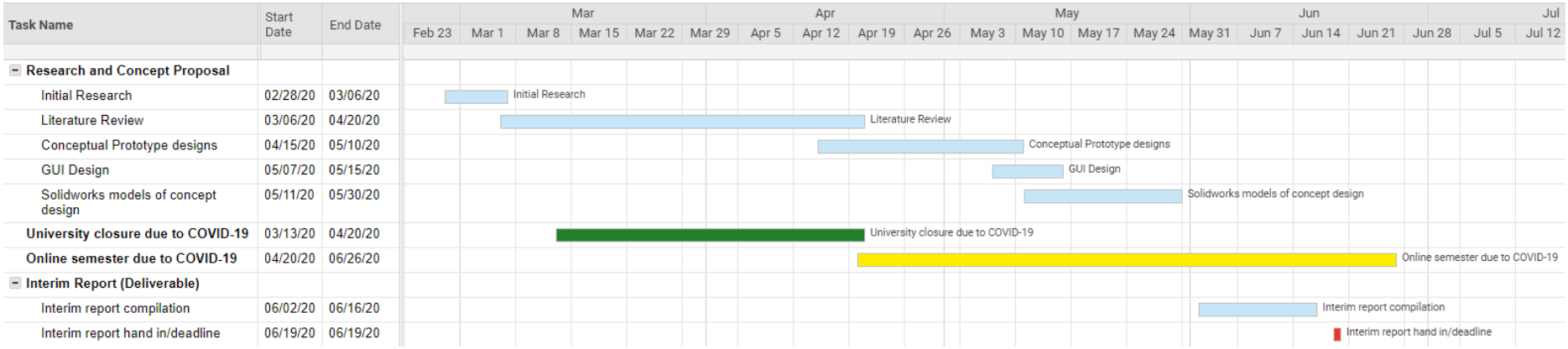
Original Gantt Chart

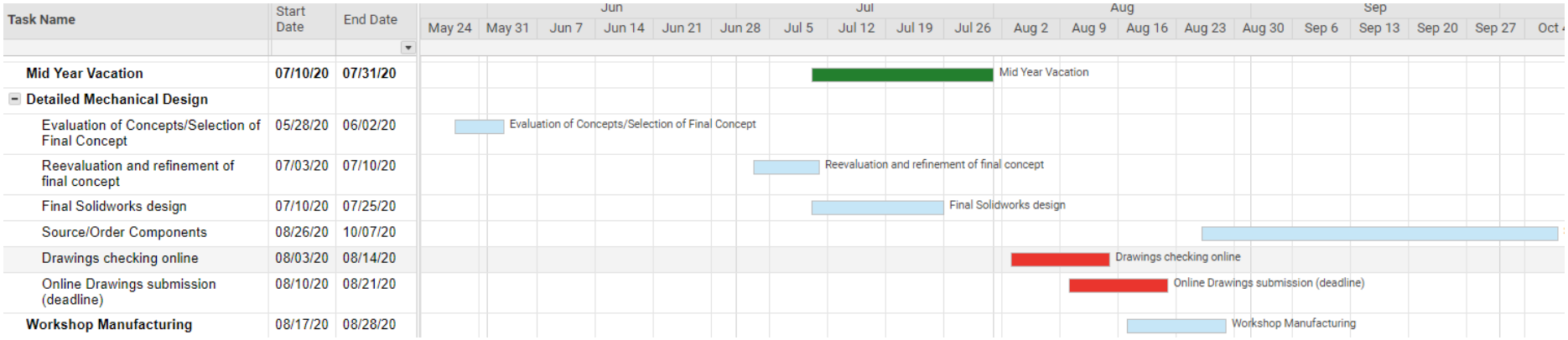


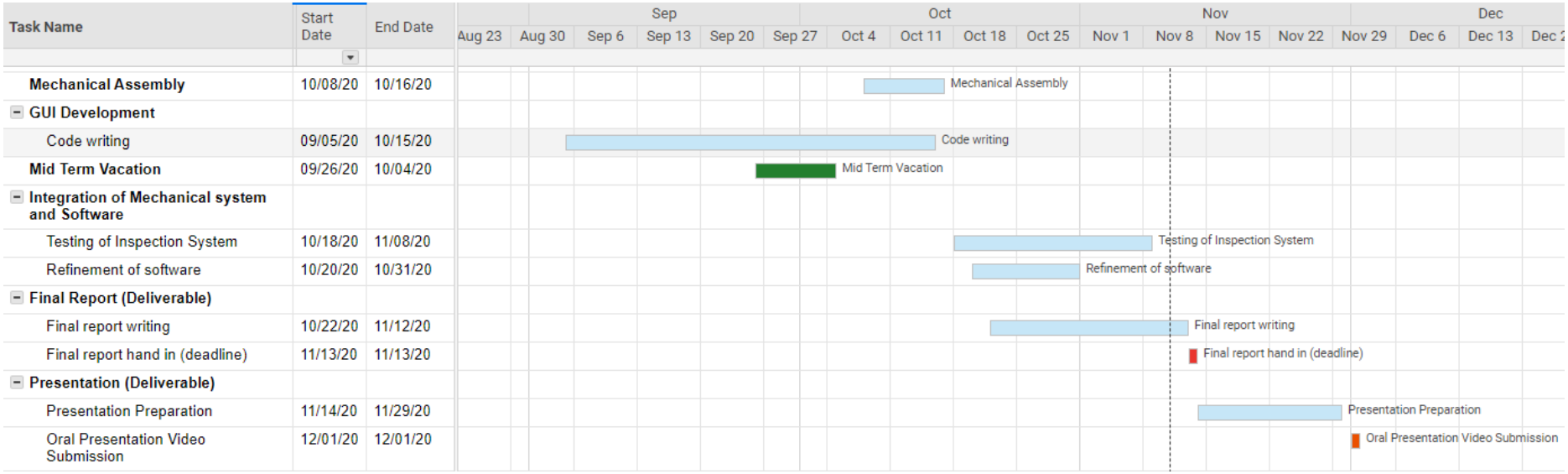




Altered Gantt Chart




Task Name
Start Date
End Date
Jun
Jul
Aug
Sep
May 24
May 31
Jun 7
Jun 14
Jun 21
Jun 28
Jul 5
Jul 12
Jul 19
Jul 26
Aug 2
Aug 9
Aug 16
Aug 23
Aug 30
Sep 6
Sep 13
Sep 20
Sep 27
Mid Year Vacation 07/10/20 07/31/20
Detailed Mechanical Design
Evaluation of Concepts/Selection of Final Concept 05/28/20 06/02/20
Reevaluation and refinement of final concept 07/03/20 07/10/20
Final Solidworks design 07/10/20 07/25/20
Source/Order Components 08/26/20 10/07/20
Drawings checking online 08/03/20 08/14/20
Online Drawings submission (deadline) 08/10/20 08/21/20
Workshop Manufacturing 08/17/20 08/28/20
Mid Year Vacation
Evaluation of Concepts/Selection of Final Concept
Reevaluation and refinement of final concept
Final Solidworks design
Drawings checking online
Online Drawings submission (deadline)
Workshop Manufacturing


Task Name
Start Date
End Date
Sep
Oct
Nov
Dec
Aug 23
Aug 30
Sep 6
Sep 13
Sep 20
Sep 27
Oct 4
Oct 11
Oct 18
Oct 25
Nov 1
Nov 8
Nov 15
Nov 22
Nov 29
Dec 6
Dec 13
Mechanical Assembly 10/08/20 10/16/20
GUI Development
Code writing 09/05/20 10/15/20
Mid Term Vacation 09/26/20 10/04/20
Integration of Mechanical system and Software
Testing of Inspection System 10/18/20 11/08/20
Refinement of software 10/20/20 10/31/20
Final Report (Deliverable)
Final report writing 10/22/20 11/12/20
Final report hand in (deadline) 11/13/20 11/13/20
Presentation (Deliverable)
Presentation Preparation 11/14/20 11/29/20
Oral Presentation Video Submission 12/01/20 12/01/20
Mechanical Assembly
Code writing
Mid Term Vacation
Testing of Inspection System
Refinement of software
Final report writing
Final report hand in (deadline)
Presentation Preparation
Oral Presentation Video Submission

From the above it can be seen that a lot more time needed to be allocated to certain activities than originally expected, such as the sourcing of components (which included shipping), the code writing of the GUI and the testing of the inspection system. The altered Gantt chart also makes changes to the deliverable due dates that were altered (such as the drawing deadline). Although more time was needed for some tasks, the project was completed on time, which reflects well on the project planning.

22.2 Appendix B - Risk Assessment Forms

Department of Mechanical Engineering rev 1.0

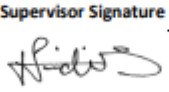
Risk Assessment Form

- **Risk Assessments are not required for simple workshop activities covered by the Safety Declaration. However permission must be obtained from the Workshop Manager before commencing any activity in the Workshop.**
- **A new Risk Assessment form needs to be completed for each major activity within your project.**
- **You are required to include this document (signed) in your bound project submission and mount a copy next to any rig / apparatus you are using.**

Your Name	Miguel Garcia Naude
Your Supervisor	Mr Dirk Findeis
Project title and number	Refine the Digital Shearography system based on a Raspberry Pi PC, a camera and miniature shearing device
Area Safety Warden	Mr Dirk Findeis

This Section to be completed by the student (Must be typed and the declaration signed)			
Location where the activity will be done	Non-Destructive Testing Lab (EM 105)		
Describe the activity (use attachment with diagrams if needed)	Non-destructive testing of specimens using a digital shearography inspection device and a laser.		
Names of persons involved in this activity	Miguel Garcia Naude		
Describe in detail the risks you (and others) will face during this activity and the potential consequences of your activities	Many lasers can cause damage to the human eye and in some cases, blindness. Eye damage can result whether the light radiation is direct or reflected. Due to the ongoing Coronavirus Pandemic there is constant risk of virus spread. Without following protective protocols, there is potential for infection from one person to another.		
Does this activity involve the use of any materials (chemicals, gasses, etc.) which may be hazardous to health, or the environment	No		*If Yes list the chemicals / gasses to be used and attach the MSDS(s) for these materials.*
Does this activity involve any equipment / device designed or built by you which is to be plugged into mains electricity?	No		*If Yes please consult with Mr Richard Whittemore or* Mr Maysam Soltanian (Electrical Machines Lab) *before connecting to the mains / switching on.*
Does the activity involve any new equipment / devices designed by you which will be pressurized with a gas or volatile liquids?	No		*If Yes please check the relevant SANS Pressurised Equipment Regulations and consult with A/Prof Fuls before testing.*
What precautions are required to protect against the risks detailed above?	Safe operation of the laser should always be maintained. When the laser is in operation, one should not stand in front of or look into the laser to decrease the risk of damage to one's eyes.		
Describe the personal protective equipment (PPE) required during this activity – specify in detail.	Face masks are always to be worn to decrease the potential chance of being infected or infecting someone else with Covid-19. The defusal of the laser light in this case means that there is no need to wear protective laser glasses		
Describe the shutdown procedure in detail.	One should stand out of the path of the laser light and turn the laser off.		
Describe any relevant emergency procedures, e.g. spillage response etc.	If one has blind spots that do not fade away after being exposed to laser light, a specialist should be contacted, specifically an Optometrist or an Ophthalmologist. If the situation is serious and requires more attention, a retina specialist should be contacted.		

I declare that I am aware of the risks associated with this activity and will take all necessary steps to mitigate these risks.	**Student Signature**	**Date**
		18/10/2020

I am aware of the student's intended activity, and have provided the necessary guidance, inputs, and oversight.	**Supervisor Signature**	**Date**
		19/10/2020

This section to be completed by the Area Safety Warden		
Level of supervision required (Please tick relevant block)	A = work may not take place without supervisor/warden present.	
	B = work may not take place without a 2nd party present.	
	C = no specific extra supervision requirements.	
I am satisfied that the student is aware of the risks associated with this activity and grant approval for it to proceed.	**Signature**	**Date**

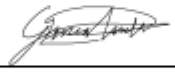
Department of Mechanical Engineering rev 1.0

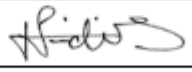
Risk Assessment Form

- **Risk Assessments are not required for simple workshop activities covered by the Safety Declaration. However permission must be obtained from the Workshop Manager before commencing any activity in the Workshop.**
- **A new Risk Assessment form needs to be completed for each major activity within your project.**
- **You are required to include this document (signed) in your bound project submission and mount a copy next to any rig / apparatus you are using.**

Your Name	Miguel Garcia Naude
Your Supervisor	Mr Dirk Findeis
Project title and number	Refine the Digital Shearography system based on a Raspberry Pi PC, a camera and miniature shearing device
Area Safety Warden	Mr Dirk Findeis

This Section to be completed by the student (Must be typed and the declaration signed)			
Location where the activity will be done	At a desk, at home		
Describe the activity (use attachment with diagrams if needed)	Sitting for long periods of time, programming code.		
Names of persons involved in this activity	Miguel Garcia Naude		
Describe in detail the risks you (and others) will face during this activity and the potential consequences of your activities	Back pain can result from sitting for long periods of time as it puts huge stress on the back, neck and spine. Sitting for long periods of time is also linked to risks of increased blood pressure & blood sugar, increased fat build-up around the waist and higher cholesterol levels. This leads to increased risk of cardiovascular disease and diabetes amongst others.		
Does this activity involve the use of any materials (chemicals, gasses, etc.) which may be hazardous to health, or the environment	No		*If Yes list the chemicals / gasses to be used and attach the MSDS(s) for these materials.*
Does this activity involve any equipment / device designed or built by you which is to be plugged into mains electricity?	No		*If Yes please consult with Mr Richard Whittemore or* Mr Maysam Soltanian (Electrical Machines Lab) *before connecting to the mains / switching on.*
Does the activity involve any new equipment / devices designed by you which will be pressurized with a gas or volatile liquids?	No		*If Yes please check the relevant SANS Pressurised Equipment Regulations and consult with A/Prof Fuls before testing.*
What precautions are required to protect against the risks detailed above?	• Take regular breaks from sitting, getting up from the desk at regular intervals. • If possible, try a standing desk or a high table computer. • When not programming, perform the activities while standing up (if possible). • Use an ergonomic chair if possible.		
Describe the personal protective equipment (PPE) required during this activity – specify in detail.	None		
Describe the shutdown procedure in detail.	None		
Describe any relevant emergency procedures, e.g. spillage response etc.	None		

I declare that I am aware of the risks associated with this activity and will take all necessary steps to mitigate these risks.	**Student Signature**	**Date**
		18/10/2020

I am aware of the student's intended activity, and have provided the necessary guidance, inputs, and oversight.	**Supervisor Signature**	**Date**
		9/11/2020

This section to be completed by the Area Safety Warden			
Level of supervision required (Please tick relevant block)	A = work may not take place without supervisor/warden present.		
	B = work may not take place without a 2nd party present.		
	C = no specific extra supervision requirements.		
I am satisfied that the student is aware of the risks associated with this activity and grant approval for it to proceed.	**Signature**		**Date**

22.3 Appendix C - Budgeting and Financial Management

After having investigated the cost of certain expenses required to build the prototype, it was determined that it would be impossible to complete the project without exceeding the allocated budget of R1500. While components that could be designed and manufactured would only take up a portion of the budget, the cost of the optical components far exceeds it. It was first suggested by the supervisor, Mr Dirk Findeis, that the allocated budget would not be enough to acquire all the components required for the project. Therefore, it was authorised that more than R1500 could be spent on the project. Below is a table showing all the purchases made for the project, the total amount spent on the project being R4595.123, the majority of which can be attributed to the lenses and lens mounts that needed to be purchased from an overseas supplier. The funding provided came from the supervisor's research grant and the purchasing of expensive items was managed by Ms Glass.

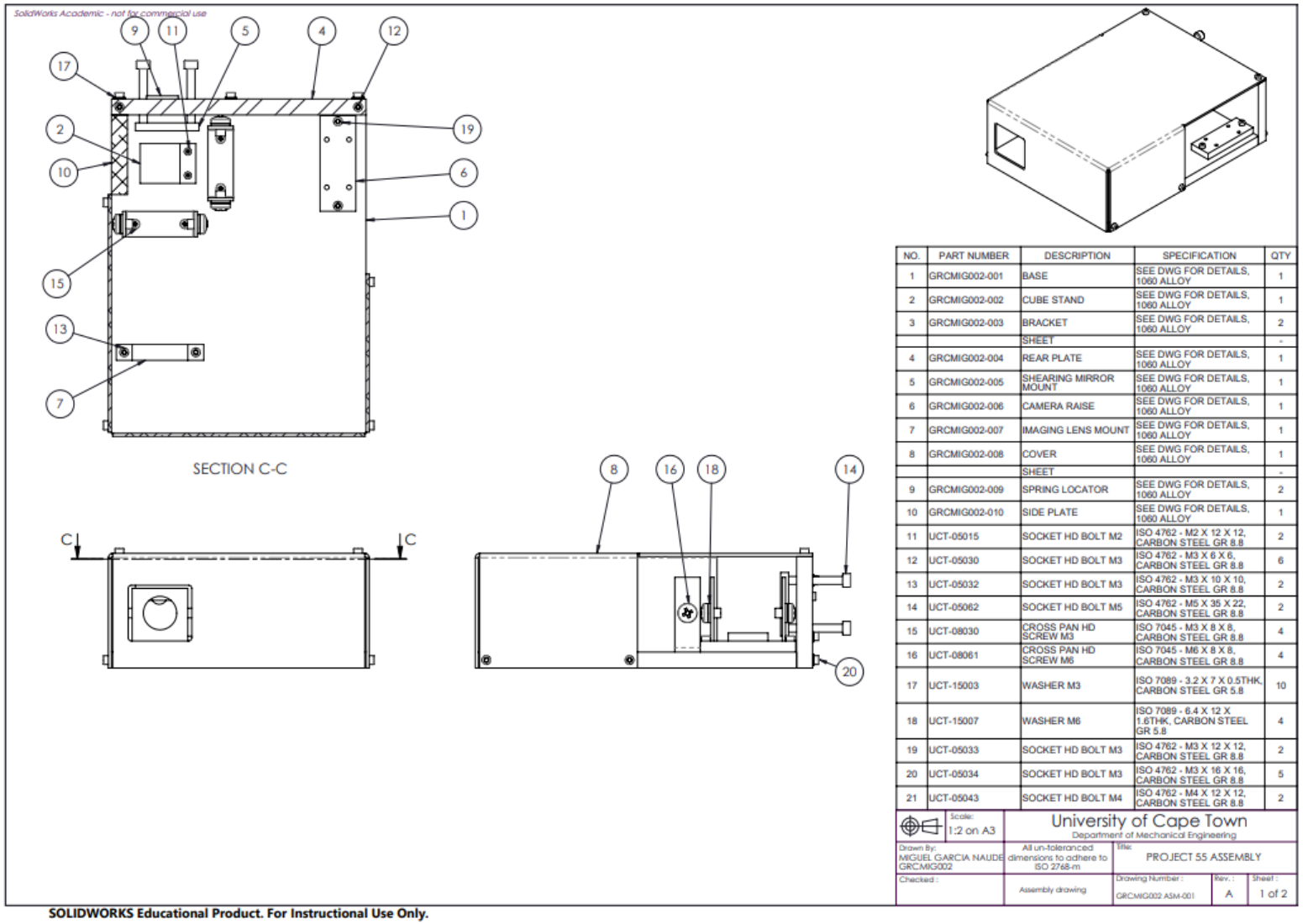
Parts/Expense	Service Provider	Quantity	Price per Unit [R]	Total Price [R]
Purchases of Optics and Optical Related Components				
25.0mm Optic Diameter, Optic Mount	Edmund Optics	2	846.6	1693.2
25.0mm Dia. x 60.0mm FL, Uncoated, Plano-Convex Lens	Edmund Optics	2	547.8	1095.6
Shipping	UPS	N/A		669.31
Import Invoives Tax Invoice	UPS	N/A		592.5
Purchases for manufacturing of components				
Bracket	Vulcan Steel	2	30.8	61.6
Cover	Vulcan Steel	1	94.093	94.093
Aluminium Sheet - M82 12mm X 2520X1270mm	Non-Ferrous Metal Works (SA) (Pty) Ltd	1	60	60
Aluminium Sheet - 6082T6 - 8mm X 2520mm X 1270mm	Non-Ferrous Metal Works (SA) (Pty) Ltd	1	100	100
Aluminium Sheet - 6082T6 - 10mm X 2520mm X 1270mm	Non-Ferrous Metal Works (SA) (Pty) Ltd	1	190	190
Additional Purchases				
Fasteners	Boltit	As per BOM		35.42
Tension Spring	Jackhammer's	1		3.4
			Total	R 4595.12

The difference between the amount spent and the original allocated budget is R3095.12. However, purchases relating to the optical components account for almost R4000 of the total expenses. While this is a large amount of money to be spending on a project, it was necessary. However, these

components will not only be used for this project but will likely be used for future iteration and development of the Digital Shearography device. This can therefore be thought of as a future investment for the mechanical engineering department.

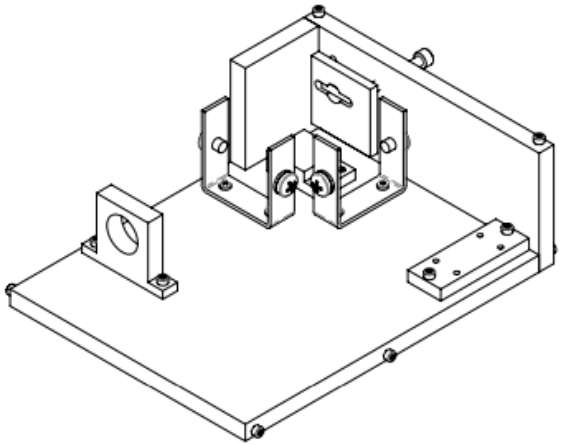
22.4 Appendix D - Final Design Drawings

22.4.1 Assembly

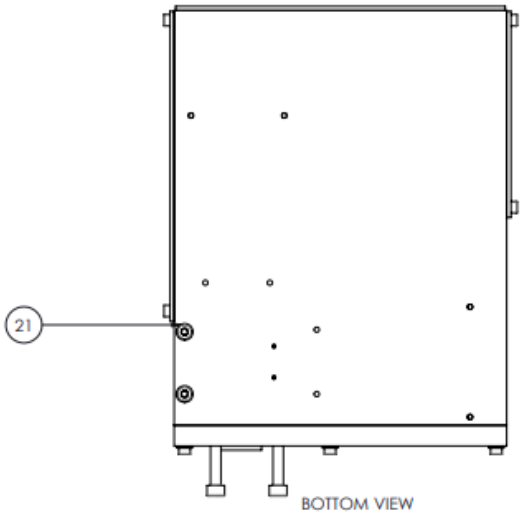


NO.	PART NUMBER	DESCRIPTION	SPECIFICATION	QTY
1	GRCMIG002-001	BASE	SEE DWG FOR DETAILS, 1060 ALLOY	1
2	GRCMIG002-002	CUBE STAND	SEE DWG FOR DETAILS, 1060 ALLOY	1
3	GRCMIG002-003	BRACKET	SEE DWG FOR DETAILS, 1060 ALLOY	2
		SHEET		-
4	GRCMIG002-004	REAR PLATE	SEE DWG FOR DETAILS, 1060 ALLOY	1
5	GRCMIG002-005	SHEARING MIRROR MOUNT	SEE DWG FOR DETAILS, 1060 ALLOY	1
6	GRCMIG002-006	CAMERA RAISE	SEE DWG FOR DETAILS, 1060 ALLOY	1
7	GRCMIG002-007	IMAGING LENS MOUNT	SEE DWG FOR DETAILS, 1060 ALLOY	1
8	GRCMIG002-008	COVER	SEE DWG FOR DETAILS, 1060 ALLOY	1
		SHEET		-
9	GRCMIG002-009	SPRING LOCATOR	SEE DWG FOR DETAILS, 1060 ALLOY	2
10	GRCMIG002-010	SIDE PLATE	SEE DWG FOR DETAILS, 1060 ALLOY	1
11	UCT-05015	SOCKET HD BOLT M2	ISO 4762 - M2 X 12 X 12, CARBON STEEL GR 8.8	2
12	UCT-05030	SOCKET HD BOLT M3	ISO 4762 - M3 X 6 X 6, CARBON STEEL GR 8.8	6
13	UCT-05032	SOCKET HD BOLT M3	ISO 4762 - M3 X 10 X 10, CARBON STEEL GR 8.8	2
14	UCT-05062	SOCKET HD BOLT M5	ISO 4762 - M5 X 35 X 22, CARBON STEEL GR 8.8	2
15	UCT-08030	CROSS PAN HD SCREW M3	ISO 7045 - M3 X 8 X 8, CARBON STEEL GR 8.8	4
16	UCT-08061	CROSS PAN HD SCREW M6	ISO 7045 - M6 X 8 X 8, CARBON STEEL GR 8.8	4
17	UCT-15003	WASHER M3	ISO 7089 - 3.2 X 7 X 0.5THK, CARBON STEEL GR 5.8	10
18	UCT-15007	WASHER M6	ISO 7089 - 6.4 X 12 X 1.6THK, CARBON STEEL GR 5.8	4
19	UCT-05033	SOCKET HD BOLT M3	ISO 4762 - M3 X 12 X 12, CARBON STEEL GR 8.8	2
20	UCT-05034	SOCKET HD BOLT M3	ISO 4762 - M3 X 16 X 16, CARBON STEEL GR 8.8	5
21	UCT-05043	SOCKET HD BOLT M4	ISO 4762 - M4 X 12 X 12, CARBON STEEL GR 8.8	2

Scale: 1:2 on A3	University of Cape Town Department of Mechanical Engineering		
Drawn By: MIGUEL GARCIA NAUDE GRCMIG002	All un-toleranced dimensions to adhere to ISO 2768-m	Title: PROJECT 55 ASSEMBLY	
Checked :	Assembly drawing	Drawing Number : GRCMIG002 ASM-001	Rev. : A / Sheet : 1 of 2



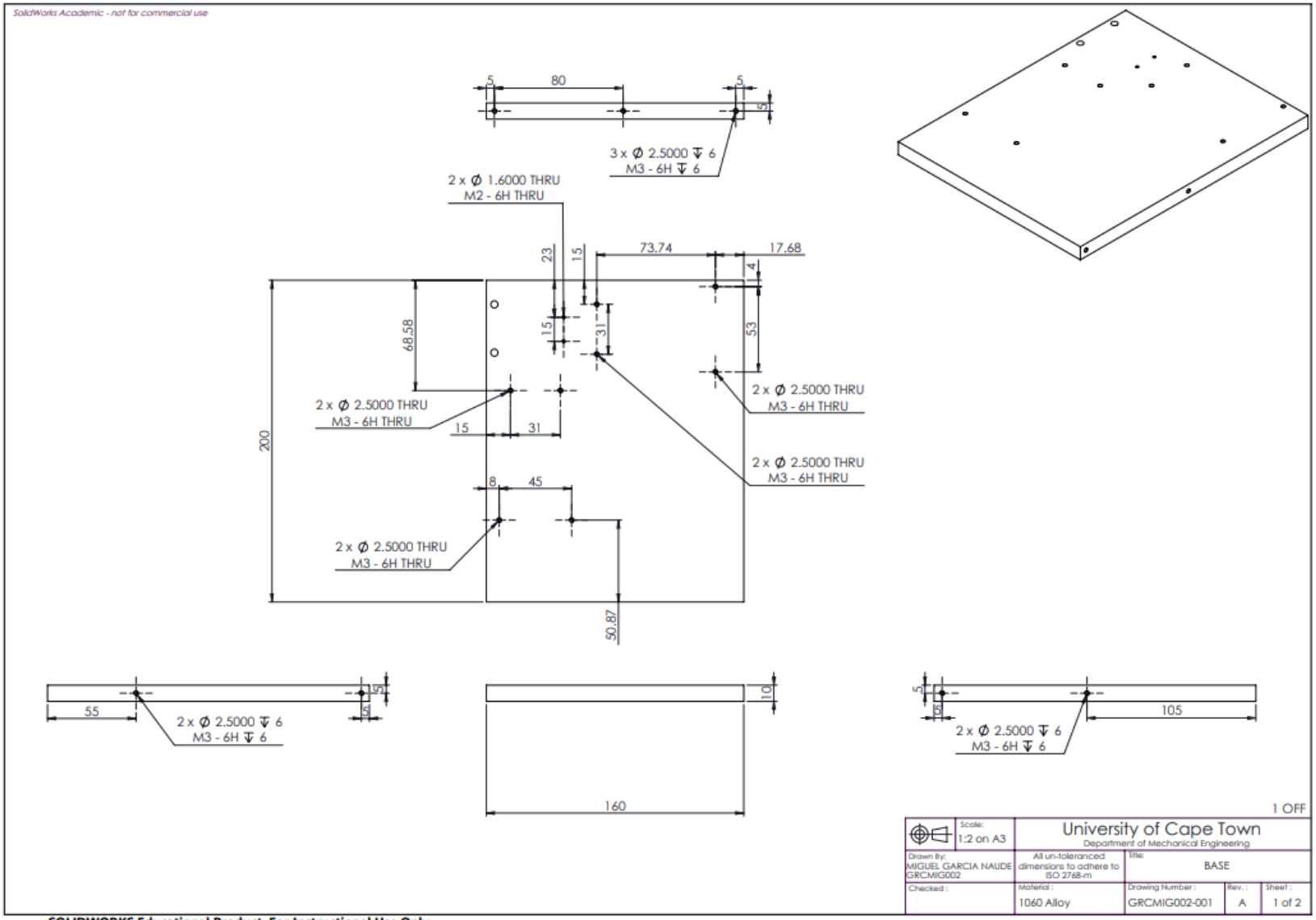
ISOMETRIC VIEW WITH COVER HIDDEN

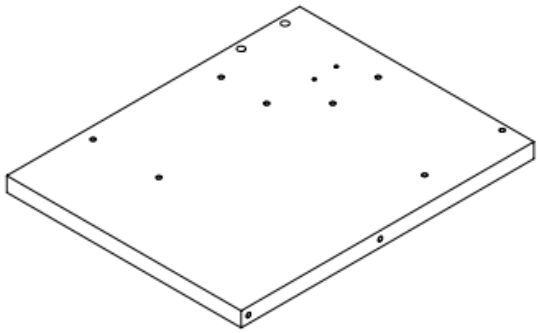


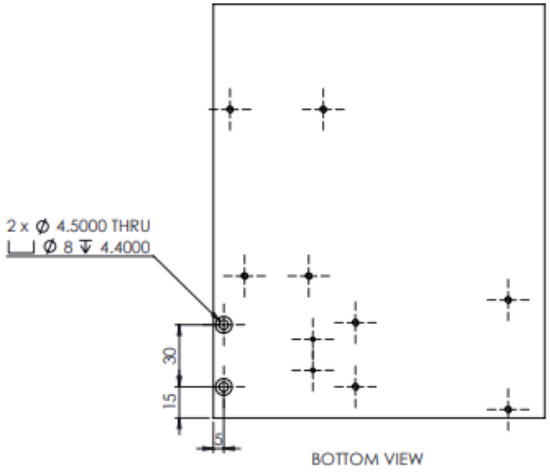
BOTTOM VIEW

Scale: 1:2 on A3	University of Cape Town Department of Mechanical Engineering			
Drawn By: MIGUEL GARCIA NAUDE GRCMIG002	All un-toleranced dimensions to adhere to ISO 2768-m	Title: PROJECT 55 ASSEMBLY		
Checked :	Assembly drawing	Drawing Number : GRCMIG002 ASM-001	Rev. : A	Sheet : 2 of 2

22.4.2 Base

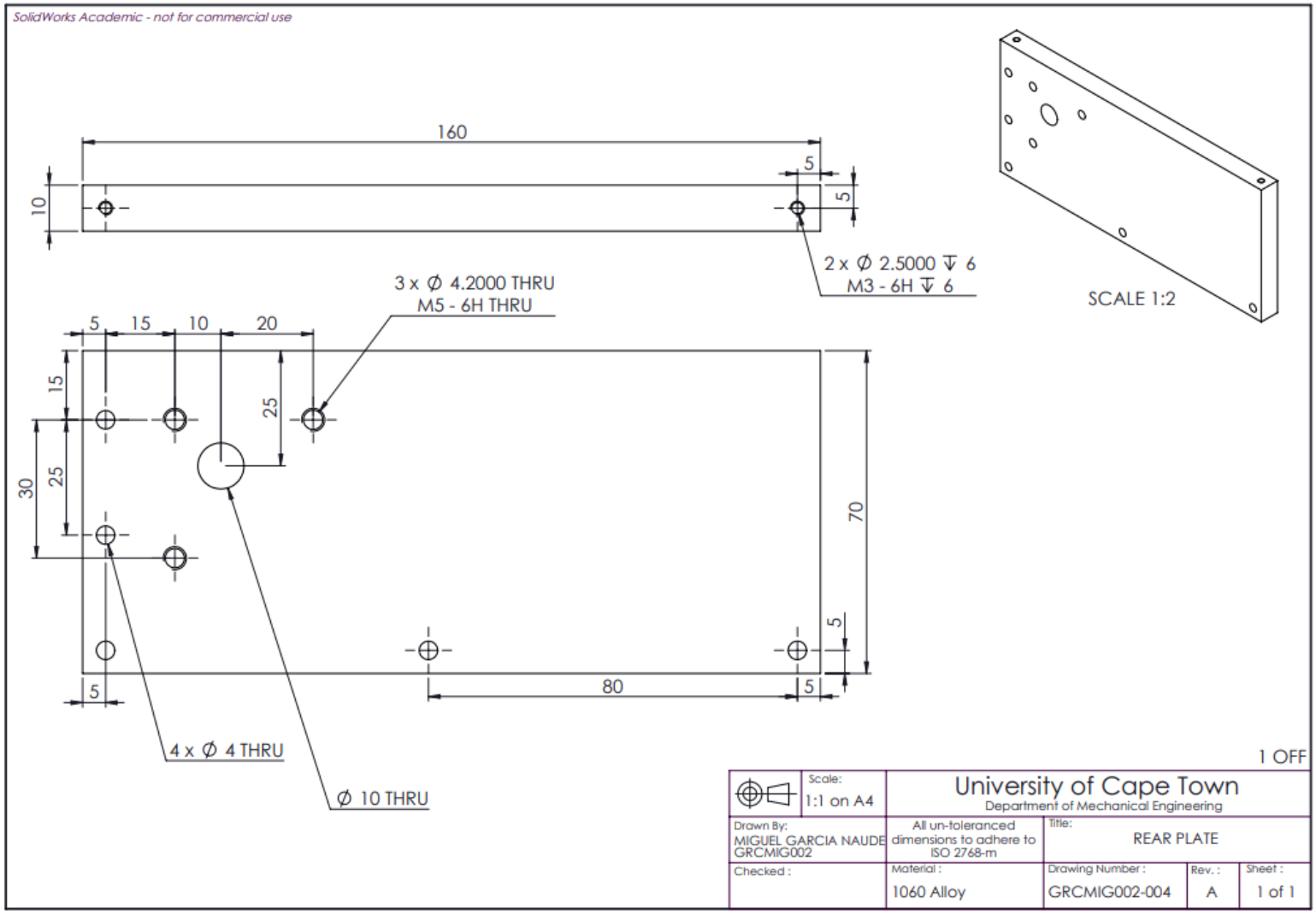






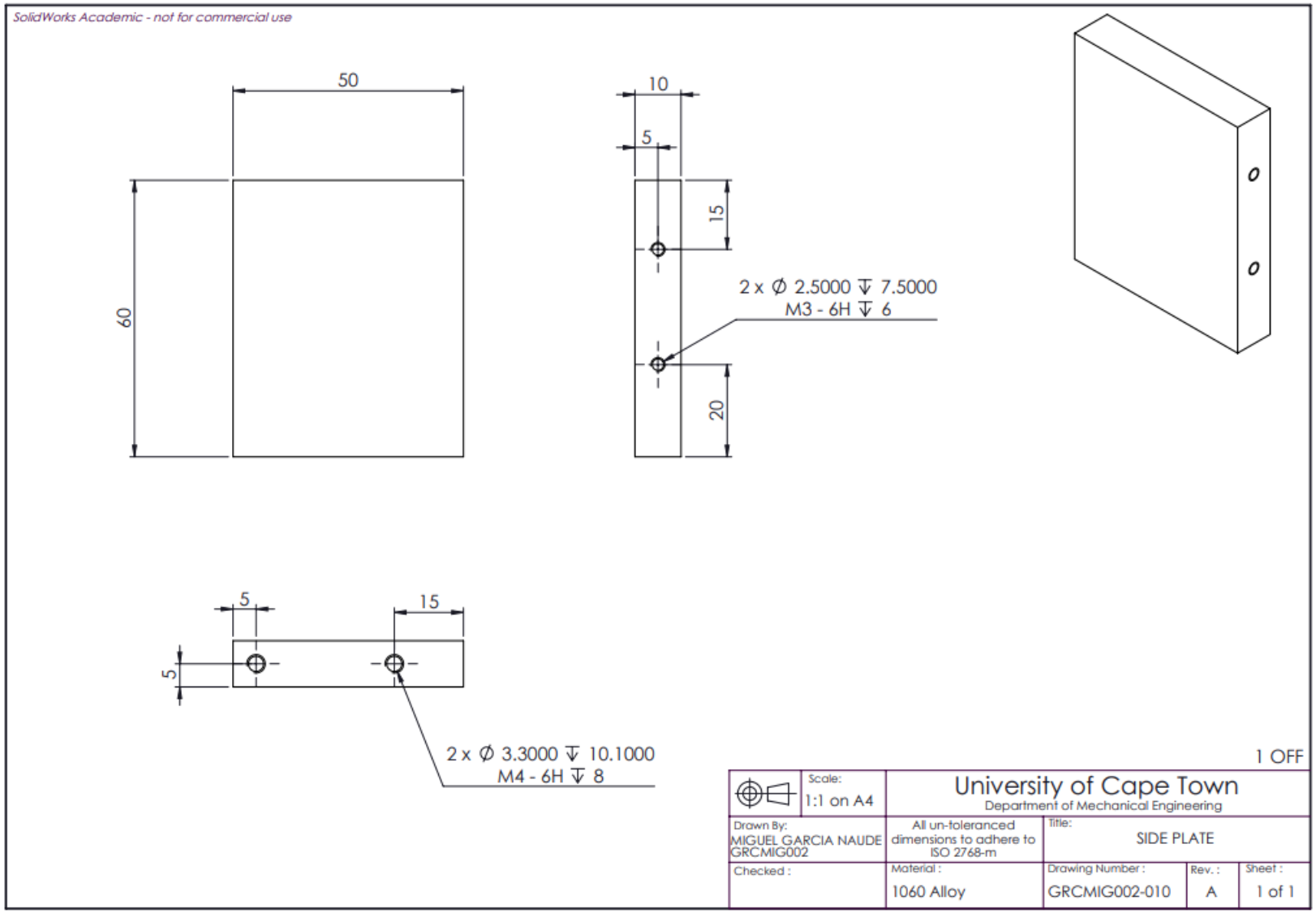
Scale: 1:2 on A3	University of Cape Town Department of Mechanical Engineering			
Drawn By: MIGUEL GARCIA NAUDE GRCMIG002	All un-toleranced dimensions to adhere to ISO 2768-m	Title: BASE		
Checked :	Material : 1060 Alloy	Drawing Number : GRCMIG002-001	Rev. : A	Sheet : 2 of 2

22.4.3 Rear Plate



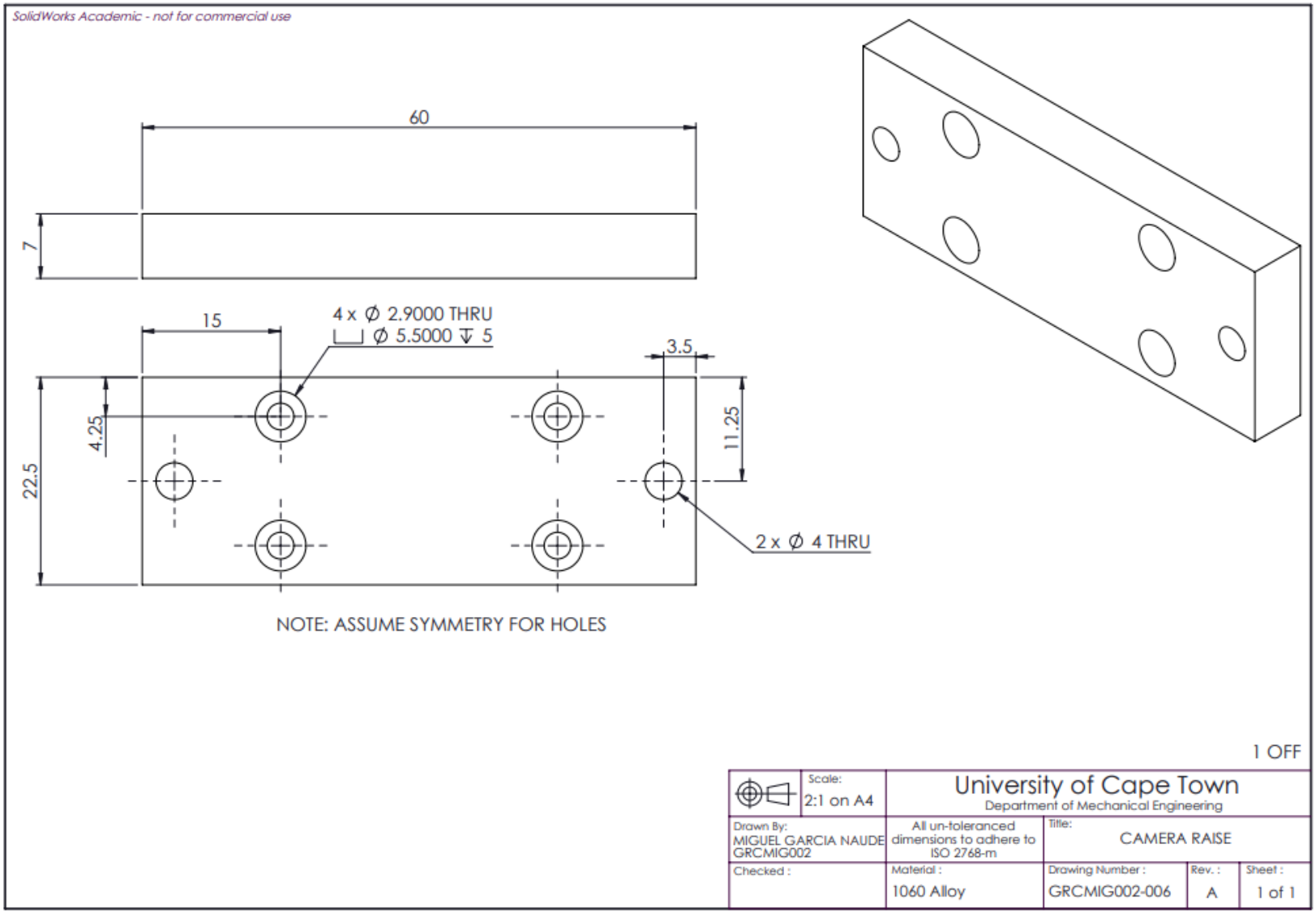
SOLIDWORKS Educational Product. For Instructional Use Only.

22.4.4 Side Plate

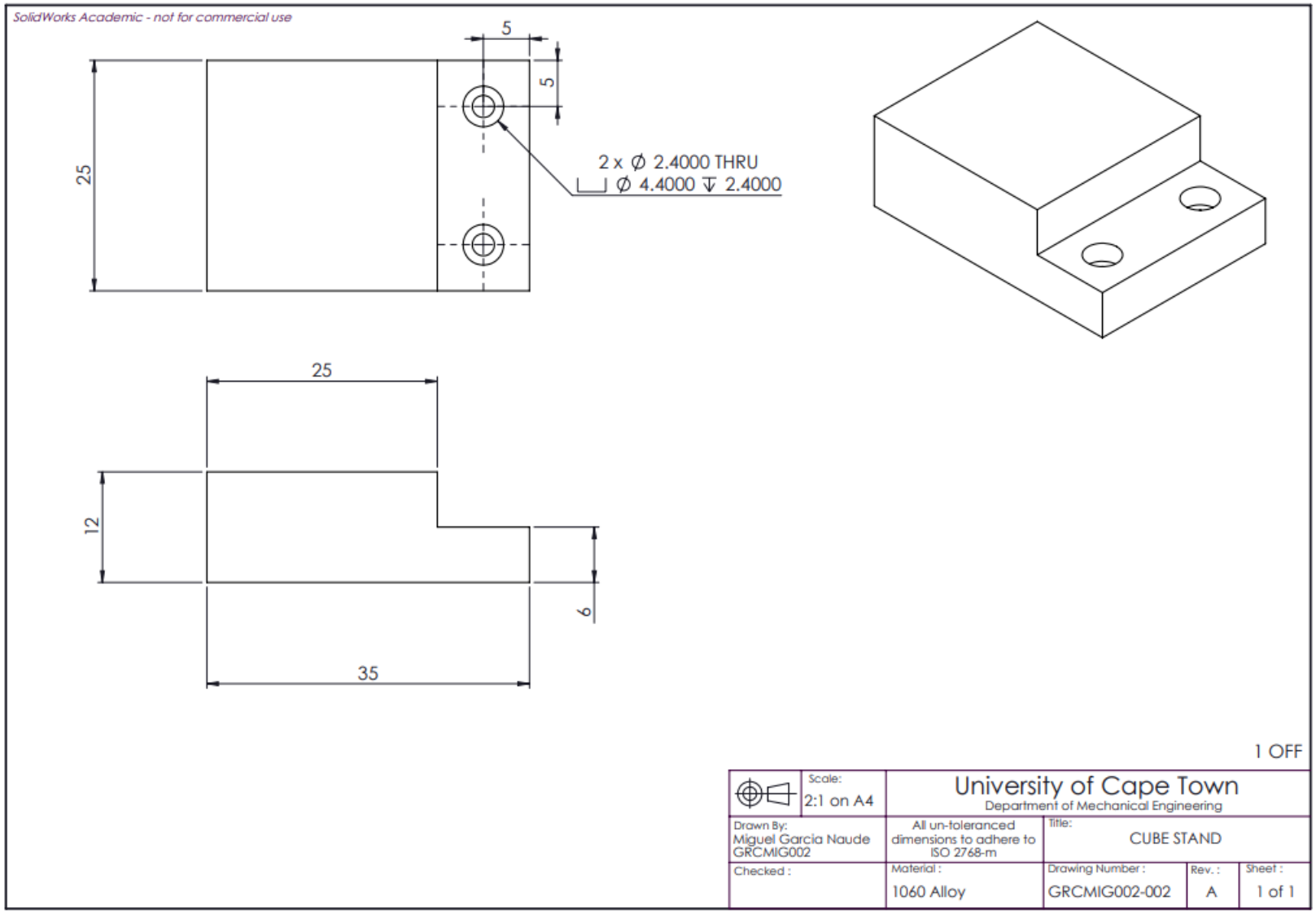


SOLIDWORKS Educational Product. For Instructional Use Only.

22.4.5 Camera Mount

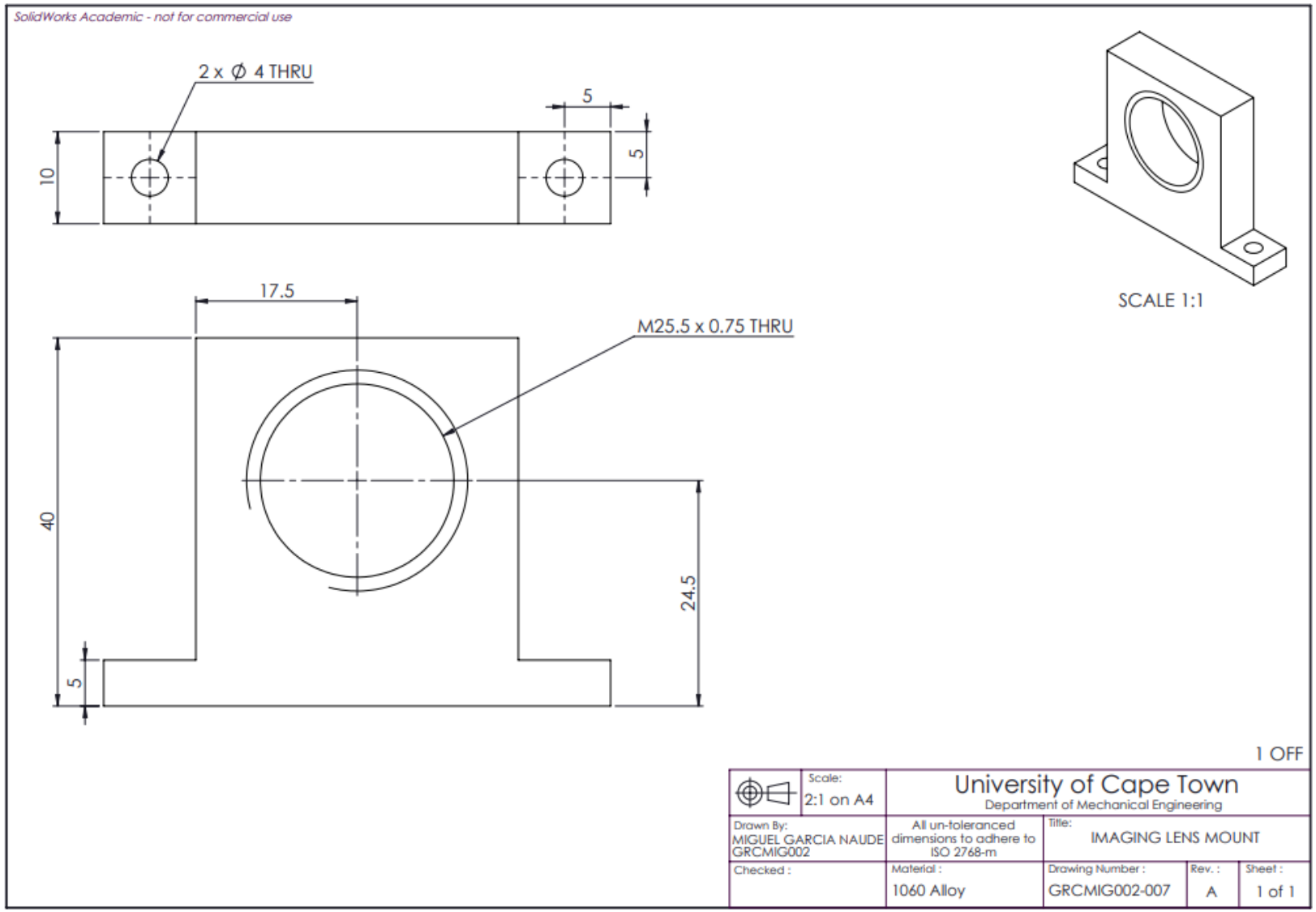


22.4.6 Cube Beam Splitter Mount



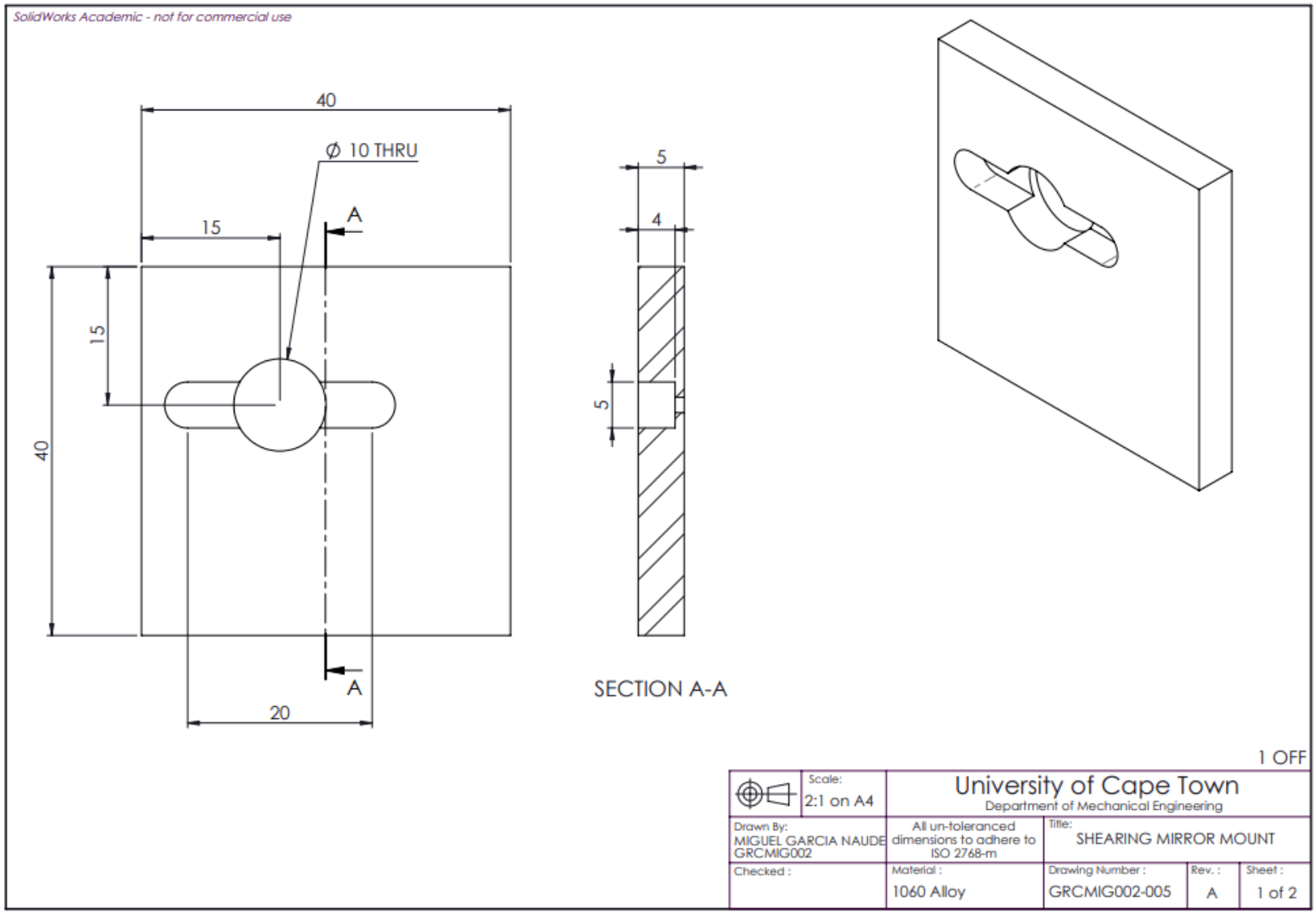
SOLIDWORKS Educational Product. For Instructional Use Only.

22.4.7 Imaging Lens Mount

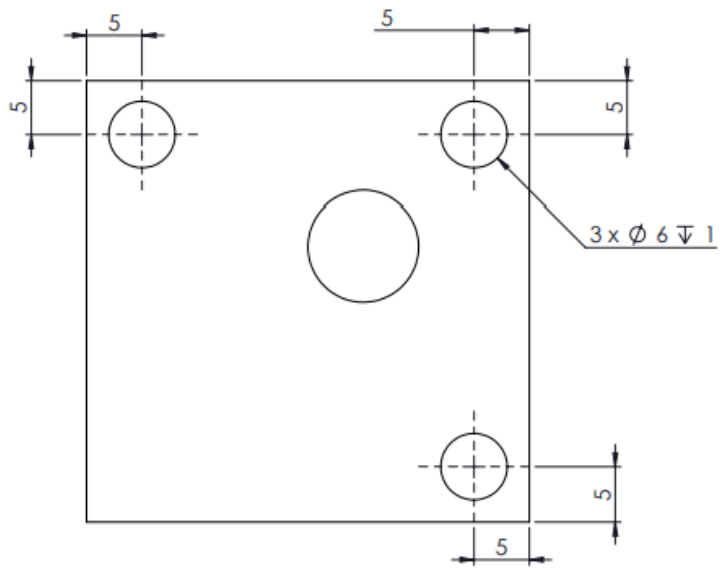


SOLIDWORKS Educational Product. For Instructional Use Only.

22.4.8 Shearing Mirror Mount

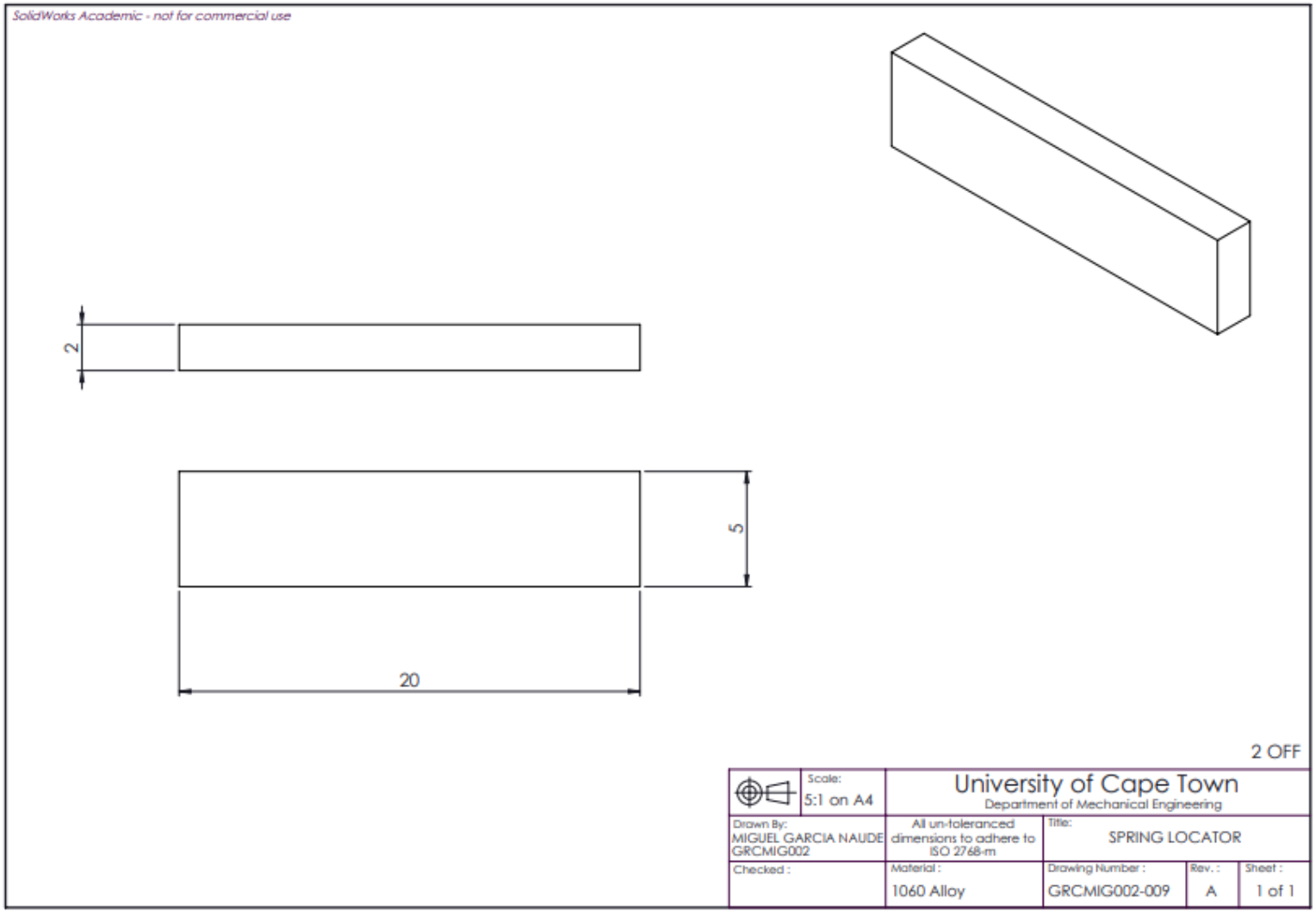


SOLIDWORKS Educational Product. For Instructional Use Only.

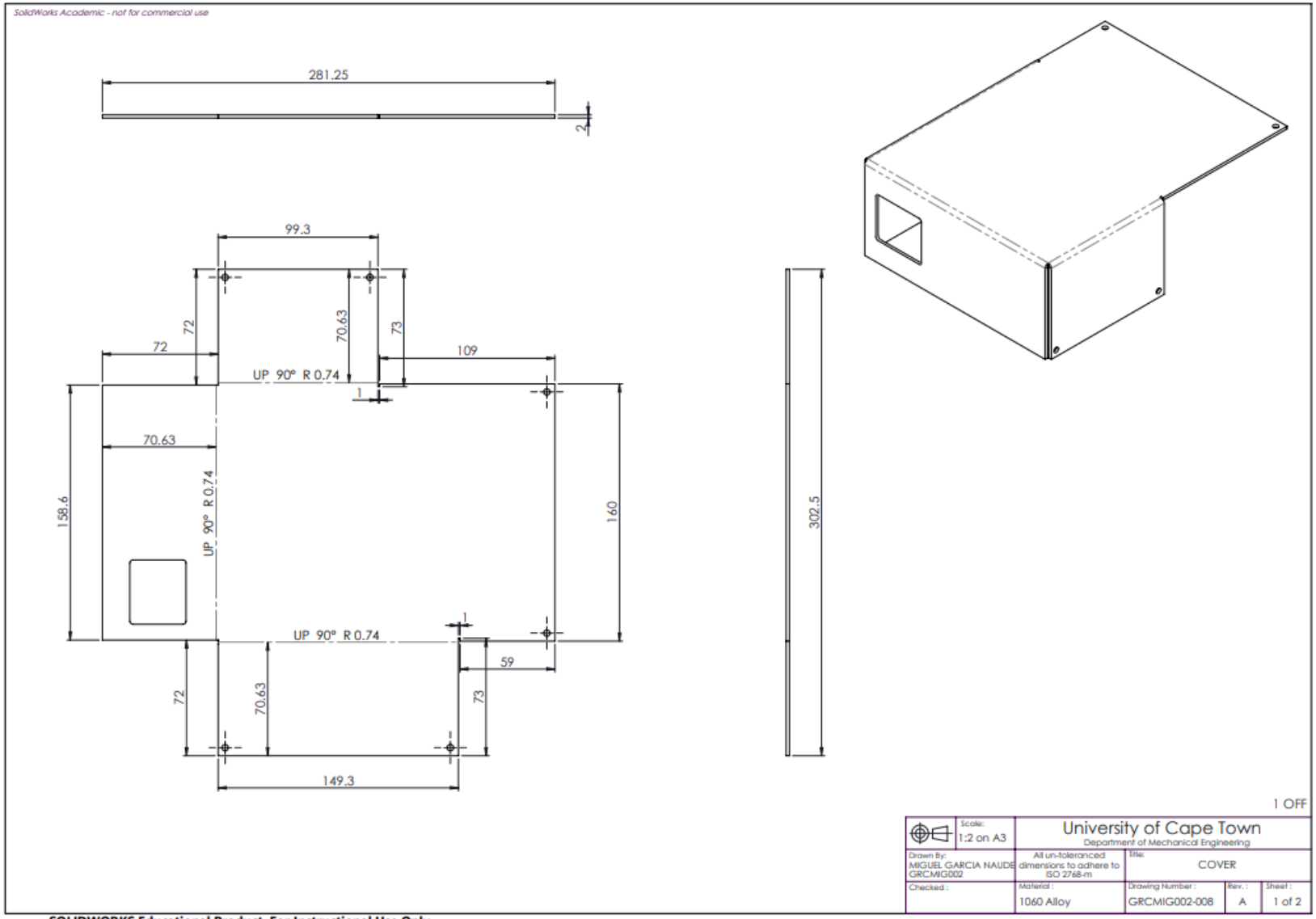


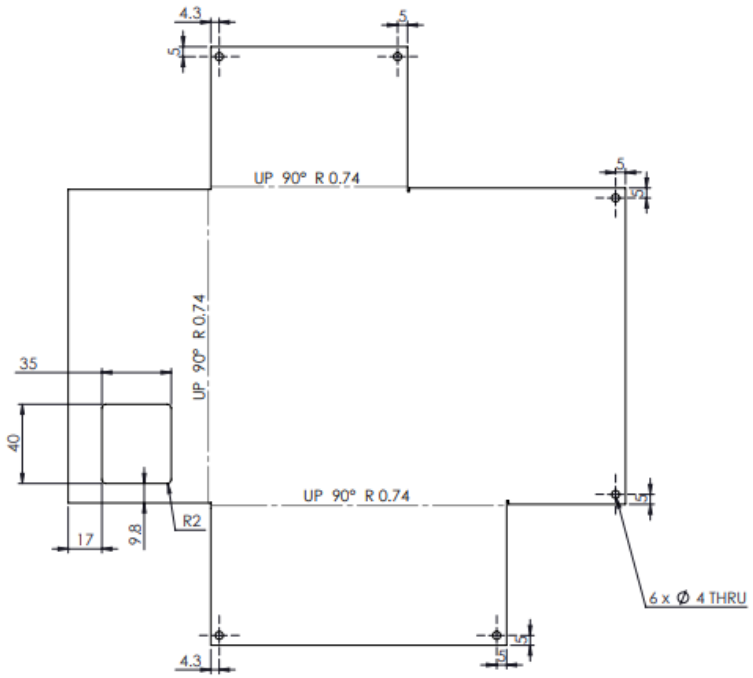
Scale: 2.1 on A4	University of Cape Town Department of Mechanical Engineering			
Drawn By: MIGUEL GARCIA NAUDE GRCMIG002	All un-toleranced dimensions to adhere to ISO 2768-m	Title: SHEARING MIRROR MOUNT		
Checked :	Material : 1060 Alloy	Drawing Number : GRCMIG002-005	Rev. : A	Sheet : 2 of 2

22.4.9 Spring Locator



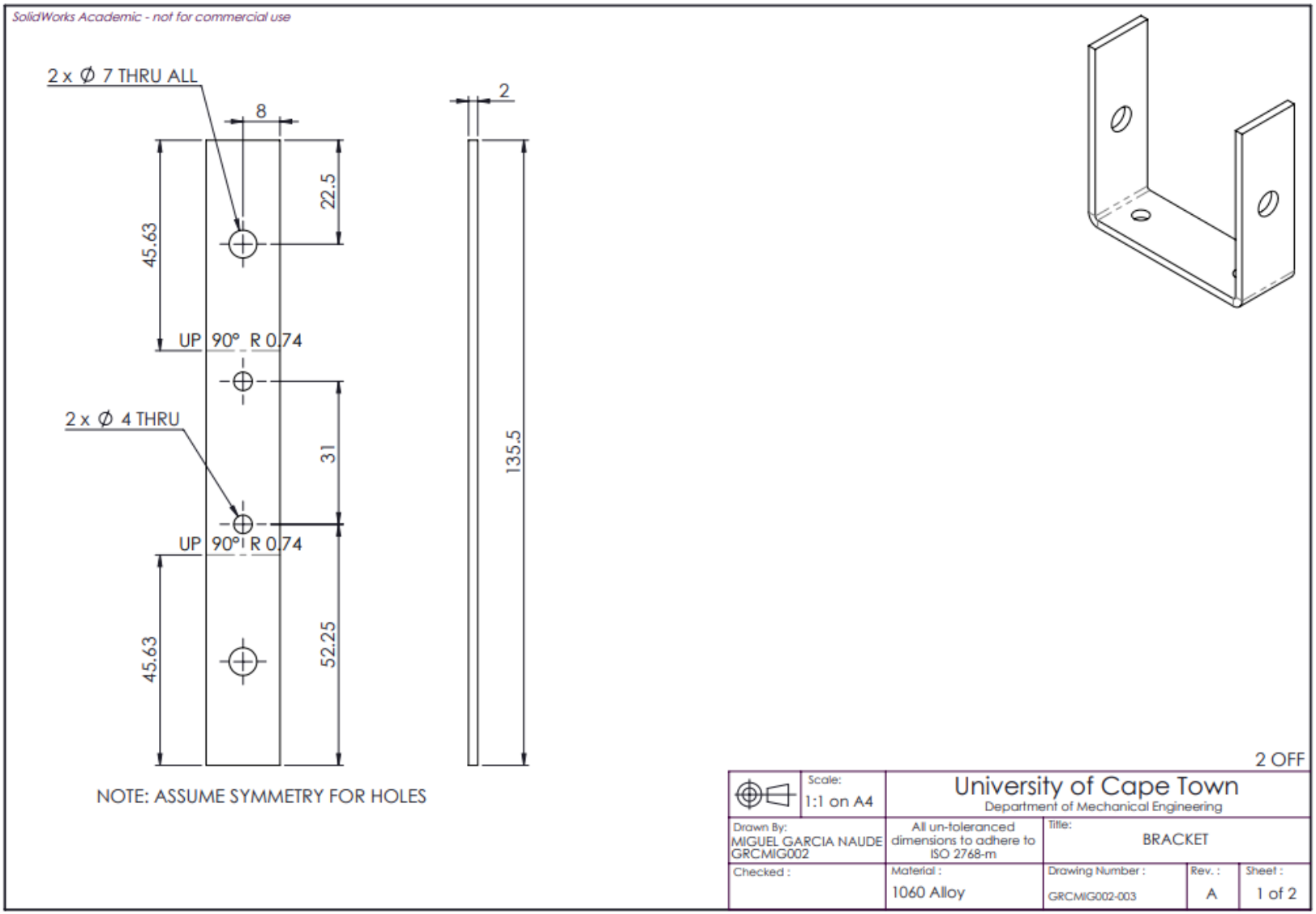
22.4.10 Cover

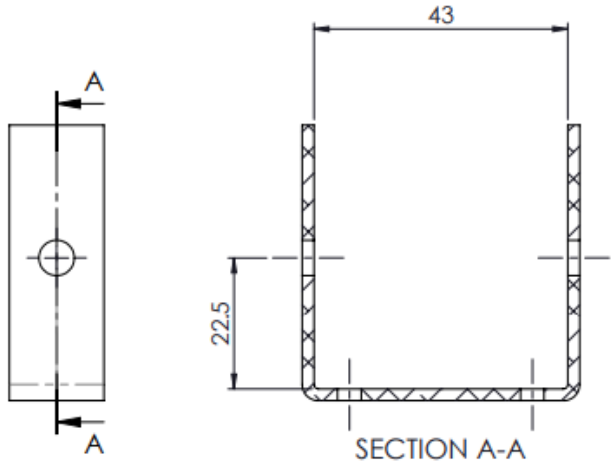

SolidWorks Academic - not for commercial use
281.25
2
99.3
72
72
70.63
73
109
UP 90° R 0.74
1
70.63
158.6
UP 90° R 0.74
160
302.5
UP 90° R 0.74
1
59
72
70.63
73
149.3
1 OFF
Scale: 1:2 on A3
University of Cape Town
Department of Mechanical Engineering
Drawn By: MIGUEL GARCIA NAUDE GRCMIG002
All un-toleranced dimensions to adhere to ISO 2768-m
Title: COVER
Checked :
Material : 1060 Alloy
Drawing Number : GRCMIG002-008
Rev. : A
Sheet : 1 of 2
SOLIDWORKS Educational Product. For Instructional Use Only.



Scale: 1:2 on A3	University of Cape Town Department of Mechanical Engineering			
Drawn By: MIGUEL GARCIA NAUDE GRCMIG002	All un-toleranced dimensions to adhere to ISO 2768-m	Title: COVER		
Checked :	Material : 1060 Alloy	Drawing Number : GRCMIG002-008	Rev. : A	Sheet : 2 of 2

22.4.11 Brackets



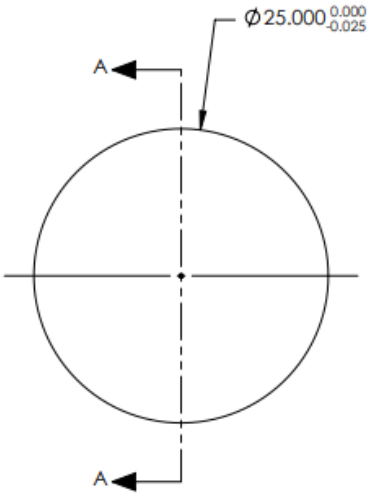


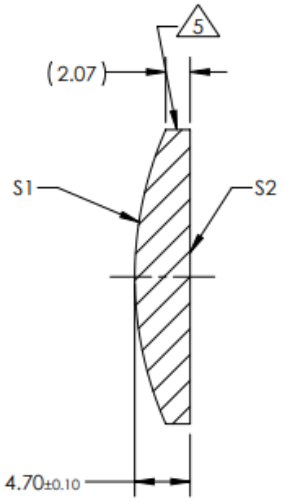
Scale: 1:1 on A4	University of Cape Town Department of Mechanical Engineering			
Drawn By: MIGUEL GARCIA NAUDE GRCMIG002	All un-toleranced dimensions to adhere to ISO 2768-m	Title: BRACKET		
Checked :	Material : 1060 Alloy	Drawing Number : GRCMIG002	Rev. : A	Sheet : 2 of 2

22.4.12 60 mm focal length lens, 25 mm diameter (Obtained from Edmund Optics) [37]

NOTES:

1. SUBSTRATE:
   GRADE A FINE ANNEALED
   SCHOTT: N-BK7 517/642
2. ROHS COMPLIANT
3. CENTERING TOLERANCE (AT 587.6nm):
   BEAM DEVIATION (HALF ANGLE): <1 ARCMIN
4. COATING (APPLY ACROSS COATING APERTURE)
   S1: NONE
   S2: NONE
5. FINE GRIND SURFACE
6. POWER, IRREGULARITY, AND SURFACE QUALITY SPECIFICATIONS APPLY ACROSS CLEAR APERTURE
7. FOCAL LENGTH (EFL): 60.00mm±1%
   BACK FOCAL LENGTH (BFL): 56.90mm
8. PROTECTIVE BEVEL AS NEEDED
9. DESIGN WAVELENGTH: 587.6nm





SECTION A-A

FOR INFORMATION ONLY:
DO NOT MANUFACTURE PARTS TO THIS DRAWING

	S1	S2
SHAPE	CONVEX	PLANO
RADIUS	31.01	INFINITY
SURFACE QUALITY	40 - 20	40 - 20
MIN CLEAR APERTURE	Ø24.00	Ø24.00
MIN COATING APERTURE	N/A	N/A
POWER AT 632.8nm	3.00 RINGS	3.00 RINGS
IRREGULARITY AT 632.8nm	0.50 RINGS	0.50 RINGS

SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE
DIMENSIONS ARE FOR REFERENCE ONLY

Edmund Optics®

THIRD ANGLE PROJECTION

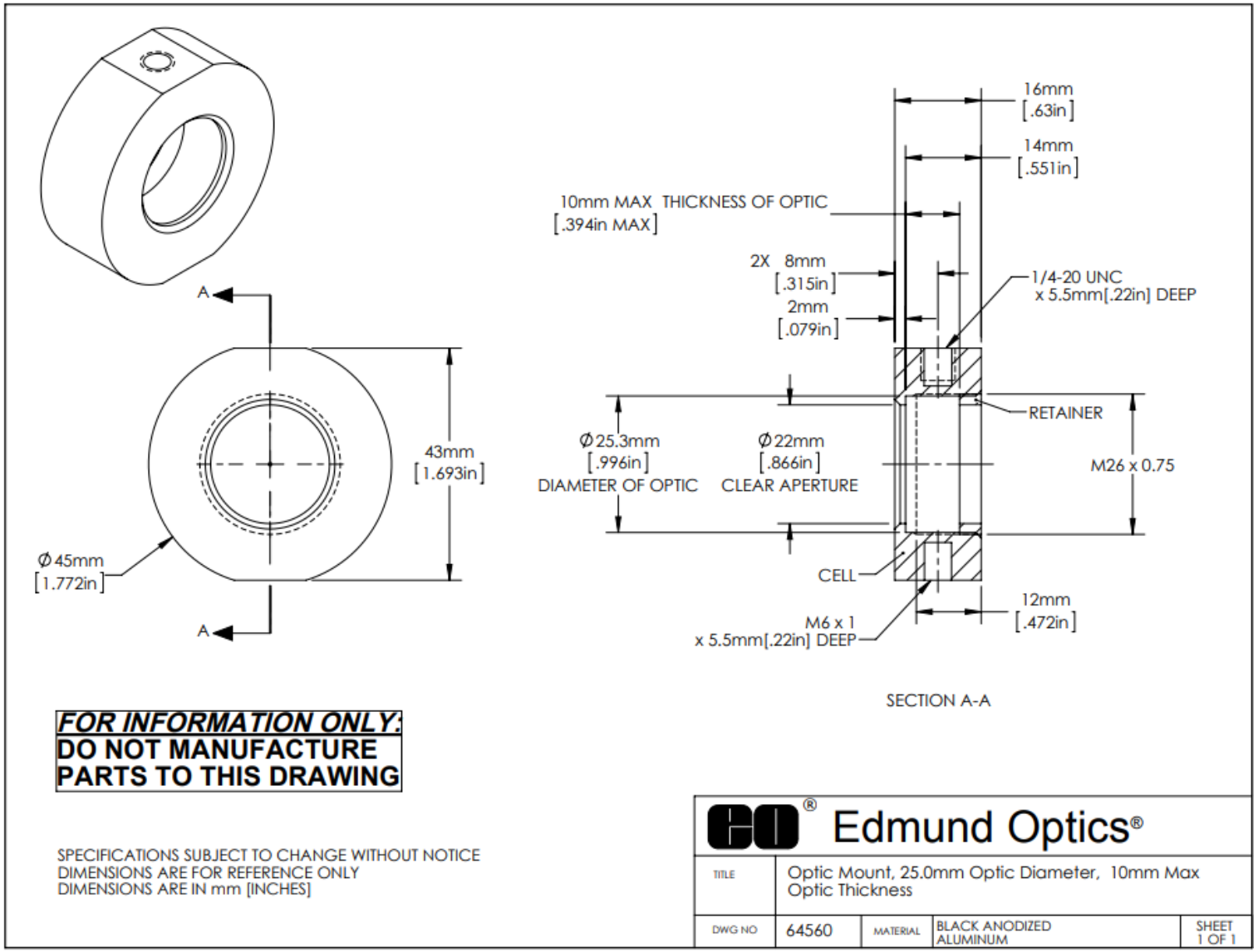
TITLE: 25.0mm Dia. x 60.0mm FL, Uncoated, Plano-Convex Lens

ALL DIMS IN: mm

DWG NO: 45127

SHEET 1 OF 1

22.4.13 Optic Mount, 25 mm Optic Diameter [36]



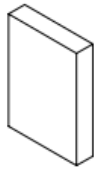
22.4.14 Mirrors (Supplied by supervisor) [52]

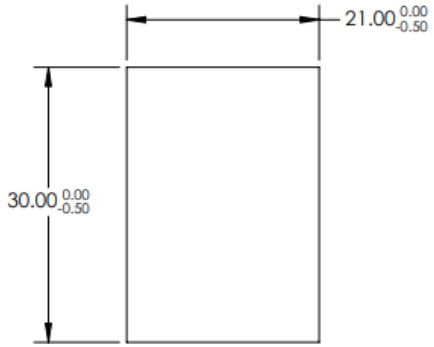
NOTES:

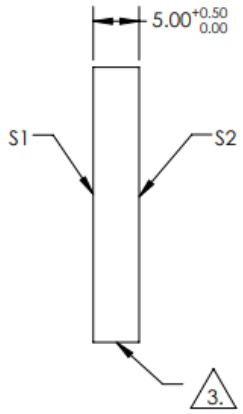
1. SUBSTRATE:
   BOROFLOAT®
2. COATING SPECIFICATION:

   S1: Protected Aluminum (400-2000nm)
   R(AVG) >85% @ 400 - 700nm
   R(AVG) >90% @ 400 - 2000nm

3. FINE GROUND SURFACE
4. ROHS COMPLIANT







FOR INFORMATION ONLY:
DO NOT MANUFACTURE PARTS TO THIS DRAWING

SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE
DIMENSIONS ARE FOR REFERENCE ONLY

	S1	S2
SHAPE	PLANO	PLANO
SURFACE QUALITY	60-40	GROUND
CLEAR APERTURE	17.85 x 25.50	N/A
BEVEL	PROTECTIVE AS NEEDED	PROTECTIVE AS NEEDED

THIRD ANGLE PROJECTION | ALL DIMS IN mm | TITLE: 21 x 30mm Protected Aluminum, λ/4 Mirror | DWG NO: 32454 | SHEET 1 OF 1

Edmund Optics®

22.4.15 25 mm Cube Beamsplitter (supplied by supervisor) [53]

NOTES:

1. SUBSTRATE MATERIAL:
   SCHOTT: N-SF11 785/258

   EQUIVALENT SUBSTITUTIONS ALLOWED

2. CENTERING TOLERANCE (AT 587.6nm):
   BEAM DEVIATION: <3 arc min.

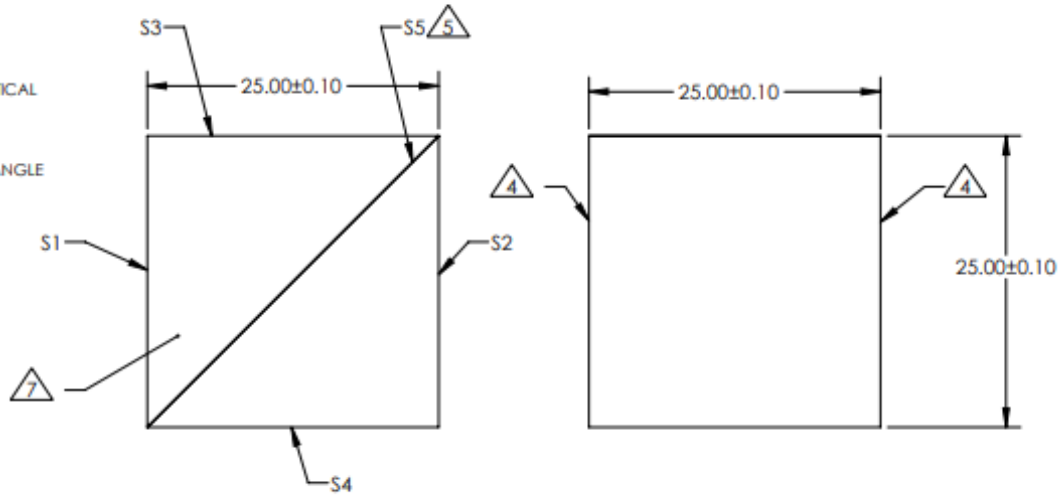
3. COATING(APPLY ACROSS CLEAR APERTURE)
   COATING SPECIFICATIONS APPLY FROM 420 TO 680nm
   S1, S2, S3, & S4: R(AVG) <0.5%
   S5: MULTILAYER DIELECTRIC TO ACHIEVE
   Tp > 90%
   Rs > 99%
   EXTINCTION RATIO: Tp/Ts > 500:1

4. FINE GROUND SURFACE

5. ELEMENTS TO BE CEMENTED WITH NORLAND OPTICAL ADHESIVE NOA 61

6. PROTECTIVE BEVEL AS NEEDED

7. MARK IDENTIFIES BEAMSPLITTER COATED RIGHT ANGLE PRISM



FOR INFORMATION ONLY:
DO NOT MANUFACTURE PARTS TO THIS DRAWING

SPECIFICATIONS SUBJECT TO CHANGE WITHOUT NOTICE
DIMENSIONS ARE FOR REFERENCE ONLY

	S1, S2, S3, S4	S5
SHAPE	PLANO	PLANO
SURFACE ACCURACY AT 632.8nm (P-V)	1/8 WAVE	1/8 WAVE
SURFACE QUALITY	40 - 20	40 - 20
MIN CLEAR APERTURE	22.50 x 22.50mm	31.82 x 22.50mm

EFL (AT 587.6nm)	NA	Edmund Optics®	
BFL (AT 587.6nm)	NA	TITLE	Polarizing Cube Beamsplitter, 25mm, VIS
ALL DIMS IN	mm	DWG NO	49002 SHEET 1 OF 1

22.5 Appendix E - FlyCapture2 SDK Programming Basics and Usage

The following section details the basics of setting up the Chameleon3 camera for use in a Python environment using the PyCapture2 library which is to be installed and imported.

As highlighted in the PyCapture2 API reference [54] and the example code provided [55], the computer needs to verify that at least one camera is detected. This is done using the following [54]:

1. The Globally Unique Identifier (GUID) of the camera needs to be identified:

```
bus = PyCapture2.BusManager()
```

2. The number of cameras connected to the PC need to be determined:

```
num_cams = bus.getNumOfCameras()
```

3. If at least one camera is not detected, the application should close:

```
if not num_cams:
    print('Insufficient number of cameras. Exiting...')
    exit()
```

Listing 16: Identifying the Chameleon3 Camera

Secondly, a camera object needs to be created and connected to one of the cameras on the bus. An example of this is given in the example code provided and looks like the following [54][55]:

1. Create a camera object:

```
c = PyCapture2.Camera()
```

2. Select the camera on the 0^{th} index (to distinguish from other cameras if more than one is connected), this returns the GUID of the camera:

```
uid = bus.getCameraFromIndex(0)
```

3. Connect the camera object to the physical camera:

```
c.connect(uid)
```

Listing 17: Connecting to the Chameleon3 Camera

Lastly, data can be captured from the camera to the image buffers. From this point, image data can be manipulated, displayed and saved. Once one has completed using the camera, capture of data from the camera to the buffers is halted and the camera is disconnected [54]:

1. To start capturing data from the camera to the image buffers:

```
c.startCapture()
```

2. To retrieve data from the image buffers:

```
buffer_image = c.retriveBuffer()
```

3. To stop capturing data from the camera to the image buffers:

```
c.stopCapture()
```

4. To disconnect the camera object from the physical camera:

```
c.disconnect()
```

Listing 18: Capturing data from the Chameleon3 Camera

22.6 Appendix F - GUI Application Code

22.6.1 app.py file

```
# Imports
import tkinter as tk
from tkinter import ttk
from frames import VideoFrame, ControlsFrame, SetupFrame
# Above: import Frame classes from frames folder

from controller import Controller
import PyCapture2

#Set DPI awareness (needed for some newer screens)
try:
    from ctypes import windll
    windll.shcore.SetProcessDpiAwareness(1)
except:
    pass


#Class for Root of application
class DS_GUI(tk.Tk):                        #create a class of type Tk class
    def __init__(self, *args, **kwargs):    #initialise the class
        super().__init__(*args, **kwargs)   #call the constructor of the Tk
                                            #class to inherit properties

        self.controller = None
        screen_width = self.winfo_screenwidth()     #get the width of the screen
        screen_height = self.winfo_screenheight()   #get the height of the screen

        #Configurations of the Root
        self.title("Digital Shearography - Project 55")
```

```
        # Above: title of application
        self.geometry("{}x{}+0+0".format(screen_width, screen_height))
        # Above: Set the size of the window to fullscreen by default

        self.columnconfigure(0, weight=1)   #configure weights of the columns and rows
        self.rowconfigure(0, weight=1)


        #Main Container to Hold the Class Frames (video, control and setup)
        main_container = ttk.Frame(self)
        main_container.grid(row=0, column=0)

        main_container.columnconfigure(0, weight=2)
        main_container.columnconfigure(1, weight=1)
        # Above: column 0 of main_container is given 2 times the space
        #        of row 1

        main_container.rowconfigure(0, weight=1)
        main_container.rowconfigure(1, weight=7)
        # Above: row 0 of main_container is given 7 times the space
        #        of row 1


        # Frames inside main container
        # 1. ControlsFrame
        self.controls_frame = ControlsFrame(main_container, controller)
        self.controls_frame.grid(row=0, column=0, columnspan=2, pady=(10), sticky="NSEW")

        # 2. VideoFrame
        self.video_frame = VideoFrame(main_container, screen_width, screen_height, c, 33)
        # Above: pass in parent frame, screen height & width, camera object and delay between refreshing video frames
```

```
        self.video_frame.grid(row=1, column=0, pady=(30), padx=(30), sticky="NSEW")
        self.video_frame.display_video()

        # 3. SetupFrame
        self.setup_frame = SetupFrame(main_container, controller)
        self.setup_frame.grid(row=1, column=1, pady=(30), padx=(30), sticky="NSEW")

    def set_controller(self, controller):   #pass in Controller class as the controller
        self.controller = controller
        # Below: create instances of the frame classes
        # so that they can be accessed in the controller
        controller.set_video_frame(self.video_frame)
        controller.set_controls_frame(self.controls_frame)
        controller.set_setup_frame(self.setup_frame)



################################################################################################################
################################################################################################################

######################### CAMERA SETUP ########################

bus = PyCapture2.BusManager()                   # identify the Globally Unique Identifier (GUID) of the camera
num_cams = bus.getNumOfCameras()                # determine the number of cameras connected to the PC
print('Number of cameras detected:', num_cams)  # print the number of cameras detected
if not num_cams:                                # if no cameras are detected exit the application
    print('Insufficient number of cameras. Exiting...')
    exit()

# Select camera on 0th index
c = PyCapture2.Camera()             # Assign Camera object to C
```

```
uid = bus.getCameraFromIndex(0)     # select camera on 0th index
c.connect(uid)                      # connect camera object to physical camera

# Disable On-board Image Processing
c.setProperty(type = PyCapture2.PROPERTY_TYPE.AUTO_EXPOSURE, onOff = False)  #Disable Auto Exposure

print('Starting image capture...')
c.startCapture()                    # start capturing data from the camera to the image buffers

#######################################################################################################
#######################################################################################################

controller = Controller()           # create object controller of the Controller() class
app = DS_GUI()                      # create a DS_GUI object
app.set_controller(controller)      # use the DS_GUI.set_controller() methods to assign the controller
app.mainloop()                      # call DS_GUI inherited method, mainloop() to run the application

try:                                # try:
    c.stopCapture()                 # stop capturing data from the camera to the image buffers
    print("stopping capture")
    c.disconnect()                  # disconnect the camera object from the physical camera
    print("camera disconnected")

except:                             # except:
    exit()                          # close application
```

22.6.2 controls_frame.py

```
# Imports
import tkinter as tk
from tkinter import ttk

class ControlsFrame(ttk.Frame):     #create a class of type ttk.Frame
    def __init__(self, parent, controller): #initialise the class
        super().__init__(parent)    #call the constructor of the ttk.Frame
                                    #class to inherit its and the parent's
                                    #properties

        # Main Control Frame
        main_control_frame = ttk.Frame(self)
        main_control_frame.grid(row=0, column=0)

        # Inside Main Control Frame
            #Snapshot and New window frame & widgets
        snapshot_frame = ttk.Frame(main_control_frame, padding=15)
        snapshot_frame.grid(row=0, column=0)

        # New Inspection button
        self.new_inspection_button = ttk.Button(snapshot_frame, text="New Inspection", state="disabled", command=control
ler.reset)
        self.new_inspection_button.grid(row=0, column=0)

        # Snapshot Frame button
        self.snapshot_button = ttk.Button(snapshot_frame, text="Snapshot Frame", state="enabled", command=controller.sav
e_snapshot)
        self.snapshot_button.grid(row=0, column=1)
```

```
            #V ideo Controls Frame and Buttons
        video_controls_frame = ttk.Frame(main_control_frame, padding=15)
        video_controls_frame.grid(row=0, column=1)

        # Play button
        self.play_button = ttk.Button(video_controls_frame, text="Play", command=controller.play, state="disabled")
        self.play_button.grid(row=0, column=0)

        # Pause button
        self.pause_button = ttk.Button(video_controls_frame, text="Pause", command=controller.pause, state="disabled")
        self.pause_button.grid(row=0, column=1)

        # Stop button
        self.stop_button = ttk.Button(video_controls_frame, text="Stop", command = controller.stop, state="disabled")

        self.stop_button.grid(row=0, column=2)


            # Real Time Video/Fringe Pattern Frame and Buttons
        video_mode_frame = ttk.Frame(main_control_frame, padding=20)
        video_mode_frame.grid(row=0, column=2)

        # Video Mode button
        self.real_time_video_button = ttk.Button(video_mode_frame, text="Video Mode", state="disabled", command = contro
ller.display_video)
        self.real_time_video_button.grid(row=0, column=0)

        # Fringe Pattern Mode button
        self.fringe_pattern_button = ttk.Button(video_mode_frame, text="Fringe Pattern Mode", state="disabled", command
= controller.display_fringes)
        self.fringe_pattern_button.grid(row=0, column=1)
```

```
    # Images folder (enter directory location for saved images)
image_folder_frame = ttk.Frame(main_control_frame, padding=20)
image_folder_frame.grid(row=0, column=3)

# Image Folder label
image_folder_label = ttk.Label(image_folder_frame, text="Image Save Location")
image_folder_label.grid(row=0, column=0)

# Image Folder Entry
self.folder_directory = tk.StringVar()
self.folder_entry = ttk.Entry(image_folder_frame, width=20, textvariable = self.folder_directory)
self.folder_entry.grid(row=1, column=0)

# Set Folder button
self.folder_set_button=ttk.Button(image_folder_frame, text="Set Folder", command = controller.get_save_location)

self.folder_set_button.grid(row=1, column=1)


    # Initialisation File Import
initialisation_frame = ttk.Frame(main_control_frame, padding = 20)
initialisation_frame.grid(row=0, column=4)

# Initialisation File label
import_config_file_label = ttk.Label(initialisation_frame, text="Initialisation file")
import_config_file_label.grid(row=0, column=0)
```

```
        # Import config button
        self.config_button = ttk.Button(initialisation_frame, text="Import Config File", command=controller.set_config)

        self.config_button.grid(row=1, column=0)
```

22.6.3 video_frame.py

```
# Imports
import tkinter as tk
from tkinter import ttk
import PIL as pl
from PIL import Image, ImageTk, ImageEnhance
import cv2
import numpy as np

class VideoFrame(ttk.Frame):    # create a class of type ttk.Frame
    def __init__(self, parent, screen_width, screen_height, cam, delay):
                                    # initialise the class
        super().__init__(parent)    # call the constructor of the ttk.Frame
                                    # class to inherit its and the parent's
                                    # properties

        self.screen_width = screen_width    # integers passed in from app.py
        self.screen_height = screen_height  # integers passed in from app.py

        self.cam = cam      # camera object that is passed into the VideoFrame class
        self.delay = delay  # the delay between frame updates of the video display

        self.video_pause = False            # A flag used to determine if video must be paused or playing
        self.display_fringes_flag = False   # A flag to be used to determine if fringes or real time video
                                            # must be displayed
        self.mode_flag = False              # A flag used to alter the inspection mode from regular to 4f mode
        self.ref_image = None               # An object to store the reference image once it is taken
        self.img = None                     # An object to store the current camera image
```

```
        # Camera Variable Default Values
        self.brightness_value = 1.0         # brightness value used in image enhancement
        self.contrast_value = 1.0           # contrast value used in image enhancement

        # Frames
        video_frame = ttk.Frame(self)       # create the video_frame frame
        video_frame.grid()


        # Video Label
        self.video_display_label = ttk.Label(video_frame)
        self.video_display_label.grid(row=0, column=0)


    def capture_reference_image(self):          # method to capture and save reference image (only to be called once)
        ref_buffer = self.cam.retrieveBuffer()  # retrieve data from the camera buffer
        self.ref_image=np.array(ref_buffer.getData(),dtype="uint8").reshape((ref_buffer.getRows(),ref_buffer.getCols()))

        # Above: convert ref_buffer to numpy array to be used in subtraction of images

        if self.mode_flag == True:  # If 4f inspection mode is enabled, rotate numpy image 180 degrees
            self.ref_image = np.flip(self.ref_image, axis=0)
            self.ref_image = np.flip(self.ref_image, axis=1)

        self.ref_img = pl.Image.fromarray(self.ref_image)
        # Above: convert numpy image array to image object

    def display_video(self):    #method to display real-time video or fringe patterns to display
        image = self.cam.retrieveBuffer()   # retrieve data from the camera buffer
        cv_image1 = np.array(image.getData(),dtype="uint8").reshape((image.getRows(),image.getCols()))
        # Above: convert image to numpy array
```

```
    if self.mode_flag == True:  # If 4f inspection mode is enabled, rotate numpy image 180 degrees
        cv_image1 = np.flip(cv_image1, axis=0)
        cv_image1= np.flip(cv_image1, axis=1)

#Display real time video or fringe patterns depending on the state of the flag
    if not self.display_fringes_flag:   # if fringe pattern display mode is disabled
        self.img = pl.Image.fromarray(cv_image1)
        # Above: convert numpy image array to image object

        contrastImg = ImageEnhance.Contrast(self.img)
        contrastedImage = contrastImg.enhance(self.contrast_value)
        # Above: adjusting image to desired contrast

        brightnessImg = ImageEnhance.Brightness(contrastedImage)
        Enhanced_img = brightnessImg.enhance(self.brightness_value)
        # Above: adjusting image to desired brightness

        self.photo = Enhanced_img.resize((self.screen_width-700,self.screen_height-300))
        # Above: resize image

        display_image = ImageTk.PhotoImage(self.photo)
        # Above: convert image to tkinter compatible photo image

        self.video_display_label.config(image=display_image)
        self.video_display_label.image = display_image
        # Above: display image on tkinter label

    elif self.display_fringes_flag==True:   #if fringe pattern display mode is enabled
        self.difference = cv2.absdiff(cv_image1,self.ref_image)
        #Above: subtract the current image from the reference image
```

```
        self.img = pl.Image.fromarray(self.difference)
        # Above: convert numpy image array to image object

        contrastImg = ImageEnhance.Contrast(self.img)
        contrastedImage = contrastImg.enhance(self.contrast_value)
        # Above: adjusting image to desired contrast

        brightnessImg = ImageEnhance.Brightness(contrastedImage)
        Enhanced_img = brightnessImg.enhance(self.brightness_value)
        # Above: adjusting image to desired brightness

        self.photo = Enhanced_img.resize((self.screen_width-700,self.screen_height-300))
        # Above: resize image

        display_image = ImageTk.PhotoImage(self.photo)
        # Above: convert image to tkinter compatible photo image

        self.video_display_label.config(image=display_image)
        self.video_display_label.image = display_image
        # Above: display image on tkinter label

    if not self.video_pause:    #if the display is not paused (note below)
        self.video_display_label.after(self.delay, lambda: self.display_video())
        # Above: call the display_video() method after delay number of milliseconds to refresh display


def pause_video(self):  # method to pause video display
    self.video_pause = True

def play_video(self):   # method to resume video display
```

```
        self.video_pause = False
        self.display_video()

    def stop_video(self):   # method to stop video display and inspection process
        # Set the video display to a black image:
        self.stop_image = Image.open("./Assets/stop.JPG").resize((self.screen_width-700,self.screen_height-300))
        self.stop_photo = ImageTk.PhotoImage(self.stop_image)

        # Set video_display_label to black image when inspection is stopped
        self.video_display_label.config(image=self.stop_photo)
        self.video_display_label.image = self.stop_photo
```

22.6.4 setup_frame.py

```
#Imports
import tkinter as tk
from tkinter import ttk

class SetupFrame(ttk.Frame):            #create a class of type ttk.Frame
    def __init__(self, parent, controller): #initialise the class
        super().__init__(parent)        #call the constructor of the ttk.Frame
                                        #class to inherit its and the parent's
                                        #properties

        self.columnconfigure(0, weight=1)

        #inspection setup frame
        inspection_setup_frame = ttk.Frame(self)
        inspection_setup_frame.grid(row=0, column=0)

        #Inside inspection setup frame
            #Inspection Setup Label and Start Inspection Button
        inspection_setup_label = ttk.Label(inspection_setup_frame, text="INSPECTION SETUP")
        inspection_setup_label.grid(row=0, column=0, pady = 5)

        self.start_inspection_button = ttk.Button(inspection_setup_frame, text = "START INSPECTION", command = controlle
r.start)
        self.start_inspection_button.grid(row=1, column=0, pady=5, ipadx=40, ipady=15)

            #Inspection Mode
        inspection_mode_frame = ttk.Frame(inspection_setup_frame)
        inspection_mode_frame.grid(row=2, column=0, pady=5)

        # Inspection Mode label
```

```
        inspection_mode_label = ttk.Label(inspection_mode_frame, text="Inspection Mode")
        inspection_mode_label.grid(row=0, column=0, columnspan=2)

        # Normal Mode button
        self.normal_mode_button = ttk.Button(inspection_mode_frame, text="Normal", state = "disabled", command = control
ler.set_normal_mode)
        self.normal_mode_button.grid(row=1, column=0)

        # 4f mode button (will rotate image 180 degrees if active)
        self.four_f_mode_button = ttk.Button(inspection_mode_frame, text="4f", command = controller.set_four_f_mode)

        self.four_f_mode_button.grid(row=1, column=1)


            #Inspection Time Limit Frame, Label, check button, entry field
        inspection_time_limit_Frame = ttk.Frame(inspection_setup_frame)
        inspection_time_limit_Frame.grid(row=3, column=0, pady=5)

        # Inspection time limit label
        inspection_time_limit_label = ttk.Label(inspection_time_limit_Frame, text="Inspection Time Limit (minutes)", pad
ding = 10)
        inspection_time_limit_label.grid(row=0, column=0, columnspan=3, sticky="EW")

        # Inspection time limit Checkbutton
        self.inspection_time_limit_option = tk.BooleanVar()

        self.inspection_time_limit_check_button = ttk.Checkbutton(inspection_time_limit_Frame,
        text="Enable",
        variable=self.inspection_time_limit_option,
        command = controller.inspection_time_limit_command,
        onvalue=True, offvalue=False)
```

```
        self.inspection_time_limit_check_button.grid(row=1, column=0, columnspan=3)

        # Inspection time limit Entry
        self.inspection_time_limit_mins = tk.DoubleVar()
        self.inspection_time_limit_entry = ttk.Entry(inspection_time_limit_Frame, width=6, textvariable=self.inspection_
time_limit_mins, state="disabled")
        self.inspection_time_limit_entry.grid(row=2, column=0)

        # Inspection time limit set button
        self.inspection_time_limit_set_button = ttk.Button(inspection_time_limit_Frame, text="Set", state="disabled", co
mmand = controller.inspection_time_limit_set)
        self.inspection_time_limit_set_button.grid(row=2, column=1)

        # Inspection time limit value label (displays the set value)
        self.inspection_time_limit_value_label = ttk.Label(inspection_time_limit_Frame, text="None")
        self.inspection_time_limit_value_label.grid(row=2, column=2)


            #Frame Capture Interval
        Frame_capture_interval_frame = ttk.Frame(inspection_setup_frame)
        Frame_capture_interval_frame.grid(row=4, column=0, pady=5)

        # Frame Capture Interval label
        Frame_capture_interval_label = ttk.Label(Frame_capture_interval_frame, text="Frame Capture Interval (seconds)",
padding = 10)
        Frame_capture_interval_label.grid(row=0, column=0, columnspan=3, sticky="EW")

        # Frame Capture Interval Checkbutton
        self.frame_capture_interval_option = tk.BooleanVar()

        self.frame_capture_interval_check_button = ttk.Checkbutton(Frame_capture_interval_frame,
```

```
        text="Enable",
        variable=self.frame_capture_interval_option,
        command = controller.frame_capture_interval_command,
        onvalue=True, offvalue=False)
        self.frame_capture_interval_check_button.grid(row=1, column=0, columnspan=3)

        # Frame Capture Interval Entry
        self.frame_capture_interval_seconds = tk.IntVar()
        self.frame_capture_interval_entry = ttk.Entry(Frame_capture_interval_frame, width=6, textvariable=self.frame_cap
ture_interval_seconds, state="disabled")
        self.frame_capture_interval_entry.grid(row=2, column=0)

        # Frame Capture Interval set button
        self.frame_capture_interval_set_button = ttk.Button(Frame_capture_interval_frame, text="Set", state="disabled",
command = controller.frame_capture_interval_set)
        self.frame_capture_interval_set_button.grid(row=2, column=1)

        # Frame capture interval value label (displays the set value)
        self.frame_capture_interval_value_label = ttk.Label(Frame_capture_interval_frame, text="None")
        self.frame_capture_interval_value_label.grid(row=2, column=2)


            #Reference Image Delay
        reference_image_delay_frame = ttk.Frame(inspection_setup_frame)
        reference_image_delay_frame.grid(row=5, column=0, pady=5)

        # Reference Image Delay label
        reference_image_delay_label = ttk.Label(reference_image_delay_frame, text="Reference Image Delay (seconds)", pad
ding = 10)
        reference_image_delay_label.grid(row=0, column=0, columnspan=3, sticky="EW")
```

```
        # Reference Image Delay Checkbutton
        self.reference_image_delay_option = tk.BooleanVar()

        self.reference_image_delay_check_button = ttk.Checkbutton(reference_image_delay_frame,
        text="Enable",
        variable=self.reference_image_delay_option,
        command = controller.reference_image_delay_command,
        onvalue=True, offvalue=False)
        self.reference_image_delay_check_button.grid(row=1, column=0, columnspan=3)

        # Reference Image Delay Entry
        self.reference_image_delay_seconds = tk.IntVar()
        self.reference_image_delay_entry = ttk.Entry(reference_image_delay_frame, width=6, textvariable=self.reference_i
mage_delay_seconds, state="disabled")
        self.reference_image_delay_entry.grid(row=2, column=0)

        # Reference Image Delay set button
        self.reference_image_delay_set_button = ttk.Button(reference_image_delay_frame, text="Set", state="disabled", co
mmand = controller.reference_image_delay_set)
        self.reference_image_delay_set_button.grid(row=2, column=1)

        # Reference Image delay value label (displays the set value)
        self.reference_image_delay_value_label = ttk.Label(reference_image_delay_frame, text="None")
        self.reference_image_delay_value_label.grid(row=2, column=2)


        ######################## CAMERA VARIABLES ########################
        #Camera Variables Frame
        camera_variables_frame = ttk.Frame(self)
        camera_variables_frame.grid(row=1, column=0, pady=15)
```

```
#Inside Camera Variables Frame
    #Camera Variable Label
camera_variable_label = ttk.Label(camera_variables_frame, text="Camera Variable Control", padding = 5)
camera_variable_label.grid(row=0, column=0, columnspan=3)

    #Brightness Frame, Label and Entry
brightness_frame = ttk.Frame(camera_variables_frame)
brightness_frame.grid(row=1, column=0)

brightness_label = ttk.Label(brightness_frame, text="Brightness", padding =5)
brightness_label.grid(row=0, column=0)

self.brightness_value_label = ttk.Label(brightness_frame, text="1.0", padding = 5)
self.brightness_value_label.grid(row=0, column=1)

self.brightness = tk.DoubleVar()
self.brightness_entry = ttk.Entry(brightness_frame, width=6, textvariable=self.brightness)
self.brightness_entry.grid(row=1, column=0)
self.brightness_entry.delete(0, 'end')

# Brightness set button
brightness_set_button = ttk.Button(brightness_frame, text="Set", command = controller.set_brightness)
brightness_set_button.grid(row=1, column=1)

# Brightness reset button
brightness_reset_button = ttk.Button(brightness_frame, text="Reset", command = controller.reset_brightness)
brightness_reset_button.grid(row=1, column=2, sticky="EW")


    #Contrast Label and Entry
contrast_frame = ttk.Frame(camera_variables_frame)
```

```
        contrast_frame.grid(row=2, column=0)

        contrast_label = ttk.Label(contrast_frame, text=" Contrast ", padding = 5)
        contrast_label.grid(row=0, column=0)

        self.contrast_value_label = ttk.Label(contrast_frame, text="1.0", padding = 5)
        self.contrast_value_label.grid(row=0, column=1)

        self.contrast = tk.DoubleVar()
        self.contrast_entry = ttk.Entry(contrast_frame, width=6, textvariable=self.contrast)
        self.contrast_entry.grid(row=1, column=0)
        self.contrast_entry.delete(0, 'end')

        # Contrast set button
        contrast_set_button = ttk.Button(contrast_frame, text="Set", command = controller.set_contrast)
        contrast_set_button.grid(row=1, column=1)

        # Contrast reset button
        contrast_reset_button = ttk.Button(contrast_frame, text="Reset", command = controller.reset_contrast)
        contrast_reset_button.grid(row=1, column=2, sticky="EW")
```

22.6.5 controller.py

```
#Configuration File Setup
from configparser import ConfigParser
config_file = 'config.ini'
config = ConfigParser()
config.read(config_file)

class Controller():                 # create the Controller class
    def __init__(self):             # initialise the class
        self.video_frame = None     # video_frame object to which VideoFrame class is assigned
        self.controls_frame = None  # controls_frame object to which ControlsFrame class is assigned
        self.setup_frame = None     # setup_frame object to which SetupFrame class is assigned

        self.inspection_time_limit_flag = False
        # Above: flag state determines if inspection time limit is going to be used or not
        self.reference_image_delay_flag = False
        # Above: flag state determines if reference image delay is going to be used or not
        self.frame_capture_interval_flag = False
        # Above: flag state determines if frames saving at a set interval is going to be used or not

        self.save_count = 0             # Object to store the image save count
        self.interval_save_count = 0    # Object to store count of images saved at set intervals

        # set the reference image delay, frame capture interval & inspection time limit to 0 default
        # (used as a check condition later)
        self.reference_image_delay = 0
        self.frame_capture_interval = 0
        self.inspection_time_limit = 0

        self.save_location = str()      # Object to store desired save location of images
        self.save_location_flag = False # Flag state determines if custom save location is used or not
```

```
######################### METHODS FOR SETTING FRAMES TO CONTROL ########################
# assign the frame classes in the DS_GUI class to the
# objects of the controller (allows for attributes
# of the frame classes to be accessed in the controller class)

def set_video_frame(self, frame):
    self.video_frame = frame

def set_controls_frame(self, frame):
    self.controls_frame = frame

def set_setup_frame(self, frame):
    self.setup_frame = frame


######################### METHODS INSPECTION MODE ########################
# methods to set the inspection mode
# if 4f mode is activated, the resultant image is flipped 180 degrees before
# being displayed, as is required for a 4f system

def set_normal_mode(self):
    self.setup_frame.normal_mode_button["state"] = "disabled"
    self.setup_frame.four_f_mode_button["state"] = "enabled"
    self.video_frame.mode_flag = False

def set_four_f_mode(self):
    self.setup_frame.normal_mode_button["state"] = "enabled"
    self.setup_frame.four_f_mode_button["state"] = "disabled"
    self.video_frame.mode_flag = True
```

```
    ######################### METHODS FOR VIDEO CONTROL ########################
    # methods for controlling the video display
    # allow for the display to be paused, resumed or stopped

    def pause(self):    #method for pausing video
        self.video_frame.pause_video()  #executes the pause_video() method from thee video frame class when pause() meth
od is called
        self.controls_frame.pause_button["state"] = "disabled"
        self.controls_frame.play_button["state"] = "enabled"
        self.controls_frame.stop_button["state"] = "enabled"


    def play(self):     #method to play video
        self.video_frame.play_video()
        self.controls_frame.play_button["state"] = "disabled"
        self.controls_frame.pause_button["state"] = "enabled"
        self.controls_frame.stop_button["state"] = "disabled"


    def stop(self):     #method to stop video/inspection
        self.video_frame.stop_video()
        self.controls_frame.play_button["state"] = "disabled"
        self.controls_frame.pause_button["state"] = "disabled"
        self.controls_frame.stop_button["state"] = "disabled"

        self.controls_frame.snapshot_button["state"] = "disabled"
        self.controls_frame.real_time_video_button["state"] = "disabled"
        self.controls_frame.fringe_pattern_button["state"] = "disabled"

        self.frame_capture_interval = 0     #if frames are set to be captured at set intervals, stop capturing frames
```

```
# methods to control what is displayed
def display_fringes(self):  # display fringes when fringe display button is pressed
                            # results in real-time subtraction of reference image from current image
    self.video_frame.display_fringes_flag = True
    self.controls_frame.fringe_pattern_button["state"] = "disabled"
    self.controls_frame.real_time_video_button["state"] = "enabled"


def display_video(self):    #display real time video when video button is pressed
    self.video_frame.display_fringes_flag = False
    self.controls_frame.fringe_pattern_button["state"] = "enabled"
    self.controls_frame.real_time_video_button["state"] = "disabled"


######################### METHODS FOR IMAGES & SAVING ########################

def save_snapshot(self):    #method to save images when snapshot button is hit
    if self.save_location_flag:     #if a custom save location has been provided
        try:                        #try save the image to the provided location
            snapshot_name = self.save_location+"\\snapshot"+str(self.save_count)+".jpeg"
            self.video_frame.photo.save(snapshot_name)

        except:                     #if an error occurs, save the image to the default save location
            snapshot_name = "snapshot"+str(self.save_count)+".jpeg"
            self.video_frame.photo.save(snapshot_name)

    else:                           #else, save the image to the default save location
        snapshot_name = "snapshot"+str(self.save_count)+".jpeg"
        self.video_frame.photo.save(snapshot_name)
```

```
        self.save_count = self.save_count+1     #increment the save count by 1


    def save_ref_image(self):   #method to save the reference image once it is taken
        if self.save_location_flag:
            try:
                self.video_frame.ref_img.save(self.save_location + "\\reference_image.jpeg")
            except:
                self.video_frame.ref_img.save("reference_image.jpeg")

        else:
            self.video_frame.ref_img.save("reference_image.jpeg")


    def frame_capture_save(self):    #method to save images at set intervals
        frame_capture_interval_milliseconds = int(self.frame_capture_interval*1000)

        if self.frame_capture_interval_flag: # if frame capture interval Checkbutton is checked
            if self.frame_capture_interval > 0: # checking that the interval (seconds) set is valid
                if self.save_location_flag: # save frames to desired save location
                    try:
                        frame_capture_name = self.save_location + "\\interval_capture"+str(self.interval_save_count)+".j
peg"
                        self.video_frame.photo.save(frame_capture_name)

                    except:
                        frame_capture_name = "interval_capture"+str(self.interval_save_count)+".jpeg"
                        self.video_frame.photo.save(frame_capture_name)

                else:   # save frames to default save location
                    frame_capture_name = "interval_capture"+str(self.interval_save_count)+".jpeg"
```

```
                    self.video_frame.photo.save(frame_capture_name)

                self.interval_save_count = self.interval_save_count+1
                self.setup_frame.start_inspection_button.after(frame_capture_interval_milliseconds, lambda: self.frame_c
apture_save())


    def get_save_location(self): # method to retrieve desired save location from entry
        self.save_location = self.controls_frame.folder_directory.get()
        self.save_location_flag = True  # set the save_location_flag to true


    ######################### METHOD FOR RESETTING INSPECTION ########################
    # resets button states and set inspection parameters so that new inspection can be configured
    def reset(self):
        self.setup_frame.start_inspection_button["state"] = "enabled"

        self.controls_frame.play_button["state"] = "disabled"
        self.controls_frame.pause_button["state"] = "disabled"
        self.controls_frame.stop_button["state"] = "disabled"

        self.controls_frame.snapshot_button["state"] = "enabled"
        self.controls_frame.new_inspection_button["state"] = "disabled"

        self.controls_frame.folder_set_button["state"] = "enabled"
        self.controls_frame.config_button["state"] = "enabled"

        self.video_frame.ref_image = None   # reset reference image variable to None
        self.video_frame.display_fringes_flag = False           #change display mode back to real time video display
        self.controls_frame.real_time_video_button["state"] = "disabled"
        self.controls_frame.fringe_pattern_button["state"] = "disabled"
```

```
    self.video_frame.mode_flag = False                      #change inspection made back to normal inspection
    self.setup_frame.four_f_mode_button["state"] = "enabled"
    self.setup_frame.normal_mode_button["state"] = "disabled"

    self.setup_frame.inspection_time_limit_check_button["state"] = "enabled"
    self.setup_frame.frame_capture_interval_check_button["state"] = "enabled"
    self.setup_frame.reference_image_delay_check_button["state"] = "enabled"

    self.inspection_time_limit = 0                          #reset inspection time limit
    self.setup_frame.inspection_time_limit_value_label["text"] = "None"

    self.frame_capture_interval = 0                         #reset frame capture interval
    self.setup_frame.frame_capture_interval_value_label["text"] = "None"

    self.reference_image_delay = 0                          #reset reference image delay
    self.setup_frame.reference_image_delay_value_label["text"] = "None"

    if self.video_frame.video_pause:    #if the display has been paused and stopped, display real time video again
        self.video_frame.play_video()


######################### METHODS FOR START INSPECTION ########################
# starts the inspection procedure
# starts the time count for relevant inspection parameters that have been set
# disables widgets that should not be used during the inspection
def start(self):
    self.controls_frame.pause_button["state"] = "enabled"   #enable pause button

    # disabled buttons, Checkbuttons and entries of inspection parameters
    self.setup_frame.inspection_time_limit_check_button["state"] = "disabled"
```

```
        self.setup_frame.inspection_time_limit_set_button["state"] = "disabled"
        self.setup_frame.inspection_time_limit_entry["state"] = "disabled"

        self.setup_frame.frame_capture_interval_check_button["state"] = "disabled"
        self.setup_frame.frame_capture_interval_check_button["state"] = "disabled"
        self.setup_frame.frame_capture_interval_entry["state"] = "disabled"

        self.setup_frame.reference_image_delay_check_button["state"] = "disabled"
        self.setup_frame.reference_image_delay_set_button["state"] = "disabled"
        self.setup_frame.reference_image_delay_entry["state"] = "disabled"

        self.setup_frame.start_inspection_button["state"] = "disabled"

        self.controls_frame.new_inspection_button["state"] = "enabled"

        self.controls_frame.folder_entry["state"] = "disabled"
        self.controls_frame.folder_set_button["state"] = "disabled"

        self.controls_frame.config_button["state"] = "disabled"

        self.setup_frame.normal_mode_button["state"] = "disabled"
        self.setup_frame.four_f_mode_button["state"] = "disabled"

        # call inspection
        self.reference_image()      #call take_reference_image method
        self.frame_capture_save()   #call frame_capture_save
        self.inspection_limit()     #call inspection_limit method


    def reference_image(self):
        # determines when reference image will be taken
```

```
        # then takes reference image using take_reference_image() method
        if not self.reference_image_delay_flag:
            # if reference image delay flag is False, a reference image will be taken immediately after start
            self.take_reference_image()

        elif self.reference_image_delay_flag:
            #if reference image delay flag is true, reference image will only be taken after the set seconds
            if self.reference_image_delay > 0:
                reference_image_delay_milliseconds = int(self.reference_image_delay*1000)
                self.setup_frame.start_inspection_button.after(reference_image_delay_milliseconds, lambda: self.take_ref
erence_image())
                # Above: after set seconds take reference image

            else:
                self.take_reference_image()
                # Above: if a delay is not properly inputted, it will default to no delay


    def take_reference_image(self):
        # takes reference image by calling capture_reference_image
        # from video_frame and enables fringe pattern button
        self.video_frame.capture_reference_image()
        self.controls_frame.fringe_pattern_button["state"] = "enabled"
        self.save_ref_image()   #once reference image is taken, save the image


    def inspection_limit(self):
        if self.inspection_time_limit_flag: # if inspection time limit checkbutton is checked
            if self.inspection_time_limit > 0:  #checking that the time limit set is valid
                inspection_time_limit_milliseconds = int(self.inspection_time_limit*1000*60)
```

```
                self.setup_frame.start_inspection_button.after(inspection_time_limit_milliseconds, lambda: self.inspecti
on_limit_set())
                # Above: after set minutes stop inspection by called inspection_time_limit_set() method


    def inspection_limit_set(self):
        self.pause()
        self.video_frame.video_display_label.after(500, lambda: self.stop())
        # Above: after 500 ms stop inspection and disconnect camera




    ######################### METHODS FOR INSPECTION SETUP ########################

    def inspection_time_limit_command(self):
        # Sets flags to true or false and enables/disables
        # buttons & entry based on checkbuttons status
        if self.setup_frame.inspection_time_limit_option.get():
            self.setup_frame.inspection_time_limit_entry["state"] = "enabled"
            self.setup_frame.inspection_time_limit_set_button["state"] = "enabled"

            self.inspection_time_limit_flag = True  #inspection time limit flag (used in start() method)

        elif self.setup_frame.inspection_time_limit_option.get()==False:
            self.setup_frame.inspection_time_limit_entry["state"] = "disabled"
            self.setup_frame.inspection_time_limit_set_button["state"] = "disabled"

            self.inspection_time_limit_flag = False
```

```
        self.setup_frame.inspection_time_limit_value_label["text"] = "None"     #sets the text of the label to none

        self.inspection_time_limit = 0      #set inspection time limit to 0 if disabled

    def frame_capture_interval_command(self):
        # Sets flags to true or false and enables/disables
        # buttons & entry based on checkbuttons status
        if self.setup_frame.frame_capture_interval_option.get():
            self.setup_frame.frame_capture_interval_entry["state"] = "enabled"
            self.setup_frame.frame_capture_interval_set_button["state"] = "enabled"

            self.frame_capture_interval_flag = True
            # Above: inspection time limit flag (used in start method later)

        elif self.setup_frame.frame_capture_interval_option.get()==False:
            self.setup_frame.frame_capture_interval_entry["state"] = "disabled"
            self.setup_frame.frame_capture_interval_set_button["state"] = "disabled"

            self.frame_capture_interval_flag = False

            self.setup_frame.frame_capture_interval_value_label["text"] = "None"
            self.frame_capture_interval = 0     #set frame capture interval to 0 if disabled


    def reference_image_delay_command(self):
        # Sets flags to true or false and enables/disables
        # buttons & entry based on checkbuttons status
        if self.setup_frame.reference_image_delay_option.get():
            self.setup_frame.reference_image_delay_entry["state"] = "enabled"
            self.setup_frame.reference_image_delay_set_button["state"] = "enabled"
```

```
            self.reference_image_delay_flag = True
            # Above: inspection time limit flag (used in start method later)

        elif self.setup_frame.reference_image_delay_option.get()==False:
            self.setup_frame.reference_image_delay_entry["state"] = "disabled"
            self.setup_frame.reference_image_delay_set_button["state"] = "disabled"

            self.reference_image_delay_flag = False

            self.setup_frame.reference_image_delay_value_label["text"] = "None"
            self.reference_image_delay = 0     #set reference image delay to 0 if disabled


    def inspection_time_limit_set(self):
        # sets the inspection time limit time when set button is pressed
        try:    # try set inspection time limit to value in entry
            self.inspection_time_limit = self.setup_frame.inspection_time_limit_mins.get()
            self.setup_frame.inspection_time_limit_value_label["text"] = str(self.inspection_time_limit)

        except: # if an error occurs, disabled inspection time limit
            self.inspection_time_limit = 0
            self.setup_frame.inspection_time_limit_value_label["text"] = "None"


    def frame_capture_interval_set(self):
        # sets the frame capture interval time when set button is pressed
        try:    # try set frame capture interval to value in entry
            self.frame_capture_interval = self.setup_frame.frame_capture_interval_seconds.get()
            self.setup_frame.frame_capture_interval_value_label["text"] = str(self.frame_capture_interval)

        except: # if an error occurs, disabled frame capture interval
```

```
            self.frame_capture_interval = 0
            self.setup_frame.frame_capture_interval_value_label["text"] = "None"

    def reference_image_delay_set(self):
        # sets the reference image delay time when set button is pressed
        try:    # try set reference image delay to value in entry
            self.reference_image_delay = self.setup_frame.reference_image_delay_seconds.get()
            self.setup_frame.reference_image_delay_value_label["text"] = str(self.reference_image_delay)

        except: # if an error occurs, disabled reference image delay
            self.reference_image_delay = 0   #if this remains 0, there will be no reference image delay
            self.setup_frame.reference_image_delay_value_label["text"] = "None"

    ######################### METHODS FOR CAMERA VARIABLES ########################

    def set_brightness(self):   # method to set image brightness
        try:    # try set brightness value to value in entry
            self.video_frame.brightness_value = self.setup_frame.brightness.get()
            self.setup_frame.brightness_value_label["text"] = str(self.video_frame.brightness_value)

        except: # if an error occurs, reset the image brightness
            self.video_frame.brightness_value = 1.0
            self.setup_frame.brightness_value_label["text"] = str(self.video_frame.brightness_value)


    def reset_brightness(self): # method to reset the image brightness
        self.video_frame.brightness_value = 1.0
        self.setup_frame.brightness_entry.delete(0, "end")
        self.setup_frame.brightness_value_label["text"] = str(self.video_frame.brightness_value)
```

```
def set_contrast(self): # method to set image contrast
    try:    # try set contrast value to value in entry
        self.video_frame.contrast_value = self.setup_frame.contrast.get()
        self.setup_frame.contrast_value_label["text"] = str(self.video_frame.contrast_value)

    except: # if an error occurs, reset the image contrast
        self.video_frame.contrast_value = 1.0
        self.setup_frame.contrast_value_label["text"] = str(self.video_frame.contrast_value)


def reset_contrast(self):   # method to reset the image contrast
    self.video_frame.contrast_value = 1.0
    self.setup_frame.contrast_entry.delete(0, "end")
    self.setup_frame.contrast_value_label["text"] = str(self.video_frame.contrast_value)


######################### METHOD FOR CONFIG FILE ##########################

def set_config(self):
    # method to configure inspection parameters and camera variables
    # to values given in the file config.ini, when the Import Config File
    # button is pressed

    #configure contrast
    try:    # try set contrast value to value in config file
        self.video_frame.contrast_value = float(config['cameraVariables']['CONTRAST'])

    except: # if an error occurs, set image contrast to default
        self.video_frame.contrast_value = 1.0
```

```
#configure brightness
try:    # try set brightness value to value in config file
    self.video_frame.brightness_value = float(config['cameraVariables']['BRIGHTNESS'])

except: # if an error occurs, set image brightness to default
    self.video_frame.brightness_value = 1.0


#configure inspection time limit
try:    # try set inspection time limit value to value in config file
    self.inspection_time_limit = float(config['inspectionSetup']['INSPECTION_TIME_LIMIT'])
    self.setup_frame.inspection_time_limit_value_label["text"] = str(self.inspection_time_limit)
    self.inspection_time_limit_flag = True

except: # if an error occurs, disable inspection time limit
    self.inspection_time_limit = 0
    self.setup_frame.inspection_time_limit_value_label["text"] = "None"


#configure frame capture interval
try:    # try set frame capture interval value to value in config file
    self.frame_capture_interval = float(config['inspectionSetup']['FRAME_CAPTURE_INTERVAL'])
    self.setup_frame.frame_capture_interval_value_label["text"] = str(self.frame_capture_interval)
    self.frame_capture_interval_flag = True

except: # if an error occurs, disable frame capture interval
    self.frame_capture_interval = 0
    self.setup_frame.frame_capture_interval_value_label["text"] = "None"


#configure reference image delay
```

```
        try:    # try reference image delay value to value in config file
            self.reference_image_delay = float(config['inspectionSetup']['REFERENCE_IMAGE_DELAY'])
            self.setup_frame.reference_image_delay_value_label["text"] = str(self.reference_image_delay)
            self.reference_image_delay_flag = True

        except: # if an error occurs, disable reference image delay
            self.reference_image_delay = 0
            self.setup_frame.reference_image_delay_value_label["text"] = "None"


        self.controls_frame.config_button['state'] = 'disabled'
        # Above: disable the config_button after importing config file
```

22.6.6 __init__.py

```
# Init file to facilitate importing of frame classes
from frames.controls_frame import ControlsFrame
from frames.setup_frame import SetupFrame
from frames.video_frame import VideoFrame
```

22.6.7 config.ini

```
[inspectionSetup]
INSPECTION_TIME_LIMIT= 0.0
FRAME_CAPTURE_INTERVAL= 0.0
REFERENCE_IMAGE_DELAY= 0.0

[cameraVariables]
CONTRAST= 1.0
BRIGHTNESS = 1.0
```

22.7 Appendix G - GUI README

The Digital Shearography GUI Application is an application to be used for Digital Shearography inspection devices utilizing the FLIR Integrated Imaging Solutions Chameleon3 CCD Camera. This application provides functions and operations needed to manage the Shearographic inspection process and display the fringe patterns that result from real-time subtraction of images using Digital Shearography.

Prior to using the application on a new PC, the PyCapture2 python extension, that is required for the operation of the Chameleon3 camera, needs to be installed. For instructions on how to extract the PyCapture2 package on Windows, please refer to the README_PyCapture2.txt file, which is provided by FLIR Integrated Imaging Solutions.

This Python Application also requires the installation of the OpenCV, Pillow and configparser libraries. This is done by:

1. Click on the Windows start button in the bottom left of the screen and search for 'command prompt'. Click on the Windows command prompt and wait for a black window to open.

2. Type "pip install configparser" and click the enter button to install the configparser library.

3. Type "pip install Pillow" and click the enter button to install the Pillow library.

4. Type "pip install opencv-python" and click the enter button to install the OpenCV library.

Once the Required Python Libraries are installed, the application may be used. Prior to opening the application, the Chameleon3 USB 3.0 camera must be connected to the PC. The application is opened by running the app.py file. Upon opening, the application the display window displays the camera view. Instructions for use of the application for a Non-destructive testing process are given below.

1. Prior to beginning the inspection process, the inspection setup on the right side of the application needs to be configured. Depending on the inspection device being used, the inspection mode should be set to normal or 4f mode.

2. Next, the desired inspection parameters (inspection time limit, frame capture interval, reference image delay) should be set. These can be set manually under the inspection setup or pre-set in the config.ini file and then imported using the Import Config File. The latter is recommended when running many NDT inspections with the same required inspection parameters and camera variable values.

3. The desired camera variables should be set (under Camera Variable Control). Setting a value greater than 1.0 increases the image enhancement (brightness, contrast), while applying a value between 0.0 and 1.0 decreases the image enhancement. The original image can be reverted to by using the Reset buttons.

4. If it is desired that the images to be saved are saved to a custom folder or location, the directory of this location, e.g. C:\Users\pc\Desktop\Final_Year_Project_App\test_inspection , should be pasted into the Image Save Location entry at the top of the application, after which the Set Folder

button should be pressed. If no save location is set, the images will save to the location of the applications install.

5. The inspection can begin by hitting the START INSPECTION button.

6. Once the reference image has been taken (immediately after pressing the START INSPECTION button or at the time set in the inspection setup), the camera display can be changed to display the real-time subtraction of the reference image from the current camera image by pressing the Fringe Pattern Mode button. One can revert back to displaying the real-time camera view by pressing the Video Mode Button.

7. The display can be paused or stopped using the Video Controls in the top panel.

8. Images that are displayed in the display window can be randomly saved by pressing the Snapshot Frame button in the top left of the application window.

9. Once the inspection is completed, a new inspection may be configured by pressing the New Inspection button in the top left of the application window.

For more information go to: https://github.com/migsdigs/Final_Year_Project

22.8 Appendix H – Calculation of different inspection system's angular field of view

22.8.1 Existing System

The Michelson Interferometer and the camera of the existing inspection system have mismatching angular field of views (AFOV).

The AFOV of the Camera is given by:

$$AFOV\ [°] = 2\tan^{-1}\frac{h}{2f}$$

Where h is the width or height of the camera sensor and f is the focal length of the imaging lens.

The width of the camera sensor is 4.8 mm and the height of the camera sensor is 3.6 mm. The imaging lens of the camera has a focal length of 8 mm.

Therefore, the horizontal AFOV of the camera is: $2\tan^{-1}\frac{4.8}{2(8)} = 33.4°$

And the vertical AFOV of the camera is: $2\tan^{-1}\frac{3.6}{2(8)} = 25.4°$

The AFOV of the Michelson Interferometer is given by:

$$AFOV = 2\tan^{-1}\frac{a}{2(2a + d_1 + d_2 + d_3)}$$

The side length of the cube beam splitter, a is 25 mm; d_1, the distance between the cube and the reference mirror is 1 mm; d_2, the distance between the cube and the shearing mirror is 6 mm; and the distance between the cube and the imaging lens is approximately 10 mm.

Therefore the (horizontal and vertical) AFOV of the Interferometer is:

$$2\tan^{-1}\frac{25}{2(2(25) + 1 + 6 + 10)} = 21.1°$$

22.8.2 Concept A

Concept A attempts to increase the AFOV of the Michelson Interferometer to $25°$ from $21.4°$.

Concept A plans to decrease the distance between the cube and the shearing mirror to 2 mm and the distance between the cube and the imaging lens to 3 mm.

The new AFOV of the Interferometer would be approximately:

$$2\tan^{-1}\frac{25}{2(2(25) + 1 + 2 + 3)} = 25.2°$$

22.8.3 Concept B

This concept overcomes the AFOV limitation of the Michelson Interferometer by implementing a 4f system. In this concept, the AFOV is given by that of the camera/imaging lens combination. Hence the AFOV is represented by:

$$AFOV\ [°] = 2\tan^{-1}\frac{h}{2f}$$

If the same 8 mm imaging lens that was used in the existing prototype is used in a prototype of concept B, the inspection system would have an approximate horizontal AFOV of $33.4°$ and vertical AFOV of $25.4°$.

22.8.4 Final Design

The final design is based on the same theory as concept B. This means that the angular field of view of the final design should theoretically be represented by the equation:

$$AFOV\ [°] = 2\tan^{-1}\frac{h}{2f}$$

Using the 8mm focal length imaging lens and the Chameleon3 Camera, the final design should have a horizontal AFOV of $33.4°$ and vertical AFOV of $25.4°$.

22.9 Appendix I – Chameleon3 Camera Specifications

The CCD camera that has been provided is the Chameleon3 CM3-U3-13S2M-CS. The relevant specifications of the camera are listed in the table below.

Table 17 showing the relevant specifications of the Chameleon3 USB3 camera [56]

Specification	
Frame rate	30 FPS
Lens Mount	C5-mount
Pixel size	3.75 μm
Resolution	1288 x 964
Sensor type	CCD
Sensor Format	1/3 inches
ADC size	12-bit
Sensor name	Sony ICX445
Flash memory	1 MB non-volatile memory
Image buffer	16 MB
Available trigger modes	Standard, bulb, low smear, overlapped, multi-shot
On board image processing	Colour interpolation, gamma, saturation, sharpness, hue
Mass	54.9 grams
Interface	USB 3.1 and GPIO
Dimensions (Width, Height, Length)	44 mm, 35 mm, 19.5 mm
Power requirements	5-24 V via GPIO Or 5 V via USB3
Operating system compatibility	Windows or Linux

22.10 Appendix J - UI-1241LE-NIR-GL (uEye) Camera Specifications

The second camera that has been provided is the IDS UI-1241LE-NIR-GL camera. The relevant specifications of the camera are listed in the table below.

Table 18 showing the relevant specifications of the IDS UI-1241LE-NIR-GL camera [57]

Specification	
Frame rate	25.8 FPS
Lens Mount	S-mount
Pixel size	5.30 μm
Resolution	1280 x 1024
Sensor type	CMOS
Sensor Format	1/1.8"
ADC size	10-bit
Sensor name	EV76C661ABT
Flash memory	N/A
Image buffer	N/A
Available trigger modes	N/A
On board image processing	N/A
Mass	16 grams
Interface	USB 2.0 and GPIO
Dimensions (Width, Height, Length)	36.0 mm, 36.0 mm, 20.2 mm
Power requirements	3.3 V via Mini-B USB cable
Operating system compatibility	Windows or Linux

IDS also provide a downloadable IDS Software Suite that includes an API programming interface that allows users to create their own uEye programs in Windows or Linux. This API includes commands which provide access to and control of parameters and functions of the uEye camera. Applications for the uEye camera may be developed in Python, C/C++, C# or Delphi [58].