**Stereo Image Calibration for Size Measurement**

by

ALIF BIN JASMIN
18003682

Dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Engineering (Hons)
(Computer Engineering)

September 2023

Universiti Teknologi PETRONAS
32610 Seri Iskandar
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

**Stereo Image Calibration for Size Measurement**

by

ALIF BIN JASMIN

18003682

A project dissertation submitted to the
Computer Engineering Programme
Universiti Teknologi PETRONAS
In partial fulfillment of the requirements for the
BACHELOR OF ENGINEERING (HONS)
(COMPUTER ENGINEERING)

Approved by,

_____

Dr. Lila Iznita

UNIVERSITI TEKNOLOGI PETRONAS

SERI ISKANDAR, PERAK

September 2023

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

_____

ALIF BIN JASMIN

# ABSTRACT

This project focuses on improvements of botanical research and analysis using stereo image processing. Measuring the length of flowers plays a significant role in various botanical research applications, such as phenotypic analysis, growth monitoring, and floral development studies. In this project, we propose a novel approach for measuring flower length using stereo imaging on a portable platform. To achieve robust and precise measurements, we integrate computer vision algorithms, such as bounding boxes and instance segmentation, into the stereo imaging pipeline. These algorithms allow us to isolate the flowers from the background and extract their prominent features, such as petals or stem, for length estimation. Additionally, we explore the potential of deep learning-based approaches, specifically Convolutional Neural Networks (CNNs), for flower recognition and localization within the stereo images. To evaluate the effectiveness of our approach, we conducted experiments using a dataset of stereo images captured by a custom made portable platform under various lighting and environmental conditions. Finally, after successfully training the CNN model the model can be used in a platform which can be hosted locally or publicly on the internet using python code.

# ACKNOWLEDGEMENTS

I am immensely grateful for the collective support and guidance I received throughout my final year project. My supervisor, Dr Lila, played a pivotal role in shaping the project's direction, providing invaluable insights, and fostering an environment of academic growth. The dedication and expertise of my lecturers, who imparted knowledge and challenged me to think critically, were instrumental in the project's development. My heartfelt appreciation extends to my parents for their unwavering encouragement and understanding during this academic journey. Lastly, my sincere thanks go to my colleagues whose collaborative spirit and shared enthusiasm added a dynamic element to the project, making the experience both enriching and enjoyable. The collective influence of my supervisor, lecturers, parents, and colleagues has undoubtedly contributed to the success and personal growth achieved in the culmination of this project.

TABLE OF CONTENTS

LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 Background of Studies

### 1.1.1 Botanical research

Botanical research is a field dedicated to the scientific investigation and study of plants. It encompasses various disciplines within botany, such as taxonomy, ecology, genetics, and more. By delving into these aspects, researchers can expand our understanding of plant life and its intricate workings. Botanical research plays a vital role in several fields, including agriculture, medicine, ecology, conservation, and biotechnology. By uncovering the secrets of plants, scientists can develop improved farming techniques, discover new medicinal compounds, understand ecosystem dynamics, preserve endangered species, and explore innovative applications in biotechnology. We can use stereo image processing is introduced to have a more exponential advancement in the botanical research field.

### 1.1.2 Machine learning

Machine learning is a subfield of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computer systems to improve their performance on a specific task through learning from data, without being explicitly programmed. The primary goal of machine learning is to develop systems that can generalize patterns from existing data and make predictions or decisions when presented with new, unseen data. There are many algorithms which can be used for oject detection, few of the most famous ones are R-CNN, YOLOv5, RetinaNet, CenterNet and so

on. In this experiment we will solely focus on R-CNN

The purpose of having a machine learning model in this project is to identify objects (eg flowers) in images. From the identified objects, we can extract information such as the location of the object in that specific picture, general length of the object in pixel and also what kind of object is inside the image (such as the flower names). All this will be highly dependant on how we train the model and what kind of datasets do we feed into that model.

### 1.1.3 Stereo image processing

Stereo image processing is a technique employed to extract depth information from a pair of images captured from slightly different viewpoints. This method relies on algorithms that analyze the disparity between corresponding pixels in the two images to estimate the 3D structure of the scene. By reconstructing depth information, stereo image processing enables us to perceive the spatial characteristics of the captured environment. This technology finds applications in various domains, including computer vision, robotics, virtual reality, and augmented reality. It allows for more immersive experiences, precise object detection, and accurate scene reconstruction, enhancing our interaction with digital environments.

### 1.1.4 Setup of a portable platform

To enhance the accessibility and usability of the standardized measurement protocol, the development of a portable platform could greatly facilitate on-site data collection. A portable platform would allow researchers to use the measurement software directly in the field, eliminating the need for manual data entry or post-processing. With an application created using python, researchers can easily capture and record flower length measurements in real-time, along with relevant metadata such as location, date, and environmental conditions. This immediate data collection not only improves efficiency but also reduces the chances of errors or data loss during the transfer from

field notes to a computer system. Furthermore, a portable platform for floral trait measurement could offer additional features to enhance the research process. For instance, the application could provide image analysis capabilities, allowing researchers to capture images of flowers and automatically measure their lengths using computer vision algorithms. This would not only streamline the measurement process but also enable researchers to handle large datasets of botanical data more efficiently.

In order to get this project to work, a set of portable cameras are needed since the whole purpose of this project is to capture pictures of plants in a botanical garden. It is indeed possible to bring some plants to the working area and start recording from there but in order to do that, we would have to cut off the plants which might cause problems in its growth. An example of portable cameras which can be programmed to use image processing is a micro-controller such as arduinos, esp32s, raspberrypis and many more. These products are suggested because they have the ability to run codes in multiple coding languages, use cameras and are also able to connect to the internet which makes them a good Internet of Things(IoT) product.

## 1.2 Problem Statement

Accurate measurement of floral traits is essential for gaining insights into plant diversity, evolutionary processes, and ecological interactions. Flower length, in particular, is a critical morphological characteristic that can provide valuable information about plant species. However, the lack of a standardized and systematic approach for measuring flower length presents a significant challenge. Without a consistent method, researchers may encounter difficulties in comparing and analyzing floral data across different studies. This inconsistency hinders our ability to fully understand the patterns and relationships among floral traits and their ecological significance.

To address this challenge, it is crucial to develop a standardized protocol for measuring flower length. Such a protocol would provide clear guidelines and procedures for researchers to follow, ensuring consistency and accuracy in their measurements. The protocol should include details on the positioning of the flower, the specific points to

measure from, and the tools or instruments required for precise measurements. Additionally, it would be beneficial to establish a common unit of measurement to facilitate data comparison and synthesis across studies. By implementing a standardized approach, researchers can effectively utilize flower length as a morphological trait and unlock its potential as a valuable indicator of plant diversity and ecological interactions.

On another note, unlike most objects which has distinguished features such as our electronic devices with fixed shapes and sizes or animals with their specific parts like the shape of their eyes or ears, flowers also comes with many shapes and sizes but even though each species have their own distinct features, flowers tend to exist on an environment with similar features such as on a bed of grass or the present of trees which usually have the same colors as the most part of the flowers. This is a crucial part in image processing as the earlier parts of image processing is to separate the object of interest with its background. To address this problem, green scaling was used as it has more shade when we convert an image from a full red, green, blue (RGB) colored images. Another crucial step in order to separate flowers from a scene of similar color and shapes is the proper training of the CNN model. The reason why CNN was chosen is because the algorithm is made strictly for image processing and it uses a lot of nodes to compare one single image with its model. In other words, CNN processes an image a multiple number of times to extract known features of the flower part by part. We can also set the number of layer to put on the algorithm to increase the accuracy of the extraction. Take note that 'CNN' is referred to the whole family of the CNN algorithm such as R-CNN, faster R-CNN, Mask-CNN and such.

To enhance the accessibility and usability of the standardized measurement protocol, the development of a portable platform could greatly facilitate on-site data collection. A portable platform would allow researchers to use the measurement software directly in the field, eliminating the need for manual data entry or post-processing. With a portable application, researchers can easily capture and record flower length measurements in real-time, along with relevant metadata such as location, date, and environmental conditions. This immediate data collection not only improves efficiency but also reduces the chances of errors or data loss during the transfer from field notes

to a computer system. Furthermore, a portable platform for floral trait measurement could offer additional features to enhance the research process. For instance, the application could provide image analysis capabilities, allowing researchers to capture images of flowers and automatically measure their lengths using computer vision algorithms. This would not only streamline the measurement process but also enable researchers to handle large datasets more efficiently.

Additionally, the portable platform could integrate with existing databases or platforms for data sharing and collaboration, fostering a more interconnected and collaborative research environment. In conclusion, the accurate measurement of floral traits, particularly flower length, is crucial for understanding plant diversity, evolutionary processes, and ecological interactions. The lack of a standardized approach poses challenges in utilizing this morphological characteristic effectively. Developing a standardized measurement protocol and a portable platform for on-site data collection can address these challenges, providing researchers with a consistent and efficient method for gathering floral trait data. By doing so, we can enhance our understanding of plant biology, foster collaboration among researchers, and contribute to broader scientific knowledge in the field of botany.

## 1.3 Objective

The objective of this project is to use stereo image processing to estimate size of flowers. The techniques and/or steps used are as follows:

- To train a Machine Learning model which will use in image processing.
- To be able to calibrate a stereo image using two cameras to estimate depth of the image.
- To estimate the actual size of an object in a stereo image using depth and the distance of the object from the cameras.

## 1.4 Scope of Work

The scope of the study encompasses three main areas:

- A setup of a pair of cameras for stereo image
- Object detection using machine learning
- Object size estimation using stereo image depth.

### 1.4.1 Portable dual camera setup

A successful camera pair setup is essential as a stereo image processing wont work unless the set up is not complete. Successful refers to having both cameras to be able to record scenes live, being able to communicate with the server, calculates the most accurate calibration of the cameras, and to be sturdy enough to be brought anywhere. The cameras must also be aligned properly, the distance between them should not change under any circumstances and they must be, at all times, protected from dust and water. The cameras used are 2 ESP32-CAM devices which are embedded with c++ to stream the cameras pictures taken frame by frame in an infinite loop. Using any computer devices such as our laptops, tables or even raspberrypi, we will be able to stream the frames from the ESP32-CAM using python via http request protocol. Ofcourse, the streamed frames saved by the python code will be remapped to using both cameras' internal parameters in order to get depth from both picture. The most important aspect is that the setup must be portable to ensure an outmost flexibility of the project so that it can be brought to the field for botanical researches.

### 1.4.2 Object detection

Object detection and recognition involve the development of algorithms and techniques to identify and locate specific objects within an image or a scene. In the context of floral research, this could be applied to detect and recognize flowers or specific floral structures. By accurately detecting and recognizing floral objects, researchers can

efficiently analyze and measure their morphological traits, contributing to a deeper understanding of plant diversity and evolution.

The first step of using any machine learning models is to train the model using a dataset which is specific to the project. Since we will be using instance segmentation and recognition, the dataset needs to be labelled one by one with the coordinates of the exact location and shape of all involved objects in each pictures.

Instance segmentation is a technique whereas the algorithm will point out the exact location and shape of specific objects in a picture. These coordinates will then be used to separate the object with the background. Separating the exact location of objects will give the algorithm the ability to use these coordinates to compare the location of the left and right eye of the stereo setup. Not only that, we can also estimate the size of the object, in pixels, using the coordinates we got from the machine learning model.

### 1.4.3 Size estimation

Actual object size estimation is another crucial aspect within the scope of the study. This entails the development of computational methods that can automatically measure the length of flowers or specific floral components. By leveraging image processing and computer vision techniques, researchers can extract relevant features from floral images and estimate their lengths accurately.

The technique used on this project is based on the concept of similar triangles (refers to **??**). With the said concept, we can estimate the actual size of an object by comparing the size of it in pixels with the scale which best fits the conversion of the pixels to real life measurements based on the distance of the object from the cameras. This automation eliminates the need for manual measurements, saving time and improving measurement precision. Automatic length measurement can significantly enhance the efficiency of data collection in botanical research, enabling researchers to analyze larger datasets and explore broader patterns and trends across plant species.

## 1.5 Research Question

- Is it possible to accurately estimate the distance as well as the size of an object in an image.
- How accurate can depth be when we manually calibrate a self setup dual camera prototype.

# CHAPTER 2

# LITERATURE REVIEW

The literature review chapter plays a crucial role in this research study by providing a comprehensive overview of existing knowledge and research related to stereo image processing, object recognition, and length measurement. Through an in-depth analysis of relevant academic articles and scholarly papers, we aim to identify the key concepts, methodologies, and findings that have shaped the field. Specifically, we examine the significance of stereo image processing techniques such as preprocessing, segmentation, and depth estimation in accurately measuring object length. Additionally, we explore the use of Convolutional Neural Networks (CNN) for object recognition and classification, particularly in the context of flower recognition. By reviewing the literature, we aim to identify gaps, trends, and challenges in the existing body of knowledge, which will inform our research objectives and contribute to the advancement of stereo image processing and object measurement techniques. This literature review chapter serves as a foundation for our research, providing a comprehensive understanding of the current state of research and guiding our approach to address the research questions and goals of the study.

## 2.1 Significance of Flower Size in Botanical Research

Flower size plays a significant role in botanical research, particularly in understanding plant-pollinator interactions and reproductive strategies. Dellinger et al. ([1]) highlight the importance of flower shape in facilitating successful pollination. They emphasize that flower shape is crucial in ensuring a proper fit with pollinators and maximizing the transfer of conspecific pollen, especially in functionally specialized systems. Their

study, which utilizes geometric morphometrics, provides valuable insights into determining the ”fittest” floral shape and its implications for pollination success.

In terms of reproductive assurance, flower size has been found to have a significant effect. Research by Brad F. K. and Elizabeth E. demonstrates that intact flowers generally produce significantly more seeds than emasculated flowers. This finding indicates that larger flowers have a reproductive advantage, potentially due to their ability to attract more pollinators or facilitate more efficient pollen transfer. The study highlights the importance of flower size in ensuring reproductive success and highlights the benefits of self-reproducing as a mechanism for reproductive assurance.

Additionally, flower size is associated with trade-offs in pollination success and fecundity. Kettle et al. ([2]) highlights the existence of a trade-off between flower size and the number of flowers produced among species. They suggest that this trade-off contributes to a reduction in the variance of fecundity among different species. In other words, larger flowers may invest more resources into attracting pollinators and producing a higher number of seeds per flower, while smaller flowers may produce a greater number of flowers to compensate for their reduced size. This trade-off underscores the intricate relationship between flower size, pollination success, and reproductive output.

Collectively, these studies emphasize the significance of flower size in botanical research. Flower size influences pollination success, reproductive assurance, and fecundity, which are essential aspects of plant fitness and evolutionary dynamics. Understanding the relationship between flower size and its implications for plant-pollinator interactions can shed light on the adaptive strategies and ecological implications of floral traits.

## 2.2 Machine Learning Algorithm

There are various subsections of visions in computer vision, few of the well known are image classification, object detection, region of interest and instance object detection. In this project we will be using instance oject detection because the algorithm will provide us with the details that we will need to measure the size of the object. Two

of the most famous algorithm for instance segmentation are Mask R-CNN (Regional Convulation Nueral Network) and YOLOv5 (You Only Look Once).

Table 2.1: Mask R-CNN vs. YOLOv5

| | Mask R-CNN | YOLOv5 |
|---|---|---|
| Advantages | • Provides precise instance segmentation, allowing for accurate measurement of flower size. | • known for real-time object detection. |
| Disadvantages | • It can be computationally expensive. | • They may sacrifice some precision in segmentation compared to Mask R-CNN. |

The difference between Mask R-CNN and YOLOv5 are depicted in 2.1. Mask R-CNN is an extension of Faster R-CNN that adds a branch for predicting segmentation masks in parallel with the existing bounding box and class predictions. They are most widely used for precise instance segmentation. This algorithm provides precise instance segmentation, allowing for accurate measurement of flower size. The disadvantage of it is that it can be computationally expensive.

YOLOv5(or YOLOv4) is designed for real-time object detection and has been extended for instance segmentation. This algorithm are also some of the popular versions that have been used for instance detection tasks. They are known for real-time object detection and can be adapted for measuring flower size. The only drawback to it is that they may sacrifice some precision in segmentation compared to Mask R-CNN.

### 2.2.1 CNN, R-CNN

Convolutional Neural Networks (CNNs) represent a foundational architecture in the domain of computer vision, particularly adept at image processing and feature extraction. Their primary purpose lies in image classification tasks, where the objective is to

assign labels or categories to entire images. CNNs operate by applying convolutional filters to input images, capturing local features in a hierarchical manner through convolutional and pooling layers. These layers allow the network to learn increasingly complex representations as it progresses, and fully connected layers aggregate these features for the final classification. CNNs are trained end-to-end on labeled datasets, optimizing their parameters through backpropagation to minimize the difference between predicted and actual outputs.

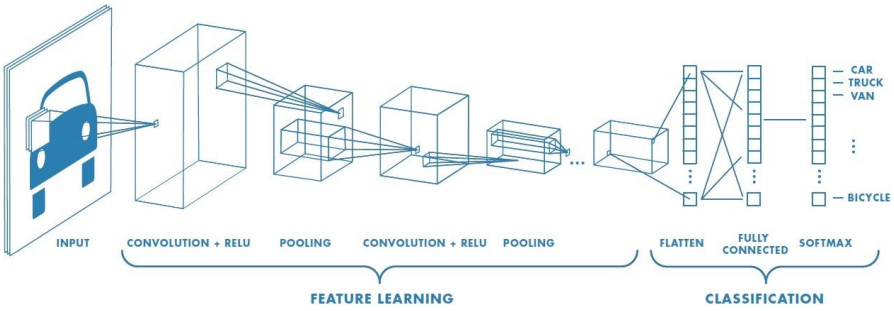



Figure 2.1: R-CNN Architecture

In contrast to CNNs, Region-based Convolutional Neural Networks (R-CNNs) are specialized architectures designed explicitly for object detection tasks. Object detection involves not only classifying objects within an image but also precisely localizing them. R-CNNs introduce a region proposal mechanism to CNNs, generating candidate regions likely to contain objects using methods like Selective Search or EdgeBoxes. Each proposed region is then processed independently through a CNN for feature extraction, and these features are utilized for both object classification and bounding box regression. R-CNNs involve a more intricate training process, including pre-training the CNN for feature extraction on a large dataset and fine-tuning the region proposal and object detection components for the specific detection task.

While both CNNs and R-CNNs leverage convolutional layers for feature extraction, their key differences lie in their intended tasks and architectures. CNNs are versatile, excelling in image classification, whereas R-CNNs extend their capabilities by incorporating mechanisms for object localization. The intricate training process of R-CNNs involves pre-training on a larger dataset for feature extraction and subsequent fine-tuning

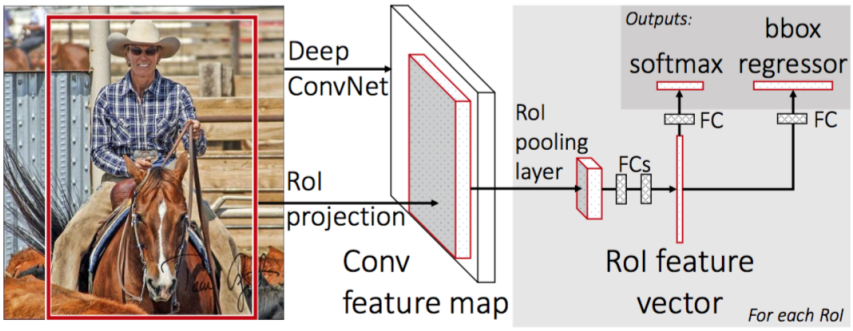



Figure 2.2: Caption

for the specific object detection task. In essence, CNNs serve as foundational components for image understanding, while R-CNNs are tailored to the nuanced demands of object detection, precisely locating and classifying multiple objects within an image.

### 2.2.2 Mask R-CNN

In another case, mask R-CNNs represent a significant evolution beyond R-CNNs, specifically addressing the need for instance segmentation in addition to object detection. The key innovation lies in the inclusion of an additional branch dedicated to predicting segmentation masks alongside the existing branches for object classification and bounding box regression. This enables Mask R-CNNs to not only classify and locate objects but also to provide a pixel-wise delineation of each object instance within an image. The mask prediction branch facilitates the creation of precise segmentation masks, making Mask R-CNNs highly effective in scenarios requiring detailed understanding of object boundaries.

Mask R-CNN (Mask Region-based Convolutional Neural Network) is an extension of the R-CNN architecture designed to address the task of instance segmentation, which involves not only detecting objects within an image but also precisely delineating the boundaries of individual instances. Developed by Facebook AI Research (FAIR), Mask R-CNN builds upon the success of Faster R-CNN (a faster variant of R-CNN) by adding a dedicated branch for mask prediction. The network is capable of simultaneously performing object detection, bounding box regression, and pixel-wise instance segmentation.

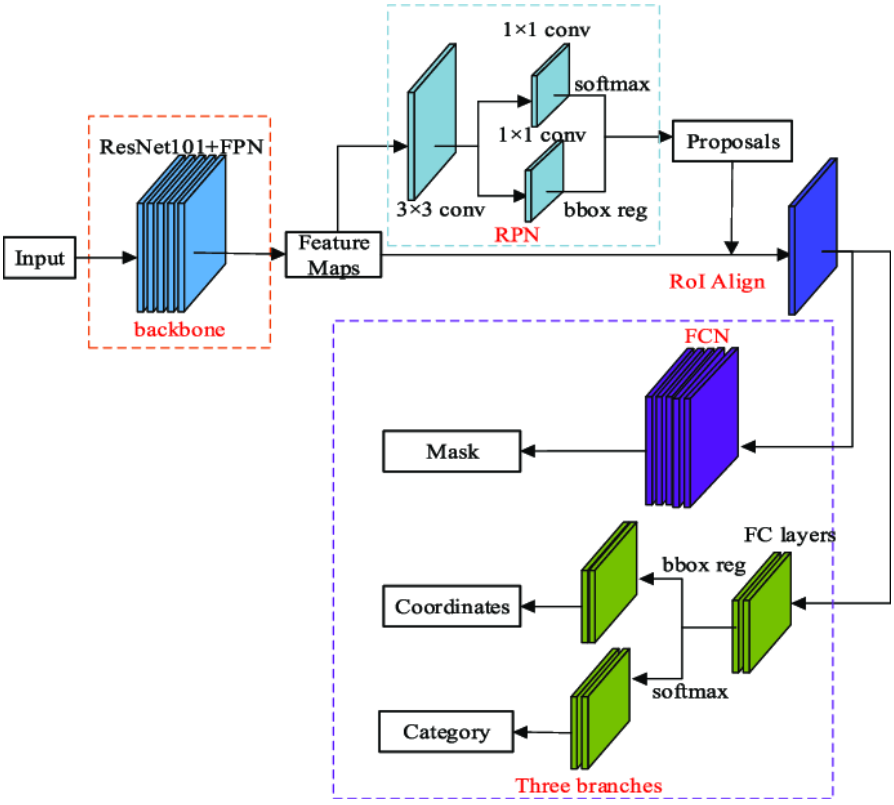



Figure 2.3: Mask R-CNN Architecture

Mask R-CNN typically utilizes a pre-trained CNN (such as ResNet) as the backbone network to extract hierarchical features from the input image. The pre-trained weights help the network learn rich and generalizable features. Region Proposal Network (RPN) is employed to generate region proposals—candidate bounding boxes likely to contain objects. This step efficiently narrows down the search space for objects within the image. The proposed regions are processed through a series of RoI (Region of Interest) pooling layers and fully connected layers, similar to the original R-CNN architecture. These layers extract features from each proposed region, enabling subsequent tasks. The network simultaneously performs object classification to determine the class of each detected object and bounding box regression to refine the location of the bounding box around the object. A unique aspect of Mask R-CNN is the additional branch dedicated to predicting segmentation masks for each proposed region. The mask branch consists of a series of convolutional layers that generate pixel-wise masks corresponding to the object instances within the proposed regions. Mask R-CNN is trained in an end-to-end fashion, incorporating both the classification and segmentation tasks. The

loss function consists of three components: object classification loss, bounding box regression loss, and mask segmentation loss. During training, the network learns to optimize these components simultaneously. During inference, the trained Mask R-CNN takes an input image and outputs predictions for object classes, bounding boxes, and segmentation masks. The segmentation masks provide a detailed pixel-wise delineation of each object instance within the image.

## 2.3 Stereo Imaging

Stereo image in stereo vision systems involves the use of two video cameras that are aligned in parallel and fixed in position. These cameras are carefully calibrated to ensure matching image properties such as size, color space, and lighting conditions. The calibration process is essential for achieving accurate and consistent stereo imaging, as it enables the cameras to capture synchronized and corresponding views of the scene. By aligning the cameras and ensuring their image properties are identical, potential discrepancies or distortions between the captured images are minimized, leading to more reliable and precise measurements.

The primary objective of stereo image capture is to obtain a pair of images with overlapping views. When an object of interest enters the overlapping field of view of the two cameras, it provides the necessary data for distance and size measurements. The disparity between the corresponding pixels in the stereo images allows for the estimation of the object's distance or depth from the cameras. By analyzing the disparities, which represent the apparent shift of an object between the two images, stereo vision algorithms can triangulate the object's position in 3D space. A visualization of this triangulation is depicted in figure 2.4 below:

Additionally, the size of the object can be determined by comparing the measurements in the two images. By utilizing the known baseline distance between the cameras, the parallax observed in the stereo images can be used to estimate the size or length of the object. This measurement process relies on the principle of triangulation, where the known baseline distance, disparity, and other calibration parameters are utilized to

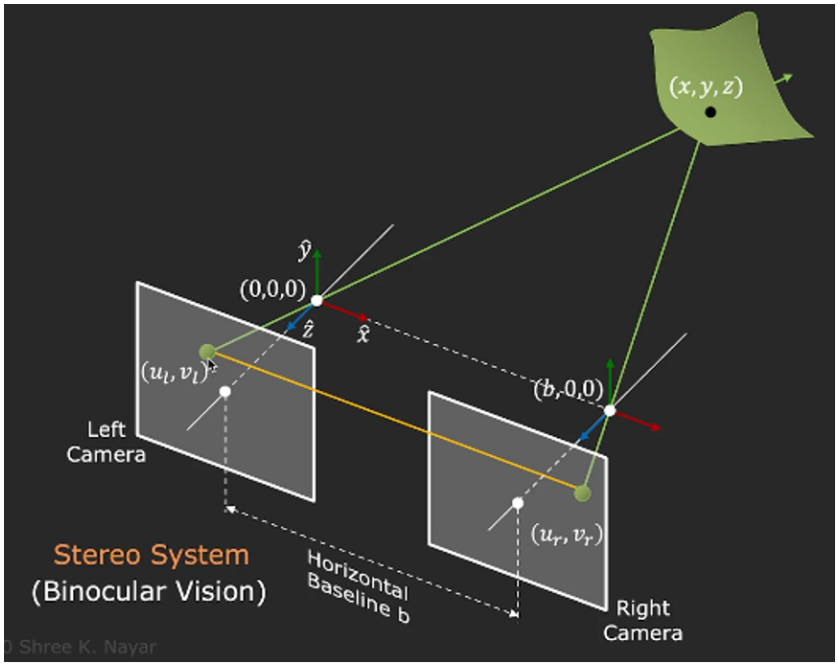



Figure 2.4: Stereo capture diagram

compute the object's actual distance and size.

### 2.3.1 Calibration and rectification

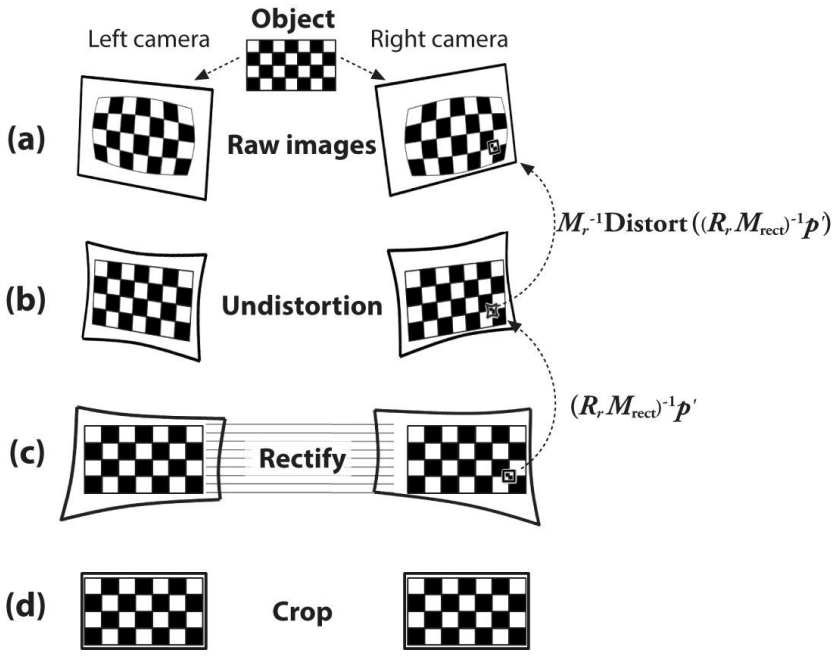



Figure 2.5: Stereo calibration flow

Stereo image calibration and rectification is a process in computer vision that involves aligning and calibrating a pair of cameras, known as a stereo camera setup. The goal is to determine the intrinsic and extrinsic parameters of each camera in the stereo pair, as well as the relationship between them. This calibration process is crucial for accurate 3D reconstruction and depth estimation in stereo vision applications. There are a few key concepts of stereo calibration and rectification.

Firstly, intrinsic parameters represent the internal characteristics of each camera, such as focal length, principal point, and lens distortion. Extrinsic parameters describe the spatial relationship between the two cameras in the stereo rig, including their relative orientation and translation. Calibration calculates these parameters to establish the baseline and alignment between the cameras. Once the intrinsic and extrinsic parameters are determined, it becomes possible to calculate the transformation matrix that relates the 3D world coordinates of a point to its corresponding image coordinates in both cameras.

Other than that, lens distortion is common in camera systems and can impact the accuracy of depth measurements. Calibration helps correct for radial and tangential distortions. Epipolar geometry describes the geometric relationship between corresponding points in stereo images. It defines the epipolar lines along which the corresponding points must lie.

Lastly, Rectification is often performed after calibration to transform the images in such a way that corresponding points in the stereo images lie along the same scanlines. This simplifies stereo matching. Figure 2.5 is the general flow of stereo calibration and rectification.

### 2.3.2 Disparity Map Generation

Depth map generation is a process in computer vision that involves estimating the distance or depth information for each pixel in an image. This is particularly important in applications like 3D reconstruction, robotics, and augmented reality, where understanding the spatial layout of a scene is crucial. The depth map provides a representation

of the scene's geometry, indicating how far objects are from the camera.

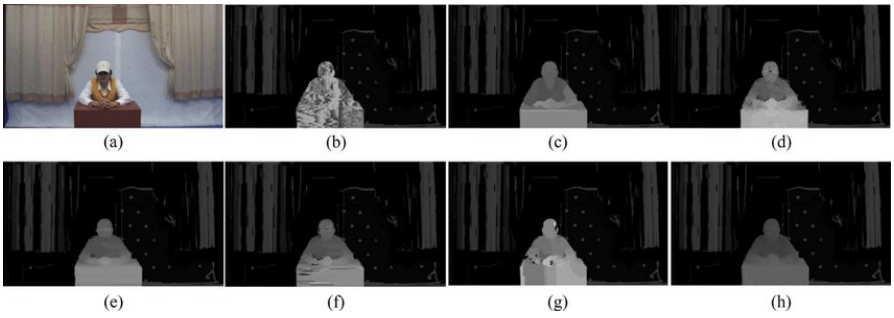



Figure 2.6: Example of a depth map

Depth map generation often relies on stereo vision, which involves using a pair of calibrated cameras (stereo cameras) to capture the same scene from slightly different viewpoints. The relative displacement between corresponding points in the two images is used to calculate the depth information.Stereo cameras needs to be calibrated and rectified as stated earlier in order to obtained depth map.

Corresponding features (such as corners or keypoints) are identified in the rectified stereo images. This process is known as feature matching. The next step is to calculate diaprity. Disparity represents the horizontal shift between corresponding points in the left and right stereo images. It is calculated by finding the disparity map, which indicates the pixel-wise differences in horizontal position between matched features.

Once the disparity map is obtained, the depth for each pixel can be calculated using the triangulation principle. The depth is inversely proportional to the disparity, and the baseline distance between the stereo cameras is a key factor in this calculation.

# CHAPTER 3

# METHODOLOGY

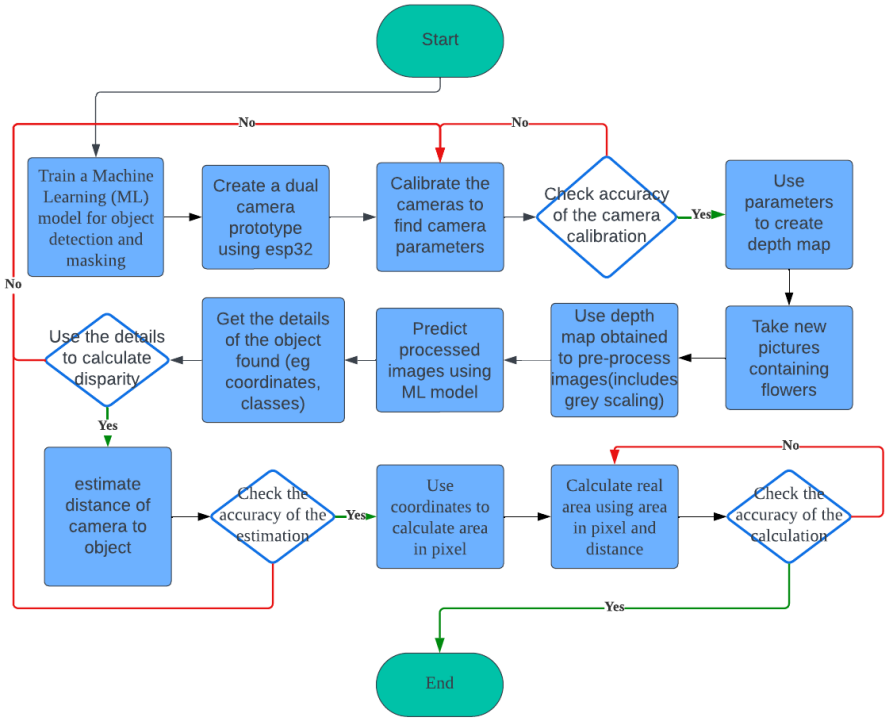



Figure 3.1: Project flowchart

In Fig. 3.1, We can see the whole process of this stereo image processing project. The project can basically be devided into 6 sections which are, Machine learning model training, stereo image setup and calibration, depth mapping, object detection using current model, estimation of the object's distance and lastly the size estimation of the actual object. The project starts with training a machine learning model using the Mask R-CNN algorithm. Then, we set up two cameras placed in parallel to take 2 images for stereo image processing. The cameras will be controlled using python via localhost. After setting up the cameras, we will need to calibrate the cameras to get the physical

internal characteristics of the cameras such as its focal length. both cameras will be calibrated together to generate a depth map that is unique to the combination of the two cameras. By looking at the output image generated by the depth map, we can check weather or not the cameras are calibrated properly. After proper calibration of the camera, we can proceed by taking pictures of flowers. The pictures taken will straight away be remapped using the depth map generated earlier.

The generated image will be used for object prediction using the machine learning trained in the first step. Predictions were done after remapping of the images taken and not before it. After the prediction, we will obtain the location of the object inside the images and with the coordinates of the location we can just find the difference of the coordinates by comparing the two objects in separate cameras, and that difference is called disparity and it can only be obtained using the left and right 'eyes' of a stereo scene between two images. Using the disparity, we can estimate the distance of the image from the camera using a formula that will be mentioned later in this report. The coordinates will also be used to calculate the area of the object, in pixel, inside the images. The average area of both objects will be used to calculate the area in cm. The real area will be obtained when compared with the distance using the formula that will also be mentioned later.

## 3.1 Model training

### 3.1.1 Dataset labelling

Generally, there are various ways to label a dataset for machine learning training. How to label a dataset will be depending on what king of machine learning model are we going to train. For example, if we want to train an image classification model, we can simply separate our dataset into separate folders labelled with its class name. Without any further actions, the dataset is already considered as labelled and it can be used to strain an image classifier machine learning model. In our case, we will be doing instance segmentation whereas each image in the dataset should be given a coordinate of where the object actually is. By training a model with this kind of dataset, the algorithm will then given us the specific coordinates of the location of the object in the image, just

like how we labelled them.

For this part of the task we used a website application called Roboflow which has a very friendly user interface (UI) which makes labeling each image fairly easy. An example of the labelling process is shown in fig3.2. The labelling process took around one week to label 300 images which is divided to 100 images for each class of flowers.

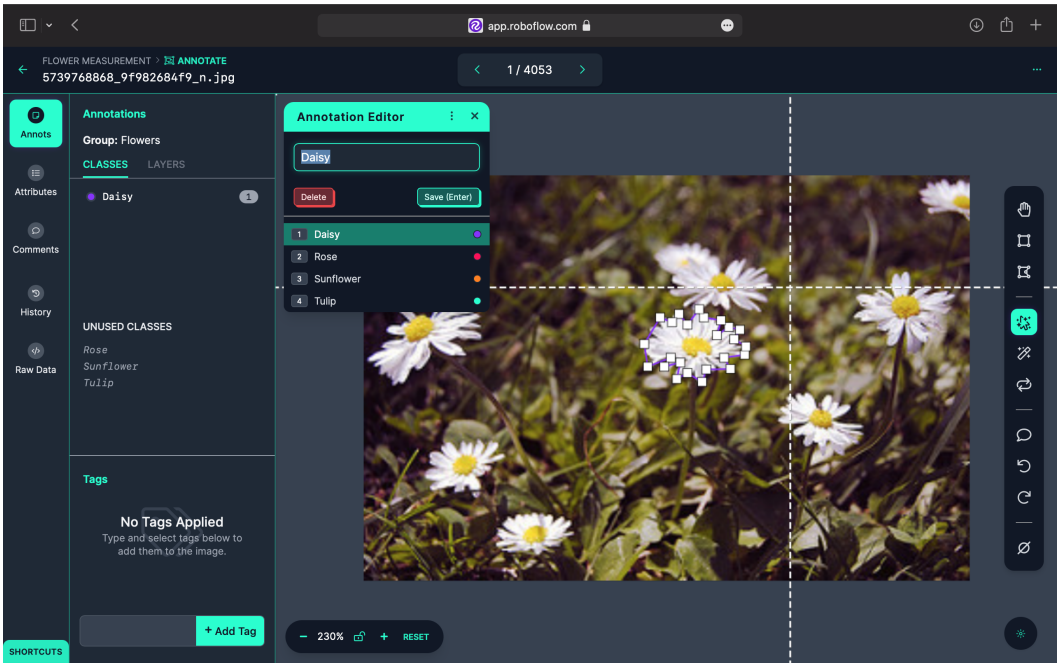



Figure 3.2: Example of labelling in roboflow

### 3.1.2 Image pre-processing

In stereo image processing, downsampling or downscaling the resolution of images is a common practice to improve computation speed. The original high-resolution images, such as 2784x1856 pixels, are reduced in size to a smaller resolution, like 320x240 pixels. By doing so, the computational load is reduced, as there are fewer pixels to process and analyze. This downsampling step is particularly useful in real-time applications or situations where quick processing is required.

Additionally, converting the images from RGB color space to green scale is another technique employed in stereo image processing. Grayscale images represent each pixel with a single intensity value, whereas RGB images represent each pixel with three color

channels (red, green, and blue). By converting the images to green scale, the computational complexity is further reduced since only one channel needs to be processed. An example and comparison between grey scaling and green scaling is shown in figure 3.3 with RGB, greyscale and green scale are arranged respectively.

Figure 3.3: Example of greyscalling

While there is some loss of information when converting to grayscale, particularly in terms of color details, it does not significantly affect the accuracy of distance measurements. The distance measurement in stereo vision primarily relies on the disparities between corresponding pixels in the stereo images, which are calculated based on the differences in pixel intensity or brightness. This disparity calculation is based on the variations in pixel intensity rather than color information. Therefore, converting the images to grayscale or green scale color space does not have a significant impact on the accuracy of distance measurement, as long as the brightness and intensity information necessary for calculating disparities are preserved.

In conclusion, downsampling the image resolution and converting the images to

grayscale or green scale color space are techniques used to improve the computation speed in stereo image processing. While there is some loss of color information, it does not have a substantial effect on the accuracy of distance measurements, as distance estimation primarily relies on disparities calculated from pixel intensity variations. These techniques allow for faster and more efficient processing of stereo images without compromising the accuracy of distance measurements.

### 3.1.3 Model training

## 3.2 Dual camera setup


Figure 3.4: Dual camera setup

The first component of the setup is two cameras, preferably the same kind of camera. In this case, two esp32-cam was used including its cam-mb board. The reason to why esp32 was used is because they are the cheapest option with the best features available. The cameras are better than arduino cameras and they have build-in WiFi module which has higher reliability rate compared to an arduino with a separate WiFi module. A reaspberrypi can be said has the same features but the price is almost 8

times more expensive and the camera qualities do not great difference. With all that into consideration, buying two esp32-cam is more worth it.

### 3.2.1 Setup schematic diagram

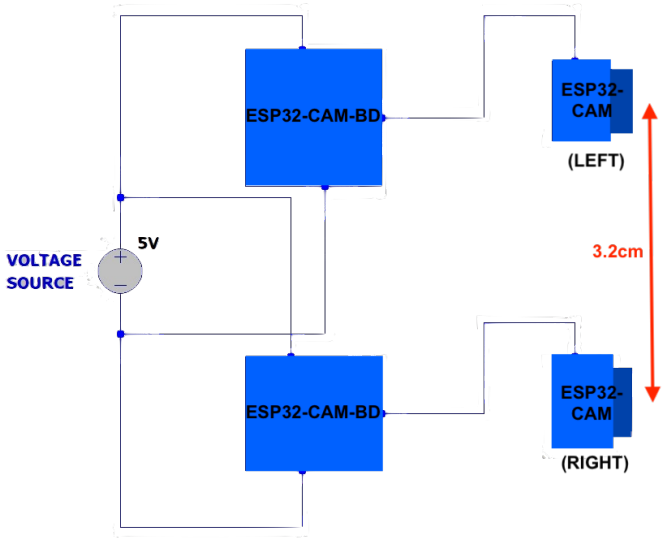
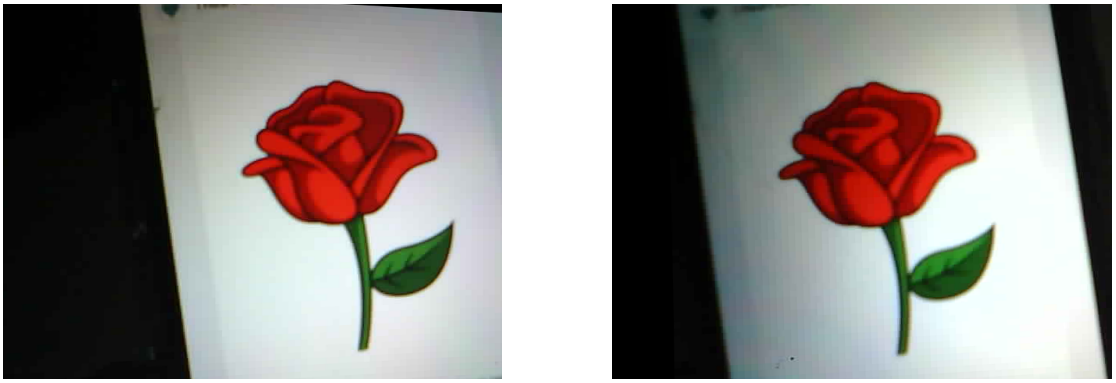



Figure 3.5: Setup schemetics

Figure 3.5 shows the schematic diagram of the dual camera setup. On each cameras, the setup are made of a power source, an esp32-CAM-BD to give power as well as embed the c++ code into the camera calles esp32-CAM, as well as the esp32-CAM itself which has the lense and the basic camera needs to take a picture. This camera will connect to the laptop wirelessly via local area network and the picture frames will be captured by a pyhton code run by the laptop.

Another needed component is a casing to keep both cameras in place. The casing must be hard so that the cameras wont move around once it has been plastered on it using Tack-Its. Two holes has been punctured through it to allow wires to pass through to power up both of the esp32-cam. The setup is shown in figure3.4 with the distance between cameras are roughly 3.1cm. Simply speaking, the esp32-cam was programmed using cpp programming language on PlatformIO which makes them stream the images

into a localhost web. The images will then be extracted using python code via localhost http protocol.



(a) Image L (b) Image R

Figure 3.6: Dual camera testing

## 3.3 Stereo Image Calibration

To calibrate, first we will need few left and right set of images taken using our prototype. It is a common practice to use checkerboards to calibrate our cameras since python openCV has a set of functions to calibrate cameras using chekerboard images (refer to the python code B). The functions mentioned were findChessboardCorners(), cornerSubPix(), drawChessboardCorners() and lastly calibrateCamera(). After running all the functions, the code will produce each cameras' internal characteristics. The characteristics of the left and right cameras were compared and depth map was generated from it. In this experiment 6 pictures were taken in total for the calibration process. The images were stored inside the laptop on separate left and right folders before it is being reused to calibrate. Figure 3.7 is the comparison (stereo matching) of rectified left and right images.

## 3.4 Stereo image processing

### 3.4.1 Object detection and disparity

By using the CNN model that was trained using roboflow, our python code can now predict the location of the objects inside both of the images. The trained model

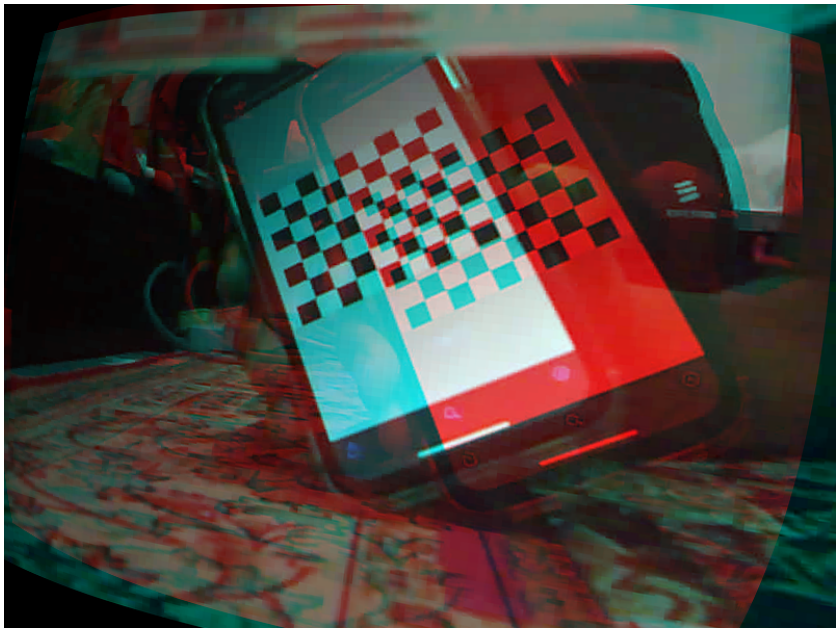


Figure 3.7: Comparison of rectified left and right images

will generate the coordinates of the objects. The coordinates obtained will be used to calculate the disparity of the cameras. The formula of disparity are as equation 3.1 below:

$$disparity = centerL - centerR \tag{3.1}$$

Where centerL and centerR is are the center points of the objects on the left and right image repectively.

### 3.4.2 Distance estimation

Now that we have obtained disparity, estimating distance is as easy as plugging the value into the formula 3.2([3]). Where FocalLength is the value of focal length obtained after stereo calibration and the DistanceBetweenCamera is the physical distance between the two esp32-cam in the setup.

$$Distance = FocalLength \times \frac{DistanceBetweenCamera}{disparity} \tag{3.2}$$

### 3.4.3 Size of object

Size in this case is refered to the area of the object. We can get the area of the object in the images in pixel units by first calculate the total perimeter of the flower. There are no direct formula to calculate area from only perimeter therefore we have to assume every flower's area is close to an area of a circle. This could be done by using the width value we got from the CNN model. This width value unit will be in pixel therefore we can calculate the area of the flower by suing formula of circle in formula3.3 where d is the diameter or width of the flower.

$$Area = \frac{1}{4} \times \pi \times d^2 \tag{3.3}$$

### 3.4.4 Actual area of object

There are 2 parts in calculating the actual real life area of an object. We will have to calculate two scale factors, one for the conversion of area in pixel to area in cm and another one is to re-scale the first scale factor to include the distance of the object from the cameras. The scale factors are derived from the similar triangle theory (see fig 3.8).

$$\frac{XY}{AB} = \frac{YZ}{BC} = \frac{XZ}{AC} = k$$

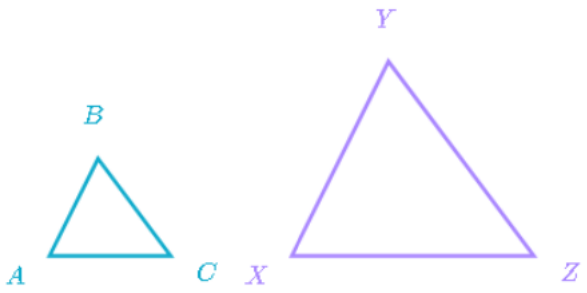



Figure 3.8: Similar triangle concept

Scale factor is a mathematical concept that plays a crucial role in describing the proportional relationship between corresponding linear measurements of two similar figures. When dealing with similar geometric shapes, such as triangles or rectangles,

the scale factor is defined as the ratio of the lengths of corresponding sides. Specifically, if you have two similar figures, A and B, with corresponding sides a a and b b, the scale factor k k is represented by the ratio a b b a .

It's important to note that the scale factor is a unitless quantity, as it signifies a ratio of lengths. In the context of similar figures, the scale factor establishes a proportional relationship between corresponding sides. This means that if two figures are similar, the ratios of the lengths of their corresponding sides remain constant.

The application of the scale factor is particularly evident in scenarios involving the enlargement or reduction of geometric figures. When you multiply all the lengths of a figure by a scale factor greater than 1, you obtain an enlarged version of the figure. Conversely, if you multiply by a scale factor between 0 and 1, you achieve a reduced version of the figure. This principle is essential in various geometric contexts, allowing for the precise manipulation of shapes based on their proportional relationships.

To illustrate, consider two similar triangles with corresponding sides ab and xy. If ab=4 and xy=2, the scale factor k is 4/2 = 2. This implies that every length in the larger triangle is twice the length of the corresponding side in the smaller triangle. Understanding and applying the scale factor is fundamental in geometry, providing a basis for solving problems related to the enlargement, reduction, and proportional relationships between lengths in similar figures.

To find the first scale factor of pixel to centimeters, we can use the pixel value of the object in the image divided by the actual known value of the object and we will get the scale factor.

$$ScaleFactor1 = \frac{AreaInPx}{AreaInCm} \tag{3.4}$$

The second scale factor is derive by comparing the first scale factor with the distance of the object from the camera. Hence it is a repetitive process using formula 3.5 until we reach a reliable trend line.

$$ScaleFactor2 = \frac{ScaleFactor1(i) - ScaleFactor1(i+1)}{distance(i) - distance(i+1)} \tag{3.5}$$

$$ScaleFactor3 = distance \times ScaleFactor2 \tag{3.6}$$

Lastly, we can get a dynamic scale factor which changes as the distance of the cameras to the object changes. With this ScaleFactor3 we now do not need to worry about updating a the scale factor every time we take a picture since it will automatically be recalculated dependent on the distance. Therefore, theoretically we can estimate the actual area of an object no matter how far or close the object is to our stereo camera. The table below 3.1 explains the differences between the three types of scale factors:

| Scale Factor | Function |
|---|---|
| 1 | To calculate the scale comparing the conversion of pixel to real life measurement units (cm in this case). |
| 2 | To find the average differences of scale factor 1 dependent on the distance of the respective scale factor. |
| 3 | To find the actual scale factor in the specific distance (just like scale factor 1 but it changes as distance changes). |

Table 3.1: Comparison of Scale Fators

### 3.5 Gantt Chart

The Gantt Chart contains this whole final year project timeline (both FYP1 and FYP2). As can be seen its starts from week 1 and stretches all the way to week 25. During FYP1(up until week 12), everything that has been done was literature reviews as a huge amount of time were needed to understand the topics relating to this project such as Machine Learning and Stereo Image Processing. The implementation of all the knowledge gained back in FYP1 brought to the start of FYP2 where we started building the stereo camera setup and we started writing codes for the project.

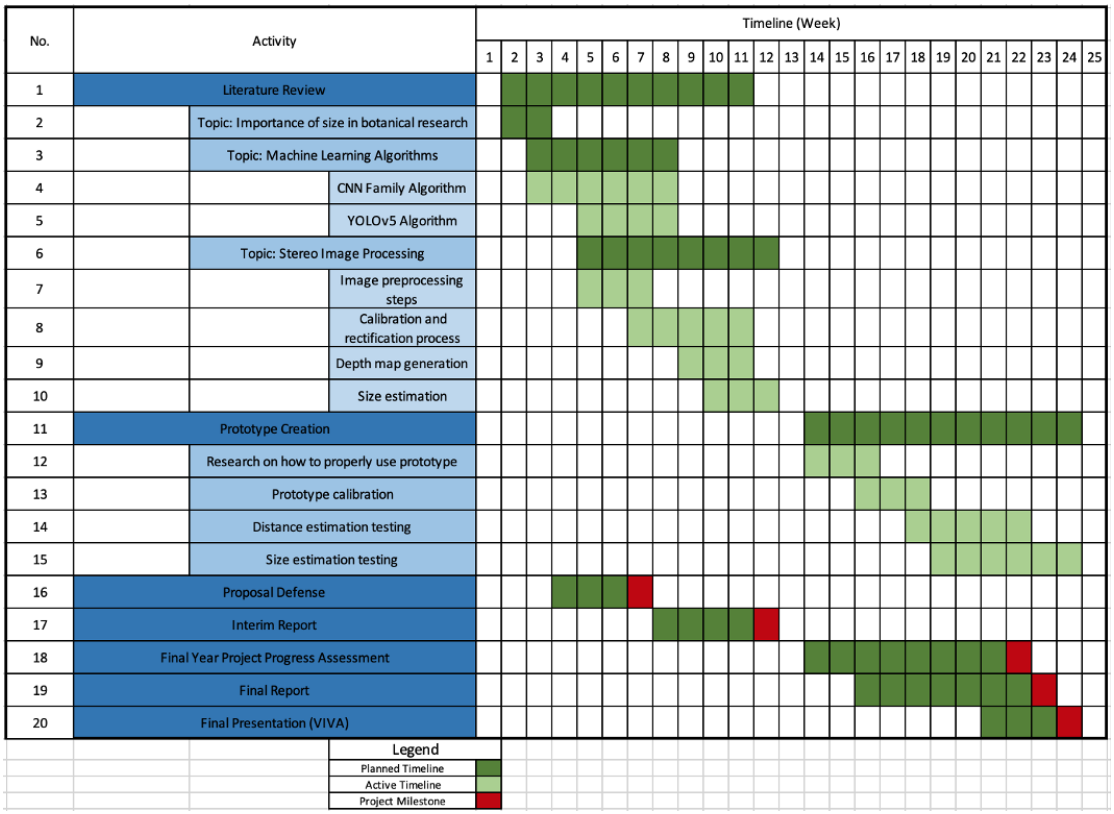



Figure 3.9: Gantt Chart

# CHAPTER 4

# RESULTS AND DISCUSSION

The results of this project can be summarized into three(3) milestones. The first one is the accuracy of the distance estimation, second is scale factor generated by comparing real and digital and lastly is the area of the object in real life depending on the scale factor. Figure 4.1 shows the example output of russing the scale factor script written in python.


Figure 4.1: Example output of measuring the distance and scale factor

## 4.1 Machine learning

The graphs in figure 4.2 are loss function graphs. They show how the loss function changes over time during training. Each line corresponds to a different type of loss:

- train/box_loss: This measures how well the model predicts the bounding boxes of objects in the training data.
- train/seg_loss: This measures how well the model predicts the segmentation of objects in the training data.

- train/cls_loss: This measures how well the model classifies objects in the training data.

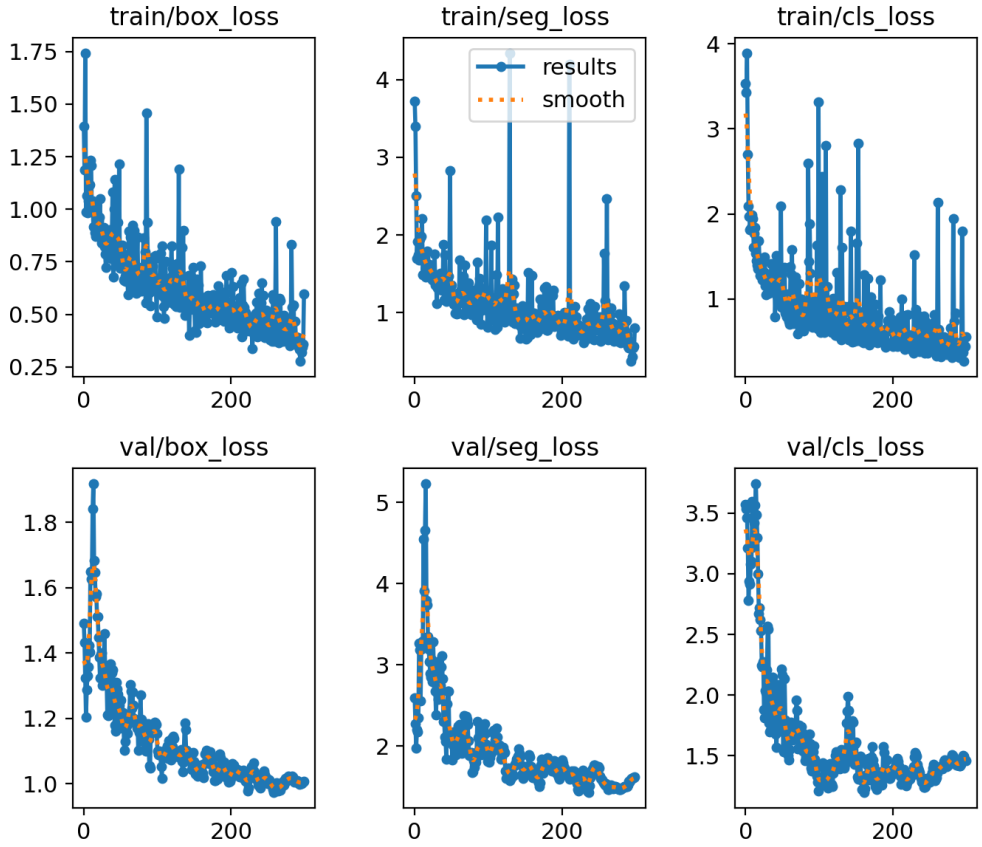



Figure 4.2: Results of model training (1)

The loss function is a measure of how bad the model's predictions are. The goal of training is to minimize the loss function. As the model trains, it learns to make better predictions, and the loss function should decrease.

The graphs show that the training loss is decreasing over time for all three types of loss. This means that the model is learning to make better predictions. The validation loss is also decreasing, which means that the model is generalizing well to unseen data. Overall, these graphs suggest that the model is training well and is likely to perform well on new data.

The figure 4.3 shows a series of scatter plots that track the changes in precision and recall metrics for both the border (B) and the mask (M) over 200 training epochs. Each plot has two lines, one for precision and one for recall.

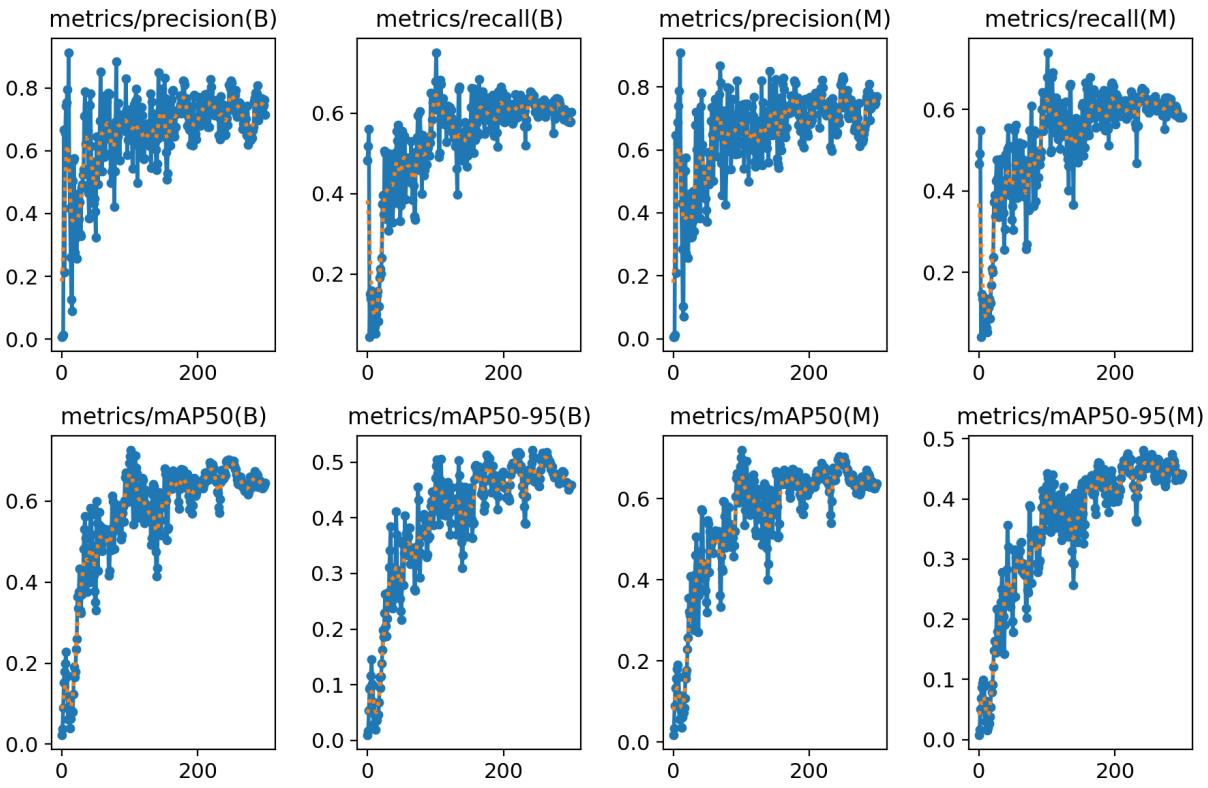



Figure 4.3: Results of model training (2)

Precision refers to the ability of the model to correctly identify relevant pixels (i.e., pixels that belong to the object of interest). A higher precision value indicates that the model is less likely to include irrelevant pixels in its predictions.

Recall refers to the ability of the model to identify all relevant pixels (i.e., not missing any pixels that belong to the object of interest). A higher recall value indicates that the model is less likely to miss relevant pixels in its predictions.

Here's a more detailed breakdown of each plot:

- metrics/precision(B) and metrics/recall(B): These plots track the precision and recall of the model for the border pixels. We can see that both precision and recall start high and remain relatively stable throughout the training process. This suggests that the model is good at identifying and segmenting the border of the object of interest.

- metrics/precision(M) and metrics/recall(M): These plots track the precision and

recall of the model for the mask pixels. We can see that both precision and recall start lower than for the border pixels, but they increase steadily throughout the training process. This suggests that the model initially has more difficulty identifying and segmenting the mask pixels, but it learns to do better as training progresses.

- metrics/mAP50(B) and metrics/mAP50-95(B): These plots track the mean average precision (mAP) at 50% and 95% intersection over union (IoU) thresholds for the border pixels. mAP is a commonly used metric for evaluating object detection performance. Here, we see that both mAP50 and mAP50-95 remain high throughout training, further confirming the model's good performance on border detection.

- metrics/mAP50(M) and metrics/mAP50-95(M): These plots track the mAP at 50% and 95% IoU thresholds for the mask pixels. Similar to the border mAP, both mask mAP50 and mAP50-95 increase steadily throughout training. This indicates that the model is improving its ability to segment the mask of the object of interest.

All in all, the graphs suggest that the model is performing well on both border and mask detection tasks. The precision and recall values are high for both B and M, and the mAP values are also good. This indicates that the model is able to accurately identify and segment both the border and the mask of the object of interest.

## 4.2 Distance estimation

The results of distance calculation was recorded in table 4.1. As can be seen, the expriments consists of 7 separate reading increments from 20cm to 50cm with 5cm difference. By using formula 3.2 using the disparity calculated in the calibration section.

By referring to the graph 4.4, at the moment the accuracy of the distance estimation began to decrease exponentially once it went above 35cm. This accuracy decrease may cause because of lack of proper calibration of the dual camera or it may be due to

| Actual Distance(cm) | Estimated Distance(cm) | Error% |
|---|---|---|
| 20 | 20 | 0 |
| 25 | 24 | 4 |
| 30 | 29 | 3.33 |
| 35 | 33 | 5.71 |
| 40 | 35 | 12.5 |
| 45 | 39 | 13.33 |
| 50 | 45 | 10 |

Table 4.1: Distance estimation table

inaccuracy in the prediction using the current machine learning model.

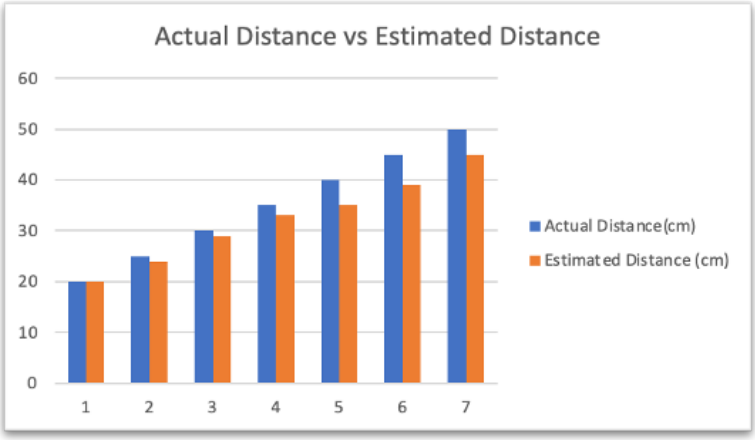



Figure 4.4: Actual vs Estimated distance graph

## 4.3 Scale Factor

An example output for the calculation of the scale factor is also in figure 4.1. As stated in the previous chapter, there will be 2 scale factors, one is dynamic which will be used to scale pixel into cm and another one is the average scale factors calculated to find the first scale factor as distance changes.

Scale factor can be calculated using formula 3.4. Now lets take the values of the first measurements. We have width equals to 171px and the actual width of the Daisy measured by a ruler is 5.2cm, plug those in formula 3.4 we will get 0.0304. The rest of the scale factors are depicted in table 4.2.

Now we find the difference between the second iteration with the first, the third iteration with the second and so on (refer eq 3.5) and we will get 0.005827, 0.0069165,

| Estimated width(cm) | Scale Factor | |
|---|---|---|
| 171 | 0.0304 | |
| 143 | 0.0362 | |
| 120 | 0.0432 | |
| 107 | 0.0486 | |
| 89 | 0.0584 | |
| 81 | 0.0638 | Actual Width(cm) |
| 71 | 0.0732 | 5.2 |

Table 4.2: scale factor table

0.0054445, 0.009829, 0.005373 and 0.009439 respectively. After that we simply have to find the average scale factor and that will be the second scale factor.

Now, with combining what we got and so far with formula 3.6, we can get the actual values from the pixel value of the object using this new formula (4.1):

$$realvalues = ValuesInPixel \times distance \times ScaleFactor2 \tag{4.1}$$

## 4.4 Area estimation

| Actual Distance (cm) | Estimated Distance (cm) | width (px) | area ($cm^2$) |
|---|---|---|---|
| 20 | 19 | 177 | 28 |
| 25 | 25 | 138 | 28 |
| 30 | 28 | 119 | 27 |
| 35 | 34 | 99 | 26 |
| 40 | 37 | 89 | 25 |
| 45 | 39 | 82 | 24 |
| 50 | 43 | 76 | 25 |

Table 4.3: Relationship of distance, width and area

Based on table 4.3, we can deduce that the the values are approximately correct in the range of +-4. Further analysis of the scale factors should be made to achieve more accurate measurements. The errors were dependant on many sources such as inaccuracy of model prediction, imperfect calibration of the dual cameras, human errors on measuring the actual size of the flowers and so on. On another notes there are many ways to minimize errors such as having many takes of the same measurements, using an ojbect with well known size, training the model with more dataset, etc.

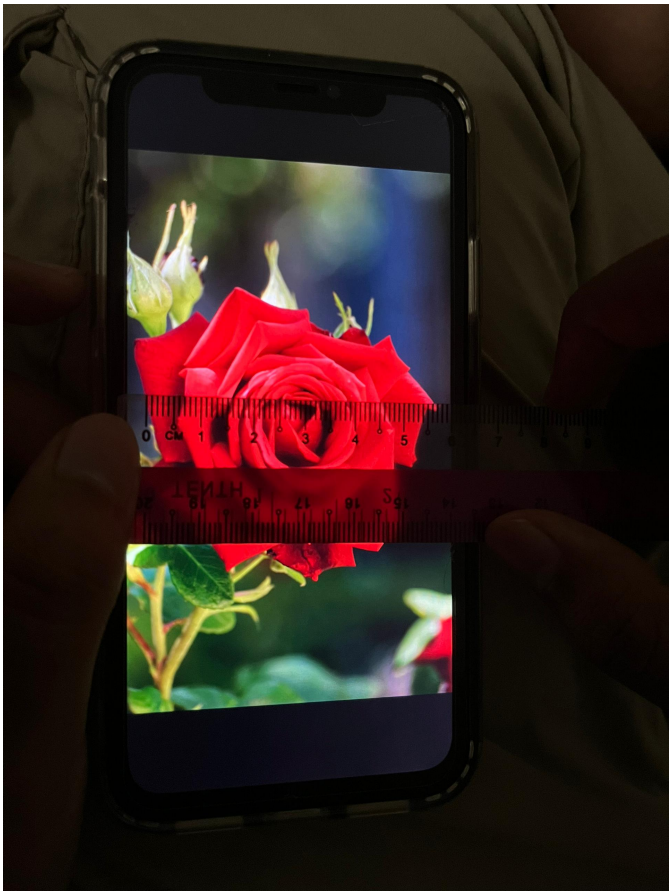


Figure 4.5: Measurement of the actual width of the object

Before we start estimating the area of the object, we need to measure the actual area of the object in real life. The measurements were taken as shown in figure 4.5. In this case, the actual width is around 5.6cm, calculating the area with formula 3.3 we will get an area of 24.63cm. Figure 4.6 shows the example output of how the final product of this project will look like.

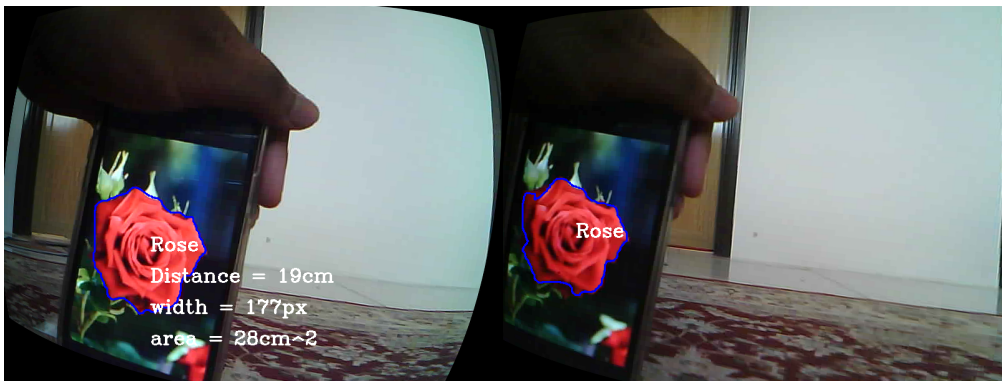



Figure 4.6: Example output of the final product

# CHAPTER 5

# CONCLUSION AND RECOMMENDATION

## 5.1 Conclusion

In conclusion, in this project we got to train our own machine learning model, use it to predict our own dataset of choice, set up our own Stereo camera and then use it to estimate distance and size of a specific object. A manual setup of stereo cameras can always be calibrated to find the most accurate rectification values by playing around with the calibration parameters and a ton of try in errors. With correct and accurate depth map generation, estimating the distance from our stereo camera setup to the object will not be a problem as long as we get to identify the location of the object in both 'eyes' of the cameras.

Machine Learning or Artificial Intelligence are no longer foreign to programmers and engineers, we could also say its a tool that everyone in this field should be familiar with. Due to that, identifying an object in an image is a side quest to this project and estimating distance accurately is very much possible with the existence of disparity. Lastly, estimating size of object was the biggest challenge in this project. Instance segmentation is a very complex computer vision training but thanks to the algorithm, we could estimate the area of an object in pixel which then can be converted to centimeters.

## 5.2 Recommendation

To start off, if this project was given more time, the size of the dataset that would be used for CNN modeling would increase to more that 1000. This is a crucial step as the more data is fed into a machine learning will mean an increase of accuracy in

our machine learning prediction. With an increased prediction confidence, it will surely increase the overall quality of the project.

Another recommendation on machine learning is to use active learning on our dynamic scale factor. As can be seen in the results and discussion chapter 4, our scale factor 2 was actually made using simple linear regression which is a subtopic in machine learning or artificial intelligence. taking more scalling examples with more diversed range of distance will surely improve our dynamic scale factor.

In stereo imaging, the depth map plays a very crucial part in ensuring the best and most accurate estimation of distance as well as size. A proper camera calibration is needed to achieve the best depth map generation. One recommendation that could be suggested to improve this is to have a proper workspace with proper objects and measuring equipment.

# REFERENCES


[1] M. Oswald D. Fernández-Fernández A. S. Dellinger, D. Hanusch and J. Schönenberger. Using geometric morphometrics to determine the “fittest” floral shape. 2023.

[2] J. Ghazoul et al C. J. Kettle, C.R. Maycock. Ecological implications of a flower size/number trade-off in tropical forest trees. 2011.

[3] Richeng Cheng. A survey: Comparison between convolutional neural network and yolo in image identification. *Journal of Physics: Conference Series*, 2020.

[4] F. K. Brad and E. Elizabeth. The reproductive assurance benefit of selfing: importance of flower size and population size.

[5] Y. Seung-jun K. Hansung and S. Kwanghoon. 3d reconstruction of stereo images for interaction between real and virtual worlds.

[6] S. Maha Y. Mohammad H. Hazem, S. Heba. Flower classification using deep convolutional neural networks.

[7] C. Meng-Che H. Jenq-Neng T. Rick W. Kresimir, L. Nathan. Automated measurements of fish within a trawl using stereo images from a camera-trawl device (camtrawl).

[8] H. Tsung-Han D. Yi-Chun, M. Muslikhin and W. Ming-Shyan. Stereo vision-based object recognition and manipulation by regions with convolutional neural network.

[9] H. Hazem A. Huthaifa, M. Saher. A flower recognition system based on image processing and neural networks.

[10] H. Hasbullah W. A. Amelia M. Yasir, N. Rahizall. Stereo vision images processing for real-time object distance and size measurement.

[11] J. Corse. Basic stereo epipolar geometry. *Electrical Engineering and Computer Science at the University of Michigan*, 2014.

[12] IEEE Yide Ma Dong Hwan Kim Yuli Chen, Member and Sung-Kee Park. Region-based object recognition by color segmentation using a simplified pcnn. *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS*, 2015.

[13] Christian Kollmitzer. Object detection and measurement using stereo images. *Electronic Engineering*.

[14] Huimin Lv Shigang Cui. Preliminary determination of the length of the main vein based on image processing.

[4] [1] [5] [6] [2] [7] [8] [9] [10] [11] [12] [3] [13] [14]

APPENDIX A

EXPERIMENT IMAGES USED

## A.1 Checkerboard used

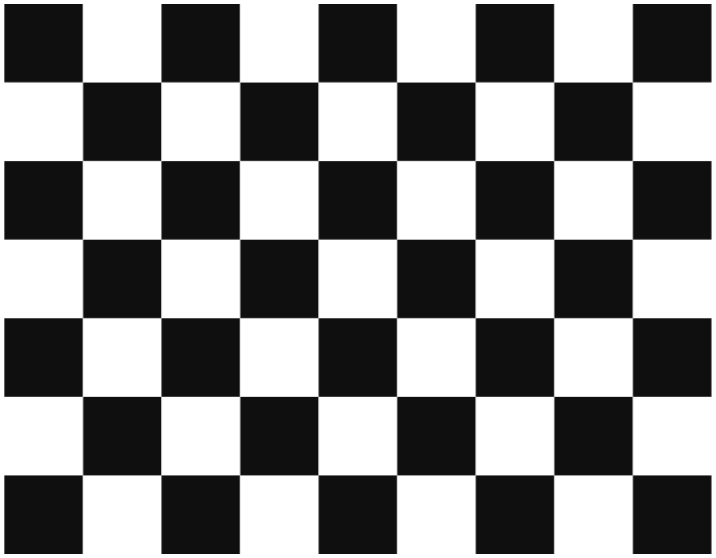


Figure A.1: 8x6 checkerboard used in calibration

## A.2 Images of flowers for testing

Figure A.2: Image of daisy

Figure A.3: Image of sunflower

Figure A.4: Image of rose

# APPENDIX B

# PROJECT CODES

## B.1 Python code for stereo calibration and rectification

```
# https://learnopencv.com/camera-calibration-using-opencv/
import cv2
import numpy as np
import glob
from tqdm import tqdm

# Define size of chessboard target.
a = 8
b = 6
chessboard_size = (a, b)
objp = np.zeros((b * a, 3), np.float32)
objp[:, :2] = np.mgrid[0:a, 0:b].T.reshape(-1, 2)

# Define arrays to save detected points
obj_points = []  # 3D points in real world space
img_points = []  # 3D points in image plane
# Prepare grid and points to display
objp = np.zeros((np.prod(chessboard_size), 3), dtype=np.float32)
objp[:, :2] = np.mgrid[0:chessboard_size[0], 0:chessboard_size[1]].T.reshape(-1,
↪  2)

pathL = "./images/Stereo/Calibration/L/"
pathR = "./images/Stereo/Calibration/R/"

# Termination criteria for refining the detected corners
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

img_ptsL = []
img_ptsR = []
obj_pts = []

```

```
for i in tqdm(range(1, 7)): # change based on number of pictures taken for the calibration
    imgL = cv2.imread(pathL + "%d.png" % i)
    imgR = cv2.imread(pathR + "%d.png" % i)
    imgL_gray = cv2.cvtColor(imgL, cv2.COLOR_BGR2GRAY)
    imgR_gray = cv2.cvtColor(imgR, cv2.COLOR_BGR2GRAY)

    outputL = imgL.copy()
    outputR = imgR.copy()

    # Looks for Chessboard corners. Parameters: Image, Pattern Size of corners
    retL, cornersL = cv2.findChessboardCorners(imgL_gray, chessboard_size, None)
    retR, cornersR = cv2.findChessboardCorners(imgR_gray, chessboard_size, None)

    if retR and retL:
        obj_pts.append(objp)
        # takes original imahe, location of corners, looks for best corner location within range. Iterative, termination criteria needed
        cornersL2 = cv2.cornerSubPix(imgL_gray, cornersL, (11, 11), (-1, -1), criteria)
        cornersR2 = cv2.cornerSubPix(imgR_gray, cornersR, (11, 11), (-1, -1), criteria)
        cv2.drawChessboardCorners(outputL, chessboard_size, cornersL2, retL)
        cv2.drawChessboardCorners(outputR, chessboard_size, cornersR2, retR)
        cv2.imshow('cornersL',outputL)
        cv2.imshow('cornersR',outputR)
        cv2.waitKey(0)

        img_ptsL.append(cornersL)
        img_ptsR.append(cornersR)

print("Calculating left camera parameters ... ")
# Calibrating left camera
retL, mtxL, distL, rvecsL, tvecsL = cv2.calibrateCamera(obj_pts, img_ptsL, imgL_gray.shape[::-1], None, None)
hL, wL = imgL_gray.shape[:2]
new_mtxL, roiL = cv2.getOptimalNewCameraMatrix(mtxL, distL, (wL, hL), 1, (wL, hL))

print("Calculating right camera parameters ... ")
# Calibrating right camera
retR, mtxR, distR, rvecsR, tvecsR = cv2.calibrateCamera(obj_pts, img_ptsR, imgR_gray.shape[::-1], None, None)
hR, wR = imgR_gray.shape[:2]
new_mtxR, roiR = cv2.getOptimalNewCameraMatrix(mtxR, distR, (wR, hR), 1, (wR, hR))

print("Stereo calibration .....")
flags = 0
```

```
72  flags |= cv2.CALIB_FIX_INTRINSIC
73  # Here we fix the intrinsic camara matrixes so that only Rot, Trns, Emat and Fmat
    ↪  are calculated.
74  # Hence intrinsic parameters are the same
75
76  criteria_stereo = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
77
78  # This step is performed to transformation between the two cameras and calculate
    ↪  Essential and Fundamenatl matrix
79  retS, new_mtxL, distL, new_mtxR, distR, Rot, Trns, Emat, Fmat =
    ↪  cv2.stereoCalibrate(obj_pts,
80
                                                                    ↪  img_ptsL,
81
                                                                    ↪  img_ptsR,
82
                                                                    ↪  new_mtxL,
83
                                                                    ↪  distL,
84
                                                                    ↪  new_mtxR,
85
                                                                    ↪  distR,
86
                                                                    ↪  imgL_gray.
87
                                                                    ↪  criteria_s
88
                                                                    ↪  flags)
89
90  # Once we know the transformation between the two cameras we can perform stereo
    ↪  rectification
91  # StereoRectify function
92  rectify_scale = 1  # if 0 image croped, if 1 image not croped
93  rect_l, rect_r, proj_mat_l, proj_mat_r, Q, roiL, roiR =
    ↪  cv2.stereoRectify(new_mtxL, distL, new_mtxR, distR,
94
                                                                ↪  imgL_gray.shape[::-1
                                                                ↪  Rot,
                                                                ↪  Trns,
95
                                                                ↪  rectify_scale,
                                                                ↪  (0,
                                                                ↪  0))
96
97  print (Q)
98  focalLengthL = (mtxL[0, 0] +mtxL[1, 1])/2
```

```
focalLengthR = (mtxR[0, 0] +mtxR[1, 1])/2
focalLength = (focalLengthL, focalLengthR)

# Use the rotation matrixes for stereo rectification and camera intrinsics for undistorting the image
# Compute the rectification map (mapping between the original image pixels and
# their transformed values after applying rectification and undistortion) for left and right camera frames
Left_Stereo_Map = cv2.initUndistortRectifyMap(new_mtxL, distL, rect_l, proj_mat_l,
                                              imgL_gray.shape[::-1], cv2.CV_16SC2)
Right_Stereo_Map = cv2.initUndistortRectifyMap(new_mtxR, distR, rect_r, proj_mat_r,
                                               imgR_gray.shape[::-1], cv2.CV_16SC2)

print("Saving parameters ......")
cv_file = cv2.FileStorage("/Users/alif/Desktop/flower/MonoParam7.xml", cv2.FILE_STORAGE_WRITE)
cv_file.write("Left_Stereo_Map_x", Left_Stereo_Map[0])
cv_file.write("Left_Stereo_Map_y", Left_Stereo_Map[1])
cv_file.write("Right_Stereo_Map_x", Right_Stereo_Map[0])
cv_file.write("Right_Stereo_Map_y", Right_Stereo_Map[1])
cv_file.write("Focal_Length", focalLength)
cv_file.write('Q', Q)
cv_file.write('T', Trns)
print(Trns)
print(focalLength)
print("Saved Parameters!")
cv_file.release()
cv2.destroyAllWindows()

# Show Rectified image
cv2.imshow("Images before rectification", np.hstack([imgL, imgR]))

Left_nice = cv2.remap(imgL, Left_Stereo_Map[0], Left_Stereo_Map[1], cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
Right_nice = cv2.remap(imgR, Right_Stereo_Map[0], Right_Stereo_Map[1], cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)

cv2.imshow("Images after rectification", np.hstack([Left_nice, Right_nice]))
cv2.waitKey(0)
cv2.destroyAllWindows()

out = Right_nice.copy()
out[:, :, 0] = Right_nice[:, :, 0]
out[:, :, 1] = Right_nice[:, :, 1]
out[:, :, 2] = Left_nice[:, :, 2]
```

```
cv2.imwrite("/Users/alif/Desktop/flower/images/Stereo/Calibration/rectifiedL.png",Left_nice)
cv2.imwrite("/Users/alif/Desktop/flower/images/Stereo/Calibration/rectifiedR.png",Right_nice)
cv2.imwrite("/Users/alif/Desktop/flower/images/Stereo/Calibration/output.png",
↪  out)

cv2.imshow("Output image", out)
cv2.waitKey(0)
```

-

## B.2 C plus+ code embedded on esp32-cam

```
#include <WebServer.h>
#include <WiFi.h>
#include <esp32cam.h>
#include <Arduino.h>

const char* WIFI_SSID = "Leplep";
const char* WIFI_PASS = "yes dapat wifi";

WebServer server(80);


static auto loRes = esp32cam::Resolution::find(320, 240);
static auto midRes = esp32cam::Resolution::find(350, 530);
static auto hiRes = esp32cam::Resolution::find(800, 600);
void serveJpg()
{
  auto frame = esp32cam::capture();
  if (frame == nullptr) {
    Serial.println("CAPTURE FAIL");
    server.send(503, "", "");
    return;
  }
  Serial.printf("CAPTURE OK %dx%d %db\n", frame->getWidth(), frame->getHeight(),
                static_cast<int>(frame->size()));

  server.setContentLength(frame->size());
  server.send(200, "image/jpeg");
  WiFiClient client = server.client();
  frame->writeTo(client);
}

void handleJpgLo()
{
```

```
34    if (!esp32cam::Camera.changeResolution(loRes)) {
35      Serial.println("SET-LO-RES FAIL");
36    }
37    serveJpg();
38  }
39
40  void handleJpgHi()
41  {
42    if (!esp32cam::Camera.changeResolution(hiRes)) {
43      Serial.println("SET-HI-RES FAIL");
44    }
45    serveJpg();
46  }
47
48  void handleJpgMid()
49  {
50    if (!esp32cam::Camera.changeResolution(midRes)) {
51      Serial.println("SET-MID-RES FAIL");
52    }
53    serveJpg();
54  }
55
56
57  void  setup(){
58    Serial.begin(115200);
59    Serial.println();
60    {
61      using namespace esp32cam;
62      Config cfg;
63      cfg.setPins(pins::AiThinker);
64      cfg.setResolution(hiRes);
65      cfg.setBufferCount(2);
66      cfg.setJpeg(80);
67
68      bool ok = Camera.begin(cfg);
69      Serial.println(ok ? "CAMERA OK" : "CAMERA FAIL");
70    }
71    WiFi.persistent(false);
72    WiFi.mode(WIFI_STA);
73    WiFi.begin(WIFI_SSID, WIFI_PASS);
74    while (WiFi.status() != WL_CONNECTED) {
75      delay(500);
76    }
77    Serial.print("http://");
78    Serial.println(WiFi.localIP());
79    Serial.println("  /cam-lo.jpg");
80    Serial.println("  /cam-hi.jpg");
```

```
81    Serial.println("  /cam-mid.jpg");
82
83    server.on("/cam-lo.jpg", handleJpgLo);
84    server.on("/cam-hi.jpg", handleJpgHi);
85    server.on("/cam-mid.jpg", handleJpgMid);
86
87    server.begin();
88  }
89
90  void loop()
91  {
92    server.handleClient();
93  }
```

-

## B.3 Python code to calculate scale factor

```
1   import cv2, math
2   import urllib.request
3   import numpy as np
4   from roboflow import Roboflow
5
6   #  Define known variables
7   count = 0
8   KNOWN_FOCAL_LENGTH = 758.5 # average focal length obtained in calibration.py
9   KNOWN_DISTANCE_BETWEEN_CAMERA = 3.1 # distance of camera in cm
10  SCALE_FACTOR = 0.00324
11
12  #  Define cameras left and right
13  cam1='http://172.20.10.18/cam-hi.jpg' #Left camera
14  cam2='http://172.20.10.20/cam-hi.jpg' #Right camera
15
16  #  Specify folders to save pictures taken by both cameras
17  folderL = r'./images/Stereo/Scale/L/'
18  folderR = r'./images/Stereo/Scale/R/'
19
20  #  Read the depth map xml file from calibration.py
21  cv_file = cv2.FileStorage("./MonoParam7.xml", cv2.FILE_STORAGE_READ)
22
23  Left_Stereo_Map_x = cv_file.getNode("Left_Stereo_Map_x").mat()
24  Left_Stereo_Map_y = cv_file.getNode("Left_Stereo_Map_y").mat()
25  Right_Stereo_Map_x = cv_file.getNode("Right_Stereo_Map_x").mat()
26  Right_Stereo_Map_y = cv_file.getNode("Right_Stereo_Map_y").mat()
27
```

```
28
29  def takeCurrentPic(L, R):
30      cv2.imwrite(folderL+'imageL%d.png'%count,L)
31      cv2.imwrite(folderR+'imageR%d.png'%count,R)
32      print("pictures taken.")
33
34  def objectDetection():
35
36      rf1 = Roboflow(api_key="QycZdcWgOjjB6XYdD6UG")
37      project1 = rf1.workspace().project("flower-measurement")
38      model1 = project1.version(1).model
39
40      predictL = model1.predict(folderL+'imageL%d.png'%count).json()
41
        ↪  model1.predict(folderL+'imageL%d.png'%count).save(folderL+'predictL%d.png'%count)
42      for pred in predictL['predictions']:
43          x1 = int(pred['x'])
44          y1 = int(pred['y'])
45          width1 = pred['width']
46          class1 = str(pred['class'])
47
48      # print(width1)
49
50      rf2 = Roboflow(api_key="QycZdcWgOjjB6XYdD6UG")
51      project2 = rf2.workspace().project("flower-measurement")
52      model2 = project2.version(1).model
53
54      predictR = model2.predict(folderR+'imageR%d.png'%count).json()
55
        ↪  model2.predict(folderR+'imageR%d.png'%count).save(folderR+'predictR%d.png'%count)
56      for pred in predictR['predictions']:
57          x2 = int(pred['x'])
58          y2 = int(pred['y'])
59          width2 = pred['width']
60          class2 = str(pred['class'])
61
62      if 'data1' in locals():
63          print(width1)
64          print(width2)
65          print("Object detected.")
66          return x1, x2, y1, y2, class1, class2, width1, width2
67      else:
68          print("No object present")
69          return None, None, None, None, None, None, None, None
70
71  def getDistance(disparity):
72      try:
```

```
            Distance = (KNOWN_FOCAL_LENGTH) * (KNOWN_DISTANCE_BETWEEN_CAMERA /
            ↪  disparity)
            if Distance != float('inf'):
                return Distance
            else:
                return 0
        except:
            return "No object detected"

def getArea(data):
    i = 0
    length = 0

    while(i<len(data)-1):
        dict1 = data[i]
        list1 = list(dict1.values())
        array1 = np.array(list1)
        x1 = array1[0]
        y1 = array1[1]

        dict2 = data[i+1]
        list2 = list(dict2.values())
        array2 = np.array(list2)
        x2 = array2[0]
        y2 = array2[1]

        length = length + (math.sqrt(pow((x2-x1),2)+pow((y2-y1),2)))

        i=i+1

    pie = (22/7)
    area = (1/4) * pie * pow(length,2)
    real_area = area * pow(0.0264583333,2)
    print("Area in pixel: %f" %area)
    print("Area in cm: %f" %real_area)

    return area

def showResults(classL,classR,xL, xR, yL, yR, distance,width):
    L = cv2.imread(folderL+'predictL%d.png'%count)
    R = cv2.imread(folderR+'predictR%d.png'%count)

    dL = "Distance = "+str(int(distance))+"cm"
    w = "width = "+str(int(width))+"px"
    actual_w = 5.2
    scale = "scale = "+str(float(actual_w/width))

```

```
119
        ↪  cv2.putText(L,classL,(int(xL),int(yL)),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
120
        ↪  cv2.putText(L,dL,(int(xL),int(yL+50)),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
121
        ↪  cv2.putText(L,w,(int(xL),int(yL+100)),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
122
        ↪  cv2.putText(L,scale,(int(xL),int(yL+150)),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
123
124
        ↪  cv2.putText(R,classR,(int(xR),int(yR)),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
125
126     LR = np.concatenate((L, R), axis=1)
127
128     cv2.imwrite(folderR+'scale%d.png'%count, LR)
129     cv2.imshow("predicted values",LR)
130
131 while True:
132 #  Get images from both cameras and save it to a frame
133     img_respL = urllib.request.urlopen(cam1)
134     imgnpL = np.array(bytearray(img_respL.read()),dtype=np.uint8)
135     frameL = cv2.imdecode(imgnpL,-1)
136     frameL = cv2.rotate(frameL, cv2.ROTATE_180)
137
138     img_respR = urllib.request.urlopen(cam2)
139     imgnpR = np.array(bytearray(img_respR.read()),dtype=np.uint8)
140     frameR = cv2.imdecode(imgnpR,-1)
141     frameR  = cv2.rotate(frameR, cv2.ROTATE_180)
142
143     #  Remap the frames using the depth map matrix
144     imgL_remapped = cv2.remap(frameL, Left_Stereo_Map_x, Left_Stereo_Map_y,
        ↪  cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
145     imgR_remapped = cv2.remap(frameR, Right_Stereo_Map_x, Right_Stereo_Map_y,
        ↪  cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
146
147     #  Display the remapped left and right views on a window
148     Hori = np.concatenate((imgL_remapped, imgR_remapped), axis=1)
149     cv2.imshow("live transmission", Hori)
150     key=cv2.waitKey(33)
151
152     if key==ord('k'):
153         count = count +1
154         #  1. Take current frame and save to folders. (including the models)
155         takeCurrentPic(imgL_remapped, imgR_remapped)
156         #  2. Use the model coordinates of lates taken picture
157         CoordinatesL, CoordinatesR, xCenter_L, xCenter_R, yCenter_L, yCenter_R,
            ↪  classL, classR, width1, width2 = objectDetection()
```

```
158             avg_width = (width1+width2)/2
159             if CoordinatesL and CoordinatesR != None:
160                 #  4. Calculate disparity with center of both objects in each images
                    ↪  images
161                 disparity = abs(xCenter_L - xCenter_R)
162                 print("disparity = %f"%disparity)
163                 #  4. Calculate distance using disparity
164                 distance = getDistance(disparity)
165                 print("Distance = %f"%distance)
166 
                    ↪  showResults(classL,classR,xCenter_L,xCenter_R,yCenter_L,yCenter_R,distance,avg_wi
167                 #key = cv2.waitKey(0)
168 
169         elif key==ord('l'):
170             cv2.destroyWindow("predicted values")
171 
172         elif key == ord('q'):
173             break
174 
175 cv2.destroyAllWindows()
176 print("Program closed")
```

-

## B.4 Python code of project final product script

```
1  import cv2, math
2  import urllib.request
3  import numpy as np
4  from roboflow import Roboflow
5 
6  #  Define known variables
7  count = 0
8  KNOWN_FOCAL_LENGTH = 758.5 # average focal length obtained in calibration.py
9  KNOWN_DISTANCE_BETWEEN_CAMERA = 3.1 # distance of camera in cm
10 SCALE_FACTOR = 0.00171316
11 
12 #  Define cameras left and right
13 cam1='http://172.20.10.18/cam-hi.jpg' #Left camera
14 cam2='http://172.20.10.20/cam-hi.jpg' #Right camera
15 
16 #  Specify folders to save pictures taken by both cameras
17 folderL = r'./images/Stereo/Real/L/'
18 folderR = r'./images/Stereo/Real/R/'
19 folderL = r'./images/Stereo/Real/L/'
```

```
20  folder = r'./images/Stereo/Real/'
21
22  #  Read the depth map xml file from calibration.py
23  cv_file = cv2.FileStorage("./MonoParam7.xml", cv2.FILE_STORAGE_READ)
24
25  Left_Stereo_Map_x = cv_file.getNode("Left_Stereo_Map_x").mat()
26  Left_Stereo_Map_y = cv_file.getNode("Left_Stereo_Map_y").mat()
27  Right_Stereo_Map_x = cv_file.getNode("Right_Stereo_Map_x").mat()
28  Right_Stereo_Map_y = cv_file.getNode("Right_Stereo_Map_y").mat()
29
30
31  def takeCurrentPic(L, R):
32      cv2.imwrite(folderL+'imageL%d.png'%count,L)
33      cv2.imwrite(folderR+'imageR%d.png'%count,R)
34      print("pictures taken.")
35
36  def objectDetection():
37
38      rf1 = Roboflow(api_key="QycZdcWgOjjB6XYdD6UG")
39      project1 = rf1.workspace().project("flower-measurement")
40      model1 = project1.version(1).model
41
42      predictL = model1.predict(folderL+'imageL%d.png'%count).json()
43
        ↪  model1.predict(folderL+'imageL%d.png'%count).save(folderL+'predictL%d.png'%count)
44      for pred in predictL['predictions']:
45          x1 = int(pred['x'])
46          y1 = int(pred['y'])
47          width1 = pred['width']
48          class1 = str(pred['class'])
49
50      # print(width1)
51
52      rf2 = Roboflow(api_key="QycZdcWgOjjB6XYdD6UG")
53      project2 = rf2.workspace().project("flower-measurement")
54      model2 = project2.version(1).model
55
56      predictR = model2.predict(folderR+'imageR%d.png'%count).json()
57
        ↪  model2.predict(folderR+'imageR%d.png'%count).save(folderR+'predictR%d.png'%count)
58      for pred in predictR['predictions']:
59          x2 = int(pred['x'])
60          y2 = int(pred['y'])
61          width2 = pred['width']
62          class2 = str(pred['class'])
63
64      if 'width1' and 'width2' in locals():
```

```
65          print(width1)
66          print(width2)
67          print("Object detected.")
68          return x1, x2, y1, y2, class1, class2, width1, width2
69      else:
70          print("No object present")
71          return None, None, None, None, None, None, None, None
72
73  def getDistance(disparity):
74      try:
75          Distance = (KNOWN_FOCAL_LENGTH) * (KNOWN_DISTANCE_BETWEEN_CAMERA /
            ↪  disparity)
76          if Distance != float('inf'):
77              return Distance
78          else:
79              return 0
80      except:
81          return "No object detected"
82
83  def getArea(width1,width2):
84      pie = (22/7)
85      width = (width1+width2)/2
86      area = (1/4) * pie * pow(width,2)
87      print("Area in pixel: %f" %area)
88
89      return area
90
91  def showResults(classL,classR,xL, xR, yL, yR, distance, width,area):
92      L = cv2.imread(folderL+'predictL%d.png'%count)
93      R = cv2.imread(folderR+'predictR%d.png'%count)
94
95      dL = "Distance = "+str(int(distance))+"cm"
96      w = "width = "+str(int(width))+"px"
97      actual_area = "area = "+str(int(area))+"cm^2"
98
99
        ↪  cv2.putText(L,classL,(int(xL),int(yL)),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
100
        ↪  cv2.putText(L,dL,(int(xL),int(yL+50)),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
101
        ↪  cv2.putText(L,w,(int(xL),int(yL+100)),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
102
        ↪  cv2.putText(L,actual_area,(int(xL),int(yL+150)),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2
103
104
        ↪  cv2.putText(R,classR,(int(xR),int(yR)),cv2.FONT_HERSHEY_COMPLEX,1,(255,255,255),2)
105
```

```
106         LR = np.concatenate((L, R), axis=1)
107 
108         cv2.imwrite(folder+'img%d.png'%count, LR)
109         cv2.imshow("predicted values",LR)
110 
111     while True:
112     #  Get images from both cameras and save it to a frame
113         img_respL = urllib.request.urlopen(cam1)
114         imgnpL = np.array(bytearray(img_respL.read()),dtype=np.uint8)
115         frameL = cv2.imdecode(imgnpL,-1)
116         frameL = cv2.rotate(frameL, cv2.ROTATE_180)
117 
118         img_respR = urllib.request.urlopen(cam2)
119         imgnpR = np.array(bytearray(img_respR.read()),dtype=np.uint8)
120         frameR = cv2.imdecode(imgnpR,-1)
121         frameR  = cv2.rotate(frameR, cv2.ROTATE_180)
122 
123         #  Remap the frames using the depth map matrix
124         imgL_remapped = cv2.remap(frameL, Left_Stereo_Map_x, Left_Stereo_Map_y,
            ↪  cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
125         imgR_remapped = cv2.remap(frameR, Right_Stereo_Map_x, Right_Stereo_Map_y,
            ↪  cv2.INTER_LANCZOS4, cv2.BORDER_CONSTANT, 0)
126 
127         #  Display the remapped left and right views on a window
128         Hori = np.concatenate((imgL_remapped, imgR_remapped), axis=1)
129         cv2.imshow("live transmission", Hori)
130         key=cv2.waitKey(33)
131 
132         if key==ord('k'):
133             count = count +1
134             #  1. Take current frame and save to folders. (including the models)
135             takeCurrentPic(imgL_remapped, imgR_remapped)
136             #  2. Use the model coordinates of lates taken picture
137             xCenter_L, xCenter_R, yCenter_L, yCenter_R, classL, classR, width1, width2
                ↪  = objectDetection()
138             if width1 and width2 != None:
139                 #  4. Calculate disparity with center of both objects in each images
                    ↪  images
140                 disparity = abs(xCenter_L - xCenter_R)
141                 print("disparity = %f"%disparity)
142                 #  4. Calculate distance using disparity
143                 distance = getDistance(disparity)
144                 print("Distance = %f"%distance)
145                 #  5. Calculate area of ROI
146                 area = getArea(width1,width2)
147                 avg_width = (width1 + width2)/2
148                 #  6. Write a formula to estimate the flower actual surface area
```

```
149             actual_area = area * math.pow(SCALE_FACTOR*distance,2) #
                ↪  math.pow(distance,2)
150             print("estimated area = %f"%actual_area)
151
                ↪  showResults(classL,classR,xCenter_L,xCenter_R,yCenter_L,yCenter_R,distance,avg_wi
152             #key = cv2.waitKey(0)
153
154     elif key==ord('l'):
155         cv2.destroyWindow("predicted values")
156
157     elif key == ord('q'):
158         break
159
160 cv2.destroyAllWindows()
161 print("Program closed")
```

-