IMAGE PROCESSING AND IMAGE ANALYSIS OF DEFECTS IN MICROLENS ARRAY USING SMARTPHONE CAMERA AND APPLICATION

TEOH YUANN YUN

A project report submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering (Hons) Electronic Engineering

Faculty of Engineering and Green Technology
Universiti Tunku Abdul Rahman

May 2022

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : 

Name : TEOH YUANN YUN

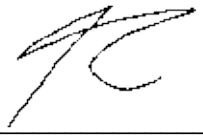
ID No. : 17AGB03738

Date : 11/9/2022

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"IMAGE PROCESSING AND IMAGE ANALYSIS OF DEFECTS IN MICROLENS ARRAY USING SMARTPHONE CAMERA AND APPLICATION"** was prepared by **TEOH YUANN YUN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons) Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : 

Supervisor : Ts. Tan Yee Chyan

Date : 21/9/2022

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2022, Teoh Yuann Yun. All right reserved.

Specially dedicated to
my beloved research supervisor, father and mother.

ACKNOWLEDGEMENTS

My appreciation for my supervisor's patience and comments is beyond words. To my supervisor, Ts. Tan Yee Chyan, I would like to extend my sincere gratitude for his patient guidance and constructive criticism of this research project.

Additionally, I also want to thank my parents for their assistance and words of encouragement. Their support, understanding, and unceasing encouragement are what enable me to complete my task successfully. Their confidence in me has sustained my enthusiasm and upbeat attitude throughout this process.

IMAGE PROCESSING AND IMAGE ANALYSIS OF DEFECTS IN MICROLENS ARRAY USING SMARTPHONE CAMERA AND APPLICATION

ABSTRACT

Microelectronics is crucial to the advancement of technology. However, industrial inspection of micro-devices is a very difficult and time-consuming operation, particularly when those devices are manufactured in large quantities utilising micro-fabrication techniques. According to Marie Freebody, the contributing editor of Photonics Media, many optical surfaces are still examined with human visual inspection system which is insufficient for getting a more precise defects inspection when the objects are small in size. Thus, automated optics inspection (AOI) machine is invented to calibrate on optics surfaces and inspect for scratches. However, according to the quotation summarized by VCTA, a manufacturer in AOI, an AOI system is expensive with a cost of around 30,000RMB to 50,000RMB in China. In this project, a smartphone camera model that imitates a light field camera (LFC) equipped with a micro-lens array (MLA), a laser and a diffuser is developed. The phone camera is calibrated with camera calibration system. Then, the MLA is positioned in front of the laser, turning it into an array of tiny cameras that emits light field. The images produced are then processed on an application on a PC which implemented image processing method to classify the defectiveness of MLA with two classes: PASS or FAIL. Additionally, the images produced are also processed on Siamese Neural Network to find image similarity and classify the defectiveness with two classes: 0 or 1. With the proposed method, the intensity distributions, contour plot, 3D surface plot, training total loss on test MLA datasets are determined, discussed and studied.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Light sensing is one of the most effective modes of perception. Throughout these years, refractive micro-optics have been shown to be a significant component of numerous high-tech systems with a variety of applications such as laser beam shaping, and lighting systems (Guenther, R. D., 2005). All of these applications have one thing in common where they all require high-quality micro-optics. An optical device having a micron-scale structure and different optical characteristics is known as a micro-lens array (MLA) (Li, Li and Gong, 2021). Because of its superior geometric properties and optical performance, MLA has become a key part in the optical system as the high-tech sector has progressed (Paschotta, 2006).

According to Gershun in 1936, light field (LF) imaging technology is the entirety of light beams in three-dimensional (3D) space in any position and direction, unlike the traditional imaging approach which only records information in two-dimensional (2D) position. With the processed light-field data, multi-view imaging, digital refocusing, depth estimation, holographic reconstruction, and other imaging applications can be realised (Georgeiv and Intwala, 2006). The most popular light-field imaging system uses a light-field camera, which is also referred to as a plenoptic camera based on MLAs. From the fundamental of plenoptic camera imaging shown in **Figure 1.1**, MLA is positioned at the focal plane of the plenoptic camera, located in

between the main lens and a CCD sensor. The charged-coupled device (CCD) is a sensor built on a collection of passive photodiodes that integrate charge over the duration of the camera's exposure time. Light and colour spectrum are transformed into electrical signals and interpreted by camera based on zeros and ones (Director, 2014). Then, each micro-lens unit captures the scene image from various view angles at a specific location (Perwass and Wietzke, 2012).

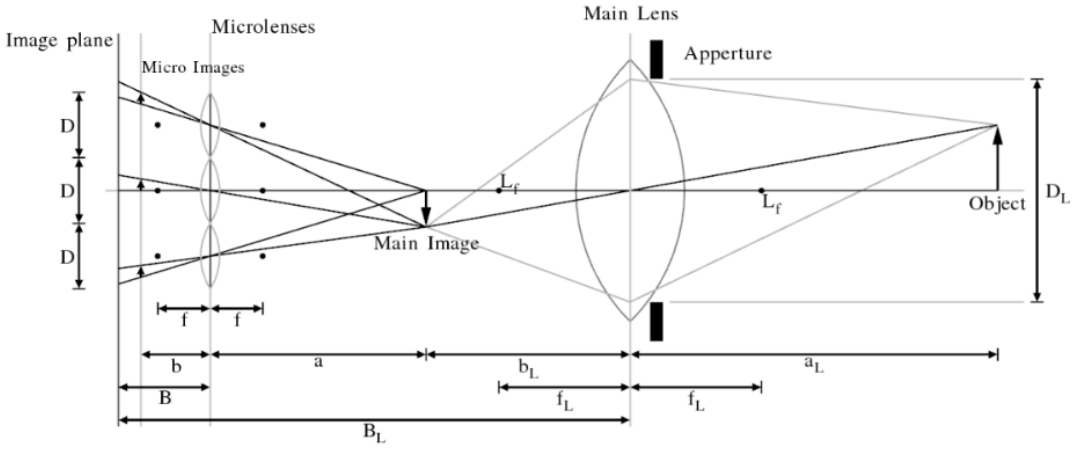


Figure 1.1: Fundamental of Plenoptic Camera Imaging (Perwass and Wietzke, 2012).

Light field cameras have evolved from the first light field camera proposed by Gabriel Lippmann in 1908 (www.tgeorgiev.net, n.d.) to the latest nanophotonic light field camera developed recently (O'Neil, 2022). Light-field cameras have been utilised extensively thanks to the development and improvement of light-field imaging. According to the research done by the Department of Electrical and Electronic Engineering of Yonsei University, light field camera is used to defend against spoofing face attacks (Kim, Ban and Lee, 2014). Furthermore, according to studies on The Plant Journal, light-field camera is used to produce a focused image and a depth image, which comprises growth details about plants (Apelt et al., 2015). Moreover, University of Maryland College Park also utilise a light field camera to eliminate images of objects that become distorted and unrecognizably blurred in the presence of strong turbulence (Wu, Ko and Davis, 2016).

In order to ensure large field data is recorded accurately, the accuracy and imaging condition of MLAs are critical in meeting the requirements of various applications in various fields such as optics, medical and astronomical fields (Abookasis and Rosen, 2004). Based on **Figure 1.2**, the geometric size of each unit of MLA is defined by the pitch (*D*), height (*h*), radius of curvature (R_u), and contact angle (θ). In order to measure these properties, equipments such as optical microscopy, electron microscopy scan, and contact profilometry can all be used.

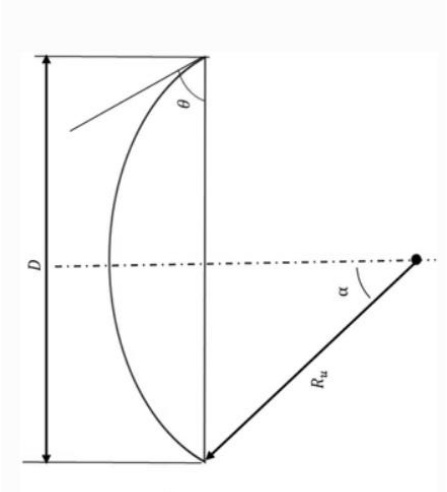


Figure 1.2: Geometry Parameters of Micro-lens Array (Yuan et al., 2018).

Then, the numerical aperture (*NA*), surface roughness, and array homogeneity can be calculated to indicate the micro-lens quality (Park et al., 2014)

$$\boldsymbol{NA = n \cdot \sin \propto} \tag{1.1}$$

where *'n'* is the refractive index between object and micro-lens, and *'∝'* is the half aperture angle which can be obtained from height (*h*) and radius of curvature (R_u) (Park et al., 2014)

$$\propto = \boldsymbol{arc \cos}\left(\frac{\boldsymbol{R_u - h}}{\boldsymbol{R_u}}\right) \tag{1.2}$$

Hence, when NA increased, the resolution and magnification of micro-lens also increased. In addition, when analysing the optical performance of the lens, surface roughness is also essential. Scattering difficulties can occur in optics with a lot of surface roughness, lowering contrast and light collection efficiency. The imaging system's array homogeneity is critical, especially when the MLA's size is vast and light retrace is required for further processing (Stevens and Miyashita, 2010).

Since, the performance of a MLA is determined by capability of the manufacturing process, measurement is important before building the MLA. However, measurement capability for MLA is a critical constraint whereby it is too small for inspections done on human eye. Hence, optical inspection is vital in optics manufacturing because it checks and maintains the quality of optics, such as inspecting surface scratches and digs according to different standards. There are two methods for an optical inspection such as manual inspection by a human or auto inspection by a machine. Manual inspection is done with naked eyes or with the help of tools such as single lens magnifiers, optical comparators, and optical microscopes. However, these assistances, necessitates the changing of the optics around, raises the risk of damage to the optics and also makes it impossible to discover defects statistically (www.electronics-notes.com, n.d.)

Thus, with the advancement of hardware capabilities as well as the capability of computer vision, various researches have been done to make MLAs defect detection as economical and easy for industrial to use as much as possible. Researches into image processing, with the help of computer vision has become more popular with as they aid in the enhancement of images for human understanding, and assisting in analysing image to retrieve information for machine interpretation, which in turn reduce the time and cost needed.

1.2 Computer Vision

Computer vision is the modelling of image processing using machine learning techniques to analyse images and to recognise patterns (Sheng, 2020). Thus, items can be discriminated, classified and ordered according to their size, pattern and so on. Image segmentation, object detection, edge detection, pattern detection, image classification, and feature matching are a few examples of computer vision techniques (Ltd, n.d.).

Computer vision system consists of camera, lens, image processing software, and machine learning accepts images as input and outputs data such as size or colour intensity. As a result, a large amount of data is required during the repeating data analysis until it can identify differences and eventually recognise images (Society, 2019).

1.2.1 Image Processing

Image processing is a subset of computer vision because a computer vision system employs image processing techniques to simulate human vision at a large scale (GeeksforGeeks, 2019). Various mathematical functions are performed in image alterations such as smoothing, sharpening, contrasting, and stretching.

In the detection of a defect through image processing, feature extraction, edge detection, morphological operators, and data training are some of the essential phases (Information Resources Management Association, 2013). In order to replace manual inspection, image processing techniques is used to detect defects to reduce time and cost. There are two approaches in image processing, such as analog image processing and digital image processing. For analog image processing, the technique for processing pictures, prints, and other tangible reproductions of images with images as input are performed. On the other hand, digital image processing entails using complex algorithms to manipulate images or information linked with an image, such as features

or bounding boxes digital image to generate information (Theteche.com, 2021). All in that, defects can be detected faster and more efficiently with the help of image processing.

1.3 Deep Learning

Deep learning has gained a lot of highlights recently, due to the successful accomplishments that were previously unattainable. It is commonly used for computer vision which is a subdivision of machine learning technique that allows computers to learn like humans do. It learns to execute categorization tasks directly from images, text, or sound and receives feedback on its performance to adjust its accuracy (Liu et al., 2021). Then, the system can independently determine whether their output is correct or not.

In addition, models are trained using a large set of labelled data and multilayer neural network architectures such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), artificial neural networks (ANNs).

1.3.1 Convolutional Neural Network (CNN)

Convolutional neural networks (CNNs) models are used in modern computer vision techniques and are the closest technical analogy to the brain (Alzubaidi et al., 2021). The layers in CNNs are divided into three categories which are the width, height, and depth as this structure aids in the recognition of various items. CNNs are commonly employed for computer vision applications and one of the earliest CNN designs which won the ImageNet visual recognition contest in 2012 is AlexNet (Tsang, 2020).

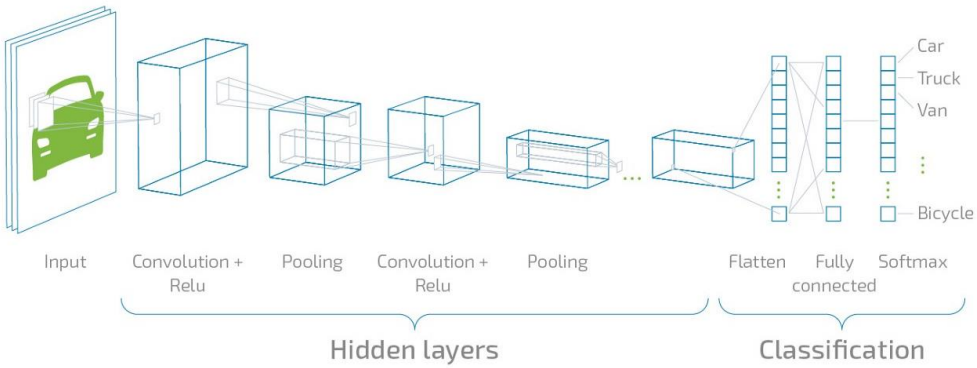


Figure 1.3: The Visualisation of Neural Network (Infopulse, 2019).

According to **Figure 1.3**, the neural network has two distinctive components which are the hidden layers and classification. For the hidden layers, the network executes a number of actions at this point in order to extract and detect certain visual features. In classification, the probability of the image being predicted by the algorithm will be determined (Infopulse, 2019).

During the image processing by a CNN, each base colour such as red, green and blue is represented as a matrix of values. When applied to colour images, these values are assessed and compressed into 3D tensors. Convolutional and pooling layers are used to process the image, condensing the most important data from each image segment into a more manageable matrix. Depending on the number of convolutional layers in the design, this step is repeated several times. The final features are then extracted using a convolutional technique and supplied to a fully connected layer (Analytics Vidhya, 2022).

For the convolution technique, kernels are used to extract the important information from the input. The construction of a RGB image is shown in **Figure 1.4**. RGB image is constructed using three distinct image planes, representing each of the main colours such as red, green, and blue. Hence, the RGB image shown is represented with 6 x 6 x 3 where the ‘3’ is the colour channels (datahacker.rs, 2018).

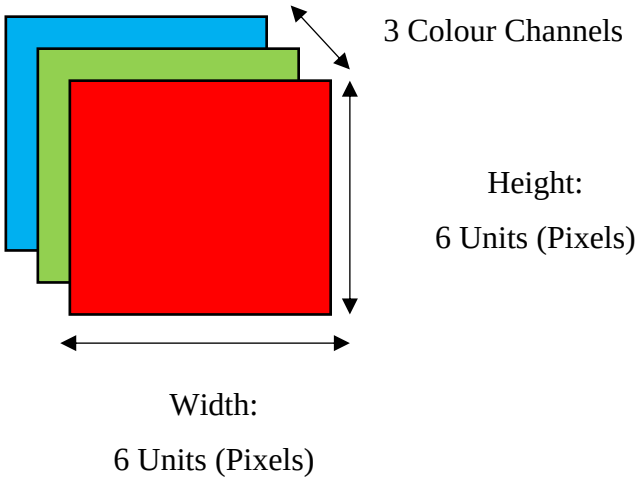


Figure 1.4: Construction of a RGB Image
(datahacker.rs, 2018).

In order to undergo image processing such as detecting edges or some other features, convolution has to be processed as shown in **Figure 1.5.** To begin, a 3 x 3 x 3 layers filter corresponding to red, green and blue channels are used.

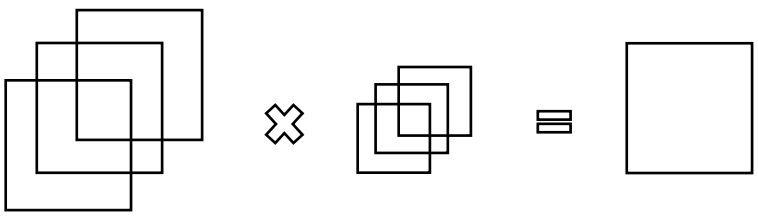


Figure 1.5: Result of a convolution applied on a RGB image
(datahacker.rs, 2018).

The ‘6’ represents the height of the image, the second ‘6’ is the width of image, and the ‘3’ is the number of channels. Similarly, the filer is also represented with height, width and number of channels. With the same number of channels for image and filter, the output can be calculated. Since the filter contains 27 numbers because of 3 x 3 x 3, these 27 numbers are taken to multiply with the corresponding numbers from red, green and blue channel. Then, by adding up those numbers, the first number of outputs is achieved (datahacker.rs, 2018).

For pooling, it is in charge of shrinking the spatial size of the convolved feature. By lowering the dimensions, it lowers the amount of CPU power needed to process the data. There are two types of pooling as shown in **Figure 1.6** such as max pooling and average pooling. Max Pooling is the process of determining the maximum value of a pixel from a section of the image that the kernel has covered. On the other hand, the average of all the data from the area of the image that the Kernel covered is returned by average pooling so that noise can be reduced by reducing the dimensionality (Mandal, 2021).

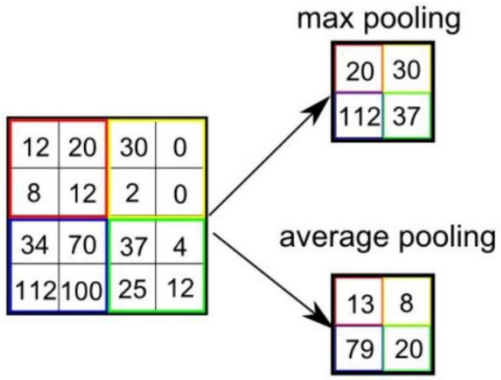


Figure 1.6: Types of Pooling

(Mandal, 2021)

In a nutshell, CNNs are widely adopted by the deep learning community because they reduce the need for manual feature extractions by convolving learnt features with input data, allowing the algorithm to extract features from images directly, which helps increase the system's classification accuracy (MathWorks, n.d.).

1.3.2 Artificial neural network (ANN)

Artificial neural networks (ANNs) are computational approximations of biological neural networks, which let the brain to perform a multitude of cognitive functions such as vision, hearing, and decision-making. To summarise, each ANN is made up of "artificial neurons", which are mathematical functions that assess input before passing it on to the next "neuron" for processing. Before transmitting the knowledge to the next

layer, each layer in the network concentrates on analysing distinct aspects (Purakkatt, 2020).

One of most basic ANNs is the feed forward neural networks with one hidden layer and three neurons. As depicted in **Figure 1.7**, information travels in one direction from left to right through the input nodes, then the hidden nodes, and ultimately the output nodes, as initially proposed by AI pioneer Frank Rosenblatt in 1958 (Cornell Chronicle, n.d.). To begin, each node in the layer represents an artificial neuron, which is represented by a function that performs the required calculations for the job at hand, such as data classification based on a given parameter. To go from the input layer to the output layer, several linear or nonlinear functions are used. Hidden layers enable the calculation of increasingly intricate functions by cascading smaller functions. In other words, a network without a hidden layer can only learn a linear decision boundary, such as categorising all the blue dots on one side and all the red dots on the other, but it can't handle more complex judgements. As a result, hidden layers contribute to the learning capabilities of the system, enabling for more complicated judgments to be made (Infopulse, n.d.)

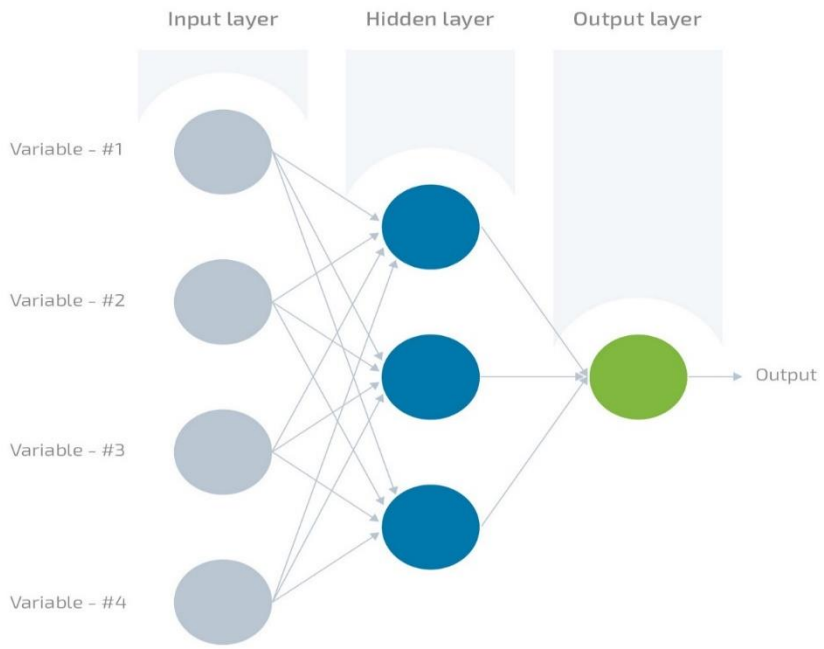


Figure 1.7: A Feed Forward Neural Network with One Hidden Layer (Infopulse, n.d.)

1.3.3 Recurrent Neural Network (RNN)

Recurrent neural networks (RNNs) are the most advanced algorithm for sequential data due to its internal memory. Because of RNNs robust internal memory, they are good on remembering input received and precise in prediction (Donges, 2019).

According to **Figure 1.8**, four types of RNNs such as One to One, One to Many, Many to One and Many to Many are shown accordingly from left to right with red circle as the input and purple colour as the output (GeeksforGeeks, 2022).

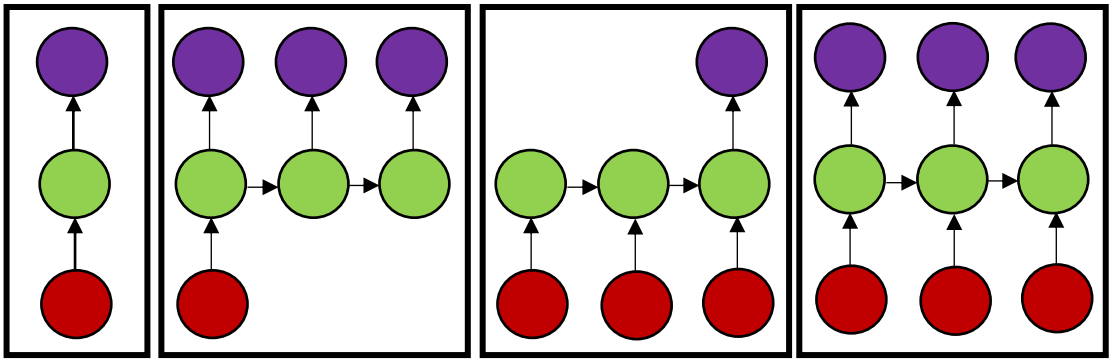


Figure 1.8: Types of RNNs

One to One, One to Many, Many to One and Many to Many (Left to Right)

The comparison of the feed-forward neural network stated on ANN with RNN is shown in **Figure 1.9.** Information only flows from the input layer to the output layer, via the hidden layers in a feed-forward neural network. Although this helps the network receives the information without any delays but this also makes their predictions poor as they barely remember the information they received. Furthermore, a feed-forward network has no concept of time order because it simply takes into account the current input. It is unable to recall anything from the past outside its schooling. However, for RNN, the information in an RNN loops. It takes into account on both of the current input and the what it has learnt from prior inputs when making a decision (Donges, 2019).

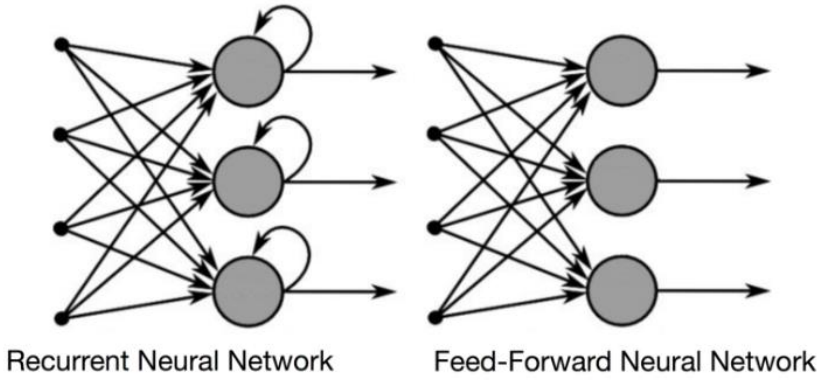


Figure 1.9: Architecture Perspective of RNN and ANN (Donges, 2019).

1.4 Problem Statements

MLA is the most important element in determining the image quality not only on a light-field camera but also on other applications such as optical microscopes, smartphone cameras and so on. This is because it is an optical component which is important in achieving uniform illumination and obtaining four-dimensional (4D) light-field information (Addoptics, 2020). Hence, high precision and rigorous registration with other components are required for MLAs. However, there are several problems on the defect's detection of MLAs:

1. Size constraints of MLAs

Micro-lenses are excessively numerous and small in size, it may be challenging to measure the surface shape and optical performance of the entire MLA. As presented by R. Stevens and T. Miyashita, scattering difficulties may occur in MLA with a high surface roughness, reducing contrast and light collection efficiency (Stevens and Miyashita, 2010). Thus, the imaging system's array constancy is critical, especially when the MLA's size is small and light retrace is required for further processing.

2. Manufacturing Errors

Manufacturing errors is a significant factor in the deterioration of the quality of light-field images. This is because the MLA parameters may change due to manufacturing technology constraints and assembling faults, which can result in surface shape problems and coupling misregistration. This will affect the plenoptic camera's spatial resolution and focusing precision, which has an impact on the transmission of light-field information. Hence, it is crucial to investigate how MLA manufacturing error affects the physical imaging process and image-degradation mechanism of the light-field camera imaging system.

3. Low in Efficient and High in Processing Cost of Fabrication Methods for MLAs

MLA’s fabrication methods are divided into two categories which are direct and indirect. The direct method eliminates the requirement for a concave 3D micro structured mask. When the material is in a thermoplastic or liquid condition, the shape of the micro-lens is usually formed by the surface tension effect, resulting in a very smooth surface (Zhu, Lai and Choi, 2010). More importantly, these processes are straightforward and cost-effective, which is preferable by the industry. However, managing the micro-lens precision remains difficult due to the fact that the geometry of the micro-lens is solely defined by adjusting the parameters such as temperature, wettability, pressure, and process time. Although there are many studies on multiple processing methods such as grinding, ultra-precision diamond turning, and micro-milling to create high precision, high-quality MLAs, still there are two major drawbacks which are low in efficiency and high in processing costs. Furthermore, new methods for fabricating MLAs such as microinjection moulding, electrochemical etching, and local grayscale oxidation have also been introduced to provide good surface quality and process small elements size. However, MLAs are still difficult to be mass produced due to their size.

4. Processing Stages of MLAs

According to Chen et al., the processing stages for MLA are also complicated. The processing cycle took to process silicon-based moulds with ultra-precision milling and injection moulding in order to produce high-quality micro-lenses was too long. The uses of rolling technology to achieve non-contact rapid rolling on a glass substrate, although resulting excellent surface quality in MLA but the control of dimension precision for each micro-lens is difficult (Chen et al., 2014).

5. High Cost

In addition, the cost is high not only because of the high cost of driving equipment for some manufacturing techniques, but also because of the cost due to the oversight of defects due to manual inspection. In the case of manual detection, when the lens becomes smaller in size, the inspection results will differ from person to person, and misjudgement may occur due to eye fatigue from extended working hours. As a result, these will not only lead to low inspection efficiency and high fault percentage but also indirectly increase the burden of cost.

Thus, an application that can differentiates the defect status of MLAs on whether it is “PASS” or “FAIL” is proposed with image processing and image analysis methods. Moreover, a Conventional Siamese Neural Network is trained and tested with the datasets obtained to compare on the similarities of test images for MLA.

1.5 Aims and Objectives

This thesis aims to study the light field imaging of MLAs captured with smartphone camera and develop an application to detect defects pieces of MLAs. In order to achieve these goals, the following objectives should be achieved:

The objectives of the thesis are shown as following:

i) To construct a light field imaging system of MLAs to simulate light field images with different type of MLAs.

ii) To design and develop an application that can perform image analysis and image processing on the light field images of MLAs obtained.

iii) To study and implement the use of Convolutional Siamese Network for similarities check on images.

iv) To investigate a better option on inspecting the defects of MLAs.

1.6 Organization of Thesis

The contents of this thesis are organized as follow:

In chapter two, related works are analyzed, and improvements to be done will be explained where a brief summary of the inspection methods on the defects of MLAs and research projects carried out by other researchers will be investigated. In chapter three, the methodology of this thesis is explained. The experimental setup for the hardware and the methods on processing the software will be discussed. In chapter four, the results and discussion of this thesis will be shown and explained. The results obtained from hardware and simulation will be compared and discussed. Moreover, the comparison on the inspections methods with the one invented will also be discussed. In chapter five, the conclusion and recommendation for future improvements will be shown. The research results on the MLA light field imaging and inspection methods are concluded. Furthermore, the possible future advancement will also be prepared and discussed.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, several papers presenting a range of methods and strategies for implementing defects detection of MLAs were investigated and reviewed. These evaluations are divided into three sections such as light field imaging on MLAs, defects detections of MLAs with image processing and neural network. Advantages and drawbacks of different techniques employed will be discussed and improved.

MLAs are optical devices made up of several small lenses that are used for a variety of purposes, including collimating, lighting, and imaging (Nussbaum et al., 1997). Due to the size constrains, industrial inspection is becoming more and more difficult to undertake, especially when vast quantities of devices are created utilising micro-fabrication techniques. In the traditional way of inspection, to discover broken lenses or faults in the metal coating, the MLA is inspected under a low magnification microscope with a connected video camera (Schambach et al., 2021). A human operator will check each of the microscopic images manually by attempting to locate and count the number of faults in the MLA in order to determine its quality. With this method, the accuracy will be inconsistent and affected due to factors such as weariness, turnover, and inconsistent defect classification. Furthermore, by using this traditional manual inspection, the inspection data record is rarely created and become inaccessible (Instrumental, 2020). Inevitably, some of those human operators who perform manual

inspections might not be technically competent which prevents them from discovering the problems and finding the unexpected defects in time.

As a result, the objectives are to first relieve the human operator from the burden of monitoring a series of images, then enhance the repeatability of the inspection process, and ultimately reduce inspection time using an automated flaw detection system.

2.2 Camera Calibration

In computer vision, camera calibration is crucial to achieve the main goal of calculating the intrinsic parameters and camera distortion parameters. These characteristics are required if photos are used to estimate the pose of objects that are identified or to scan things in 3D. Assuming a pinhole camera model as shown in **Figure 2.1**, is used to perform camera calibration. A basic camera design without a lens is the pinhole camera. When light enters the aperture, it reverses the image and projects it on the other side of the camera (Mathworks.com, 2019)

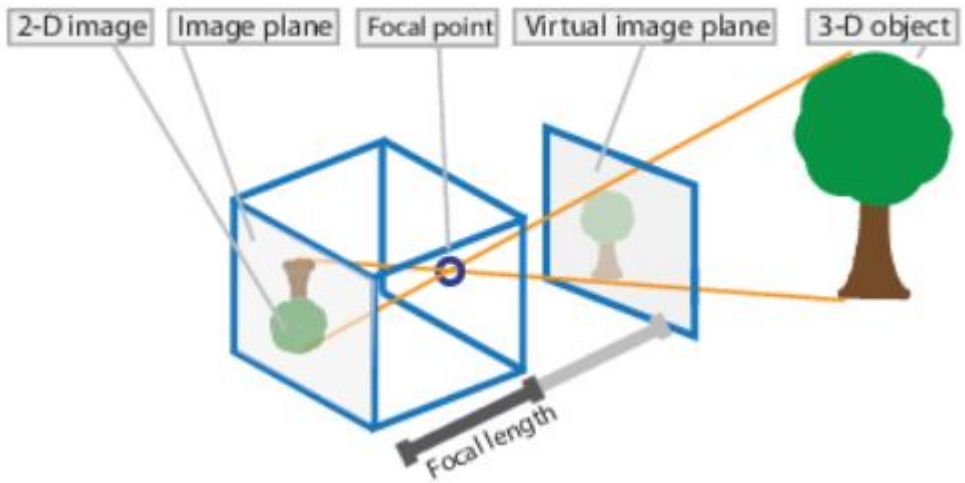


Figure 2.1: Pinhole Camera Model
(Mathworks.com, 2019).

With the calibration algorithm using the extrinsic and intrinsic parameters created, the camera matrix is computed. The extrinsic parameters indicate the camera's position in the 3-D scene. The focal length and optical centre of the camera are represented by the intrinsic characteristics.

$$P = \begin{bmatrix} R \\ t \end{bmatrix} K \tag{2.1}$$

According to **Equation 2.1**, compensated distortion and zero skew are assumed whereby *'P'* is the camera matrix, *'R'* is the extrinsic rotation, *'t'* is the extrinsic translation and *'K'* is the intrinsic matrix with the principal point (cx, cy) [pixel] and the focal lengths [pixel] in the x- and y-direction both presented (Mathworks.com, 2019).

$$K = \begin{bmatrix} \boldsymbol{fx} & \mathbf{0} & \boldsymbol{cx} \\ \mathbf{0} & \boldsymbol{fy} & \boldsymbol{cy} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \tag{2.2}$$

According to **Equation 2.2**, the calibration technique employs related camera coordinates as well as known world coordinates as input points which are used as calibration points. The points established in a world coordinate system that correspond to these image points are called object points (x, y, and z).

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{world} \tag{2.3}$$

According to **Equation 2.3,** the homogeneous transformation matrix, H connects the points in the camera co-ordinate system to object points defined in a world co-ordinate system. Currently, planar checkerboards are frequently used to provide object and image points for camera calibration. The checker corners are estimated and located from these boards' images. The intrinsic matrix and distortion parameters are contained in the camera parameters, which are utilised to estimate camera parameters.

There are many different types of checkerboard patterns shown in **Figure 2.3** which are mostly used in OpenCV tutorials. There are other board types that use circles rather than corners. There are also checkerboards with extra markers to detect partially obscured checkerboards (HAN, 2008).

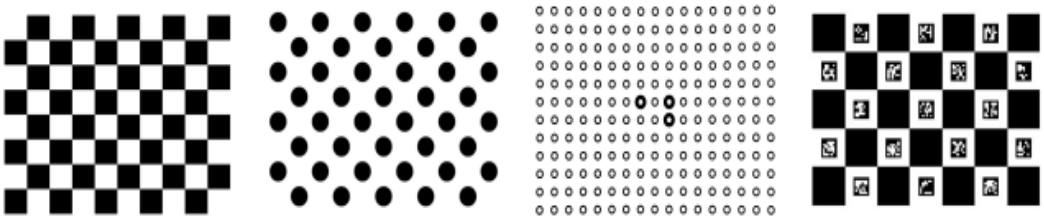


Figure 2.2: Different Types of Checkerboard Patterns (HAN, 2008).

2.3 Image Processing

The following are studies on different possible techniques for defect detection of MLAs based on image processing.

2.3.1 Image Subtraction

Image subtraction is used to emphasise the differences between two photos that have a lot of similarities. Images taken under different or very similar situations may need to be registered as it is possible that the output image has a very narrow dynamic range and needs to be rescaled to fit the display range if the photographs were taken at separate times or with different settings.

Based on **Equation 2.4**, assume image A, $f_1(m,n)$ as the test image and image B, $f_2(m,n)$ as the reference image, the rescaled output under image subtraction will be $b(m,n)$ as the unstretched difference image (Sonka and J Michael Fitzpatrick, 2009).

$$\boldsymbol{b(m,n) = f_1(m,n) - f_2(m,n)} \tag{2.4}$$

In addition, image subtraction can be used to counteract the effects of uneven illumination. In a light microscope image, when parts of the image are much brighter than the rest, the unfavourable effect can be rectified by creating a reference photograph with the same lighting variation (www.allaboutcircuits.com, n.d.). Then, when the reference photograph is subtracted from the image, the pixel values in the brighter locations will decrease significantly more than the pixel values in dimly lit areas. In the difference image, $b(m,n)$, the subtraction operation results in negative pixel values can be prevented by adding a big constant to the minuend image. However, this act might cause overflow to the input image because only a restricted number of bits is used to represent each pixel value in practise. As a result, the length of the data type has to be converted to a longer data length before adding the constant to avoid overflow. For example, if 8-bit data types are used to describe the pixel values of a grayscale image normally, then a 16-bit data type have to be utilised to complete the calculations properly.

Moreover, finding differences between two photographs is another major application of the subtraction technique. Based on **Figure 2.3**, if the input photos have the same value at a given pixel location, the grayscale value of the difference image at that point will be zero (black). On the other hand, if the input photos have different value, the differences will result in a non-zero output value and can be easily identified.

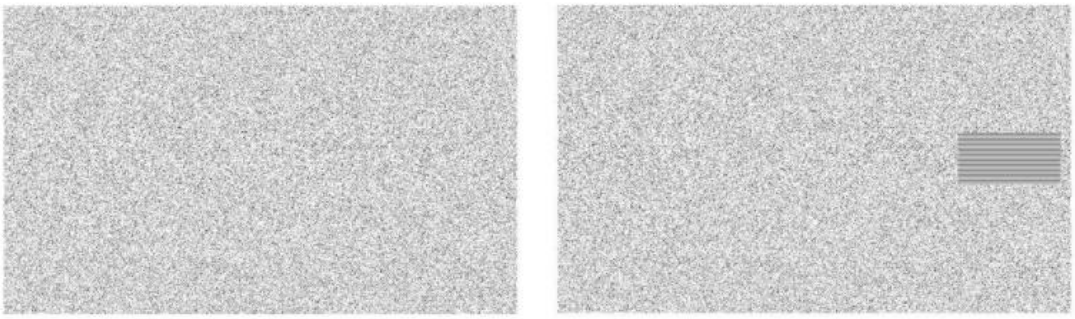


Figure 2.3: Test Image (Left) and Reference Image (Right) for Image Subtraction (www.allaboutcircuits.com, n.d.).

From **Figure 2.4**, the difference image is shown when the two images from **Figure 2.3** is compared and subtracted through the image subtraction technique leaving a hatched rectangular area difference.

Figure 2.4: Rescaled Output Image After Image Subtraction (www.allaboutcircuits.com, n.d.).

Although this method is cost-effective in terms of processing time and memory requirements, it has some drawbacks in terms of data examination. With this method necessitates the use of the test image $f_1(m, n)$ and the image needs to be perfectly aligned with the reference point. Hence, it is inevitably becoming a difficult criterion to meet subjoined. Due to the fact that the digital image resolution for the MLA is low in the implementation of inspection system. As a result, the output precision might be

impacted, which leads to confusion. Moreover, parasitic signals such as light gradients between different parts of the device under inspection are also sensitive to image subtraction (Schelkens et al., 2008). All of these factors point to image subtraction as a poor solution for micro-lens inspection.

2.3.2 Edge Detection

Edge detection is a technique used in image processing to identify object boundaries. It functions by detecting changes in light, which is useful for image segmentation and data extraction (MathWorks., n.d.). Edge detection approaches are divided into two types which are the 1D edge detection and 2D edge detection. From **Figure 2.5**, the 1D edge detection is an algorithm that scans an image down a route and look for points where the image edges and the scan line overlap. For **Figure 2.5**, the 2D edge detection is used to detect the full edge in two dimensions.

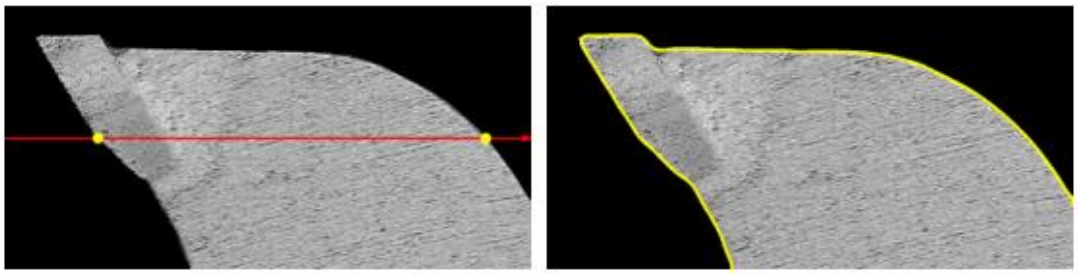


Figure 2.5: 1D Edge Detection (Left) and 2D Edge Detection (Right) (Alan Conrad Bovik, 2009).

There are several methods of edge detection techniques such as the Sobel edge detection and Canny edge detection. For Sobel edge detection shown in **Figure 2.6**, after image pre-processing such as converting image into grayscale, Sobel derivatives along the x-direction and y-direction are obtained. Then, the gradient's strength and direction at each pixel are calculated where high magnitude pixels formed an edge. For Canny edge detection, the implementation is more complicated than Sobel edge detection where edge thinning is performed after performing Sobel algorithm. Edge thinning is carried out pixel-by-pixel where the sharpest gradient is kept. Then, a

double thresholding is performed so that strong edges are selected as pixels with gradient magnitudes greater than the high threshold and appear in the outcome. Lastly, comparison among the magnitude of the processed pixel with the pixels around are performed to identify whether the edges are strong or weak. With these, the weak edges group are discarded and the strong edge groups are retained (Joshi et al., n.d.).

The key benefit of the Sobel operator is less complicated and less time consumed but the edges obtained are rough. On the other side, although the Canny technique have a more complicated and time-consuming algorithms, smoother edge is obtained (Haidar, 2021).

Figure 2.6: Image Input to Different Edge Detection Techniques
Input Image, Sobel, Canny (Left to Right)
(Haidar, 2021).

Edge detection methods are highly sensitive to noise causing the output to be inaccurate. Moreover, the kernel filter's size and coefficients are set in stone and cannot be changed to fit a specific image. The performance of edge detection method is strongly reliant on the changeable parameters leading to more blurring images. Hence, in order to help separate real picture contents from visual artefacts generated by noise, an adaptive edge-detection method is required to give a robust solution that is adjustable to the shifting noise levels of these images (Bovik and Acton, 2009).

2.4 Image Segmentation

The first and most crucial phase in an inspection system utilised in the industry is image segmentation. Whether the system is used to detect faults or cracks in the examined product, it normally includes some method of obtaining one or more photographs of the inspected object. Image segmentation is a subset of image processing which focuses on segmenting an image based on their characteristics and attributes (upGrad blog, 2021). The purpose of image segmentation is to make the image simpler so that it can be analysed easier.

With the implementation of image segmentation, certain pixels can be grouped and divided from an image using image segmentation algorithms. With labels assigned to them and additional pixels being categorised, those specifics objects which are important can be separated from those unimportant components. There are several techniques, such as the Otsu method and U-Net network which are useful for image segmentation.

2.4.1 Otsu Method

Based on the proposal by Nobuyuki Otsu in 1979, Otsu method is used for determining the threshold value with the least weighted variance between foreground and background pixels (Otsu, 2007). The fundamental idea is to consider all feasible threshold values, gauge the spread of background and foreground pixels, and then determine the spread with the smallest value.

However, in the traditional Otsu algorithm, thresholds are found by exhausting all solutions in the grey space. Thus, when the number of thresholds increased, the search dimension to be performed increased which leads to increase of complexity due to many unnecessary calculations, increasing the processing time and reducing the efficiency. Furthermore, images acquired through various channels are subjected to a

variety of random disturbances and conditions, resulting in a large amount of noise in the acquired original images. (Krishnan, 2020).

2.5 Methods Comparison

According to methods discussed above, their capacities to deal with challenges are compared and listed in **Table 2.1**.

Table 2.1: Methods Comparison of Image Subtraction, Edge Detection and Image Segmentation.

Techniques	Pros	Cons
Image Subtraction	• Good at finding differences and uneven	• Need precise alignment
Edge Detection	• Simple • Time-efficient	• Need sharp and thin edges
Image Segmentation	• Easy to analyse	• Sensitive to noise

From **Table 2.1**, the disadvantage of the image subtraction method is that it requires precise alignment of the test and reference images, edge detection requires sharp and thin edges for better efficiency and accuracy whereas image segmentation is sensitive to noise.

2.6 Transfer Learning

Transfer Learning is a machine learning technique in which a previously trained model is reused as the basis for a new model on a different task (Jason Brownlee, 2017). This

optimization method allows rapid progress while modelling the new task. As such, most researchers and data scientists prefer to begin with a pre-trained model that already knows how to identify objects and has learned general features such as edges and shapes in photos.

Since the model will be trained with datasets, a significant amount of time and resources will be required only to train the massive parameters. As a result, transfer learning has become one of the most popular methods for building a deep learning model in the real world, as it only requires training a few final classification layers on top of a base pre-trained model, saving time and data. As claimed by Andrew Ng, a machine learning researcher, "Transfer Learning leads to Industrialization." (Analytics India Magazine, 2018). Hence, transfer learning is said to be a great technique for achieving cutting-edge results and are needed in many applications nowadays.

2.6.1 Pre-Trained Deep Learning Models

A pre-trained model is one that has already been trained to handle a similar problem by someone else. Instead of starting from the beginning to tackle a comparable problem, a model that has previously been trained on another problem might be utilised to begin the learning process. Even if a pre-trained model is not 100% correct, it saves time and effort over having to start from scratch (Analytics Vidhya, 2017).

For instance, by comparing the pre-trained deep learning models with custom built CNN, employing a pre-trained model for various picture recognition tasks is advantageous. By adopting a pre-trained model, less training and work are required in constructing the model's architecture. Another advantage is the precision, using a pre-trained model rather than a custom-built convolutional neural network greatly improves the accuracy.

2.6.2 Comparison of Pre-Trained Deep Learning Models

Quality inspection at manufacturing firms to meet industry standards is time-consuming and the rejected product caused massive waste in factory capacity, consumables, labour, and cost. Hence, industrial companies are trying to leverage deep learning-based computer vision technology to automate material quality inspection during the manufacturing process. This is to decrease human intervention while maintaining or improving human-level accuracy, as well as optimising production capacity, labour costs, and other considerations.

There are several pre-trained models to be selected such as LeNet, AlexNet, VGG, GoogLeNet, Inception V3, Inception BN and so on. Based on the Top 5 error rate in the ImageNet Large Scale Visual Recognition Competition shown in **Figure 2.7**, although VGG-19 is not the best among all of the pre-trained models but according to the ILSVRC 2014 Conference, the VGG-19 is one of the most often used pre-trained image categorization models (image-net.org, n.d.). VGG-19 which is developed at the University of Oxford's Visual Graphics Group, outperformed the previous standard, AlexNet, and was soon adopted by academia and industry for image classification tasks (Ox.ac.uk, 2014).

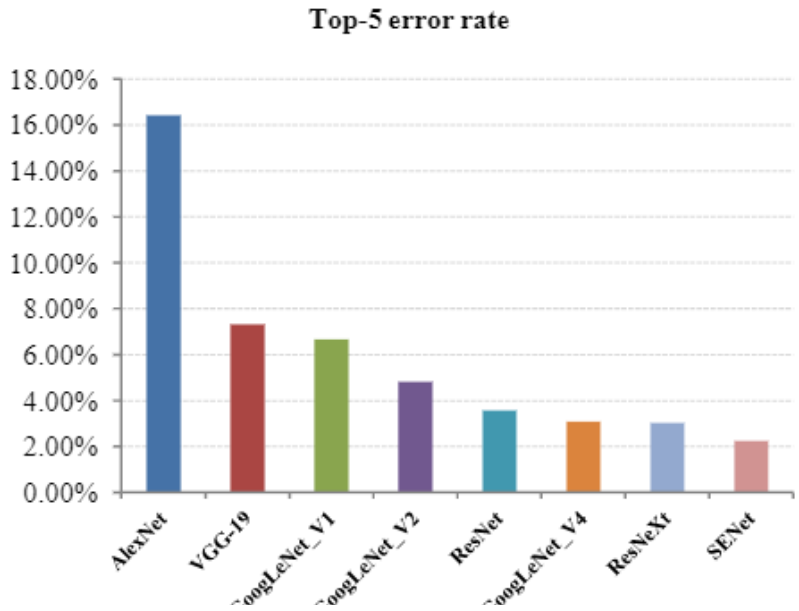


Figure 2.7: Top 5 Error Rate of ImageNet Large Scale Visual Recognition Competition (Itoh et al., 2018).

2.6.3 Classification versus One Shot Learning

In classification, the input image is fed into a series of layers and a probability distribution over all the classes is produced at the output. For example, 4 probabilities are constructed for each input image based on classes such as cat, dog, horse, or elephant. For each class, a lot of photographs during the training process are needed. If the network is only trained on four of the classes above then it can only be used to test on these classes. Thus, data gathering and regular retraining are therefore prohibitively expensive (Lamba, 2019).

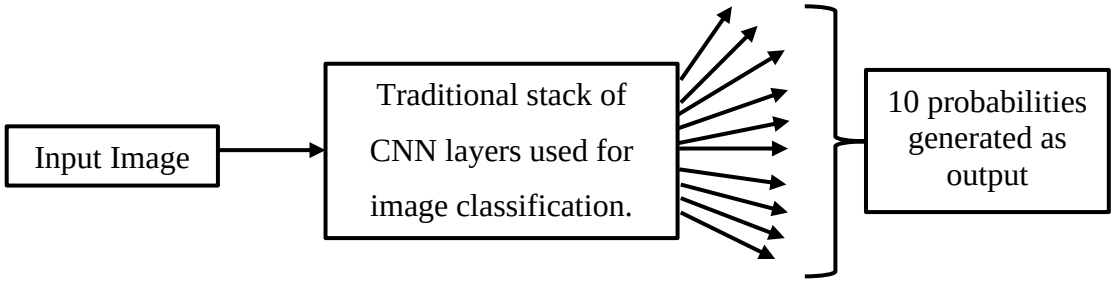


Figure 2.8: Classification Using CNN (Lamba, 2019).

In one shot learning, only one training example is needed for each class. With traditional approach shown in **Figure 2.8**, to build a facial recognition system of 10 peoples, a bunch of images for 10 of these peoples have to be gathered based on different angles or other parameters. If the facial recognition system is changed to detect another batch of people, data has to be regathered and models have to be retrained. However, with one shot learning shown in **Figure 2.9**, if new groups of peoples needs to be added into the system, the network will determine the similarity for any new instance provided to it using newly added images as the reference images.

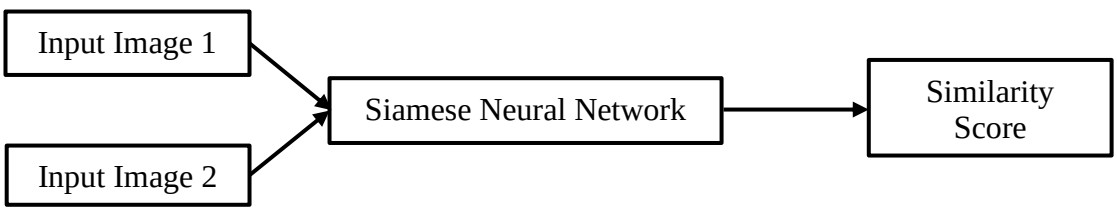


Figure 2.9: One Shot Classification (Lamba, 2019).

2.6.4 Datasets

Datasets are required for model architecture training, validation, and testing. With more data, a deep learning system has a better chance of comprehending it and making correct predictions on the unknown data. Based on the previous researches, most of the light field datasets are obtained from either Lytro Illum camera or plenoptic camera as shown in **Table 2.2**.

Table 2.2: Table of Light Field Datasets Provided Online.

Reference	Datasets	Year
A. Mousnier, E. Vural and C. Guillemot (www.irisa.fr, 2015)	Lytro First Generation Datasets	2015
Rodrigo C. Daudt and Christine Guillemot (www.irisa.fr, n.d.)	Lytro Illum Light Field Datasets	-
Christopher Hahne, Amar Aggoun, Vladan Velisavljevic, Susanne Fiebig, Matthias Pesch (Christopher Hahne et al, 2016).	Standard Plenoptic Camera Datasets.	2016
The Stanford (lightfields.stanford.edu, 2016)	Lytro Light Field Datasets	2016

Although these light field imaging datasets shown above are sufficient, they did not fulfil objectives of this project. Most of the open-source images found are showing the light field images of light field camera with built in MLAs. However, the goal of this project is to gather and investigate the light field images directly from MLAs. Hence, datasets are obtained with new methods such as smartphone camera to identify its feasibility.

2.7 Methods of Defects Detection on MLAs

The following are a few journal studies that discuss various defects detection techniques on MLAs.

2.7.1 Review Journal: *Lens Sag Measurement of Micro-lens Array Using Optical Interferometric Microscope* by Yang, S.-W., Lin, C.-S., Fu, S.-H., Yeh, M.-S., Tsou, C. and Lai, T.-H.

Based on the paper (Yang et al., 2012), researchers created a lens sag detection system for the MLA utilising an optical automatic inspection framework. The spherical MLA with its micro-lens surface illuminated with laser light was captured with CCD camera and a microscope. Then, detection is processed using an image processing technique and the outcomes were compared to computer images.

In the study, the centre position and diameter length were searched using a variety of methods such as the centroid calculation method (CCM) and the area-filling centroid method. The results of different approaches for the centre position and diameter length were discussed in order to choose the one that provided the highest level of accuracy as the primary method for making judgments. Then, the lens sag of the micro-lens was discovered after the centre position and diameter length were determined.

For the CCM, the centre coordinate can be found by adding up and averaging each dark spot coordinate or bright spot coordinate of an image that has undergone binary image processing. However, a lot of factors need to be taken into account when using the centroid calculation method, such as the need for a clear, noise-free image after binarization. These vulnerabilities would result in mistakes in the central position calculations. Next, the area-filling centroid method was proposed to improve the conditions of central position deviation. However, the dark spot coordinates and bright spot coordinates are neither averaged or accumulated by this method. In order to ensure

the computation of the central location is unaffected by noise in the image, the closed region in the area-filling centroid method is filled with a particular colour for calculation against this colour.

Using the area-filling centroid approach might compensate for the shortcomings of the standard centroid calculation method, regardless of the picture quality, allowing for a higher degree of precision in the estimated micro-lens central position. After determining the centre position and the position of the circular interference fringes, the micro-lens diameter may then be determined.

In this study, the AOI system detects lens sag in real-time. As illustrated in **Figure 2.10**, the system framework used CCD to collocate with the MLA and capture the interference image while applying the Fizeau interference principle. The spherical MLA was discovered using an innovative surface profile method and image processing tools. Although the estimation would be incorrect if the micro-lens margin was too blurry or if the observed innermost dark or bright fringes were incomplete, the technique could successfully identify lens sag in real time with regard to various sized MLAs.

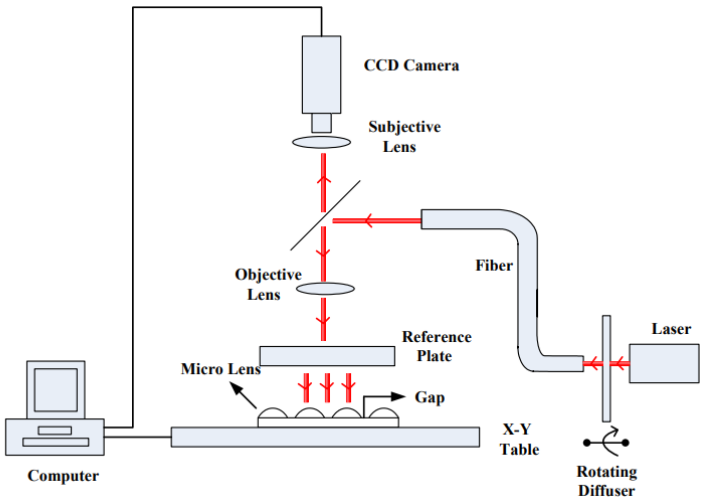


Figure 2.10: System Framework of Proposed AOI System (Yang et al., 2012).

2.7.2 Review Journal: *Local Error and Its Identification for Micro-lens Array in Plenoptic Camera* by Li, S.-N., Yuan, Y., Liu, B., Wang, F.-Q. and Tan, H.-P.

Based on the paper by Li et al., 2018, a local error model for an MLA shown in **Figure 2.11** is created to solve surface flaws brought on by manufacturing faults that might impair the transmission of light-field information.

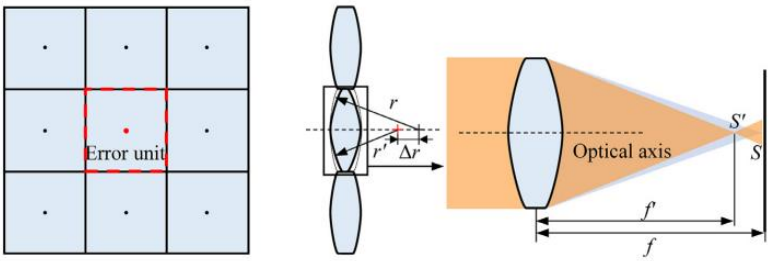


Figure 2.11: Local Error Model for MLA (Li et al., 2018).

According to the research, the local error can be divided into three primary forms such as pitch error, radius-of-curvature error, and decenter error brought on by double-sided processing based on the organisation and surface geometry of the MLA employed. These fundamental mistakes can be combined and superimposed to provide the real surface error. Since the centres of the sub-images corresponding to the micro-lenses have various offsets such as the assembly accuracy, and primary lens aberration as illustrated in **Figure 2.12**. Centre calibration and sub-image division must be applied to each micro-lens to ascertain the link between the sensor pixel and the micro-lens.

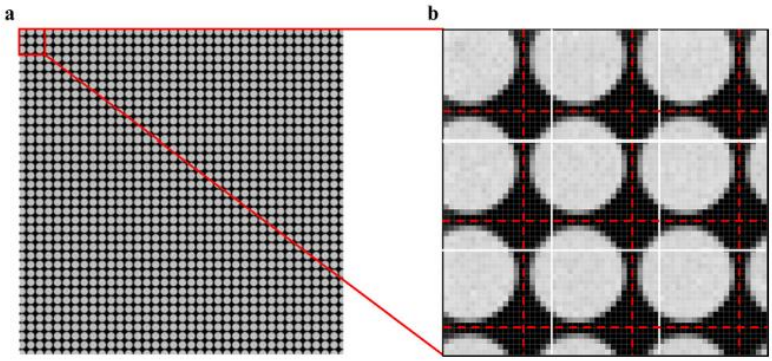


Figure 2.12: Raw Light Field Image (Li et al., 2018).

With the technique of locating error micro-lenses based on image quality evaluation indices introduced, the local imaging properties and degradation mechanism of the three fundamental flaws are examined along with mean square error (MSE) and structural similarity index measure (SSIM).

2.7.3 Review Journal: *Development of an Analysis System for Geometric Contour Error Evaluation in Ultra-precision Machining for Micro-lens Arrays* by You, H. and Yang, J.

Based on the paper by You and Yang, 2014, a system for evaluating contour errors that aids users in selecting appropriate tool path was proposed. The system made decisions correctly depending on the predicted level of a MLA's quality the users expect when they build tool paths and did computation on contour deviations. As a result, machining losses that may be caused by the surplus of data generated during the creation phase are foreseen.

Based on **Figure 2.13**, the quality of a MLA's surface is affected by a variety of factors such as spindle speed, feed rate, tool vibration, heat, tool path data and external environment factors. Therefore, by minimising the disparities between the MLAs and the design criteria that regulate these intricate aspects, the total quality of MLA can be raised. Geometric contour inaccuracy is another element that affects a MLA's quality.

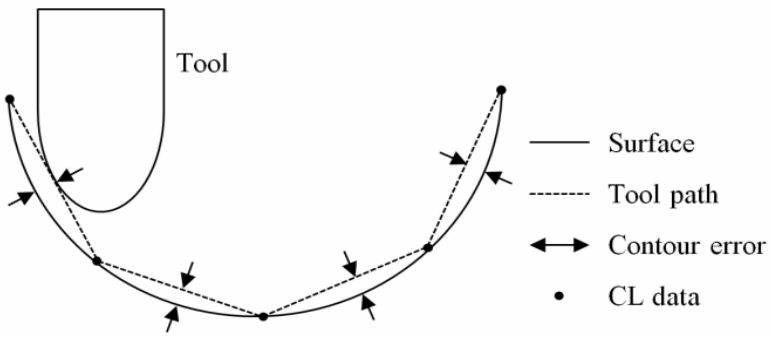


Figure 2.13: MLA's Surface

(You and Yang, 2014)

There are two types of Cartesian methods being introduced such as the cutter contact (CC) Cartesian method and the cutter location (CL) points Cartesian method. Based on **Figure 2.14**, the process of creating tool path data for the parametric method and Cartesian method are described. A free-form surface benefits the parametric method by permitting smoother workpiece machining than the Cartesian method. However, due to serial computing restrictions, computing a parametric surface's basis functions is time consuming. Despite the drawbacks, the Cartesian technique is frequently applied in real-world settings, due to the ability in calculating the tool path data quickly and simpler data structure than the parametric method. As a result, in this study, the CL-Cartesian approach is employed to produce tool path data.

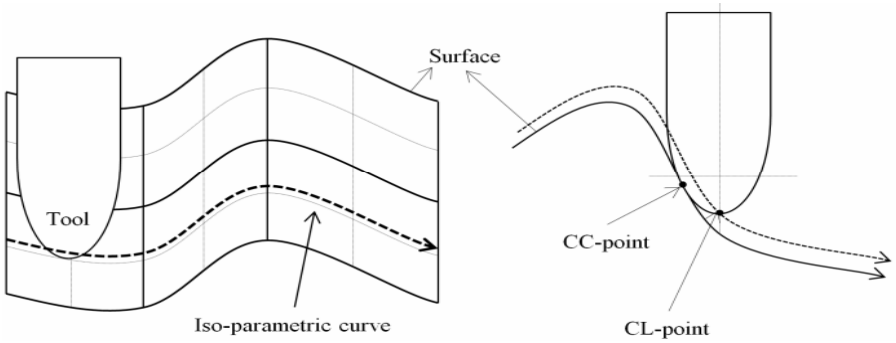


Figure 2.14 Parametric method (Left) Cartesian method (Right) (You and Yang, 2014)

2.7.4 Review Journal: *Micro-optics Metrology using Advanced Interferometry* by Reichelt, S., Bieber, A., Aatz, B. and Zappe, H.

Based on the paper by Reichelt et al., 2005, a flexible Mach-Zehnder/Twyman-Green hybrid interferometer was proposed to enables accurate measurements of radius of curvature, focal length, and lens and surface figure errors on MLAs.

The interferometer is schematically depicted in **Figure 2.15**, which consists of a Twyman-Green and Mach-Zehnder interferometer that can shift in phase simultaneously. The swivelling plane mirror in the reference arm, which switches the

interferometer between the Twyman-Green and the Mach-Zehnder modes and functions as a phase-shifting component, is the essential component of the proposed interferometer system (PZT). Furthermore, software for acquiring and analysing interferometric wavefronts is also constructed. With these methods, both the radius of curvature and the focal length of micro-lenses can be measured without removing the test part allowing the results to be compared without confusion over the real lens and without an azimuthal inaccuracy.

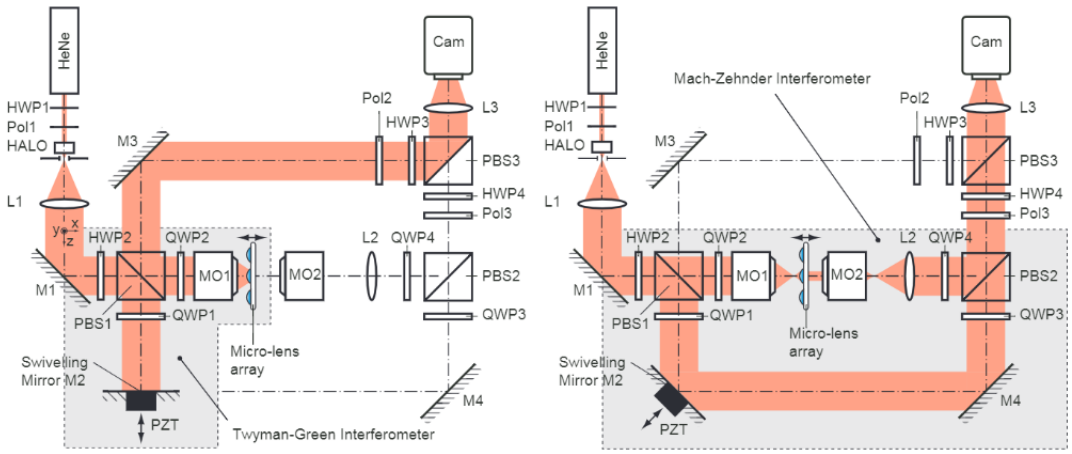


Figure 2.15: Schematic of Proposed Interferometer Methods (Reichelt et al., 2005).

According to the researchers, there are still areas for improvements such as the improving the interferometer software so that the systematic error of the interferometer can be calibrated to increase measurement accuracy, and the measurements can be automated to enable array testing.

2.7.5 Comparison of the Techniques Used by Other Researchers

In this chapter, it is evident that defect detection system for MLA is important as the accuracy and effectiveness of target reconstruction can be greatly impacted by MLA faults and lead to confusion or mismatching of 4D spatio-angular information in the image space. It can be concluded that the enhancement and optimization of

structural design, processing technology, and prepared material has been the main focus of recent research on MLA manufacturing mistake. Furthermore, since micro-lenses are excessively numerous and small in size, it may be challenging to maintain the surface shape and optical performance of the entire MLA during fabrication.

Based on the reviews done on the previous researches, the MLA's faults can be categorized into two types such as manufacturing error and measurement error. In investigating manufacturing error, Yang et al. proposed a non-contact testing method for spherical MLA of various diameters that uses the Fizeau interference principle as a lens sag detection system. Li et al. proposed a local error model for an MLA based on image quality evaluation indices. You et al. presented a system for evaluating contour errors that makes it easier to arrange the tool path logically during MLA processing and increases the accuracy of the MLA surface. Reichelt et al. proposed a flexible Mach-Zehnder/Twyman-Green hybrid interferometer to enables accurate measurements of radius of curvature, focal length, and lens and surface figure errors on MLAs. Based on **Table 2.3**, a comparison was done among all of the review journals according to their techniques, pros and cons.

Table 2.3: Summary of Comparison Among Review Journals.

No.	Authors	Type of Techniques	Pros	Cons
1	Yang, S.-W., Lin, C. S., Fu, S.-H., Yeh, M.-S., Tsou, C. and Lai, T.-H. (2012)	Image processing	-Can effectively identify lens sag of different-sized MLAs. -Noise-free for central calculation of image.	-Margin of micro-lens have to be clear. -Innermost dark or bright fringes have to complete.
2	Li, S.-N., Yuan, Y., Liu, B., Wang, F. Q. and Tan, H.-P. (2018)	Image processing	-Hard to implement	-Localization of image center need to be precise.
3	You, H. and Yang, J. (2014)	Deep learning	-Useful for shape analysis	-Poor convergence performance
4	Reichelt, S., Bieber, A., Aatz, B. and Zappe, H. (2005)	Deep Learning	-Not sensitive to intensity fluctuations.	-Hard to achieve accurate readings.

CHAPTER 3

METHODOLOGY

3.1 System Overview

In this chapter, the method and justification for the hardware and software carried out are explained. The objective of this project is to setup a model to gather light field images of MLAs as data and implement them into software developed to undergo image processing and image analysis. There are two methods such as image processing techniques and deep learning techniques used to develop the software so that the best model is selected. The block diagram of the defect detection system is shown in **Figure 3.1**.

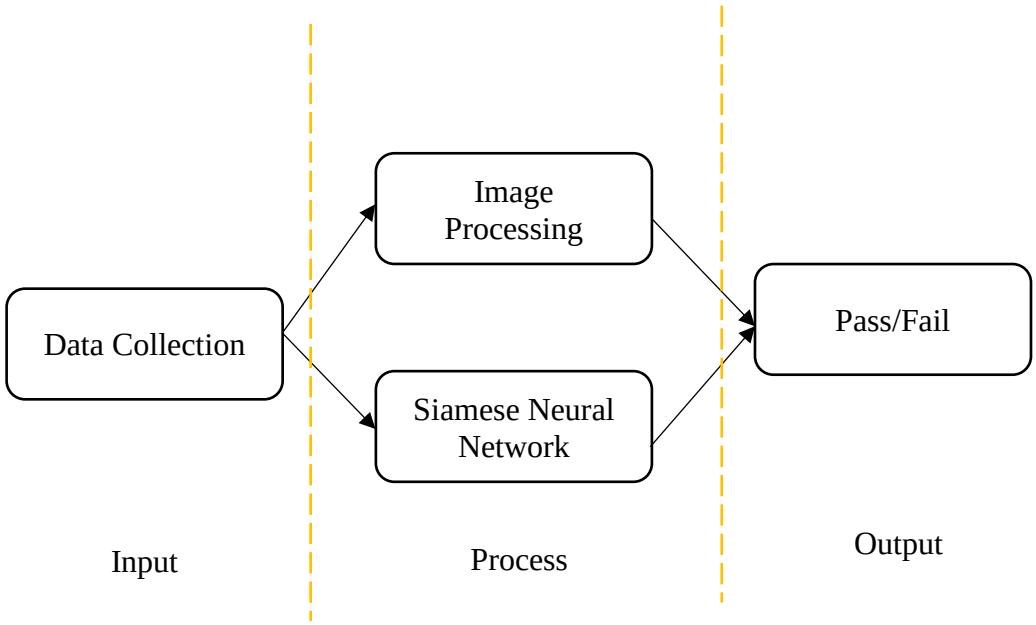


Figure 3.1: Block Diagram of Proposed Defect Detection System.

3.2 Equipment

Table 3.1 shows the list of equipment required for the development of proposed defect detection system of MLAs. The equipment part can be divided into hardware and software. Hardware forms the image capture system for data collection. Software forms the analysis system for image pre-processing, feature extraction, image classification.

Table 3.1: List of Equipment Required.

Hardware	Software
• VCSEL Lasers Multi-Mode VSCEL Power Array • iPhone 13 pro max • Bosch Laser Measure GLM 50 C Professional • MLA diffuser	• Google Colaboratory • PyCharm • Roboflow • Qt Designer

3.2.1 Hardware Configuration

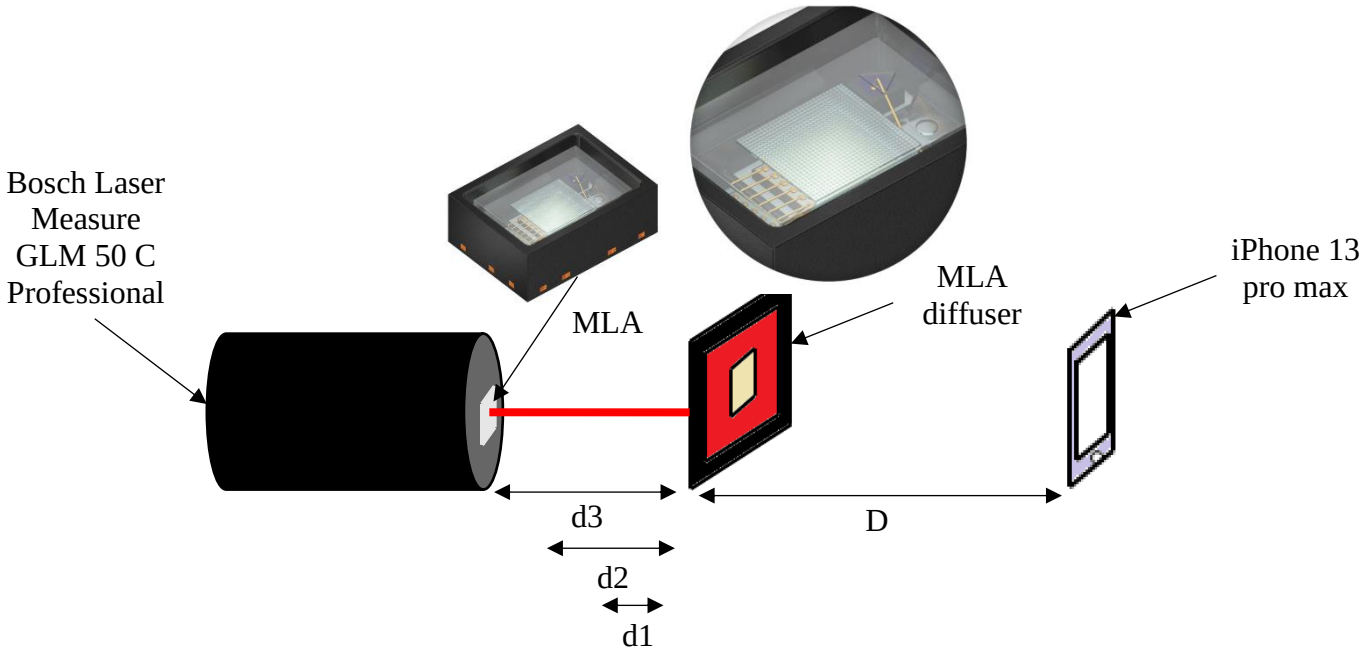


Figure 3.2: Hardware Configuration for Proposed Defect Detection System.

The hardware configuration for the proposed defect detection system of MLAs is shown in **Figure 3.2**. It consists of one Bosch Laser Measure GLM 50 C Professional, VCSEL Lasers Multi-Mode VSCEL Power Array, MLA diffuser and an iPhone 13 pro max.

From the setup, the MLA diffuser of dimensions 10cm x 10cm is placed in a distance, D=20cm from the camera. It is treated as the calibration object to calibrate the camera of iPhone 13 pro max. As illustrated in **Figure 3.2**, there are three different distances, *'d'* between the MLA and the MLA diffuser. The distance is set to 1cm, 2cm, and 3cm respectively for '*d1'*, '*d2'*, and '*d3'*. This is to observe the light field images projected on MLA diffuser when distance differed. The MLA with a wavelength of 900nm is sticked on the point where the laser beam emitted. The laser of Bosch Laser Measure GLM 50 C Professional is used as the light source of the hardware framework to illuminate the MLA surface.

3.3 Camera Calibration

In this project, camera calibration method is introduced to measure the camera parameters of iPhone 13 pro max used. The flowchart of the camera calibration system is shown in **Appendix A.** The setup of camera calibration method is performed as shown in **Figure 3.3**.

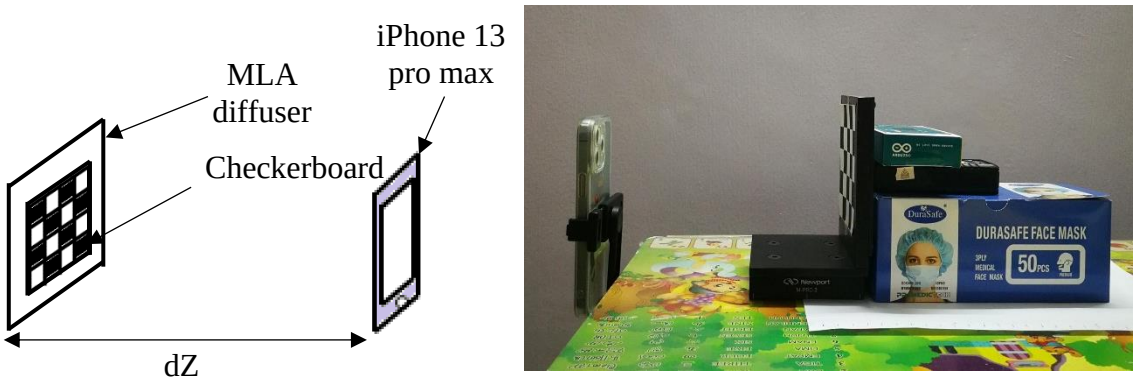


Figure 3.3: Hardware Configuration for Camera Calibration System.

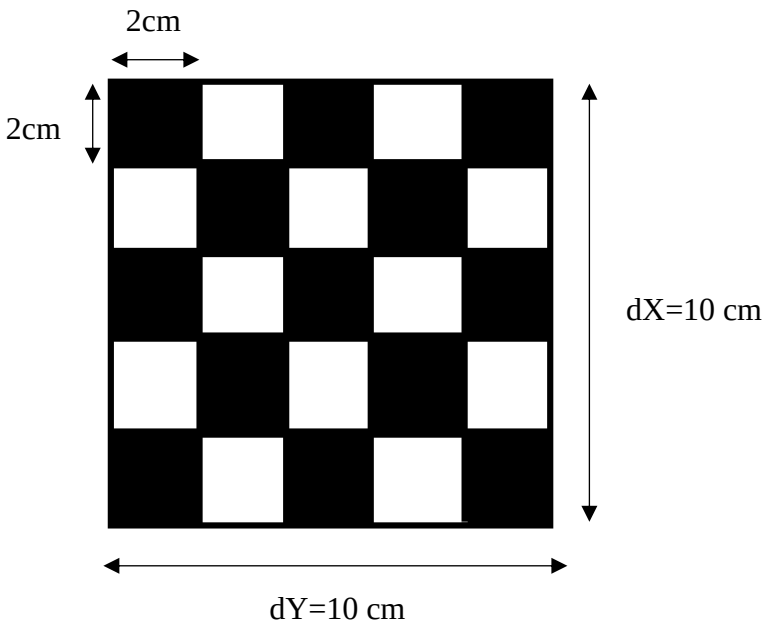


Figure 3.4: Target Plate Size.

The target plate size is shown in **Figure 3.4**, since the MLA diffuser is 10cm x 10cm, a checkerboard image of square sides 10cm each with black and white squares of sides 2cm each is printed and displayed as the calibration pattern. In the beginning, the width and length of the checkerboard image is referred to as '*dX'* and '*dY'*. The checkboard image is placed on a flat surface parallelly to the camera and in the middle of the camera's vision as shown in **Figure 3.5**. The distance between the camera and the checkerboard image, '*dZ'* is calculated and recorded. Then, a picture is captured to make sure the arrangement is straight and the sides of the checkerboard image are aligned.

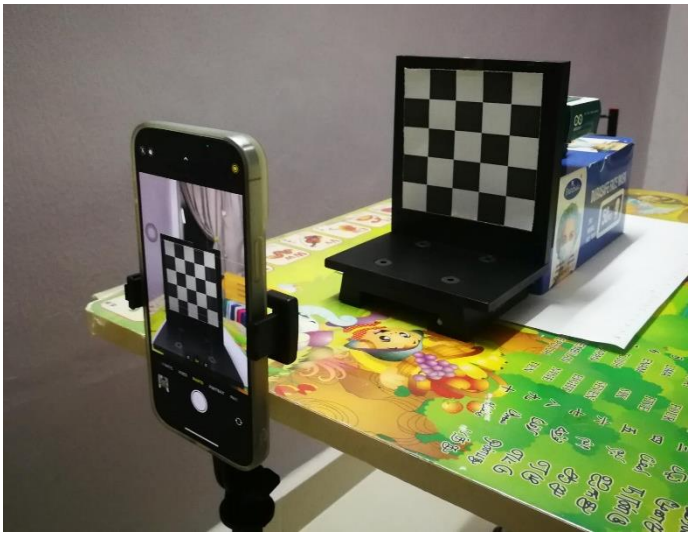


Figure 3.5: Hardware Configuration for Camera Calibration System.

With the data image loaded to the laptop, the camera calibration is performed with the camera calibration software shown in **Appendix C**. Then, the OpenCV method findChessboardCorners() is used to determine the pixel coordinates (*u*, *v*) for each 3D point in the checkerboard image. The pixels size of the image is recorded as (*dx*, *dy*) respectively for its width and height in pixel. Lastly, the output corner detection and camera parameters of checkerboard image are saved and the results obtained are discussed.

3.4 Dataset Preparation and Acquisition

There are a total three sets of data needed for the whole defect detection system for MLAs. For the camera calibration system, image data of checkerboard is captured with smartphone camera. The camera is repositioned for 7 times in different distance from the checkerboard image so that a better calibration is achieved and the best distance and height of the camera is adjusted.

For the image processing based main programme of the defect detection system of MLAs, images are obtained from the hardware as the smartphone camera light field

datasets are not available in various sources such as Kaggle, GitHub, COCO, ImageNet, and PascalVOC. The image captured are in the format of PNG format and the resolution is 3024 x 4032. There are three types of MLAs such as Reference, Flat and Narrow used to produce the light field images, and 10 images are captured respectively for all three types. Since the main objective of this project is to detect defects on MLAs, one reference image for each type is captured from the good condition MLA so that comparison can be made to compare among defects MLAs and defect-less MLAs.

After image data is captured, they are loaded to a desktop computer for further processing through a USB cable in order to maintain full resolution of images. For image processing, images are directly loaded to the software when the particular images are chosen. On the other hand, for Siamese Neural Network, the dataset is uploaded to Roboflow for annotation of custom datasets. It allows user to annotate images and categorised the groups with a pre-defined label. After annotations, all of the 30 images from the dataset are split evenly among training, validation, and testing, then exported in PascalVOC format.

3.5 Image Processing Operation

3.5.1 Installation of Required Software and Library Packages

The proposed defect detection system of MLAs is developed using PyCharm with the Python programming language.

Install PyCharm

PyCharm is a Python-specific Integrated Development Environment (IDE), that offers a wide range of crucial tools for Python developers. There are three editions of PyCharm such as community version, Edu version that are free and open sourced and professional version that requires subscription. The version of PyCharm selected

for this project is the community version which aids on the Python development. Before Python language can be used in PyCharm, Python is downloaded and installed from the main webpage of Python as shown in **Figure 3.6**. Hence, the version of Python 3.9 (64-bit) is downloaded based on the operating system of the laptop used for this project.

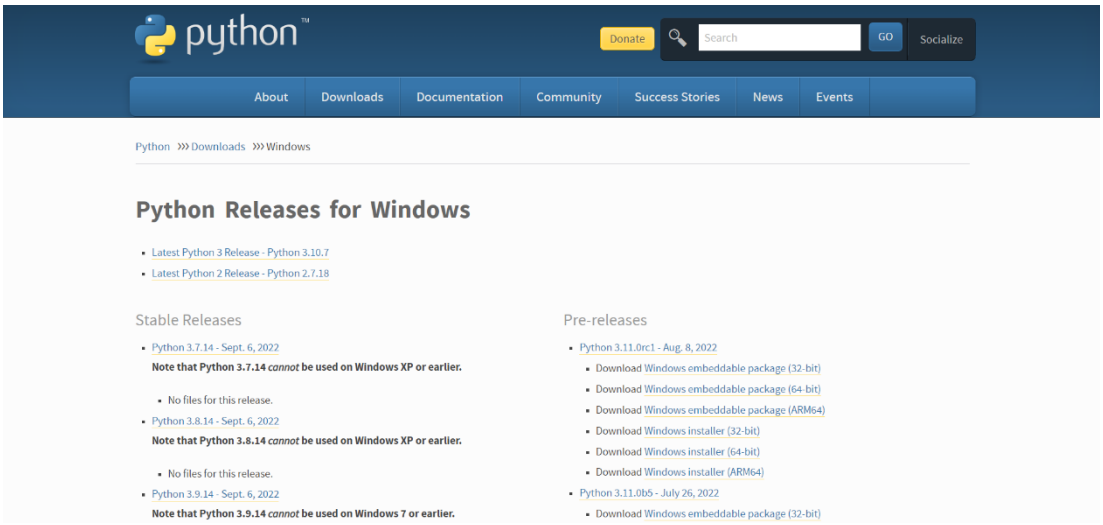
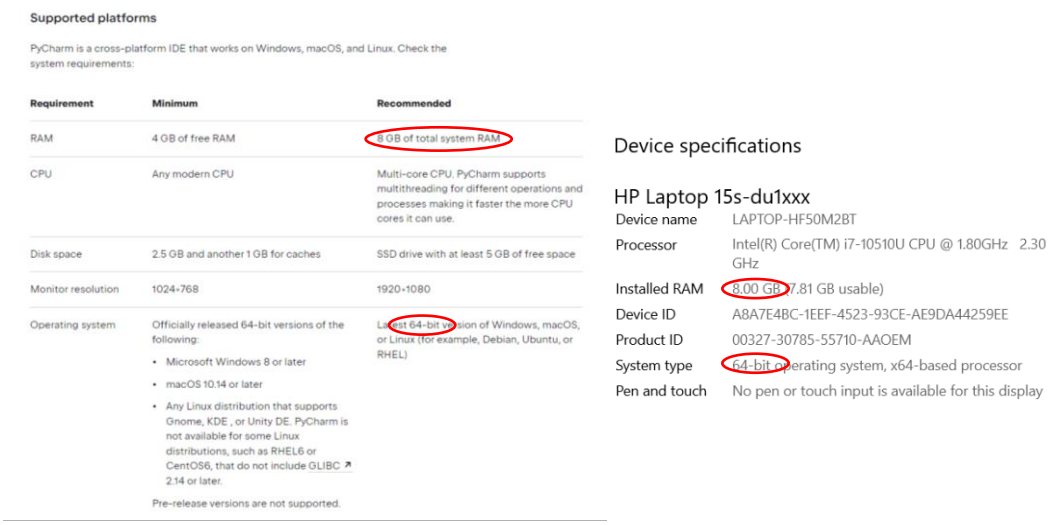


Figure 3.6: Main Webpage for Python Installation.

From the requirement suggested by PyCharm shown in **Figure 3.7**, the specifications of the laptop used for this project is inspected. According to **Figure 3.7**, the model of the laptop used for this project is HP Laptop-HF50M2BT with Intel(R) Core (TM) i7-10510U CPU @ 1.80GHz 2.30 GHz processor and 8GB RAM. Since the specifications met the requirement, PyCharm can be installed.



Supported platforms

PyCharm is a cross-platform IDE that works on Windows, macOS, and Linux. Check the system requirements:

Requirement	Minimum	Recommended
RAM	4 GB of free RAM	8 GB of total system RAM
CPU	Any modern CPU	Multi-core CPU. PyCharm supports multithreading for different operations and processes making it faster the more CPU cores it can use.
Disk space	2.5 GB and another 1 GB for caches	SSD drive with at least 5 GB of free space
Monitor resolution	1024×768	1920×1080
Operating system	Officially released 64-bit versions of the following: • Microsoft Windows 8 or later • macOS 10.14 or later • Any Linux distribution that supports Gnome, KDE , or Unity DE. PyCharm is not available for some Linux distributions, such as RHEL6 or CentOS6, that do not include GLIBC 2.14 or later. Pre-release versions are not supported.	Latest 64-bit version of Windows, macOS, or Linux (for example, Debian, Ubuntu, or RHEL)

Device specifications

HP Laptop 15s-du1xxx

Device name	LAPTOP-HF50M2BT
Processor	Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
Installed RAM	8.00 GB (7.81 GB usable)
Device ID	A8A7E4BC-1EEF-4523-93CE-AE9DA44259EE
Product ID	00327-30785-55710-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 3.7: Requirement Suggested by PyCharm (Left) Laptop Specification (Right).

After PyCharm is successfully installed, the Python interpreter downloaded is configured with the path to the Python executable in system specified.

Install Library Packages

Next, necessary library packages are installed to develop the proposed system. Instead of installing these library packages by adding the library packages through Python interpreter page in PyCharm, Pip Installs Packages (PIP) installation is used. With the command prompt in Windows launched, libraries packages are installed based on commands stated in **Table 3.2.**

Table 3.2: Pip Installation Command for Library Packages.

Library Packages	Command
OpenCV	`pip install opencv-python`
Matplotlib	`pip install matplotlib`
Pandas	`pip install pandas`
Numpy	`pip install numpy`
Os	`pip install os-sys`
Skimage	`pip install scikit-image`

Continue to the next page

	Continue from the previous page
Tkinter	`pip install tk`
Glob	`pip install glob2`

3.5.2 Image Processing Main Software

The overall process of the image processing-based software for defect detection of MLAs system is shown in **Appendix B**. The program started with performing condition check on the test image. A message on whether the test image is loaded correctly is returned. Then, the test image is duplicated and image pre-processing such as noise reduction is performed. After image pre-processing is done, line scan is performed on the test images by extracting the pixel value from row to row and column to column. Some necessary information such as type of image, image shape, pixel size, dimensions, maximum RGB value and the centre of the image are displayed as output and recorded. Lastly, the intensity distributions of the reference image and test image are plotted and the status of the test MLA is shown.

3.6 Deep Learning Main Software

The overall process of the deep learning-based software for defect detection of MLAs system is shown in **Figure 3.8**. The process is classified into five main steps such as data collection, data preparation, model fitting, model evaluation and parameter tuning.

Figure 3.8: Overall Process for Deep Learning Based Software.

Images obtained from the hardware are uploaded to Roboflow for annotation. The data set exported in PascalVOC format is then used to train the Siamese Neural Network model by uploading the zip file to Google Colaboratory. With commands stated in **Appendix E**, the file is unzipped. Necessary packages shown in **Table 3.3** are installed and imported.

Table 3.3 Pip Installation Command for Library Packages.

Library Packages	Command
Matplotlib	`pip install matplotlib`
Numpy	`pip install numpy`
Random	`pip install random`
Pillow	`pip install Pillow`
Torch	`pip install torch`

For the training verification network, three sets of datasets with different classes of MLAs are used. Then, the Siamese Network is defined and created. Moreover, the contrastive loss function is also defined. Next, iteration is performed among the datasets and training loss graph is visualised. Lastly, information on verification and one-off performance are investigated.

3.7 Cost Analysis of the Project

The total amount spent for this project are tabulated in **Table 3.4** below.

Table 3.4: Summary of Cost Analysis.

Equipment	Quantity	Remarks	Cost (RM)
VCSEL Lasers Multi-Mode VSCEL Power Array	1	Website: MouserElectronic.com	54.50
Bosch Laser Measure GLM 50 C Professional	1	Website: Lazada.com	578.30
MLA Diffuser	1	Acquired from friend	-
Phone Camera Tripod	1	Case Zone	69.00
		Total Cost	**701.8**

3.8 Project Management

The schedule of the project is shown in the Gantt chart as below:

Table 3.5: Gantt Chart for FYP

FYP	**FYP 1**					**FYP 2**			
Month	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**June**	**Jul**	**Aug**	**Sep**
Input									
Propose topic									
Learn basic Python for application invention									
Collect materials for hardware									
Process									
Build hardware									
Collect data for datasets									
Pre-process datasets									
Tweak software to optimize output									
Output									
Test the hardware and software									
Analyze data									
Final test run									
Report Writing									
Complete									

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Hardware Testing

In this chapter, hardware testing has been performed to evaluate the operation of the design and hardware used.

4.1.1 Camera Calibration

Based on **Table 4.1**, comparison of several parameters between a charge-coupled device (CCD) camera and iPhone 13 pro max camera are shown.

Table 4.1 Parameter Comparison.

Parameter	Omron FZ-SC (CCD camera)	iPhone 13 pro max camera
Resolution	0.3MP	12 MP
Aperture lens	f/8	f/1.5
Pixel pitch	7.4um	1.9um
Sensor Area	4.8mm x 3.6 mm	44 mm x 44mm
Cost	RM 5518.96	RM 4,999.00

Based on the comparison stated at **Table 4.1**, iPhone 13 pro max is suitable to be used in light field imaging of this project. This is because the resolution is higher than the CCD camera. The higher the megapixels of the resolution, the more details are recorded. Thus, a better-quality image is captured. Furthermore, a lower aperture means more light is entering the camera, which is better for low-light scenarios. This makes the iPhone 13 pro max camera with a f/1.5 better than the CCD camera with a f/8. Next, the pixel pitch for iPhone 13 pro max camera is smaller than CCD camera indicating higher pixel density and higher resolution. The bigger the sensor, the better the quality, so it is obvious that iPhone 13 pro max camera are preforming better. Last but not least, the price of the two cameras is almost the same. However, iPhone 13 pro max phone is chosen instead of a CCD camera, because according to the above conditions, the mobile phone is indeed better, which cuts the costs in buying a CCD camera that is only used for once.

4.1.1.1 Camera Calibration (Data Preparation)

The developed camera calibration system was tested with 6 checkerboard images captured from different angles shown in **Figure 4.1 and Figure 4.2.**

Figure 4.1: Checkerboard Images with 60cm, 70cm, 80cm Camera's Height (Left to Right)

Figure 4.2: Checkerboard Images with 20cm, 25cm, 30cm Camera's Distance (Left to Right)

Based on **Figure 4.3**, a snapshot of the results obtained for one of the calibration test images are shown. Four types of camera parameters such as the camera matrix, distortion coefficient, rotation vectors and translation vectors are displayed.

```
Calibration starts
Successfully saved
 Camera matrix:
[[4.97688011e+03 0.00000000e+00 7.55327754e+02]
 [0.00000000e+00 5.24459604e+03 1.13633750e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

 Distortion coefficient:
[[ 4.57801166e+00 -9.63126690e+01  4.47845982e-02  4.23182312e-03
  -1.12092591e+03]]

 Rotation Vectors:
(array([[-0.29634374],
       [-0.00950116],
       [-0.03813   ]]),)

 Translation Vectors:
(array([[-1.6951517 ],
       [-2.74880101],
       [33.67713583]]),)
```

Figure 4.3: Output Messages of Calibration Test.

4.1.1.2 Camera Calibration (Data Testing)

As previously stated, 6 test patterns are used to calibrate the camera. The software of OpenCV is presented in **Appendix C**, where 3D real-world points and their associated 2D image coordinates are loaded as input data for calibrating the camera. Checkerboard are positioned stationarily, with a camera capturing images in various places and orientations. Since, the camera was moved in accordance with how the checkerboard was kept stable at the XY plane, Z=0. This idea enables us to identify solely the X and Y values. With X=10cm and Y=10cm, the numbers are passed as (0,0), (10,0), (20,0) and so on.

Then, the number of grids is loaded to the software as to find the pattern of the checkerboard. Since, the checkerboard image used are in 5x5 grid, the checkerboard size is set as (4,4) for corner detection on the internal corners. The system returned the corner points as shown in **Figure 4.4** and 'True' if the desired pattern is obtained. Once the corners are located, cv.cornerSubPix is used to improve the accuracy. Additionally, cv.drawChessboardCorners is used to draw the pattern.

Figure 4.4: Input Test Image (Camera's Height=60cm) and Output Image with Corner Detected (Left to Right)

Figure 4.4 shows the test input image captured by proposed camera at the height of 60cm from ground and the output image after corner detection. Next, the calibration process is continued with the object points and image points obtained. The cv.calibrateCamera() function shown in **Figure 4.5** is used to obtain the camera matrix, distortion coefficients, rotation and translation vectors.

```
ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objpoints1, imgpoints1, gray.shape[::-1], None, None)
```

Figure 4.5: cv.calibrateCamera() function

Figure 4.6 and **Equation 4.1** show the camera intrinsic matrix obtained from the camera calibration system. The camera matrix obtained is rounded up to approximately:

```
print(cameraMatrix)

[[2.32921740e+03 0.00000000e+00 1.55805115e+03]
 [0.00000000e+00 2.32249606e+03 2.06920165e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

Figure 4.6: Output Messages of Camera Matrix.

$$K = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2.329 & 0 & 1.558 \\ 0 & 2.322 & 2.069 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.1)$$

Next, the camera extrinsic matrix is obtained with rotation matrix, *'R'* and translation vector *'t'* as shown in **Equation 4.2**.

$$[R \;\; t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \quad (4.2)$$

Firstly, the rotation vectors (rvecs) as shown in **Figure 4.7** is obtained from the camera calibration system. It is a 3D rotation vector that specifies the orientation as well as its rotation axis and angle.

```
[64] print(rvecs)

(array([[0.00718142],
        [0.01154581],
        [0.00997095]]), array([[0.00718142],
        [0.01154581],
        [0.00997095]]), array([[0.00718142],
        [0.01154581],
        [0.00997095]]), array([[0.00718142],
        [0.01154581],
        [0.00997095]]), array([[0.00718142],
        [0.01154581],
        [0.00997095]]), array([[0.00718142],
        [0.01154581],
        [0.00997095]]))
```

Figure 4.7: Output Messages of Rotation Vectors.

Then, the Rodrigues function, cv.Rodrigues() is used to transform the rotation vectors into a 3 x 3 rotation matrix, *'R'* as shown in **Figure 4.8.**

```
rmat = cv2.Rodrigues(rvecs[0])
print(rmat)

(array([[ 0.99988364, -0.00992902,  0.01158107],
        [ 0.01001193,  0.99992451, -0.00712352],
        [-0.01150946,  0.00723864,  0.99990756]]), array([[ 1.39272424e-07,  5.79658700e-03,  4.95767633e-03,
          5.74885129e-03, -7.18116184e-03, -9.99935496e-01,
          5.01295165e-03,  9.99935358e-01, -7.18114156e-03],
        [-1.15453136e-02,  3.62891947e-03,  9.99908115e-01,
          3.55217318e-03,  1.45274980e-07,  5.01288374e-03,
         -9.99908253e-01,  4.95760842e-03, -1.15453596e-02],
        [-9.97052031e-03, -9.99919547e-01,  3.55219346e-03,
          9.99919410e-01, -9.97058822e-03,  5.79654096e-03,
          3.62893975e-03,  5.74880525e-03,  1.53615008e-07]]))
```

Figure 4.8: Output Messages of Rotation Matrix.

Similar to the rotation vectors, translation vectors (tvecs) as shown in **Figure 4.9** result from a translation (x, y, z) from the origin.

```
print(tvecs)

(array([[-1.588528  ],
        [-3.17397106],
        [ 7.86544569]]), array([[-1.588528  ],
        [-3.17397106],
        [ 7.86544569]]), array([[-1.588528  ],
        [-3.17397106],
        [ 7.86544569]]), array([[-1.588528  ],
        [-3.17397106],
        [ 7.86544569]]), array([[-1.588528  ],
        [-3.17397106],
        [ 7.86544569]]), array([[-1.588528  ],
        [-3.17397106],
        [ 7.86544569]]))
```

Figure 4.9: Output Messages of Translation Vectors.

With the 3 x 3 rotation matrix ‘R’ and translation vector ‘t’, the extrinsic matrix is shown in **Equation 4.3**.

$$[R\ \ t] = \begin{bmatrix} 0.99988364 & -0.00992902 & 0.01158107 & -1.588528 \\ 0.01001193 & 0.99992451 & -0.00712352 & -3.17397106 \\ -0.01150946 & 0.00723864 & 0.99990756 & 7.86544569 \end{bmatrix} \quad (4.3)$$

Next, distortion coefficients are gathered as shown in **Figure 4.10** and **Equation 4.4**.

```
[24] print(dist)

     [[ 0.26348375 -0.91351622  0.00625441  0.00431559  1.7379322 ]]
```

Figure 4.10: Output Messages of Distortion Coefficients.

$$Distortion\ coefficients = (k_1\ k_2\ p_1\ p_2\ k_3)$$
$$= (0.26348375\ \ -0.391351622\ \ 0.00625441\ \ 0.00431559\ \ 1.7379322) \quad (4.4)$$

With OpenCV, the radial and tangential distortions are described as *‘k_1’, ‘k_2’, ‘k_3’ and ‘p_1’, ‘p_2’* respectively. After that, image distortion is removed by using cv.getOptimalNewCameraMatrix, With this, an undistorted image with the fewest unnecessary pixels is returned. Lastly, re-projection error is performed to provide a reliable indication of how precise the parameters were actually discovered to be. The accuracy of the parameters discovered increases with the re-projection error's distance from zero. Thus, cv.projectPoints is used to transform the object point to the image point with the intrinsic, distortion, rotation, and translation matrices. Based on **Figure 4.11**, the absolute norm between the results of transformation and the corner locating procedure is determined.

```
# Reprojection Error
mean_error = 0

for i in range(len(objpoints2)):
    imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], tvecs[i], cameraMatrix, dist)
    error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2)/len(imgpoints2)
    mean_error += error

print( "total error: {}".format(mean_error/len(objpoints)) )
```

```
total error: 0.023798287568719798
```

Figure 4.11: Output Messages of the Total Error.

A strong camera calibration has a total error of less than 1.0. Thus, the camera calibration for this test image used is good enough with only 0.024 total error obtained. However, as shown in **Table 4.2**, more positions are investigated so that the least total error location is obtained for our proposed system.

Table 4.2: Total Error for Different Input Image.

Checkerboard Image	Total Error
Height of camera stand from the ground	
Image 1 (h=60cm)	0.023798287568719798
Image 2 (h=70cm)	0.02381852821789704
Image 3 (h=80cm)	0.019693988304370975
Distance of camera to checkerboard image	
Image 4 (d=20cm)	0.014942440184795791
Image 5 (d=25cm)	0.019201723750899242
Image 6 (d=30cm)	0.022835818320492418
Camera with Height=80cm and Distance=20cm	
Image 7 (selected)	0.01650393425789723

Based on the camera calibration system, images with corner detected and their respective parameters are shown in **Figures 4.12 to 4.14.**

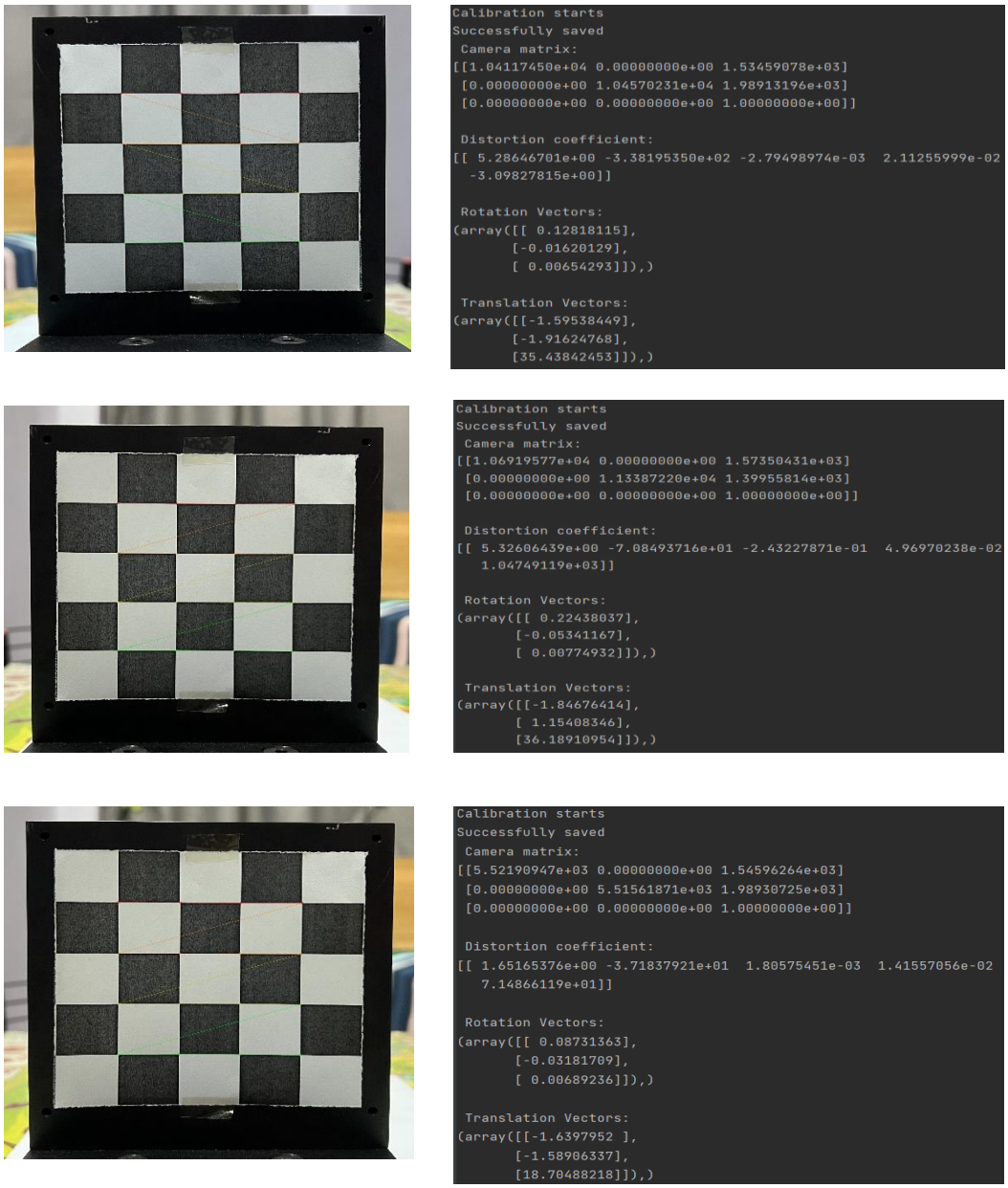
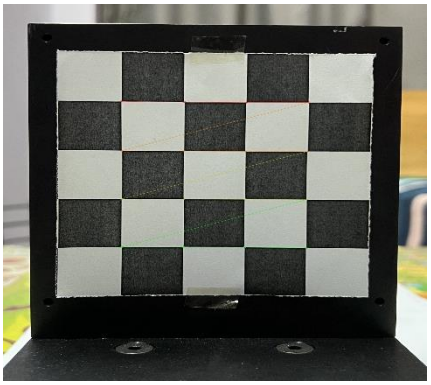
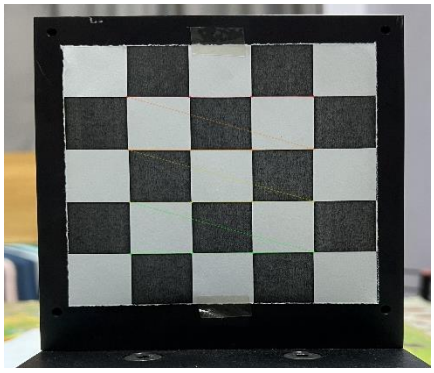


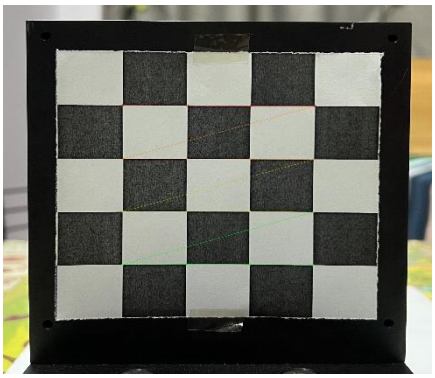
Figure 4.12: Corner Detected Output Images and Output Messages of Camera Parameter for (Camera's Height = 60, 70, 80cm)



```
Calibration starts
Successfully saved
 Camera matrix:
[[9.81125790e+02 0.00000000e+00 1.56005648e+03]
 [0.00000000e+00 9.76287790e+02 2.04300140e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

 Distortion coefficient:
[[ 0.06871189 -0.11244048  0.00180465  0.00228344  0.10635644]]

 Rotation Vectors:
(array([[0.01275824],
       [0.01175739],
       [0.00970087]]),)

 Translation Vectors:
(array([[-1.82479965],
       [-1.83098921],
```

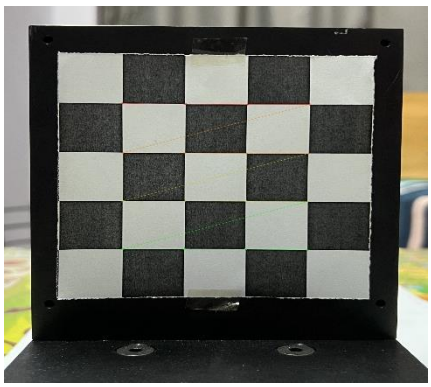


```
Calibration starts
Successfully saved
 Camera matrix:
[[1.04117450e+04 0.00000000e+00 1.53459078e+03]
 [0.00000000e+00 1.04570231e+04 1.98913196e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

 Distortion coefficient:
[[ 5.28646701e+00 -3.38195350e+02 -2.79498974e-03  2.11255999e-02
  -3.09827815e+00]]

 Rotation Vectors:
(array([[ 0.12818115],
       [-0.01620129],
       [ 0.00654293]]),)

 Translation Vectors:
(array([[-1.59538449],
       [-1.91624768],
       [35.43842453]]),)
```



```
Calibration starts
Successfully saved
 Camera matrix:
[[5.52190947e+03 0.00000000e+00 1.54596264e+03]
 [0.00000000e+00 5.51561871e+03 1.98930725e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

 Distortion coefficient:
[[ 1.65165376e+00 -3.71837921e+01  1.80575451e-03  1.41557056e-02
   7.14866119e+01]]

 Rotation Vectors:
(array([[ 0.08731363],
       [-0.03181709],
       [ 0.00689236]]),)

 Translation Vectors:
(array([[-1.6397952 ],
       [-1.58906337],
       [18.70488218]]),)
```

Figure 4.13: Corner Detected Output Images and Output Messages of Camera Parameter for (Camera's Distance = 20, 25, 30cm)



```
Calibration starts
Successfully saved
 Camera matrix:
[[9.81125790e+02 0.00000000e+00 1.56005648e+03]
 [0.00000000e+00 9.76287790e+02 2.04300140e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

 Distortion coefficient:
[[ 0.06871189 -0.11244048  0.00180465  0.00228344  0.10635644]]

 Rotation Vectors:
(array([[0.01275824],
       [0.01175739],
       [0.00970087]]),)

 Translation Vectors:
(array([[-1.82479965],
       [-1.83098921],
```

Figure 4.14: Corner Detected Output Image and Output Messages of Camera Parameter for (Camera's Height = 60cm Camera's Distance = 20cm)

4.1.1.3 Camera Calibration (Calculations)

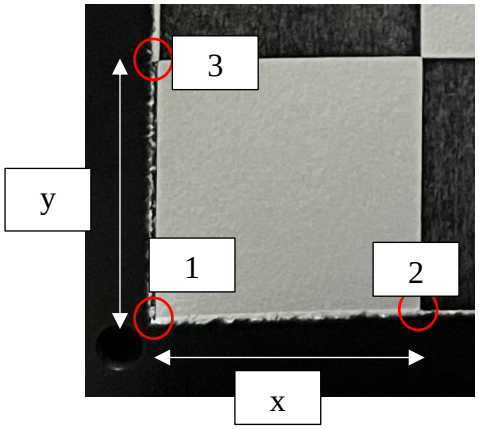


Figure 4.15: Enlarge of Selected Checkerboard Image's Corner.

After checkerboard image is selected, the image is opened with "Paint" software. Then, the pixel value for three of the red points as labelled in **Figure 4.15** are obtained and recorded in **Table 4.3**.

Table 4.3: Pixel Value for Three Red Points.

Point	[x,y]
1	[732,2847] pixels
2	[1035, 2832] pixels
3	[727, 2540] pixels

Since one particular box of the checkerboard is 20mm x 20mm, calculations below are performed.

$$x = \frac{1\ pixel}{(1035 - 732)\ pixels} x\ 20mm = 0.066mm = 66\ micron$$

$$y = \frac{1\ pixel}{(2847 - 2540)\ pixels} x\ 20mm = 0.065mm = 65\ micron$$

Field of View, *'fov'*:

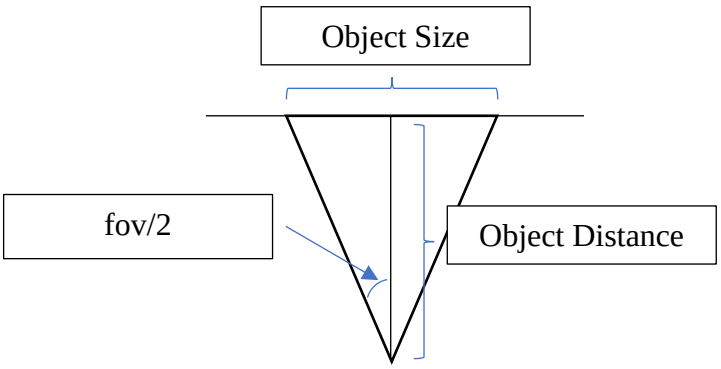


Figure 4.16: Field of View.

$$\tan\theta = \frac{10/2cm}{1cm}$$

$$\theta = 78.69°$$

$$fov = 78.69° \times 2 = 157.38°$$

4.2 Software Testing

In this section, image processing-based software and deep learning-based software are used during the software testing process to test and examine the performance and operation of the installed software to ascertain whether the software can get the desired results.

4.2.1 Image Processing Based Software Simulation

In this project, the defect detection system for MLAs was written in Python language using PyCharm software. The Python code is listed in **Appendix D**. As previously stated, 10 images are captured for each type of MLA with 1 reference image used to make comparison among defects MLAs. Based on **Figure 4.17**, the selection

of loaded image is specified to PNG file type. This is to make sure that the loaded image and the reference image are in the same file type.

```
#######################    Select File    ##############################
root = Tk()
root.filename = filedialog.askopenfilename(initialdir = "/",title = "Select file",filetypes = (("png files","*.png"),("all files","*.*")))
print (root.filename) #returns file path
root.filename_2 = filedialog.askopenfilename(initialdir = "/",title = "Select file",filetypes = (("png files","*.png"),("all files","*.*")))
print (root.filename_2) #returns file path
```

Figure 4.17: Code Snippet for Image Selection.

After test image is loaded, the path of both of the reference image and test image are shown in the output messages based on **Figure 4.18**.

```
Reference image path: C:/Users/YY/Downloads/new/Reference.png
Test image path: C:/Users/YY/Downloads/new/Reference_Flip.png
```

Figure 4.18: Output Messages for Path of Both Images.

Next, the test image is proceeded to image pre-processing, where formatting actions colour adjustment and noise removal are taken as shown in **Figure 4.19.** In skimage, the Gaussian blur is a low-pass filtering technique. It is frequently employed to eliminate Gaussian noise, or random noise from the image, which is essential to the proposed system because when processing an image, edges or the lines that signify a change from one group of related pixels in the image to another dissimilar group, are a crucial idea in the identification of objects in an image. When an image is implemented with Gaussian blur, the rapid fluctuations in pixel intensity is averaged. to assess the focusing and imaging capabilities of the MLA, an optical imaging experiment was developed.

```
# Apply GaussianBlur to reduce image noise
image1 = cv2.GaussianBlur(img, (5, 5), 0)
image2 = cv2.GaussianBlur(img2, (5, 5), 0)
```

Figure 4.19: GaussianBlur() Function.

From the code snippets in **Figure 4.20**, the syntax of GaussianBlur() function is shown.

```
dst = cv2.GaussianBlur(src, ksize,signmaX,signmaY)
```

Figure 4.20: GaussianBlur() Function.

By comparing the statement in **Figure 4.20**, the parameters are explained. The output image is represented by *'dst'* and the input image is represented by *'src'*. Furthermore, *'ksize'* is the Gaussian Kernel Size. Hence, when the *'ksize'* is set to [5 5], it means the height and width of the Gaussian Kernel size is 5 respectively. Lastly, *'signmaX'* and *'sigmaY'* represents the Kernel standard deviation along x-axis and y-axis.

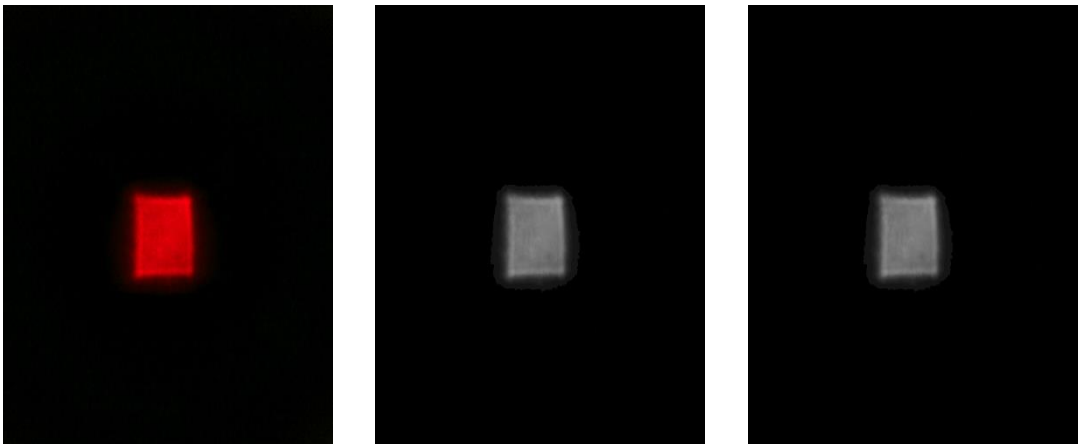


Figure 4.21: Input Image, Gaussian Blur Applied Image [5 5] and Gaussian Blur Applied Image [7 7] (Left to Right)

From **Figure 4.21**, the input image is loaded into the system and the Gaussian kernel size is changed. When the Kernel size is increased from (5 5) to (7 7), the image is becoming blur. This is because a larger kernel will blur the image more than a smaller kernel since a larger kernel has more values averaged into it.

Next, after pre-processing is done, the information of the test image is generated. From **Figure 4.22**, the image type, image shape, image dimensions, maximum RGB value and the center of the image is localised.

```
Type of the image for image 1: <class 'numpy.ndarray'> Type of the image for image 2: <class 'numpy.ndarray'>

Shape of image 1: (3968, 2976) Shape of image 2 : (3968, 2976)
height of image 1 3968 pixels height of image 2 3968 pixels
width of image 1 2976 pixels width of image 2 2976 pixels
Dimension of Image 1 2 Dimension of Image 2 2
Image size of image 1 11808768 Image size of image 211808768
Maximum RGB value in image 1 147 Maximum RGB value in image 2 147
Minimum RGB value in image 1 0 Minimum RGB value in image 2 0
Center of the image (x) {} 1488
Center of the image (y) {} 1984
```

Figure 4.22: Output Messages for Information of Images.

It is important to understand the pixel information of an image because each pixel value specifies its brightness and intended colour. Without the information extracted, proper image processing on light field image is limited.

Table 4.4: Extracted Information.

Information	Reference image	Test image
Image type	Numpy.ndarray	Numpy.ndarray
Image shape	3968,2976	3968,2976
Image dimension	2	2
Image Size	11808768	11808768
Maximum RGB value	147	147
Center	1488, 1984	1488, 1984

From **Table 4.4**, the information extracted are the same for both of the reference image and test image. The ‘numpy.ndarray’ which is the primary data structure in NumPy is a shortened form for N-dimensional array. Data in a ‘numpy.ndarray’ is simply referred to as an array when using NumPy. It is a memory array with a fixed size that holds data of the same type, like integers or floating-point numbers. Since, images are kept in ‘numpy.ndarray’, ‘ndarray. Shape’ is used to obtain the image's size and dimensions. Moreover, since both of the data are converted to grayscale, their channels equal to 2. This is because 1 channel existed in a monochromatic image with one number per pixel. Two channels represented a

grayscale image and three channels are presented in a more typical image with three (R, G, and B) integers per pixel.

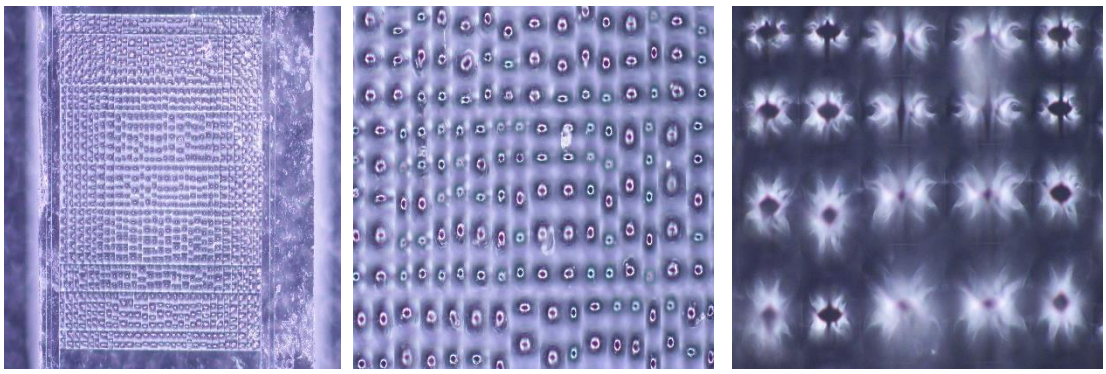


Figure 4.23: Optical Image Taken with Microscope (x10, x20,x50) (Left to Right)

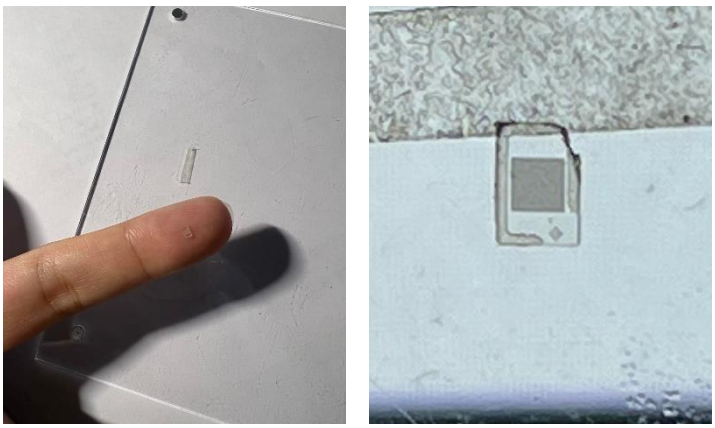


Figure 4.24: Image Taken with Phone Camera (Left to Right)

For investigation, the optical image for the Reference MLA in microscope magnification of x10, x20 and x50 are captured and showed in **Figure 4.23**. The reference MLA is focussed at the focal plane so that its focal point image is obtained. Although these images are not necessary for the proposed system but comparison on the observation among optical image and naked eyes shown in **Figure 4.24** are beneficial.

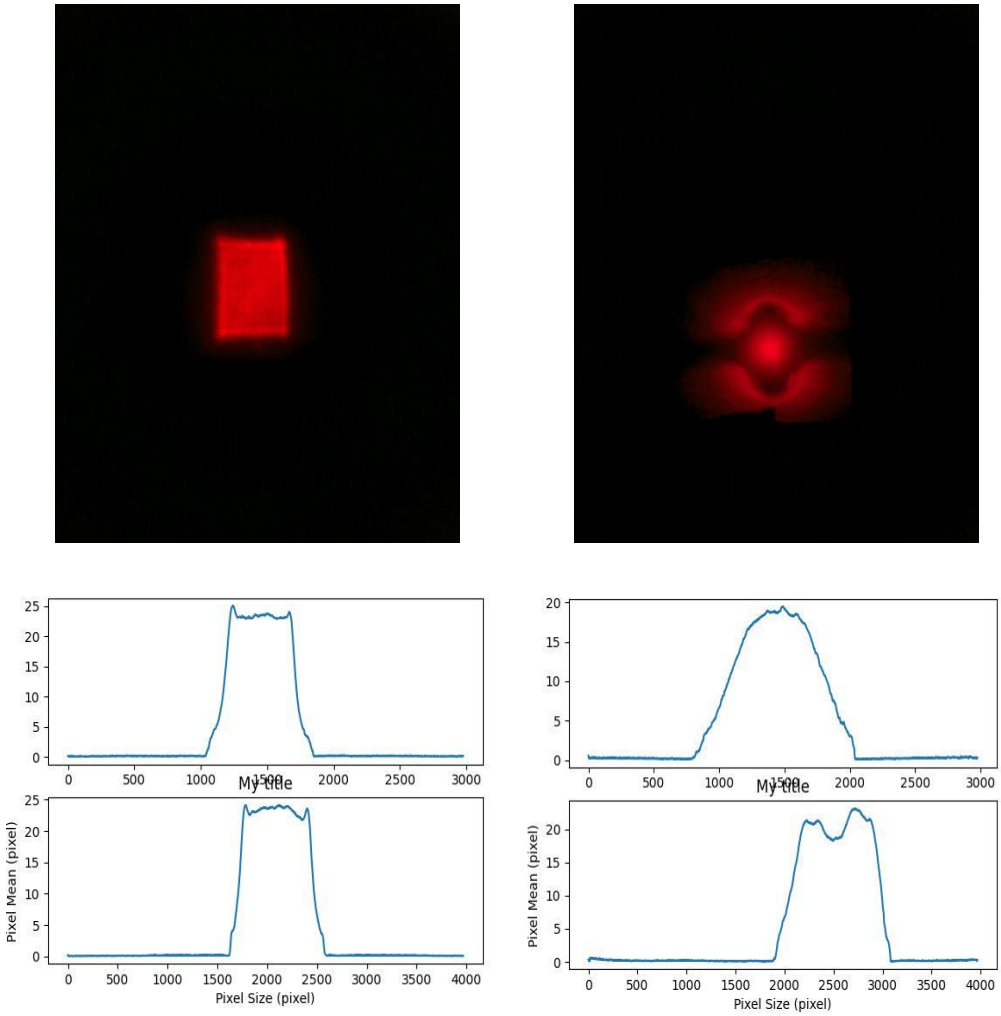


Figure 4.25: Laser Beam Profile and Intensity Distribution for Reference MLA and Reference Flipped MLA (Left to Right)

Based on **Figure 4.25**, the laser beam profile and the intensity distributions of reference MLA and flipped reference MLA from hardware setup are shown. For the reference image, the center axis of the emerging beam is perpendicular to the target plane as the beam energy is collected at the target plane's middle and formed a Gaussian-like energy distribution. For the test image, a peak and a sag are observed in the intensity distributions for line scan of image from left to right (x-direction) and line-scan from up to bottom (y-direction).

```
Pixel mean value at the center for image 1: 127
Pixel mean value at the center for image 1: 28
********************Comparison Status**********************
               Both image have the same size
=                            FAIL
```

Figure 4.26: Output Messages for System.

Additionally, localization of the image pixel means' value at the center of both images are performed to further confirm the comparison results. Hence, the outputs shown in **Figure 4.26** with pixel mean value at the center for image 1 and image 2 equals to 127 and 28 respectively. Since the center point is the most sensitive parts to detect the intensity distributions, so comparison of pixel mean value are also performed at the center of image. The comparison results are presented as "FAIL" because the pixel mean value of flipped MLA and the reference image are not identical.

In order to investigate on more data images, the reference MLA is scratched and the laser beam profile is observed. Based on **Figure 4.27**, the scratch is appearing as a white spot in the center of the laser beam profile.

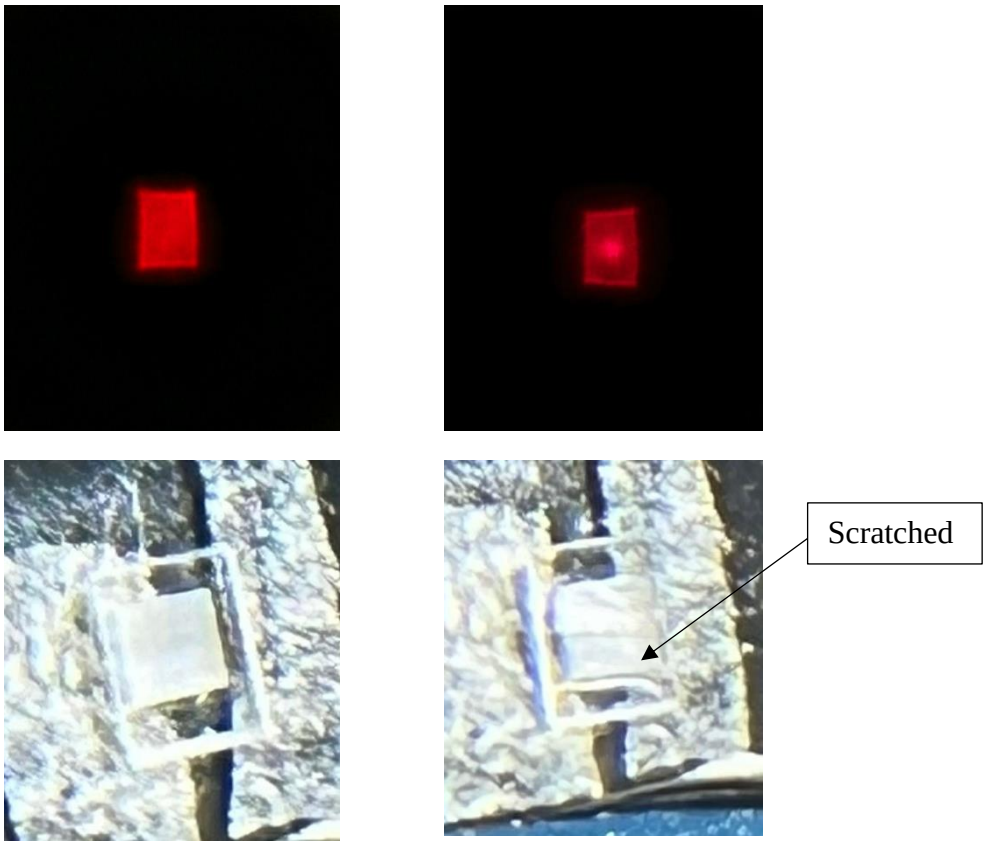
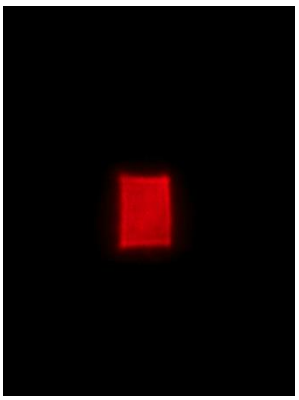
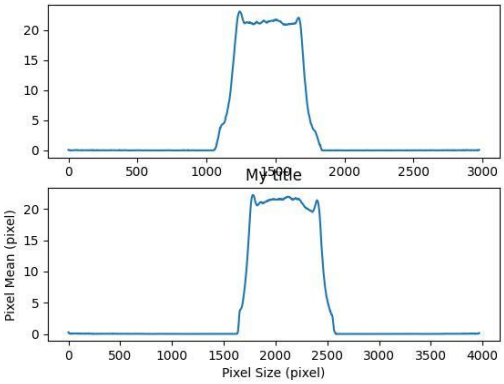
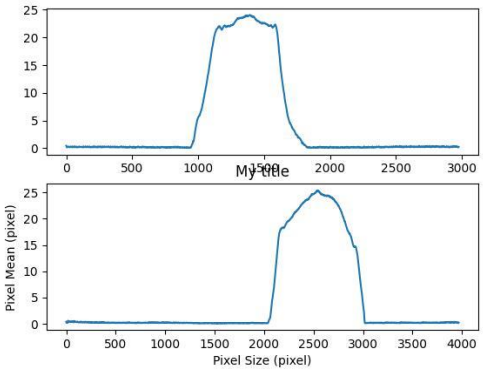
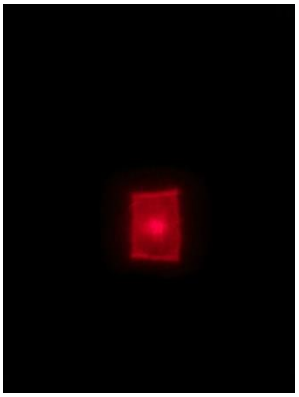
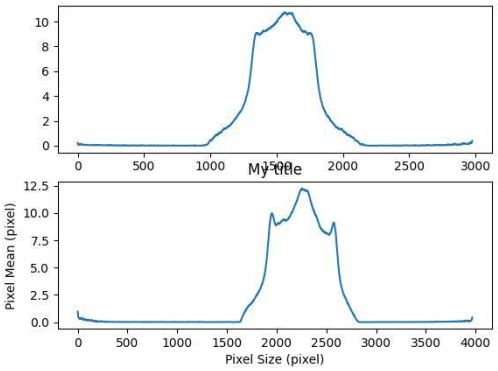
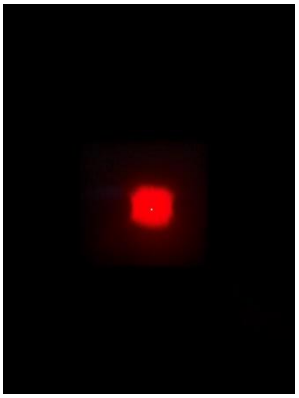
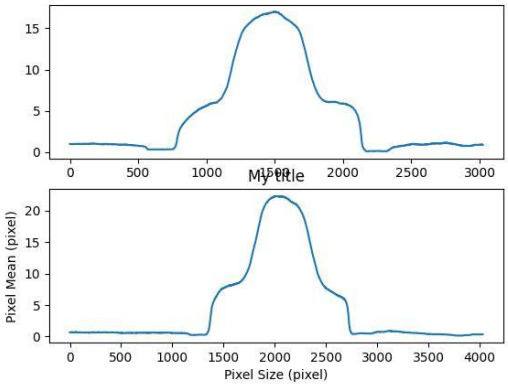
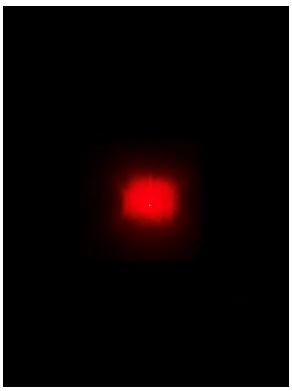
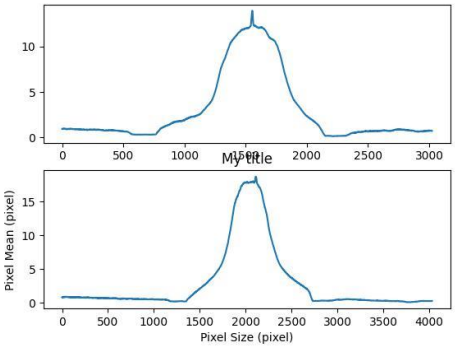


Figure 4.27: Laser Beam Profile and Image Taken for Reference MLA and Reference Scratched MLA (Left to Right)

Based on **Table 4.5**, the intensity distributions and status for MLAs are visualised.

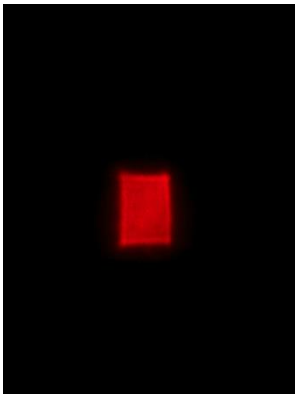
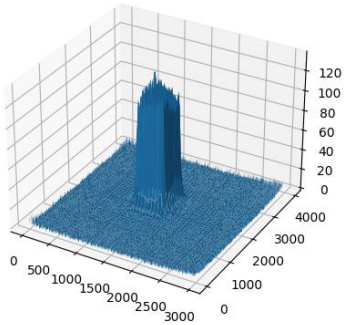
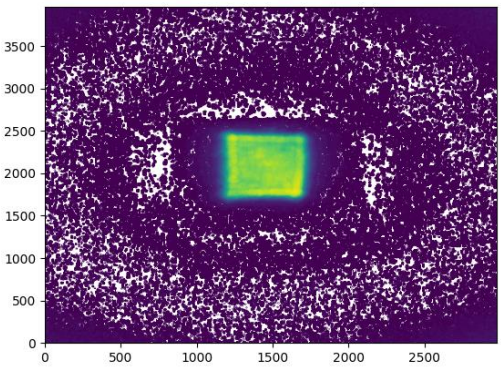
Table 4.5: Summary of Surface Plot and Contour Plot of the Pixel Intensity.

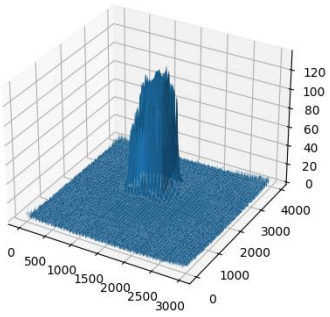
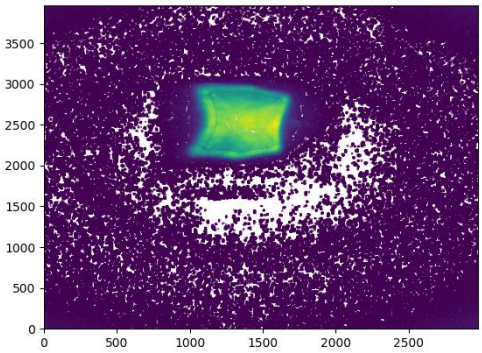
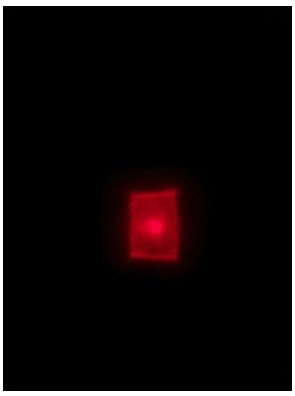
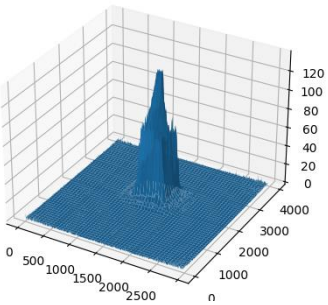
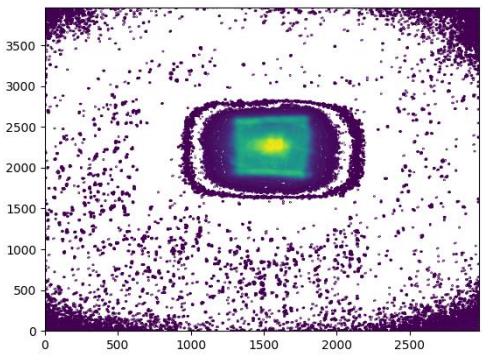
Laser beam profile	Intensity Distributions and Status
Reference MLA 	 **Pass**
Distorted Reference MLA 	 **Fail**
Scratched MLA 	 **Fail**

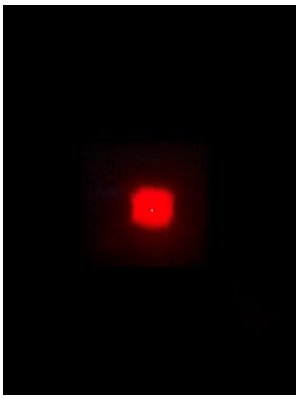
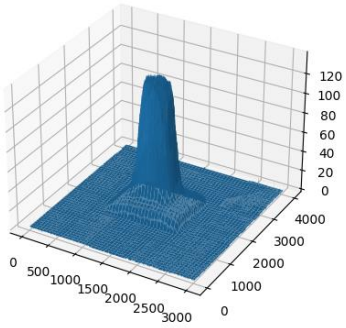
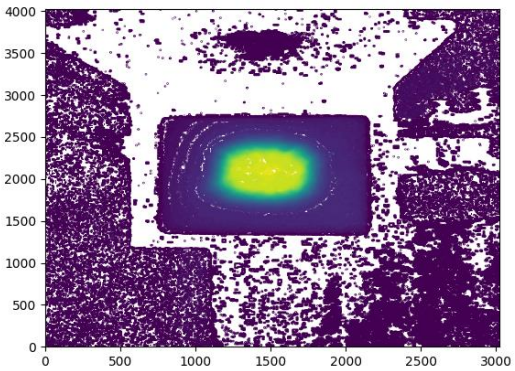
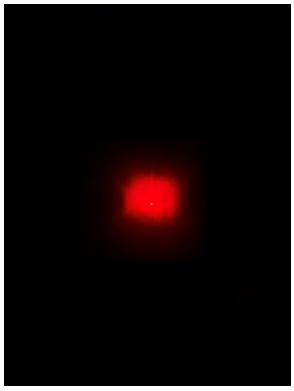
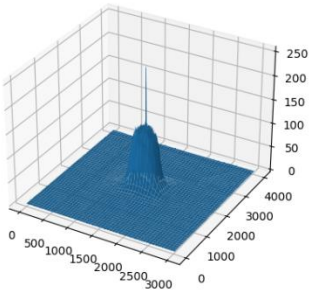
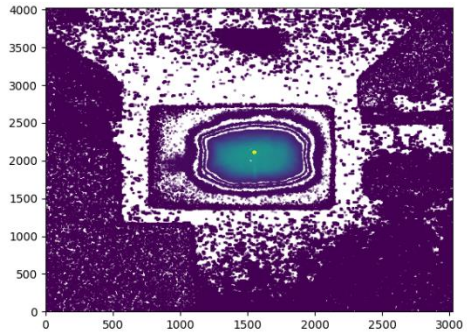
Narrow MLA 	 **Pass**
Distorted Narrow MLA 	 **Fail**

Based on **Table 4.6**, the surface plot and contour plot of the pixel intensity for MLAs are visualised. The Python code for surface plot and contour plot with Qt Designer .UI code is attached in **Appendix F** and **Appendix G** respectively.

Table 4.6: Summary of Surface Plot and Contour Plot of the Pixel Intensity.

Laser beam profile	Surface Plot and Contour Plot
Reference MLA 	 

Distorted Reference MLA 	 
Scratched Reference MLA 	 

Narrow MLA 	 
Narrow Distorted MLA 	 

4.2.2 Deep Learning Based Software Simulation

The proposed defect detection system for MLAs is based on the architecture of one-shot learning, Siamese Neural Network shown in **Figure 4.28**.

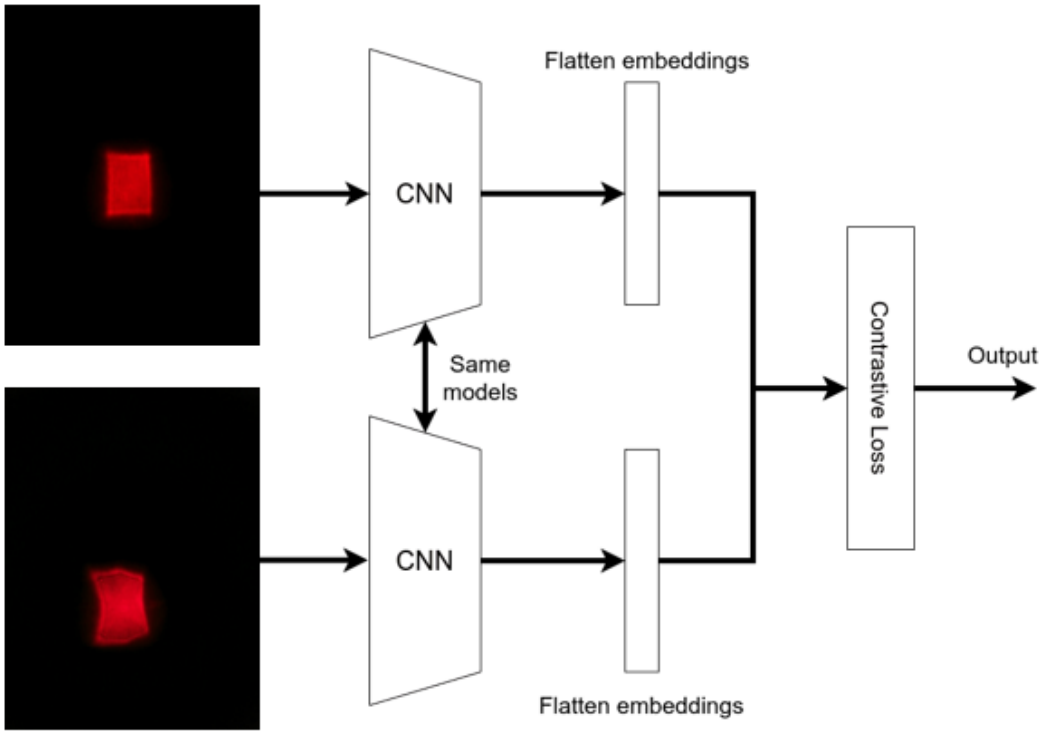


Figure 4.28: Architecture of Siamese Neural Network.

All of the steps are conducted on Google Colaboratoy with built in GPU. The command shown in **Figure 4.29** is used to check whether the GPU is enabled. With the output messages shown below reverted, GPU is enabled.

```
import tensorflow as tf
tf.test.gpu_device_name()
```

```
'/device:GPU:0'
```

Figure 4.29: Code Snippet of GPU Check

The Siamese Neural Network model are trained on the MLAs dataset previously described and are divided into three parts whereby 69% for training, 20%

for validation and 12% for testing. Each of the dataset have pairs of values *'Xi'* and *'Yi'*, where *'Xi'* represents the input and *'Yi'* represents the result. Hence, the system is recalled that the input is a pair of images and the result is a similarity score between 0 and 1. Based on **Table 4.7**, the explanation above is visualised.

Table 4.7 Summary of Siamese Neural Network System

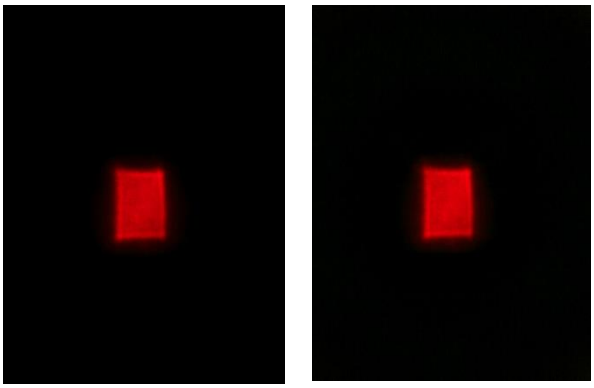
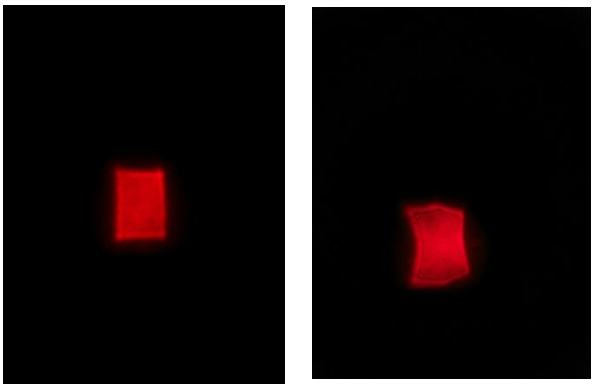
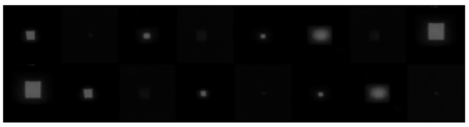
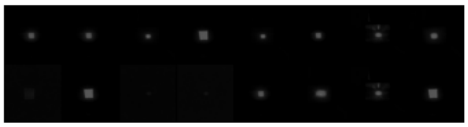
Pairs	Labels
	1 (same)
	0 (not same)

Figure 4.30 showed the visualization of outcome before training of Siamese Neural Network are processed. It is observed that 8 sets of images are randomly selected to perform similarity compare. The final results showed [0 1 1 0 0 0 1 1] where "0" indicates that the compare set are same and "1" indicates that the compare set are not same. Hence, from the messages generated, 4 out of 8 sets indicates same, "1". The simple dataloader is run for three times and the output obtained are still the same where the "0" and "1" are evenly distributed among 8 datasets. The only changes is just that the images used for comparison.

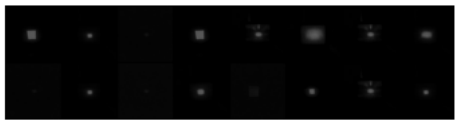
```
[10] # Create a simple dataloader just for simple visualization
     vis_dataloader = DataLoader(siamese_dataset,
                             shuffle=True,
                             num_workers=2,
                             batch_size=8)

     # Extract one batch
     example_batch = next(iter(vis_dataloader))

     # Example batch is a list containing 2x8 images, indexes 0 and 1, an also the label
     # If the label is 1, it means that it is not the same , label is 0, same
     concatenated = torch.cat((example_batch[0], example_batch[1]),0)

     imshow(torchvision.utils.make_grid(concatenated))
     print(example_batch[2].numpy().reshape(-1))
```



```
[0. 1. 1. 0. 0. 0. 1. 1.]
```



```
[0. 0. 0. 1. 1. 1. 0. 1.]
```



```
[1. 0. 0. 1. 1. 1. 0. 0.]
```

Figure 4.30: Output of Random Data Images.

As a result, in order to feed the Siamese Network as input, pairs of images together with the target variable, as illustrated in **Figure 4.30** are created. Although the MLAs light field images are depicted here, pairs are randomly allocated from all the datasets in the training set.

The CNN Siamese model with contrastive loss function are trained using two identical CNN models and the output was input into a contrastive loss function to calculated the loss value. Finally, two images were fed into the CNN Siamese models to test for similarity. Both tests were run using the identical set of 25 pairs of MLA light field images and the EPOCH values are set to 100 to determine the training loss. The outcome is recorded in **Table 4.8.**

Table 4.8: Training Loss for 100 Epochs.

Epoch	Training Loss
10	1.443
20	1.247
30	1.332
40	1.607
50	1.523
60	0.557
70	1.694
80	0.910
90	0.564
100	**0.292**

Based on **Figure 4.31**, the plot of training loss which is the indicator of how well Siamese Neural Network model fits the training data is shown. In other words, the model's error on the training set is evaluated. When the epoch values are set to 100, the training loss dropped from 1.443 to 0.292. The training results obtained are consider good because the lower the loss, the better the model is trained.

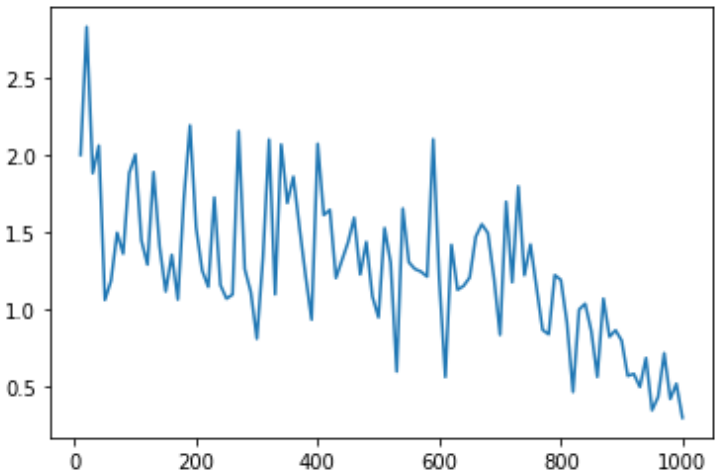


Figure 4.31: Plot of Training Loss (Epochs=100)

The training loss is obtained and recorded in **Table 4.9**, when the epoch value is increased to 200. From **Table 4.8**, the training loss is 1.018 which is not as good as epoch 100. This is because overfitting of the training dataset occurred when there are

too many epochs. The system should terminate training as soon as the model performance is sufficient for the validation dataset.

Table 4.9: Training Loss for 200 Epochs.

Epoch	Training Loss
10	0.766
20	1.540
30	1.915
40	1.132
50	2.029
60	1.776
70	1.256
80	1.471
90	1.153
100	1.643
110	0.975
120	1.258
130	1.394
140	1.564
150	1.973
160	1.682
170	1.208
180	1.478
190	0.729
200	**1.018**

Based on **Figure 4.32**, the plot of training loss is shown. When the epoch values are set to 200, the training loss increased from 0.766 to 1.018. The training results obtained are not as good as epoch 100 due to overfitting.

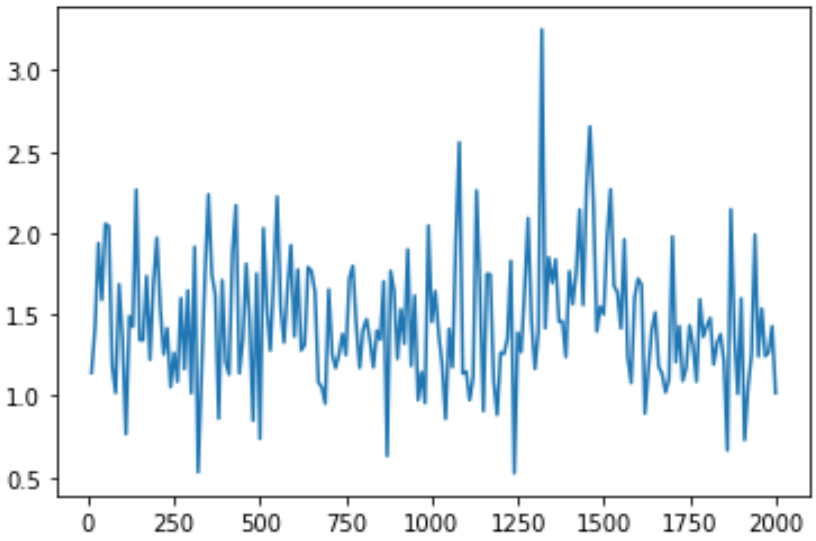


Figure 4.32: Plot of Training Loss (Epochs=200)

The training loss is obtained and recorded in **Table 4.10,** when the epoch value is decreased to 50. From **Table 4.10**, the training loss is 1.412 which is worsen as compared to epoch 200. This is because underfitting occurred when there are too few epochs.

Table 4.10: Training Loss for 50 Epochs.

Epoch	Training Loss
10	2.097
20	1.551
30	1.096
40	1.098
50	**1.412**

Based on **Figure 4.33**, the plot of training loss is shown. When the epoch values are set to 50, the training loss increased from 1.349 to 1.412. The training results obtained are bad as compared to epoch 100 and epoch 200 due to underfitting.

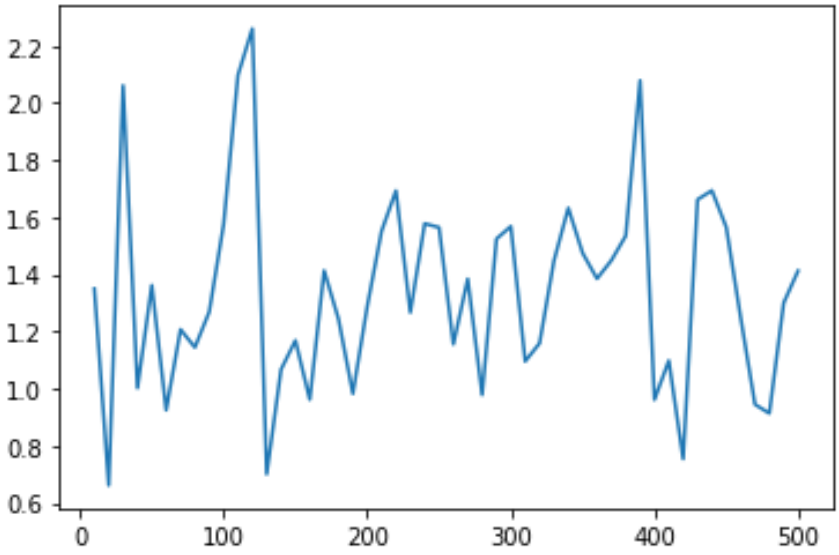


Figure 4.33: Plot of Training Loss (Epochs=50)

With model trained on 100 epochs selected, validation and testing of the trained model is performed. Based on **Figure 4.34**, the similarity score is generated for each of the image pairs. The higher the dissimilarity value, the greater the differences between two images.

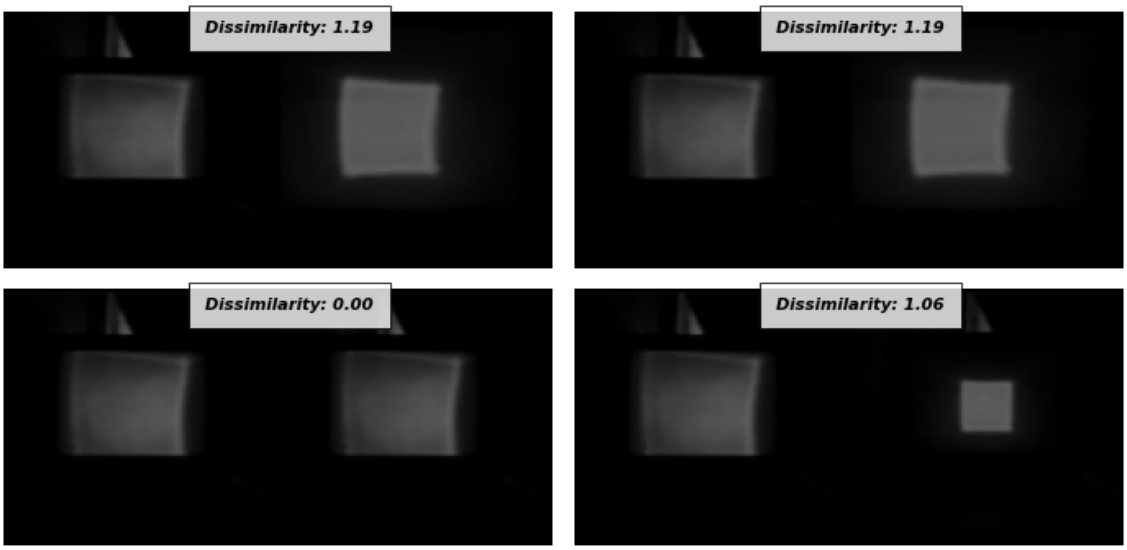


Figure 4.34: Similarity Score for Each Image Pairs.

4.3 Comparison Between Two Proposed Method

In a nutshell, comparison between two of the image processing-based system and deep learning-based system are performed. In particular, deep learning is frequently used than computer vision in the future. With the aid of deep learning, tasks like image classification, semantic segmentation, object identification, and simultaneous localization and mapping with increased its precision. All in that, more researches between deep learning and image processing are needed to figure out a method that can combine two of the approaches.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusion

The project “Image Processing and Image Analysis of Defects in Microlens Array Using Smartphone Camera and Application” which aims to study the light field imaging of MLAs captured with smartphone camera and developing an application to detect defects of MLAs has been conducted successfully.

In the camera calibration system, the total error was around 0.0165 pixel which makes a good camera calibration. This is because a good camera calibration has a final total error of less than 1.0 and as low as below 0.4 for a strong camera calibration.

In the image-based defects detection system for MLAs, the light field images of MLAs are detected and classified to “PASS” or “FAIL” based on the analysis on intensity distributions, contour plot and 3D surface plot.

In deep learning defects detection system for MLAs, the Siamese Neural Network model is trained, validated and tested based on datasets collected. Since the number of datasets collected are small, the Siamese network is selected as single training example is all that is needed for the one-shot classification model to make a prediction. Last but not least, classification of the similarity of two images are performed with similarity percentage shown.

In a nutshell, despite that the proposed systems are not tested in the field, the system's performance was assessed, the findings were satisfactory. However, the system still has significant limitations, and those constraints need to be addressed, and some other improvements and alterations can be done. so that the technology can be used in the real world in the future.

5.2 Limitations and Recommendations

The limitations and recommendations of this system are as follows:

1. Hardware Setup

The alignments of equipment are important in obtaining precise results. Dictating the camera, MLA diffuser, MLA and light source have to be in parallel, a table that have the same height with camera tripod must be prepared. This is to make sure that no misalignment occurred.

2. Image Intensity

Intensity is the most important parameter in light field imaging system. The setup of smartphone camera has to be checked before laser beam profile are captured. This is because the images are taken under dark condition and this might turn on the night mode of smartphone camera. With this, the image taken will be in contrast with high intensity and leads to inaccurate results.

3. Size of MLA Diffuser

The size of MLA diffuser used in this project is 10cm x10cm. Although the size is sufficient for current datasets, the MLA diffuser might not be able to cover wide angle laser beam profile if other types of MLA are experimented. Furthermore, with small size MLA diffuser, there are not many options in adjusting the distance between MLA and MLA diffuser Hence, laser beam profile is small and might lead to inaccurate results.

4. Model Training of Siamese Network

Although Siamese Network are famous for training model with small datasets. Small dataset is difficult to keep out many samples, and the test data's number of observations may be too low to provide an accurate performance estimate. Furthermore, the risk of overfitting increases as training datasets get smaller since there are less examples for the models to learn from. Hence, different type of MLAs shall be bought to conduct the experiments so that more datasets can be obtained.

In a nutshell, in order to overcome the shortcomings noted and increase the robustness of defect detection system for MLAs to handle unexpected environment, improvements shall be done in the future to achieve a better result.

REFERENCES

Abookasis.D and Rosen.J, (2004). *Microlens Array Help Imaging Hidden Objects for Medical Applications.* Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems, 2004. ICECS 2004., 2004, pp. 431-434, doi: 10.1109/ICECS.2004.1399710.

Addoptics (2020). *The Benefits and Applications of Microlens Arrays*. [online] Addoptics. Available at: https://www.addoptics.nl/optics-explained/microlens-arrays/#:~:text=Microlens%20arrays%20are%20most%20often [Accessed 7 Sep. 2022].

Alan Conrad Bovik (2009). *The Essential Guide to Image Processing*. Amsterdam; Boston: Academic Press, Cop.

Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O.,Santamaría, J., Fadhel, M.A., Al-Amidie, M. and Farhan, L. (2021). *Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions.* Journal of Big Data, 8(1).

Analytics Vidhya. (2016). *Artificial Intelligence Demystified.* [online] Available at: https://www.analyticsvidhya.com/blog/2016/12/artificial-intelligence-demystified/. [Accessed 26 March. 2022].

Analytics Vidhya. (2017). *Master Transfer Learning by Using Pre-trained Models in Deep Learning.* [online] Available at: https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/.

Analytics India Magazine. (2018). Si*gnificance Of Transfer Learning in The World Of Deep Learning*. [online] Available at: https://analyticsindiamag.com/transfer-learning-deep-learning-significance/ [Accessed 26 March. 2022].

Anon, (2020). *OptiNspec - Automated Optics Inspection Machine*. [online] Available at: https://wavelength-oe.com/systems-and-software/optinspec/ [Accessed 10 Sep. 2022].

Apelt, F., Breuer, D., Nikoloski, Z., Stitt, M. and Kragler, F. (2015). *Phytotyping 4D: A Light-Field Imaging System for Non-Invasive and Accurate Monitoring of Spatio Temporal Plant Growth.* The Plant Journal, 82(4), pp.693–706. doi:10.1111/tpj.12833.

Bovik, A.C. and Acton, S.T. (2009). *Basic Linear Filtering with Application to Image Enhancement.* The Essential Guide to Image Processing, pp.225–239. doi:10.1016/b978-0-12-374457-9.00010-x.

Christopher Hahne, Amar Aggoun, Vladan Velisavljevic, Susanne Fiebig, Matthias Pesch. (2016). *Figshare.com.* [online] Available at: https://figshare.com/articles/dataset/Raw_image_data_taken_by_a_standard_plenoptic_camera/3362152 [Accessed 26 March. 2022].

Cornell Chronicle. (n.d.). *Professor's Perceptron Paved the Way for AI – 60 Years* Too Soon. [online] Available at: https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon. [Accessed 26 March. 2022].

datahacker.rs (2018). *#006 CNN Convolution On RGB Images*. [online] Master Data Science. Available at: https://datahacker.rs/convolution-rgb-image/ [Accessed 7 Sep. 2022].

Director, A.C. (2014). *Digital Video Surveillance and Security.* Elsevier Science & Technology.

docs.opencv.org. (n.d.). *OpenCV: Camera Calibration.* [online] Available at: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html. [Accessed 26 March. 2022].

Donges, N. (2019). *Recurrent Neural Networks 101: Understanding the Basics of* RNNs and LSTM. [online] Built In. Available at: https://builtin.com/data-science/recurrent-neural-networks-and-lstm. [Accessed 26 March. 2022].

EDITOR, M.F., CONTRIBUTING (n.d.). *Inspecting for Lens Defects Only Scratches* the Surface. [online] www.photonics.com. Available at: https://www.photonics.com/Articles/Inspecting_for_Lens_Defects_Only_SScratche_the/a67214 [Accessed 10 Sep. 2022].

GeeksforGeeks. (2019). *Difference Between Image Processing and Computer Vision.* [online] Available at: https://www.geeksforgeeks.org/difference-between-image-processing-and-computer-vision/. [Accessed 26 March. 2022]

GeeksforGeeks. (2022). *Types of Recurrent Neural Networks (RNN) in Tensorflow.* [online] Available at: https://www.geeksforgeeks.org/types-of-recurrent-neural-networks-rnn-in-tensorflow/ [Accessed 7 Sep. 2022].

Gershun A (1939) *The Light Field.* J Math Phys 18(1–4):51 151. https://doi.org/10.1002/sapm193918151

Georgeiv, T. and Intwala, C. (2006). *Light Field Camera Design for Integral View Photography.* [online] www.semanticscholar.org. Available at: https://www.semanticscholar.org/paper/Light-Field-Camera-Design-for-Integral View-Georgeiv-Intwala/b2bd28f56a84807f577cc54e7e23c4d63e2a42e8 [Accessed 7 Sep. 2022].

Guenther, R.D. (2005). *Encyclopaedia of Modern Optics*. 1, A - F. Amsterdam: Elsevier Acad. Press.

Haidar, L. (2021). *Sobel vs. Canny Edge Detection Techniques: Step by Step Implementation.* [online] Medium. Available at: https://medium.com/@haidarlina4/sobel-vs-canny-edge-detection-techniques-step-by-step-implementation-11ae6103a56a [Accessed 10 Sep. 2022].

HAN, L. (2008). Object detection module based on implementation of Java and OpenCV. Journal of Computer Applications, 28(3), pp.773–775. doi:10.3724/sp.j.1087.2008.00773.

ilp.mit.edu. (n.d.). *A Wide-Angle View on Micro-Optics Innovation | ILP*. [online] Available at: https://ilp.mit.edu/node/48940 [Accessed 10 Sep. 2022].

Infopulse. (n.d.). *The Executive Guide to Neural Networks and Deep Learning for Businesses.* [online] Available at: https://www.infopulse.com/blog/the-executive-guide-to-neural-networks-and-deep-learning-for-businesses [Accessed 26 March. 2022].

Information Resources Management Association (2013). *Image Processing: Concepts, Methodologies, Tools, and Applications.* Hershey, Pa: Information Science Reference.

Jason Brownlee. (2017). *A Gentle Introduction to Transfer Learning for Deep Learning.* [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/transfer-learning-for-deep-learning/. [Accessed 26 March. 2022]

Joshi, M., Tech Scholar, M., Vyas Head, A. and Program, P. (n.d.). *Comparison of Canny Edge Detector With Sobel and Prewitt Edge Detector Using Different Image Formats.* [online] Available at: https://www.ijert.org/research/comparison-of-canny-edge-detector-with-sobel-and-prewitt-edge-detector-using-different-image-formats-IJERTCONV2IS03009.pdf. [Accessed 26 March. 2022]

Kim, S., Ban, Y. and Lee, S. (2014). *Face Liveness Detection Using a Light Field Camera. Sensors*, 14(12), pp.22471–22499. doi:10.3390/s141222471.

Krishnan, M. (2020). *Otsu's Method for Image Thresholding Explained and Implemented.* [online] Muthukrishnan. Available at: https://muthu.co/otsus-method-for-image-thresholding-explained-and-implemented/. [Accessed 26 March. 2022]

Lamba, H. (2019). *One Shot Learning with Siamese Networks using Keras.* [online] Medium. Available at: https://towardsdatascience.com/one-shot-learning-with-siamese-networks-using-keras-17f34e75bb3d. [Accessed 26 March. 2022]

Li, Y., Li, K. and Gong, F. (2021). *Fabrication and Optical Characterization of Polymeric Aspherical Microlens Array Using Hot Embossing Technology*. Applied Sciences, 11(2), p.882.

Li, S.-N., Yuan, Y., Liu, B., Wang, F.-Q. and Tan, H.-P. (2018). *Local Error and Its Identification for Microlens Array in Plenoptic Camera. Optics and Lasers in Engineering*, 108, pp.41–53. doi:10.1016/j.optlaseng.2018.04.017.

lightfields.stanford.edu. (n.d.). *Stanford Lytro Light Field Archive*. [online] Available at: http://lightfields.stanford.edu/LF2016.html [Accessed 26 March. 2022].

Liu, T., Bao, J., Wang, J. and Wang, J. (2021). *Deep Learning for Industrial Image: Challenges, Methods for Enriching the Sample Space and Restricting the Hypothesis Space, and Possible Issue.* International Journal of Computer Integrated Manufacturing, pp.1–30.

Ltd, A. (n.d.). *What is Computer Vision.* [online] Arm | The Architecture for the Digital World. Available at: https://www.arm.com/glossary/computer-vision [Accessed 7 Sep. 2022].

Mandal, M. (2021). *CNN for Deep Learning | Convolutional Neural Networks (CNN).* [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/. [Accessed 26 March. 2022]

MathWorks. (n.d.). *Edge Detection.* [online] Available at: https://www.mathworks.com/discovery/edgedetection.html#:~:text=Edge%20detection%20is%20an%20image. [Accessed 26 March. 2022]

Mathworks.com. (2019). *VisibleBreadcrumbs*. [online] Available at: https://www.mathworks.com/help/vision/ug/camera-calibration.html. [Accessed 26 March. 2022]

Nussbaum, P., Völkel, R., Herzig, H.P., Eisner, M. and Haselbeck, S. (1997). *Design, Fabrication and Testing of Microlens Arrays for Sensors and Microsystems.* Pure and Applied Optics: Journal of the European Optical Society Part A, 6(6), pp.617–636. doi:10.1088/0963-9659/6/6/004.

Otsu. (2007). [online] Available at: https://engineering.purdue.edu/kak/computervision/ECE661.08/OTSU_paper.pdf [Accessed 26 March. 2022].

O'Neil, D. (2022). *New Light Field Camera Can Focus Up-Close and Far Away Simultaneously.* [online] PetaPixel. Available at: https://petapixel.com/2022/04/27/new-light-field-camera-can-focus-up-close-and far-away-simultaneously/ [Accessed 7 Sep. 2022].

Ox.ac.uk. (2014). *Visual Geometry Group - University of Oxford.* [online] Available at: https://www.robots.ox.ac.uk/~vgg/research/very_deep/. [Accessed 26 March. 2022]

Park, M.-K., Lee, H.J., Park, J.-S., Kim, M., Bae, J.M., Mahmud, I. and Kim, H.-R. (2014). *Design and Fabrication of Multi-Focusing Microlens Array with Different Numerical Apertures by using Thermal Reflow Method*. Journal of the Optical Society of Korea, 18(1), pp.71–77.

Perwass, C. and Wietzke, L. (2012). *Single lens 3D-camera with extended depth-of field. SPIE Proceedings.* doi:10.1117/12.909882. www.tgeorgiev.net. (n.d.). Todor Georgiev - Photoshop - Adobe. [online] Available at: http://www.tgeorgiev.net/Lippmann/ [Accessed 7 Sep. 2022].

Purakkatt, A. (2020). *Deep Learning — Artificial Neural Network (ANN). [online] Medium.* Available at: https://medium.com/analytics-vidhya/deep-learning-artificial-neural-network-ann-13b54c3f370f. [Accessed 26 March. 2022]

Reichelt, S., Bieber, A., Aatz, B. and Zappe, H. (2005). *Micro-Optics Metrology Using Advanced Interferometry*. SPIE Proceedings. doi:10.1117/12.612584.

R. Paschotta, "*Beam Quality Deterioration of Lasers Caused by Intracavity Beam Distortions*", Optics Express 14 (13), 6069 (2006)

Schambach, M. and Puente Leon, F.P. (2020). *Microlens Array Grid Estimation, Light Field Decoding, and Calibration.* IEEE Transactions on Computational Imaging, 6, pp.591–603. doi:10.1109/tci.2020.2964257.

Schelkens, P., Spie Europe, Alsace International, De, I. and Spie (Society (2008). *Optical and Digital Image Processing* : 7-9 April 2008, Strasbourg, France. Bellingham, Wash.: Spie, C.

Society, T.R. (2019). *Blob Detection*. [online] Medium. Available at: https://medium.com/image-processing-in-robotics/blob-detection-309226a3ea5b. [Accessed 26 March. 2022]

Sonka, M. and J Michael Fitzpatrick (2009). *Handbook of Medical Imaging*. Bellingham: Spie Press.

Stevens, R. and Miyashita, T. (2010). *Review of Standards for Microlenses and Microlens Arrays*. The Imaging Science Journal, 58(4), pp.202–212. doi:10.1179/136821910x12651933390746.

Systems» Electronics Notes. [online] Available at: https://www.electronicsnotes.com/articles/test-methods/automatic-automated-test ate/aoi-optical-inspection.php. [Accessed 26 March. 2022]

Sheng, R. (2020). *Computer Vision and Image Processing.* [online] The Startup. Available at: https://medium.com/swlh/computer-vision-and-image-processing 470ceea06b91 [Accessed 26 March. 2022].

Theteche.com. (2021). *Image Processing - Digital and Analog Image Processing.* [online] Available at: https://theteche.com/image-processing-digital-and-analog-image-processing/ [Accessed 26 March. 2022].

Tsang, S.-H. (2020). *Review: AlexNet, CaffeNet — Winner of ILSVRC 2012 (Image Classification).* [online] Coinmonks. Available at: https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in ilsvrc-2012-image-classification-b93598314160. [Accessed 15 September. 2022].

upGrad blog. (2021). *Image Segmentation Techniques [Step by Step Implementation]*. [online] Available at: https://www.upgrad.com/blog/image-segmentation-techniques/. [Accessed 26 March. 2022]

VCTA (2019). *Price of Automated Optical Inspection (AOI) System*. [online] ZhenHuaXing. Available at: https://aoi-spi.com/automated-optical-inspection-aoi-price/ [Accessed 10 Sep. 2022].

www.allaboutcircuits.com. (n.d.). *Image Arithmetic in DSP: Image Averaging and Image Subtraction* - Technical Articles. [online] Available at: https://www.allaboutcircuits.com/technical-articles/arithmetic-operations-on-images-image-addition-averaging-and-subtraction/. [Accessed 26 March. 2022]

www.electronics-notes.com. (n.d.). *What is AOI: Automatic Optical Inspection*

www.irisa.fr. (2015). *Lytro Image Dataset.* [online] Available at: https://www.irisa.fr/temics/demos/lightField/index.html [Accessed 26 March. 2022].

www.irisa.fr. (n.d.). *Lytro Illum Dataset*. [online] Available at: https://www.irisa.fr/temics/demos/IllumDatasetLF/index.html [Accessed 26 March. 2022].

Wu, C., Ko, J. and Davis, C.C. (2016). *Imaging Through Strong Turbulence With a* Light Field Approach. Optics Express, 24(11), p.11975. doi:10.1364/oe.24.011975.

Yang, S.-W., Lin, C.-S., Fu, S.-H., Yeh, M.-S., Tsou, C. and Lai, T.-H. (2012). *Lens Sag Measurement of Microlens Array Using Optical Interferometric Microscope*. Optics Communications, 285(6), pp.1066–1074. doi:10.1016/j.optcom.2011.11.109.

Yuan, W., Li, L.-H., Lee, W.-B. and Chan, C.-Y. (2018). *Fabrication of Microlens Array and Its Application: A Review*. Chinese Journal of Mechanical Engineering, 31(1).

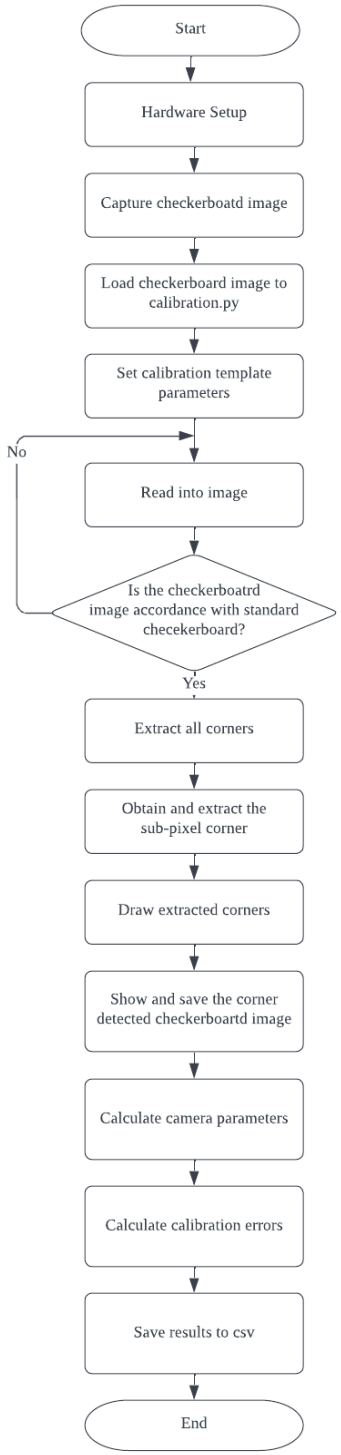
You, H. and Yang, J. (2014). *Development of An Analysis System for Geometric Contour Error Evaluation in Ultra-Precision Machining for Microlens Arrays.* Journal of Mechanical Science and Technology, 28(10), pp.4121–4129. doi:10.1007/s12206-014-0924-6.

Instrumental. (2020). Machine vision vs. manual inspection vs. Instrumental. [online] Available at: https://instrumental.com/resources/improve-product-quality/machine-vision-vs-manual-inspection-vs-instrumental/ [Accessed 10 Sep. 2022].

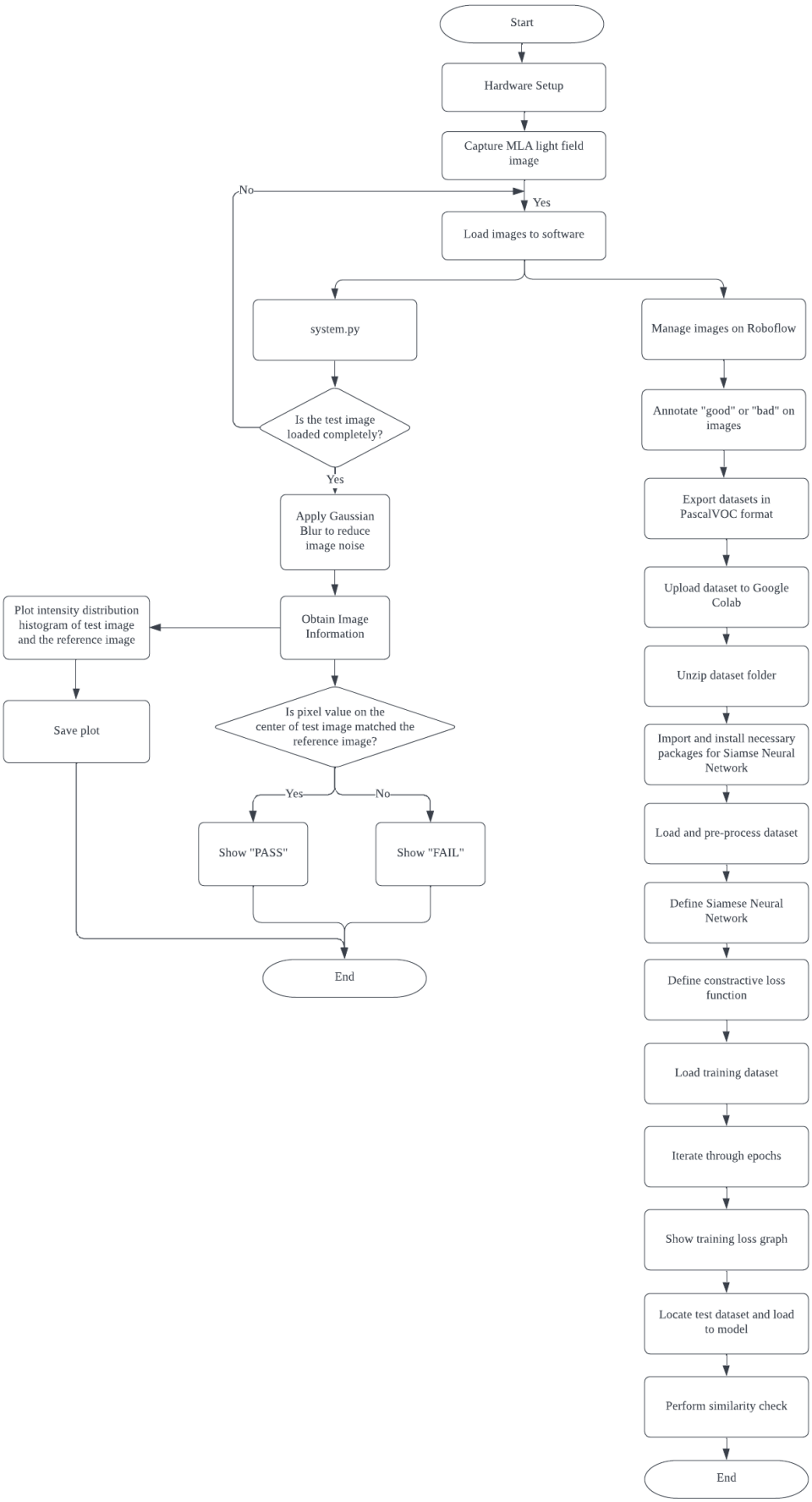
Zhu, L., Lai, P.T. and Choi, H.W. (2011). *Microlens Array on Flip-Chip LED Patterned With an Ultraviolet Micro-Pixelated Emitter*. IEEE Photonics Technology Letters, 23(15), pp.1067–1069. doi:10.1109/lpt.2011.2154323.

APPENDICES

Appendix A: Flowchart of Camera Calibration System.



Appendix B: Flowchart of Defect Detection System for MLAs.



Appendix C: Python Code for Camera Calibration System

```
# impoting needed libraries
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
import cv2
from skimage import io

# chessboaf parameters
chessboardSize = (4,4)
frameSize = (658,492)
images = r"./70h.JPG"
# termination criteria when increasing accuracy of found corne
rs later on
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER,
30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(
6,5,0)
objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np
.float32)
objp[:,:2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]]
.T.reshape(-1,2)

# Arrays to store object points and image points from all the
images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

for image in images:
    img = io.imread(images)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # Find the chess board corners
    # _retval_: Boolean (return value, True if corners obtaine
d), _corners_: the actual corners
    ret, corners = cv.findChessboardCorners(gray, chessboardSi
ze, None)
    # If found, add object points, image points (after refinin
g them)
    if ret == True:
        objpoints.append(objp)
        # Increase accuracy of corners
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-
1,-1), criteria)
        imgpoints.append(corners)
        # Draw and display the corners
        # cv.drawChessboardCorners(img, chessboardSize, corner
s2, ret)
        # cv.imshow('img', img)
        # cv.waitKey(700)
# cv.destroyAllWindows()

filename = '/calibration_1_saved.jpg'
cv2.imwrite(filename, img)
print('Successfully saved')
imgpoints1 = imgpoints[:6]
objpoints1 = objpoints[:6]
```

```
imgpoints2 = imgpoints[6:8]
objpoints2 = objpoints[6:8]

ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(obj
points1, imgpoints1, gray.shape[::-1], None, None)
print(cameraMatrix)
print(rvecs)
print(tvecs)
print(dist)
img = cv.imread("20d.JPG")
h,  w = img.shape[:2]

# input params : cameraMatrix, distCoeffs, imageSize, alpha[,n
ewImgSize[,centerPrincipalPoint]])
newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(cameraMatr
ix, dist, (w,h), 1, (w,h))
# Undistort
dst = cv.undistort(img, cameraMatrix, dist, None, newCameraMat
rix)

# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv.imwrite('calib-img/Result-calib-6.png', dst)

# Undistort with Remapping
mapx, mapy = cv.initUndistortRectifyMap(cameraMatrix, dist, No
ne, newCameraMatrix, (w,h), 5)
dst = cv.remap(img, mapx, mapy, cv.INTER_LINEAR)

# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
# Reprojection Error
mean_error = 0

for i in range(len(objpoints2)):
    imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], t
vecs[i], cameraMatrix, dist)
    error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2)/len(
imgpoints2)
    mean_error += error
print( "total error: {}".format(mean_error/len(objpoints)) )
```

Appendix D: Python Code for Image Processing Based System

```
import cv2
import matplotlib.pyplot as plt
import pandas as pd
import numpy
from tkinter import filedialog
from tkinter import *
import numpy as np

########################    Select File     ##################
##############
from numpy import arctan
from pandocfilters import Math

root = Tk()
root.filename =  filedialog.askopenfilename(initialdir = "/",t
itle = "Select file",filetypes = (("png files","*.png"),("all
files","*.*")))
print ("Reference image path:", root.filename) #returns file p
ath
root.filename_2 =  filedialog.askopenfilename(initialdir = "/"
,title = "Select file",filetypes = (("png files","*.png"),("al
l files","*.*")))
print ("Test image path:", root.filename_2) #returns file path

########################    Pre Process     ##################
##############
img_path = root.filename
img = cv2.imread(img_path, 0) # read image as grayscale. Set s
econd parameter to 1 if rgb is required
new_img = img / 255.0
numpy.set_printoptions(threshold=sys.maxsize)

img_path2 = root.filename_2
img2 = cv2.imread(img_path2, 0) # read image as grayscale. Set
 second parameter to 1 if rgb is required
new_img2 = img / 255.0
numpy.set_printoptions(threshold=sys.maxsize)

# Apply GaussianBlur to reduce image noise
image1 = cv2.GaussianBlur(img, (5, 5), 0)
image2 = cv2.GaussianBlur(img2, (5, 5), 0)

# Saving the image
filename = 'C:\\Users\\YY\\Downloads\\gaussian_saved_2.jpg'
cv2.imwrite(filename, image1)
```

```
print('Type of the image for image 1:', type(image1),'Type of
the image for image 2:',type(img2))
print()
print('Shape of image 1: {}'.format(img.shape), 'Shape of imag
e 2 : {}'.format(img2.shape))
print(f'height of image 1 {img.shape[0]} pixels', f'height of
image 2 {img2.shape[0]} pixels')
print(f'width of image 1 {img.shape[1]} pixels', f'width of im
age 2 {img2.shape[1]} pixels')
print('Dimension of Image 1 {}'.format(img.ndim), 'Dimension o
f Image 2 {}'.format(img2.ndim))
print('Image size of image 1 {}'.format(img.size), 'Image size
 of image 2{}'.format(img2.size))
print('Maximum RGB value in image 1 {}'.format(img.max()), 'Ma
ximum RGB value in image 2 {}'.format(img2.max()))
print('Minimum RGB value in image 1 {}'.format(img.min()), 'Mi
nimum RGB value in image 2 {}'.format(img2.min()))
center_x = img.shape[1]//2
center_y= img.shape[0]//2
print('Center of the image (x) {}',center_x)
print('Center of the image (y) {}',center_y)
fov1 = arctan(((img.shape[0]/2)/ 1)*2)
print('fov:',fov1)

######################   image 1   #################
arr = np.array([
    list(range(image1.shape[1]))
    for _ in range(image1.shape[0])
])

# average columns and rows
# left to right
cols = image1.mean(axis=0)
# bottom to top
rows = image1.mean(axis=1)

print(len(rows), len(cols),len(arr))

pd.options.display.max_rows = 20
df = pd.DataFrame({'x': cols})
df = pd.DataFrame(list(zip(cols)), columns=['x'])

# determining the name of the file
file_name = 'C:\\Users\\YY\\Downloads\\yy_all.csv'
# saving the excel
df.to_csv(file_name)
print('DataFrame is written to Excel File successfully.')
```

```
# plot histograms
f, ax = plt.subplots(2, 1)
ax[0].plot(range(image1.shape[1]),cols)
ax[1].plot(range(image1.shape[0]),rows)
# Add title and axis names
plt.title('My title')
plt.xlabel('Pixel Size (pixel)')
plt.ylabel('Pixel Mean (pixel)')
plt.show()
plt.savefig('C:\\Users\\YY\\Downloads\\saved1.jpg')

######################   image 2   #################
arr2 = np.array([
    list(range(image2.shape[1]))
    for _ in range(img2.shape[0])
])
# average columns and rows
# left to right
cols2 = image2.mean(axis=0)
# bottom to top
rows2 = image2.mean(axis=1)

# plot histograms
f, ax = plt.subplots(2, 1)
ax[0].plot(range(image2.shape[1]),cols2)
ax[1].plot(range(image2.shape[0]),rows2)
# Add title and axis names
plt.title('My title')
plt.xlabel('Pixel Size (pixel)')
plt.ylabel('Pixel Mean (pixel)')
plt.savefig('C:\\Users\\YY\\Downloads\\saved2.jpg')
compare1=(image1[center_y][center_x])
compare2=(image2[center_y][center_x])
print('Pixel mean value at the center for image 1:', compare1)
print('Pixel mean value at the center for image 1:', compare2)
diff= compare1- compare2
print("**********************Comparison Status******************
****")
if (img.shape == img2.shape):
    print('               Both image have the same size
        ')
if (diff<=15):
    print('=                              PASS
        ')
else:
    print('=                              FAIL
        ')
```

Appendix E: Python Code for Siamese Neural Network Based System

```
import tensorflow as tf
tf.test.gpu_device_name()
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import random
from PIL import Image
import PIL.ImageOps

import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset
import torchvision.utils
import torch
from torch.autograd import Variable
import torch.nn as nn
from torch import optim
import torch.nn.functional as F
# Creating some helper functions
def imshow(img, text=None):
    npimg = img.numpy()
    plt.axis("off")
    if text:
        plt.text(75, 8, text, style='italic',fontweight='bold',
            bbox={'facecolor':'white', 'alpha':0.8, 'pad':10})

    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

def show_plot(iteration,loss):
    plt.plot(iteration,loss)
    plt.show()
class SiameseNetworkDataset(Dataset):
    def __init__(self,imageFolderDataset,transform=None):
        self.imageFolderDataset = imageFolderDataset
        self.transform = transform

    def __getitem__(self,index):
        img0_tuple = random.choice(self.imageFolderDataset.imgs
)

#We need to approximately 50% of images to be in the same class
        should_get_same_class = random.randint(0,1)
        if should_get_same_class:
```

```
            while True:
                #Look untill the same class image is found
                img1_tuple = random.choice(self.imageFolderDat
aset.imgs)
                if img0_tuple[1] == img1_tuple[1]:
                    break
        else:

            while True:
                #Look untill a different class image is found
                img1_tuple = random.choice(self.imageFolderDat
aset.imgs)
                if img0_tuple[1] != img1_tuple[1]:
                    break

        img0 = Image.open(img0_tuple[0])
        img1 = Image.open(img1_tuple[0])

        img0 = img0.convert("L")
        img1 = img1.convert("L")

        if self.transform is not None:
            img0 = self.transform(img0)
            img1 = self.transform(img1)

        return img0, img1, torch.from_numpy(np.array([int(img1
_tuple[1] != img0_tuple[1])], dtype=np.float32))

    def __len__(self):
        return len(self.imageFolderDataset.imgs)
!wget ('/content/mla.v3i.folder.zip')
!rm -rf data
!unzip "mla.v3i.folder.zip" -d .
# Load the training dataset
folder_dataset = datasets.ImageFolder(root="./train/")
# Resize the images and transform to tensors
transformation = transforms.Compose([transforms.Resize((100,10
0)),
                                     transforms.ToTensor()
                                    ])

# Initialize the network
siamese_dataset = SiameseNetworkDataset(imageFolderDataset=fol
der_dataset,
                                        transform=transformati
on)
# Create a simple dataloader just for simple visualization
vis dataloader = DataLoader(siamese dataset,
```

```
                        shuffle=True,
                        num_workers=2,
                        batch_size=8)

# Extract one batch
example_batch = next(iter(vis_dataloader))

# Example batch is a list containing 2x8 images, indexes 0 and
 1, an also the label
# If the label is 1, it means that it is not the same , label
is 0, same
concatenated = torch.cat((example_batch[0], example_batch[1]),
0)

imshow(torchvision.utils.make_grid(concatenated))
print(example_batch[2].numpy().reshape(-1))
#create the Siamese Neural Network
class SiameseNetwork(nn.Module):

    def __init__(self):
        super(SiameseNetwork, self).__init__()

        # Setting up the Sequential of CNN Layers
        self.cnn1 = nn.Sequential(
            nn.Conv2d(1, 96, kernel_size=11,stride=4),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(3, stride=2),

            nn.Conv2d(96, 256, kernel_size=5, stride=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, stride=2),

            nn.Conv2d(256, 384, kernel_size=3,stride=1),
            nn.ReLU(inplace=True)
        )

        # Setting up the Fully Connected Layers
        self.fc1 = nn.Sequential(
            nn.Linear(384, 1024),
            nn.ReLU(inplace=True),

            nn.Linear(1024, 256),
            nn.ReLU(inplace=True),

            nn.Linear(256,2)
        )

    def forward_once(self, x):
```

```
       # This function will be called for both images
       # It's output is used to determine the similiarity
        output = self.cnn1(x)
        output = output.view(output.size()[0], -1)
        output = self.fc1(output)
        return output

    def forward(self, input1, input2):
        # In this function we pass in both images and obtain b
oth vectors
        # which are returned
        output1 = self.forward_once(input1)
        output2 = self.forward_once(input2)

        return output1, output2
# Define the Contrastive Loss Function
class ContrastiveLoss(torch.nn.Module):
    def __init__(self, margin=2.0):
        super(ContrastiveLoss, self).__init__()
        self.margin = margin

    def forward(self, output1, output2, label):
      # Calculate the euclidian distance and calculate the con
trastive loss
      euclidean_distance = F.pairwise_distance(output1, output
2, keepdim = True)

      loss_contrastive = torch.mean((1-
label) * torch.pow(euclidean_distance, 2) +
                                    (label) * torch.pow(torch.
clamp(self.margin - euclidean_distance, min=0.0), 2))


      return loss_contrastive
# Load the training dataset
train_dataloader = DataLoader(siamese_dataset,
                        shuffle=True,
                        num_workers=8,
                        batch_size=64)
net = SiameseNetwork().cuda()
criterion = ContrastiveLoss()
optimizer = optim.Adam(net.parameters(), lr = 0.0005 )
counter = []
loss_history = []
iteration_number= 0

# Iterate throught the epochs
for epoch in range(100):
```

```
    # Iterate over batches
    for i, (img0, img1, label) in enumerate(train_dataloader, 0
):

        # Send the images and labels to CUDA
        img0, img1, label = img0.cuda(), img1.cuda(), label.cud
a()

        # Zero the gradients
        optimizer.zero_grad()

        # Pass in the two images into the network and obtain tw
o outputs
        output1, output2 = net(img0, img1)

        # Pass the outputs of the networks and label into the l
oss function
        loss_contrastive = criterion(output1, output2, label)

        # Calculate the backpropagation
        loss_contrastive.backward()

        # Optimize
        optimizer.step()

        # Every 10 batches print out the loss
        if i % 10 == 0 :
            print(f"Epoch number {epoch}\n Current loss {loss_c
ontrastive.item()}\n")
            iteration_number += 10

            counter.append(iteration_number)
            loss_history.append(loss_contrastive.item())

show_plot(counter, loss_history)
# Locate the test dataset and load it into the SiameseNetworkDa
taset
folder_dataset_test = datasets.ImageFolder(root="./test/")
siamese_dataset = SiameseNetworkDataset(imageFolderDataset=fold
er_dataset_test,
                                        transform=transformatio
n)
test_dataloader = DataLoader(siamese_dataset, num_workers=2, ba
tch_size=1, shuffle=True)

# Grab one image that we are going to test
dataiter = iter(test_dataloader)
x0, _, _ = next(dataiter)
```

```
   for i in range(10):
 # Iterate over 10 images and test them with the first image (
x0)
    _, x1, label2 = next(dataiter)

    # Concatenate the two images together
    concatenated = torch.cat((x0, x1), 0)

    output1, output2 = net(x0.cuda(), x1.cuda())
    euclidean_distance = F.pairwise_distance(output1, output2)
    imshow(torchvision.utils.make_grid(concatenated), f'Dissim
ilarity: {euclidean_distance.item():.2f}')
```

Appendix F: Python Code for 3D Contour Plot and Surface Plot.

```
from PyQt5 import QtWidgets, uic, QtGui, QtCore
import cv2
import matplotlib.pyplot as plt
from PyQt5.QtWidgets import *
from PyQt5 import QtWidgets
import sys
import numpy as np

class Ui(QtWidgets.QMainWindow):
    def __init__(self):
        print("Program start.")
        super(Ui, self).__init__()
        uic.loadUi('C:\\Users\\YY\\OneDrive\\Documents\\Y4S2\\
FYP\\MLA.ui', self)

        self.button = self.findChild(QtWidgets.QPushButton, 'p
ushButton_5')  # Find the button
        self.button.clicked.connect(self.exitButtonPressed)
        self.show()

        self.button = self.findChild(QtWidgets.QPushButton, 'p
ushButton')  # Find the button
        self.button.clicked.connect(self.three_d_image)
        self.show()

    def exitButtonPressed(self):
        # destroy tkinter object
        print("Program exiting.")
        self.destroy()
        sys.exit()

def three_d_image(self):
        imbgr = cv2.imread('C:\\Users\\YY\\Downloads\\IMG_2022
0910_210838.jpg')
        imrgb = cv2.cvtColor(imbgr, cv2.COLOR_BGR2RGB)
        imlab = cv2.cvtColor(imbgr, cv2.COLOR_BGR2LAB)

        # contour map
        plt.figure(2)

        y = range(imlab.shape[0])
        x = range(imlab.shape[1])
        X, Y = np.meshgrid(x, y)
```

```
        plt.contour(X, Y, imlab[:, :, 0], 50)

        plt.show()

        # surface map
        plt.figure(3)

        ax = plt.axes(projection='3d')

        y = range(imlab.shape[0])
        x = range(imlab.shape[1])
        X, Y = np.meshgrid(x, y)

        ax.plot_surface(X, Y, imlab[:, :, 0])

        plt.show()

if __name__=="__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = Ui()
    app.exec_()
```

Appendix G: Qt Designer .UI code for 3D Contour Plot and Surface Plot GUI.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">UI
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>302</width>
    <height>251</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QPushButton" name="pushButton">
    <property name="geometry">
     <rect>
      <x>40</x>
      <y>80</y>
      <width>93</width>
      <height>28</height>
     </rect>
    </property>
    <property name="text">
     <string>Plot 3D</string>
    </property>
   </widget>
   <widget class="QPushButton" name="pushButton_5">
    <property name="geometry">
     <rect>
      <x>150</x>
      <y>80</y>
      <width>93</width>
      <height>28</height>
     </rect>
    </property>
    <property name="text">
     <string>Exit</string>
    </property>
   </widget>
```

```
  <widget class="QFrame" name="frame_3">
   <property name="geometry">
    <rect>
     <x>490</x>
     <y>360</y>
     <width>121</width>
     <height>51</height>
    </rect>
   </property>
   <property name="frameShape">
    <enum>QFrame::StyledPanel</enum>
   </property>
   <property name="frameShadow">
    <enum>QFrame::Raised</enum>
   </property>
  </widget>
 </widget>
 <widget class="QMenuBar" name="menubar">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>302</width>
    <height>26</height>
   </rect>
  </property>
 </widget>
 <widget class="QStatusBar" name="statusbar"/>
</widget>
<resources/>
<connections/>
</ui>
```