Stereo Image Calibration for Size Measurement

by

Timothy Ling Jit Huong

17008605

Dissertation submitted in partial fulfillment of the requirements for the

Bachelor of Engineering (Hons)

(Computer Engineering)

JAN 2023

Universiti Teknologi PETRONAS

Bandar Seri Iskandar

31750 Tronoh

Perak Darul Ridzuan

CERTIFICATIONS

CERTIFICATION OF APPROVAL

Stereo Image Calibration for Size Measurement

by

Timothy Ling Jit Huong
17008605

A project dissertation submitted to the
Computer Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(COMPUTER ENGINEERING)

Approved by,

_____

(Lila Iznita Izhar - Dr)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
JAN 2023

CERTIFICATION OF ORIGINALITY

This is to certify that I am not 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.

_____

TIMOTHY LING JIT HUONG

ACKNOWLEDGEMENT

Firstly, I would like to express my sincerest thanks to Dr. Lila Iznita Izhar for giving me the opportunity to work on the project under her supervision. Her guidance, suggestions and feedback throughout the project had a huge influence on the outcome of the project, and her constant and unconditional support has been invaluable. Her expertise in the field of image processing has been instrumental to the completion of the project, and I have gained much valuable knowledge and experience under her guidance. This project has given me a chance to learn a variety of image processing knowledge and the difficulties they pose. I am grateful to have the opportunity to work with an amazing supervisor on the project. Thank you, Dr. Lila.

Last but not least, I would like to thank everyone who has directly or indirectly supported me throughout this project. My friends and family and their continuous support and motivation have been very helpful to my work on this project, and Universiti Teknologi PETRONAS for giving me the opportunity to work on my Final Year Project in their campus. Thank you all very much.

ABSTRACT

The stereo imaging system is a system designed to measure the distance of objects from a stereo camera based on stereo images. Stereo images are images taken with stereo cameras where 2 cameras are placed side by side facing the same direction, which mimics the binocular vision of humans. The stereo camera setup gives computers the ability to capture stereo images, also known as 3 dimensional images, which can be used for distance measurement based on the disparity between the same point as it appears on a stereo image. An example application of this form of distance estimation is in self-driving cars, where the car has to be able to see the real world and also determine the distance of obstacles and also the size of obstacles. Proper steps have to be followed for accurate distance and size estimation. The complete setup process consists of 4 steps, Calibration, Rectification, Stereo Matching and Distance estimation. The OpenCV library is one of the most popular computer vision libraries, hence using it will streamline the stereo camera setup process, and the main program will be coded using Python. A prototype has been created as a part of the project to test the feasibility of the distance and size measurement in an unrectified stereo camera setup, and the results of the experiment are recorded as preliminary results. After the proper steps have been followed, additional prototypes have also been created to obtain the rectification matrix needed to calibrate the stereo camera, test the accuracy of the stereo calibration of the camera, allow users to select a region in the stereo image and estimate the distance and size of the region, and a simple obstacle detection algorithm that allows users to specify their range of choice. The results of the experiments show that stereo vision is able to accurately estimate the size and distance of an object up to a limited range. When estimating the size and distance further than the limited range of 5 meters, the estimation starts to deviate. Limitations for each prototype and future improvements to improve the setup were discussed, including ways to increase the accuracy and range of the prototypes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1. BACKGROUND OF STUDY

The camera is one of the most commonly found electronic equipment on a modern electronic device, most commonly found in the smartphones and personal computers used everyday. Using the camera, computers are able to view our world through the camera lens, and with computer vision algorithms, gain the ability to recognize patterns in the images they pick up from the camera, which allows them to identify different objects and even tell faces apart. However, computers with monovision would have the vision like a cyclops: viewing the world as a flat plane and without the sense of depth. Without a sense of depth, computers are unable to tell if an object is big or just really close to the camera, or vice versa. The depth of an object as it appears on a camera is crucial information for many applications such as in self-driving vehicles, hence the need for computers to be able to see the world and also be able to identify distance of objects.

Animals with 2 front facing eyes are able to see the world and differentiate the size and distance of an object through stereoscopic vision. Humans are born with 2 eyes that are slightly separated horizontally but at the same vertical height. This means that both eyes view the world at a slightly different position. The brain then computes the distance of objects based on the differences in the views of both eyes and combines them into 1 singular image. The difference of the view from both eyes creates a disparity, where closer objects create a larger disparity, while objects farther away creates less disparity between both viewpoints.

This fact can be proven with a simple experiment by placing a finger in front of your eyes and then closing one eye and then the other. The point of view of both eyes is different as the background of what the eyes see shifts, and this difference in the 2 points of view is the disparity that stereoscopic vision uses to tell the distance of objects.

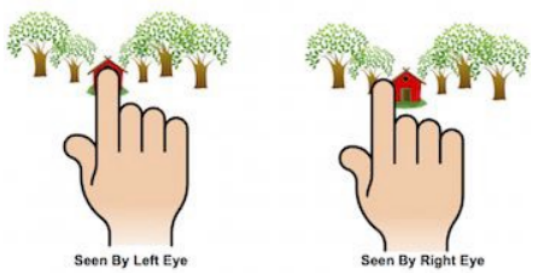

Figure 1.1: Stereo image seen by eyes

As seen in **Figure 1.1**, when our stereo vision eyes focus on the same point (the index finger), the background of the point as it appears on each eye is different, and the difference between these 2 points is described as the disparity.

By understanding stereoscopic vision, a similar method to identify distance can be implemented in computer vision. Using 2 cameras separated slightly horizontally similar to the human eyes, computers are able to view the world with 2 slightly different points of views and identify the distance of objects from the disparity of the 2 point of views.

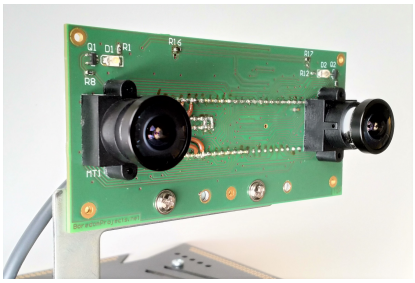

Figure 1.2: Example Stereo Camera Device

As seen in **Figure 1.2**, it is an example of a stereo camera device in one piece. However, the setup can be recreated with a makeshift setup without the need to purchase a pre-built stereo camera as it can be recreated using 2 individual cameras setup side by side to mimic a stereo camera.

To obtain the best results, both individual cameras need to be built in such a way that they would return identical results if they were taking the same picture. In the real world, however, it is very difficult to find 2 cameras that are exactly identical in terms of built quality due to manufacturing differences and error. Disadvantages of this method is that many cameras have to

be tested to find 2 cameras that match each other, which is time consuming, and the process has to be repeated if any of the cameras needs to be replaced. A more practical solution is to use software to improve the images taken from the 2 cameras to a point that the 2 cameras would have virtually identical image results when capturing the same scene at the same point and angle. These steps are the calibration and rectification stages.

Figure 1.3: Example distorted image (left) and the same image after calibration (right)

As seen in **Figure 1.3**, the left image captures a scene of the Petronas Twin Towers with a camera that is distorted, hence the twin towers appear to be slanted away from each other, while the image on the right captures the scene with a calibrated camera, where the twin towers appear vertically and parallelly aligned as it should be.

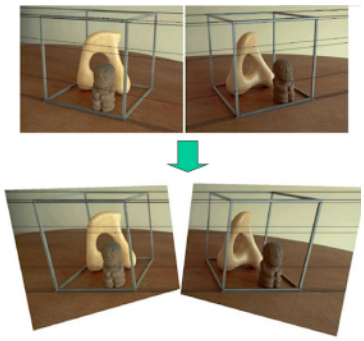

Figure 1.4: Example stereo image unrectified (top) and stereo image after rectification (bottom)

As seen in **Figure 1.4**, the top image shows a scene captured with an unrectified stereo camera, where the horizontal edges of the box on the image do not line up parallelly between the 2 stereo images, while the bottom image shows the same scene captured with a rectified stereo camera, where the horizontal edges of the box line up parallelly between the 2 stereo images.

Once the stereo camera has been calibrated and rectified, the stereo images will be aligned on the y axis, and objects in a scene that appears in the 2 images should only have a

difference on the x axis as the cameras are separated slightly horizontally but are vertically aligned. This means that when the stereo images are matched against each other, similar points in the 2 images must lie on the same horizontal line (share the same y coordinate). This process is known as Stereo Matching.

Figure 1.5: Example Stereo Camera Device

As seen in **Figure 1.5**, it shows an example of a scene taken with a stereo camera, and the stereo matching of similar points between the 2 images. The matching points on the 2 images are shown with colored horizontal lines. This process is known as stereo matching. From stereo matching, the coordinates of the matching points can be extracted. From this, the difference between the x position of the 2 matching points in the stereo image can be determined, where this difference is known as the disparity.

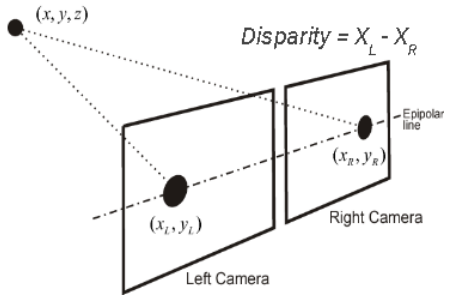

Figure 1.6:Calculating disparity from the matching results

As seen in **Figure 1.6**, the disparity is calculated based on the difference in x coordinates of the same point between the 2 scenes. Disparity is the main factor in computing distance in a stereo camera setup, where the disparity decreases exponentially the further the object moves away from the camera.

1.2. PROBLEM STATEMENTS

In current times, the application of stereo cameras for distance and size measurement is not widespread as its implementation can be quite tedious and complicated. While cameras are now common tools found in smartphones and laptops, stereo cameras are quite uncommon to find due to its complicated nature to implement and lack of concrete use in daily life. This project aims to create a custom made stereo camera implementation from scratch using 2 separate USB powered cameras and a computer in an attempt to understand and simplify its implementation, and hopefully lower the barrier of entry for the general public to create their own custom stereo camera tool. Additionally, a successful implementation of the simple stereo camera with computers would allow it to easily integrate with projects from other fields such as the navigation system in an autonomous robot or vehicle.

All cameras that are uncalibrated have different types of distortions, some slightly, and others very noticeably. These distortions are caused by the optical design of the lens, by the perspective of the camera relative to the subject, or by hardware defects and built quality. Additionally, different cameras also have different intrinsic values such as the optical center and focal length of the camera. These distortions and differences have huge effects on the resulting stereo image, which will greatly affect the stereo matching process of the stereo image. Even slight differences in the vertical alignment of points in the stereo image would cause the stereo matching algorithm to fail as the stereo image becomes too dissimilar, and the stereo matching process becomes virtually impossible. Some of the example applications of using stereo image for distance measurement are in self-driving cars.

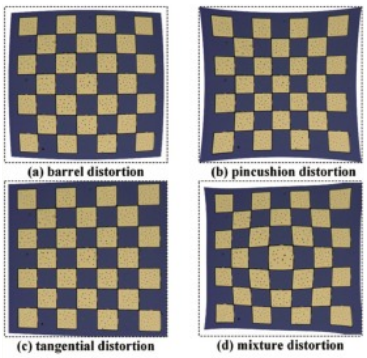

Figure 1.7:Types of camera image distortion

As seen in **Figure 1.7**, different types of camera distortions and the effects it has on an example checkerboard image is shown. Barrel distortions cause the straight lines in a scene to curl inwards like the walls of a barrel. Pincushion distortions on the other hand cause straight lines in a scene to curl outward similar to the shape of a cushion. Additionally, tangential distortions are caused when the image sensor inside a camera is not aligned perfectly parallel to the camera lens, causing the image to look tiled and some points in the image appear farther away than they really should. The mixture distortion is any combination of the distortions that cause an image to be distorted.

Figure 1.8 Example of stereo camera in self driving cars

Figure 1.8 shows an example application of stereo cameras in vehicles as a part of Hitachi Automotive Systems to enable Automatic Emergency Braking (AEB) [1].

The title of this project is "Stereo image calibration for size measurement". The purpose of this project is to be able to accurately compute the real life size of objects detected on screen from the 2 cameras' point of view. The first step to achieve the main goal is to be able to set up the cameras and ensure that the cameras are able to detect a desired object. Next, proper calibration of the individual cameras and the rectification of the stereo camera setup will need to be done. After that, the stereo matching of the stereo images will need to be done to obtain the disparity between each similar point in the 2 points of view, where a disparity map showing the disparity of the matched points in the stereo image will be generated. Once the disparity map is generated accurately, the distance can then be determined based on the disparity. Lastly, the

width of any object in the stereo image can be estimated based on the estimated distance between the camera and the object based on simple basic triangulation.

1.3. OBJECTIVES

1. To calibrate cameras for stereo vision.
    - Approach: Using Image Processing Tools to calibrate, rectify and set up 2 cameras for stereo vision.
2. To estimate the distance and size of an object from stereo images.
    - Approach: Using stereo image matching and triangulation to determine the distance and size of an object in real time.

1.4. SCOPE OF STUDY

Hardware used:

USB cameras are chosen as the camera hardware to capture the stereo images due to their affordable price and adaptability to different use cases. Next, a Stereo Camera Setup is needed to ensure the stereo cameras are set up in such a way that the results would give the best stereo image results that reduce the adjustments needed to be done for calibration and rectification.

Initially, a Raspberry Pi is chosen to be the main hardware for running the program due to its versatility, customizability and portable nature which allows for easier adaptation in different scenarios. However, as the scale of the project grew, it became clear that the computation power of the Pi is not powerful enough, hence the project was shifted to a conventional computer instead.

Software used:

Python will be used along with the OpenCV library as OpenCV contains many useful built-in functions to streamline the image processing and stereo image processing steps. Additionally, there are many existing documentations on stereo image processing using OpenCV in Python on the internet as well as through the official OpenCV documentations.

1.5. RESEARCH QUESTIONS

Based on objective 1, the question that comes up is, what are the effects of a camera's distortion on the images captured by a camera on distance and size estimation, and how can camera calibration correct these distortions and improve the results? Additionally, how does Stereo Rectification improve the accuracy of Stereo Matching and Disparity Map Generation?

Based on objective 2, the question that comes up is, what is the relationship between the disparity in stereo images and the estimation of distance and size?

CHAPTER 2
LITERATURE REVIEW

To achieve a proper stereo camera setup for distance measurement, there are a total of 4 steps in the setup process: Calibration, Rectification, Stereo Matching and Disparity Map Generation [2].

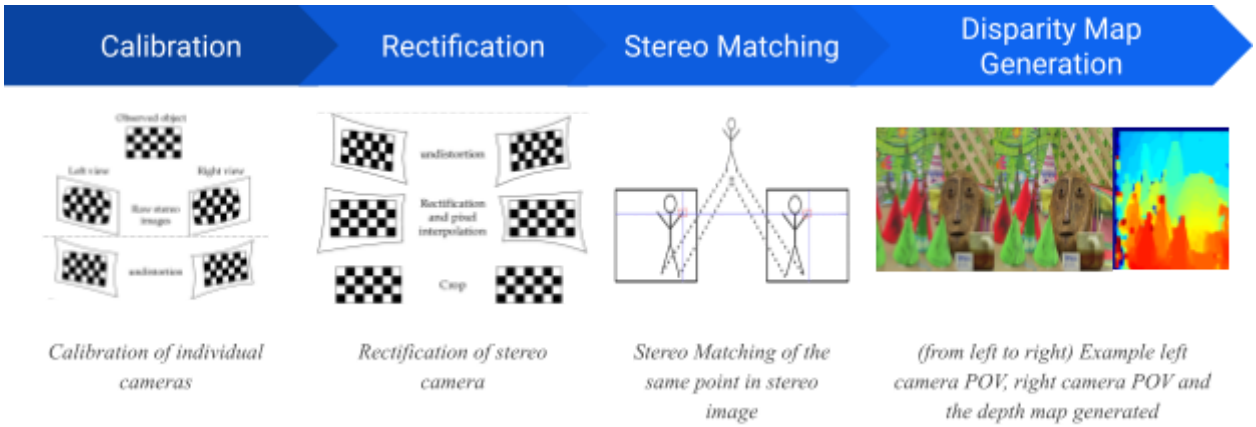

Figure 2.1:Flow chart of the complete stereo camera for distance measurement setup

As seen in **Figure 2.1**: the complete stereo camera setup firstly involves camera calibration, which removes distortions in an image taken with the cameras. Next, the camera rectification step ensures that all similar points in the stereo image line up horizontally through the use of distortions, and the unusable parts of the image after the distortion are then cropped out. After that, Stereo Matching is done to match similar points in the 2 images to determine the difference in coordinates between the same points as it appears in the 2 images. Lastly, a disparity map is generated to allow the distance estimation of each point in a stereo image.

2.1. Step 1: Calibration

The camera calibration process is the process of determining the internal and external parameters of a camera model such as orientation through analyzing images taken by the camera in a set of controlled test environments. The calibration process would usually return the intrinsic and extrinsic values of a camera which represented by a 3 x 4 camera matrix, where extrinsic values include the position and orientation of the camera in the scene, while the intrinsic parameters represent the optical center and focal length of the camera. The purpose of camera

calibration is to improve the consistency of the image result of different images when taking the images in different angles, position and environments by removing distortions.

Two of the most common forms of distortions are Radial Distortions and Tangential Distortions. In the case of radial distortions, straight lines will appear to be curved away or towards the center of the camera, and the effect becomes more noticeable when the straight lines fall on the edge of the camera view. On the other hand, tangential distortions are distortions that occur in the manufacturing process, where the image sensor does not align vertically, which results in a tilted image, where some areas of the image may look closer than expected.

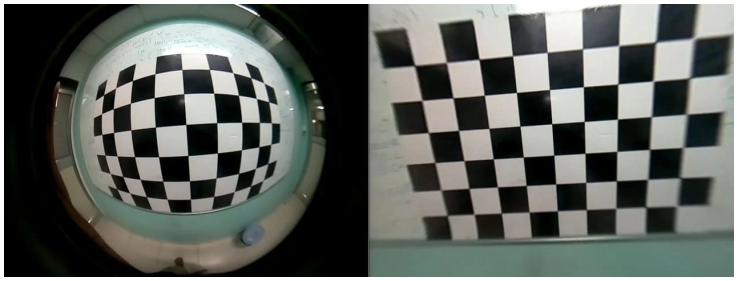

Figure 2.2:Example distorted image (left) and calibrated image (right)

As seen in **Figure 2.2**, the left image shows an image taken with an uncalibrated camera which results in a fisheye view in the resulting image and causing the straight lines in the checker box to appear curly, while the right image shows the image after the camera is calibrated, where the straight lines in the checker box are properly straight as it should.

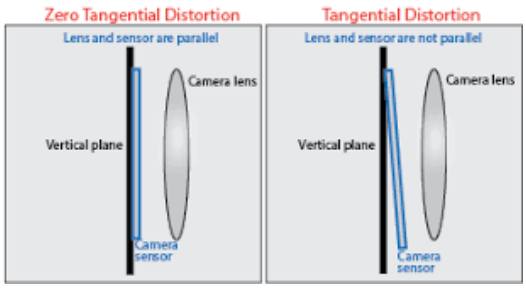

Figure 2.3: Diagram showing the cause of tangential distortion

As seen in , it shows the inside of a camera, where the image on the left shows a zero tangential distortion camera, **Figure 2.3** where the lens line up vertically with the camera sensor,

and the image on the right shows a camera with tangential distortion, where the camera sensor does not line up vertically with the camera lens.

A few algorithms have been developed for camera calibration, such as Zhang's method, Tsai's method and Bouguet's method [3]. A comparison between the methods can be seen in **Table 2.1** below.

Table 2.1: Comparison between Zhang's method, Tsai's method and Bouguet's method

Zhang's Method [4]	Tsai's Method [5]	Bouguet's Method [6]
The calibration object is placed in the field of view of the stereo camera and multiple images are taken at different viewpoints and the corners of the calibration object are detected. Using these features the camera's extrinsic and intrinsic values are obtained to describe the transformation between cameras. Based on Direct Linear Transformation, an estimated projection matrix between the calibration target and the image plane is generated. One of the most common methods.	The calibration object is placed in the field of view of the stereo camera and multiple images are taken at different viewpoints and the corners of the calibration object are detected. These features are then related to the corresponding 3D points to get the intrinsic and extrinsic parameters. Based on non-linear equations. 2-stage algorithm, the orientation and translational coordinates are first obtained, then the focal length and distortion coefficients are obtained. Slightly lower accuracy, lower computation time compared to Zhang's method.	The calibration object is placed in the field of view of the stereo camera and multiple images are taken at different viewpoints and zoom levels and the corners of the calibration object are detected. Intrinsic and extrinsic parameters are estimated by minimizing the reprojection error between the observed features and the corresponding 3D points. Based on Zhang's method, used in Matlab and OpenCV library for camera calibration.

2.2. Step 2: Rectification

In the real world, no 2 cameras are built exactly the same. However, the algorithm used for stereo matching works best when the images line up parallelly. Hence there is a need for stereo rectification. The stereo rectification process warps and distorts the stereo images to a point that they would appear to line up parallelly. Firstly, key points in the stereo image need to be identified, which is usually done with a checkerboard. These key points will be reprojected to best fit every key point on the same parallel lines [7].

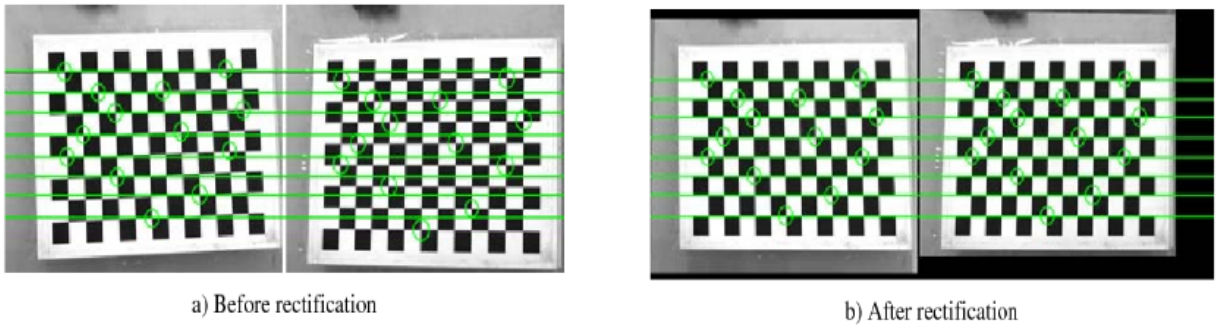

Figure 2.4: Example stereo image before rectification (left) and after rectification (right)

As seen in **Figure 2.4**, the stereo image on the left shows a stereo image taken before stereo rectification, where the checkerboard as it appears on the stereo image are not properly aligned, causing the horizontal points to not appear to line up properly between the stereo image. The stereo image on the right shows the image after proper rectification of the stereo camera where the horizontal points of the checkerboard line up parallelly between the stereo image.

A few algorithms have been developed to rectify the stereo cameras, such as Longuet-Higgins Algorithm, Hartley's Algorithm and Bouguet's Method. A comparison between the algorithms can be seen in **Table 2.2** below.

Table 2.2: Comparison between Longuet-Higgins' Algorithm, Hartley's Algorithm and Bouguet's Method

Longuet-Higgins' Algorithm [8]	Hartley's Algorithm [9]	Bouguet's Method [10]

Based on singular value decomposition (SVD). First, the epipolar lines are computed, then the horizontal offset needed to align the 2 images is computed as the rectification matrix. The correspondence problem was solved, but the method provided in the paper was unstable.	Based on Longuet-Higgins' algorithm. The fundamental matrix is computed which describes the fundamental matrix to compute the rectification transformation. Commonly used by researchers to solve the problem of relative camera placement.	Based on Hartley's method. Includes steps to optimize rectification transformations for better alignment and accuracy used in Matlab and OpenCV library for camera calibration.

2.3. Step 3: Stereo Matching

The main goal of the stereo matching process is to find the correspondence of one point in a scene between stereo images that lie on the same y coordinate. Once the correspondence for every single point in the stereo image is found, the disparity of the points can be determined, and a resulting disparity map that maps out the disparity of every correspondence point can be generated.

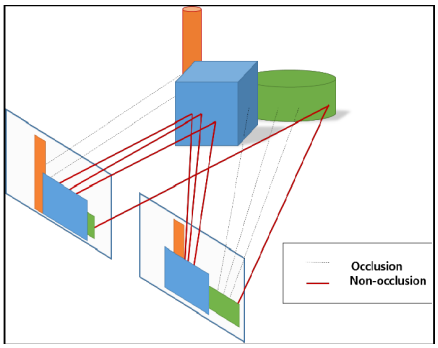

Figure 2.5:Stereo matching process. Non-occlusion points are points in the image that are visible on both cameras' POV

Figure 2.5 shows an example of the stereo matching process, where similar points as it appears in the stereo image are matched together, in this case the blue box in the scene is

matched between the stereo image. These points are non-occlusion points, which are points that are visible between the 2 cameras. On the other hand, the bottom half of the orange pillar can only be seen by the left stereo camera, while the back on the green cylinder can only be seen by the right camera. These points are known as Occlusion points, where the points are unable to be matched due to it being obscured in the other camera's point of view.

A few algorithms have been developed for stereo matching, such as Local Matching, Global Matching and Semi-Global Matching. However, none have proven to be flawless [11]. A comparison between the matching algorithms can be seen in **Table 2.3** below.

Table 2.3: Comparison between Local Matching, Global Matching and Semi-Global (Mixed) Matching

Local Matching	Global Matching	Semi-Global (Mixed) Matching
This method matches points by comparing every pixel and its neighboring pixels to find the best match. Since the images are parallel through rectification, it is only necessary to search for matching pixels on the same horizontal axis.[11]	This method compares every pixel in the image to find the best match. This method cannot be run in real time due to the high computational power and time needed. Best suited for 3D mapping. [11]	This method is a mix of both local and global matching by comparing each disparity value obtained. Fast and reliable for real time matching. [12]

2.4. Step 4: Disparity Map Generation / Distance and Size Estimation

The last step of the stereo camera setup is the disparity map generation and size estimation process. The disparity map generation step aims to give depth to images from the disparity map of the scene, where the depth of the object is calculated from the disparity. Since the disparity of every point in the stereo image is mapped out in the disparity map, and the distance of any point in the given stereo image can be estimated.

The relationship between disparity and distance can be seen in the graph below. As the object moves further away from the stereo camera, the disparity of the object as it appears in the stereo image reduces exponentially. From the estimated distance,the size of the object can be estimated using similar triangles principle.

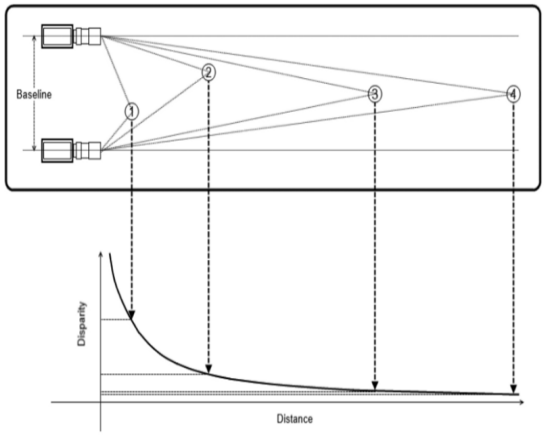

Figure 2.6:Diagram showing the relationship between the distance of an object from the camera and the disparity of that object in the stereo image [2]

As seen in **Figure 2.6**, the diagram shows that the disparity between a point as it appears in the stereo image is inversely proportional to the distance between the object and stereo camera. This means that as the object moves further away from the stereo camera, the disparity of the point between the stereo camera would decrease with diminishing disparity as the object moves farther away from the stereo camera.

A few methods have been discovered to derive the distance between the object and the stereo camera based on the disparity between the 2 objects on the stereo image, such as the Triangulation Geometry method and Mrovlje & Vrancic Method. A comparison between the methods can be seen in **Table 2.4** below.

Table 2.4: Comparison between Triangulation Geometry Method and Mrovlje & Vrancic Method

Triangulation Geometry Method - most commonly used. Reliant on focal length and	Mrovlje & Vrancic Method - sometimes used in self-driving cars. Reliant on angle of the

disparity. [13]	object from camera and disparity. [14]

CHAPTER 3
METHODOLOGY

The flow chart of the implementation of the project can be seen in **Figure 3.1**below. Firstly, the basis of the stereo camera setup is implemented using a rigid base to hold the stereo camera in place. Then, capturing of live stereo video is done using Python. Next, the stereo camera is calibrated and rectified to obtain the ideal stereo camera output for stereo matching. After that, stereo matching can be done with the live video footage. From the stereo matching results, the disparity between the matched points in the stereo image is able to be generated, from which the disparity of the points is mapped out in the disparity map. From each disparity point in the disparity map, the distance can be calculated. Lastly, based on the contour size and the value of the points in the disparity map, the distance and size of the contours in terms of their real life size is estimated.

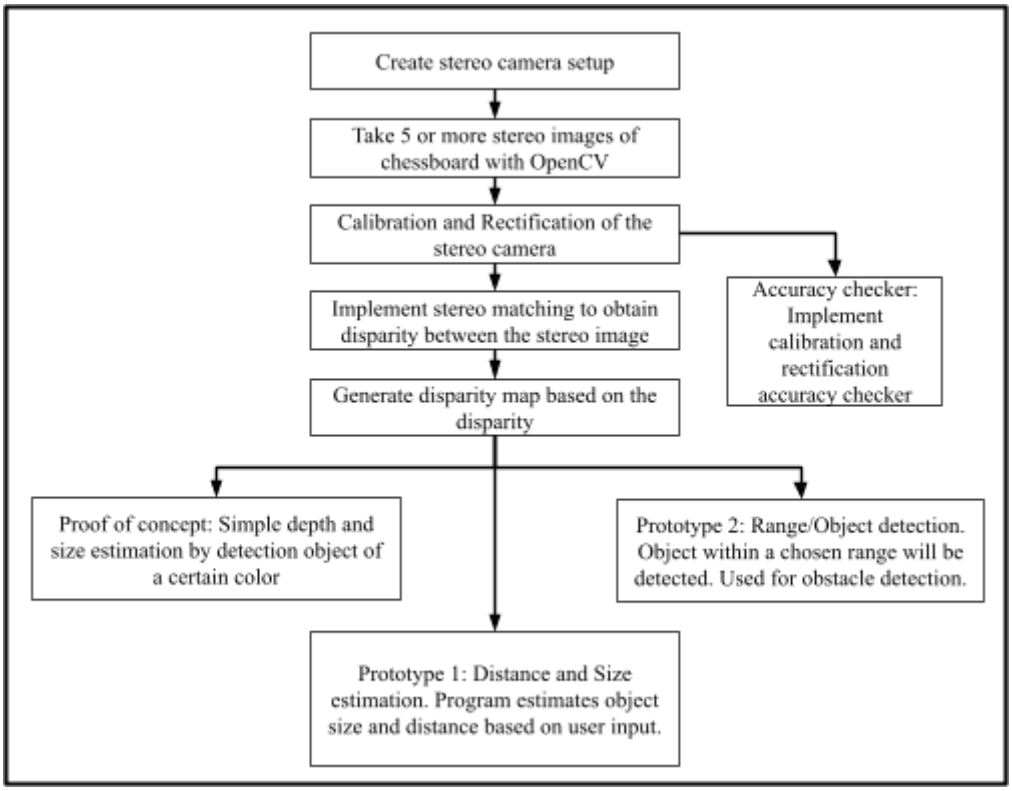

Figure 3.1: Flow Chart of Project Implementation

As seen in **Figure 3.1**, a total of 4 prototypes were created as a result. An accuracy checker program was made to determine the accuracy of the calibrated and rectified stereo

cameras (See Appendix B: Code B7: CalibrationAccuracy.py). The proof of concept program was created as an ideal prototype where the distance and size is estimated using triangulation and the disparity map was not needed (See Appendix B: Code B5: ProofOfConcept.py). Prototype 1 was created to showcase the capabilities of the stereo camera in determining the size and distance of objects from the disparity map generated (See Appendix B: Code B3: DrawBoxCropTest.py). As prototype 1 required some human input for object selection to determine the size and distance, prototype 2 was derived from prototype 1 to automatically track objects at a distance range specified by the user (See Appendix B: Code B4: ObstacleDetector.py).

3.1. Stereo Camera Setup

The camera setup involves 2 cameras, preferably identical cameras, placed slightly apart from each other on the horizontal axis, but are on the same level vertically. Additionally, the orientation of the camera has to be set up identically with each other as well. As the stereo camera is being custom made, 2 USB powered cameras are used and will be the stereo camera for this project. These 2 cameras are connected to a computer, and it will be the computer for running the programs for this project. The 2 cameras are set up and held together using a rigid base for keeping the stereo camera fixed at all times during the project. As the stereo camera is being custom made, many materials can be used as long as it keeps the cameras parallel with each other at all times during the runtime [15].

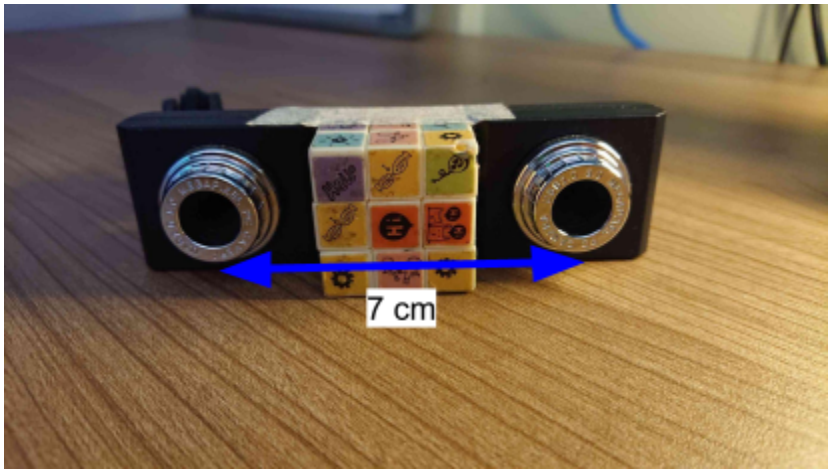

Figure 3.2: Stereo camera setup with makeshift materials used for the project

Figure 3.2 above shows the stereo camera setup that is used for this project. It is done using a rubik's cube, a small piece of wood and some paper tape. For this project, the distance between the 2 camera's centers are roughly 7 cm.

3.2. Stereo Vision

Once the custom stereo camera setup has been made, the stereo vision for distance and size measurement using the custom made stereo cameras can be done. The steps below outline the process of setting up the stereo vision for distance and size measurement.

3.2.1. Step 1: Capture the stereo images needed for stereo calibration and rectification

After setting up the stereo cameras for stereo vision purposes, Python and OpenCV are used for capturing stereo images (see Appendix B: Code B1: StereoCapture.py). Checkerboards are commonly used in image calibration and rectification as they have a consistent repeating and high contrast pattern, making them easily detectable with the algorithm chosen. Using the program, 5 stereo images of a 10x7 chessboard were taken (see Appendix A: 10x7 Chessboard). To capture a stereo image, press the 's' key, and the respective images will be saved in its designated folders (**./images/stereoLeft/** and **./images/stereoRight/** for the respective camera images) as **img**[times 's' has been pressed].

Note, if the camera ports were swapped, the resulting disparity map generated will be unusable. Before capturing any stereo images, ensure that when the program is run, any item that slides across the screen from left to right should also appear as such in the window that appears when the program is run. If the item that slides across the camera appears in the middle of the window first, then the camera ports are swapped. Swapping the camera ports around should fix this issue.

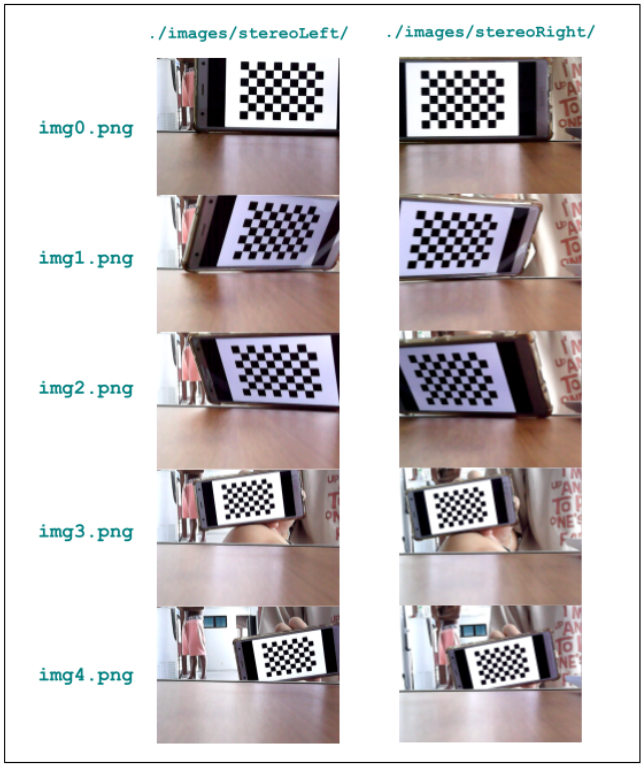

Figure 3.3: 5 stereo images taken with the stereo camera stored in the respective folders and named as img0.png to img4.png

As seen in **Figure 3.3**, the 5 stereo images of the 10x7 chessboard were taken with the stereo camera. These images were stored appropriately in their respective folders (images taken with the left stereo camera were stored in the **/stereoLeft/** folder while images taken with the right stereo camera were stored in the **/stereoRight/** folder), while the images are names similarly based on the order they were taken in (Example: **img0.png** for the first image taken, **img4.png** for the fifth image taken).

3.2.2. Step 2: Stereo Calibration and Rectification

The proposed method of camera calibration and rectification will be Bouguet's method. This is because it is the algorithm that the OpenCV Developers have used to implement their image calibration method, and successful implementations are widely documented.

Using the stereo images of the 10x7 chessboards taken from Step 1, use functions available in the OpenCV library to calibrate and rectify the stereo camera. The OpenCV library provides streamlined image processing functions and tools [16]. Below are the steps taken for the calibration and rectification process (See Appendix B: Code B2: StereoCalibration.py).

For this project, 5 stereo images of the chessboard were taken for the stereo calibration process. The calibration process works best when the images are at different positions, distances and angles. Once done, ensure that the stereo images are stored in the correct folders (**./images/stereoLeft/** and **./images/stereoRight/** for the respective camera images) and ensure that the names of the chessboard images are **img** followed by the number (Example: **img0.png**).

The following program will go through the 5 or more images taken and search for a 10x7 chessboard pattern in each of them with the function cv2.findChessboardCorners(). Users have to press space to continue processing each chessboard image specified, which will be shown to the user with the function cv2.drawChessboardCorners(). Ensure that each of the found chessboard points in the stereo image are positioned correctly.

Once done, the resulting calibrated stereo image of the last chessboard image will appear. An image where the calibrated image stacked on top of each other in red and blue will also appear. These outputs should provide an easy way to determine if the images are properly calibrated or not. Additionally, the resulting camera calibration and rectification matrix and the camera's focal length will be stored in the ./MonoParam.xml file.

Note, if any of the found chessboard points in the stereo image is incorrectly positioned (due to small image size or poor image quality), the resulting camera calibration may be incorrect. If this is the case, the stereo images of the chessboard have to be taken again.

3.2.3. Step 3: Stereo Matching and Disparity Map Generation

The proposed method of stereo matching will be Semi Global Block Matching Algorithm. This is because it is the method that the OpenCV developers have used to implement their stereo matching algorithm, and successful implementations are widely documented. Additionally, real-time implementations of Semi Global Block Matching in stereo matching and disparity map generation have been well documented.

Using the rectification matrix obtained in Step 2, the stereo matching algorithm becomes much more reliable. However, the stereo matching parameters best suited for the stereo images to be taken have yet to be determined. The resulting disparity map generated will vary from users, hence these parameters should be adjusted by the user until the desired disparity map

results are obtained. Once a satisfactory disparity map is generated, the parameters are recorded for future use. A list of the parameters adjusted and the values chosen for the project are listed in the appendix of the report (See Appendix).

There is no specific value for these parameters actors that will work for every stereo camera in every environment, and the best way to determine these values for every implementation of the stereo camera is through trial and error. A few factors that should be considered when setting the parameters include the textures present in the scene, the contrast and lighting of the image, the level of noise present in the images, and the desired output.

Using functions built into Python and OpenCV (See Appendix B: Code B6: DMapSliderTest.py), users will be provided an interface to test and determine the best parameters best suited for the stereo images they have taken.

A pseudocode is provided in the appendix of the report (see Appendix B: Code B6: DMapSliderTest.py) that provides users an interface to test the stereo matching parameters with stereo images. The parameters that will show up in a slider for the users to adjust include numDisparities, minDisparity, blockSize, preFilterType, preFilterSize, preFilterCap, textureThreshold, uniquenessRatio, speckleRange, speckleWindowSize and disp12MaxDiff.

From the rectification matrix obtained in StereoCalibration.py (see Appendix B: Code B2: StereoCalibration.py), it is load into the algorithm using cv2.FileStorage() and the program rectifies the output images using cv2.remap(). Then, the program creates the stereo semi-global block matching algorithm using cv2.StereoSGBM_create() and computes the disparity map of the rectified stereo image using leftMatcher.compute(). From this, a disparity map will be generated for the user to see.

The program will create trackbars for the adjustable stereo semi-global block matching parameters. Based on these trackbar values, the program will assign the values of these trackbars to the corresponding stereoSGBM parameter in real time. The resulting disparity map with the updated stereoSGBM values will then be shown to the user. Because there is no one parameter value that will work for every case and for everyone, this code allows the user to tweak the stereo SGBM parameters until a satisfactory result is obtained for them.

Through trial and error, the user can adjust the parameter values using the user interface until a satisfactory disparity map is obtained. These parameters are then recorded by the user and will be used in the future steps. The meaning of each parameter and the value of the parameter set for this project can be found in Appendix C: Value of Parameters for Stereo Block Matching.

3.2.4. Step 4: Prototype creations

The title of the project is "Stereo Image Calibration for Size Measurement", and the objectives of the project are to calibrate stereo cameras for stereo vision and to estimate the size and distances of an object from stereo images. From this, a few prototypes were created to achieve these objectives.

3.2.4.1. Prototype 1: CalibrationAccuracy.py

This program allows users to check the accuracy of the stereo calibration and stereo rectification that was done in step 2 (See Appendix B: Code B7: CalibrationAccuracy.py). This is done by capturing the stereo image of a chessboard after rectification and determining the difference in y coordinates between each corresponding chessboard points in the stereo image. The main objective of stereo calibration and rectification is to ensure that every point in the left stereo image lines up on the same y axis with the corresponding point in the right stereo image. Using an object that has points that can be easily detected like a chessboard, the program simulates the checking of these points and their correspondents by checking the difference in each point and their correspondent's y coordinate to determine if the accuracy of the calibration is good.

After capturing live footage of the calibrated and rectified stereo image, the user shows to the camera the 10 x 7 chessboard obtained in Appendix A: Image A1: 10x7 Chessboard. When a stable stereo image is ready, the user presses the 's' key to start the calibration accuracy checking. The algorithm will find the chessboard in the stereo image using cv2.findChessboardCorners() in both stereo images. The found chessboard points are then drawn onto the original stereo image using cv2.drawChessboardCorners()and shown to the user using cv2.imshow(). For every point in the left stereo image, the program then checks for the difference in y coordinate in the corresponding point on the right stereo image, and the difference is stored into an array. At the end of the program, the total difference in y coordinate between

every point in the left stereo image and the corresponding point in the right stereo image is calculated, and the average of the total difference is calculated. In an ideal scenario, the total difference in coordinates between the chessboard points in the left stereo image and its corresponding point in the right stereo image should be 0. However, such accuracy is very hard to achieve. Hence the resulting calibration is accepted if the average error between the points is less than 1.

3.2.4.2. Prototype 2: DrawBoxCrop.py

The program generates the disparity map of the scene, and based on user's selection, measuring the approximate distance and size of the user-highlighted area using stereo images (See Appendix B: Code B3: DrawBoxCrop.py)

After capturing live footage of the calibrated and rectified stereo image, the stereo semi-global block matching object is created using cv2.StereoSGBM_create() and the disparity map of the rectified stereo image is computed using leftMatcher.compute(). From this, a disparity map will be generated for the user to see. Then, the program is set to detect the mouse clicks of the users on the windows generated showing the disparity map and the rectified left image. The user is expected to highlight the region on the disparity map generated that the size and distance estimation should be done on. This can be done by clicking on where the starting corner of the region the user wants to select, dragging to and releasing at the opposite end corner of the region. The program will then show the selected region's highest disparity value contour in a new region. Based on the contour disparity values, a mask is generated that shows the region of the object that is detected within the region highlighted by the user. A bounding box that indicates the area of the object detected by the program is then shown to the user. The distance calculation is done based on the average disparity value of that bounding. The distance is then calculated using simple triangulation. The size of the object is calculated using the estimated distance and the bounding box shape and pixel size. The resulting bounding box is drawn onto the left rectified image, and the estimated values of the size and distance of the selected object is written at the bottom of the left rectified image.

The proposed methodology of the disparity map distance calculation is the simple triangulation method. This is because it is derived from the principle of similar triangles, where

the length of the sides of triangles with similar corner angles can be calculated with a simple formula. The similar principle is applied to the calculation of the size of the object as well.

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

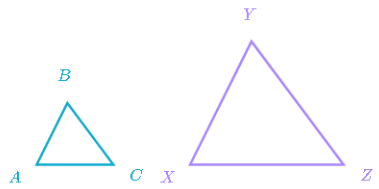

Figure 3.4: Principle of similar triangles [19]

Based on the principle of similar triangles as seen in **Figure 3.4**, the crucial information based on the simplified diagram of the stereo camera and object setup can be extracted. From this information, the following formula can be derived from the principle of similar triangles to estimate the distance between the camera and the object as seen in the Equation 3.1 below. **Figure 19** below shows the simplified diagram of the stereo camera setup and the object for easy understanding and extraction of the information [13].

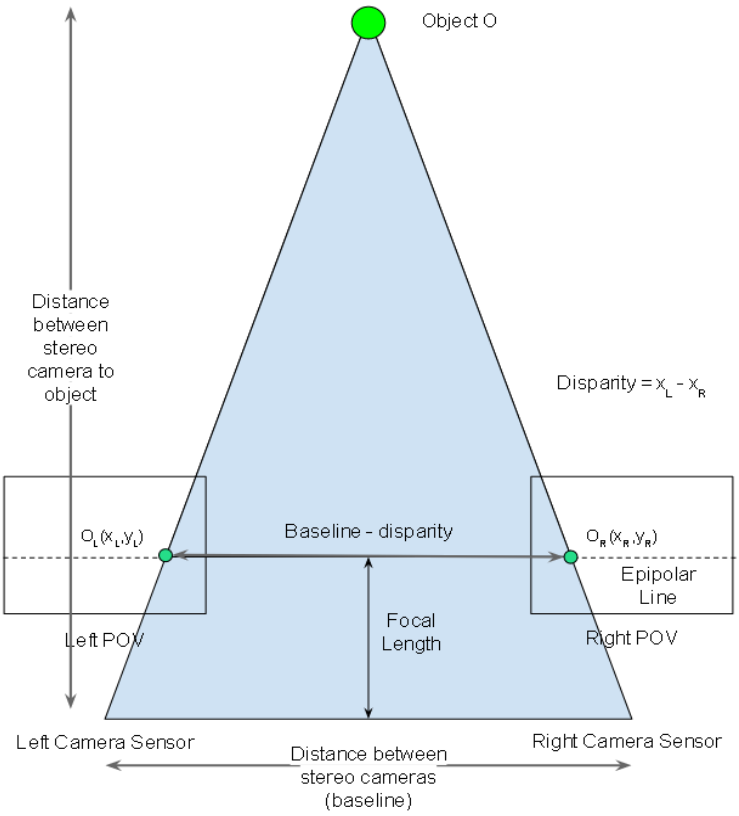

Figure 3.5:Simplified diagram showing the key information needed for distance measurement

$$Distance\ from\ camera\ to\ object\ in\ cm\ = \frac{distance\ between\ cameras\ in\ cm * focal\ length}{Disparity\ in\ pixels}$$

Equation 3.1: calculating distance of an object from the camera from the stereo image

With the estimated distance, further information can be extracted from the stereo camera setup based on a simplified diagram of the scene, and derive the formula to obtain the estimated width of the object as seen in the Equation 3.2 below. **Figure 3.6** below shows the simplified diagram of the object size measurement based on the camera's view for easy understanding and extraction of the information [18].

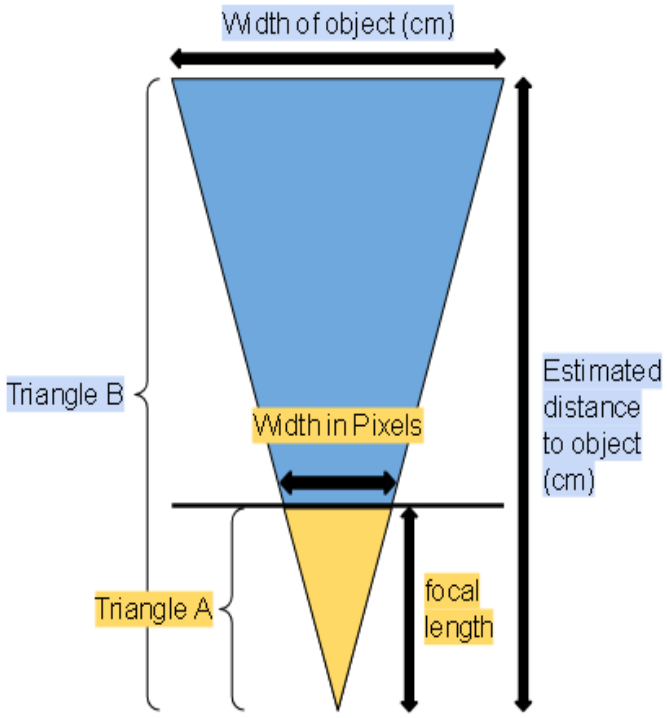

Figure 3.6: Simplified diagram showing the key information needed for size measurement

$$Width\ in\ cm\ = \frac{Estimated\ distance\ in\ cm * Width\ in\ pixels}{Focal\ Length}$$

Equation 3.2: calculating the actual width of an object from the stereo image

Based on these equations and figures extracted from the known information, the distance of an object from the stereo camera can be estimated using the disparity obtained and pixel width of the detected object. During the preliminary stages of the project, a simple prototype was created to test the capabilities of the stereo camera's distance detection capabilities as seen in the preliminary results. The principle of similar triangles is used again to estimate the distance of objects in this prototype.

3.2.4.3. Prototype 3: ProofOfConcept.py

This program detects the largest red object in the stereo images, calculates the difference in their x coordinate as the disparity, and uses it to estimate the distance and size of the red object. This program was used as a simple proof of concept to test the feasibility of the distance estimation using stereo vision. This program is meant to simulate a perfect scenario of the distance calculation method, where the object detected in both stereo images are fixed for every

case, simulating the perfect stereo matching process. From this, the estimation of disparity, distance and size in a perfect scenario can be simulated. This prototype however does not simulate the 3D reconstruction of the scene.

After capturing live footage of the calibrated and rectified stereo image, the program will then look for the largest contour of red (hue value within 0 to 179, saturation value within 166 to 255, lightness value within 184 to 255. See Appendix A: Image A2: Red Object) in the stereo image, draw a bounding rectangular box around it and determine its coordinate. Based on the x coordinate of the red object detected in the left and right stereo images, the disparity in x coordinates can be determined. With this disparity value, estimate the distance between the stereo camera and the red object using triangulation. Using the same principle, the size of the object with the estimated distance, and the estimated values of the size and distance of the detected red object is written at the bottom of the stereo image.

Based on the distance calculated, Prototype 3 was created as a simple obstacle detection algorithm that can detect any objects within a user specified range. The reason for the creation of the obstacle detection algorithm is to demonstrate the size and distance estimation capabilities of the stereo camera setup by using a common real world use case scenario: obstacle detection in robotics.

3.2.4.4. Prototype 4: ObstacleDectector.py

This program allows users to specify a range of distance that the object detector would detect. Any object that falls within the user specified minimum and maximum distance would be detected and the size of the object would be shown. (See Appendix B: Code B4: ObstacleDetector.py)

This program is used to detect any object that falls within the user defined range. The estimated distance and size of said object will be shown in text form at the bottom of the ‘original’ window. After capturing live footage of the calibrated and rectified stereo image, a window will pop out that contains 2 sliders: one for the maximum distance and another for the minimum distance, where it will define the range for objects that fall within the defined minimum and maximum range to be detected. For example, if the minimum distance is set to 1 meter and the maximum distance is set to 2 meters, then any object between 1 meter to 2 meters

from the camera will be detected. Based on this range, the contour that falls within this range in the disparity map will be generated as a mask. The mask will then be compared with the rectified left image, where the mask will indicate the location and area of the object that falls within the range, and this object will then be bound within a rectangular box that is shown to the user on the left rectified image. Then, the estimated distance and size of the object selected is calculated using the principle of similar triangles, and the estimated values of the size and distance of the detected object is written at the bottom of the left rectified image.

The results of the prototypes created can be seen in the results section of this report.

3.3. GANTT CHART

A gantt chart estimating the time needed for each part of the project can be seen below in **Figure 3.7**. Project activities listed include the deliverables for the FYP1 semester as well as an estimated time to research and develop a complete stereo camera setup for size measurement by going through the steps one by one. From week 2 to week 7, the literature review process will be undergone. During this time, research is done on the setup of the stereo camera and stereo image processing. From week 3 to week 5, a simple prototype of the stereo vision was created to test the feasibility of the project for distance and size by using basic triangulation (see Preliminary Results). The proposal defense was held on week 8, where the results of the research and prototype were presented. Next, from week 9 to week 11, the camera calibration and rectification step was researched and developed on. Then from week 12 to week 14, the stereo matching algorithm for disparity map creation was developed from week 12 to week 14, during which the stereo semi-global block matching algorithm was tested until a satisfactory result was obtained for the prototype creations. In the meantime, the Interim Report was also done between week 7 and week 12 and submitted on week 12. From week 14 to week 20, prototypes for the project were created based on the disparity map to satisfy the objectives of the project, which are stereo camera calibration and distance and size estimation of objects based on stereo images. The project progress assessment was held on week 19, where the status of the project was evaluated. From week 20 to week 23, the project undergoes finalization and testing and the results of the project are detailed in the results section of the report. Additionally, while the project undergoes the finalization, the final report will be prepared based on the findings and activities done for this project and is submitted on week 24. The Final Year Project presentation is done on week 24.

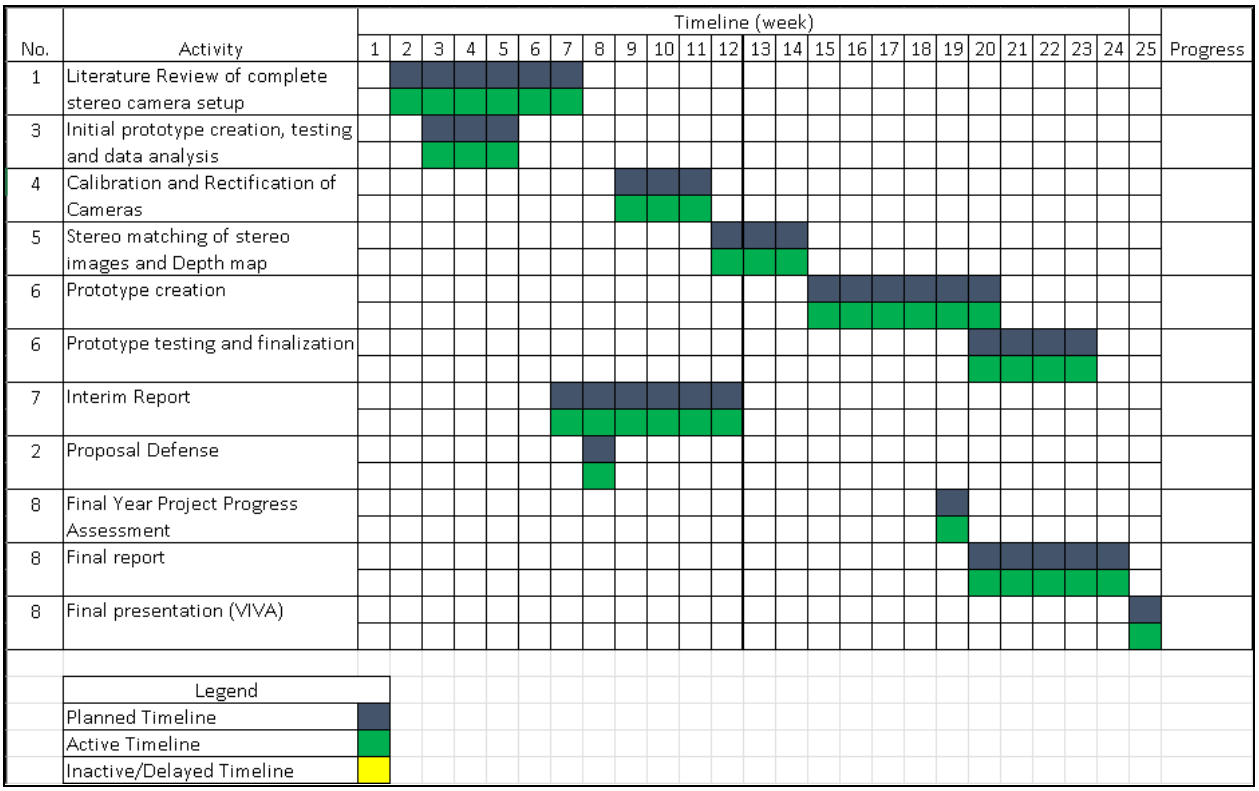

Figure 3.7:Gantt chart for the total duration of FYP 1 and FYP 2

CHAPTER 4
PRELIMINARY RESULTS

As part of the research process, a prototype of a simple uncalibrated and unrectified stereo camera setup as seen in **Figure 4.1** below was made to understand and test the feasibility of distance measurement using stereo images. The prototype was custom made with 2 monocular cameras, duct-tape, expired cards and pencils. The prototype did not undergo calibration and stereo rectification. (See Appendix B: Code B5: ProofOfConcept.py for a rectified version of this prototype)

1. A program was written that takes a snapshot of the stereo images, filters the stereo images to only show the area in the stereo image that contains red, and returns the coordinates and area of the largest red contour in the stereo images. This process is to simulate the matching of the same point in the 2 images but fixing the stereo matching to a consistent point. The example output of this process made with Python can be seen in **Figure 4.2**.

2. From the coordinates and area of the same object point returned, the disparity between the x coordinates between the red contours in the stereo image is determined.

3. Using triangulation calculations and the disparity value obtained, the distance of the red contour from the camera and size of the red contour is estimated.

4. The process is repeated to simulate a live video capture, distance estimation and size estimation.

Figure 4.1: Preliminary results: Prototype of stereo camera setup

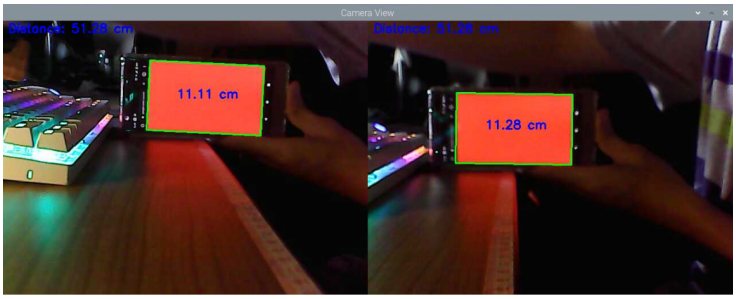

Figure 4.2: Preliminary results: Using Triangulation method for distance measurement

To observe the accuracy of the data obtained, a test is conducted by placing a red object of length 10.6 cm around 40 cm to 150 cm with 10 cm intervals away from the stereo camera. The average of the first 1000 return values of estimated distance and size is then recorded in **Table 4.1** below. The percentage of error is also calculated and recorded for each test scenario and the Equation 4.1 is shown below.

$$Percentage\ of\ error\ = \frac{|Experimental\ Value - Exact\ Value|}{|Exact\ \ Value|}\ *\ 100\%$$

Equation 4.1: percentage error between estimated and actual value of distance and size

Table 4.1: Preliminary results of uncalibrated and unrectified stereo camera prototype

Actual distance between stereo camera and red contour	Estimated distance between stereo camera and red contour	Percentage error between the estimated and actual distance	Estimated width of an object of 10.6 cm width	Percentage error between the estimated and actual width

40	10.6268	0.2528%	39.4964	1.2590%
50	10.7343	1.2670%	50.6055	1.2110%
60	10.5882	0.1113%	59.8112	0.3147%
70	10.4792	1.1396%	68.7580	1.7743%
80	10.4734	1.1943%	78.3651	2.0436%
90	10.4734	1.1943%	88.9262	1.1931%
100	10.2806	3.0132%	96.7324	3.2676%
110	9.9177	6.4368%	103.3989	6.0010%
120	9.7494	8.0245%	111.4822	7.0982%
130	9.5666	9.7491%	117.7923	9.3905%
140	9.4832	10.5358%	126.9637	9.3116%
150	9.0924	14.2226%	129.7479	13.5014%

From this preliminary results as shown in **Table 4.1**, a few observations can be deduced:

1. The percentage of error of the estimated distance compared to the actual distance between the object and the stereo camera increases as the object moves further away from the camera.

2. The percentage of error of the estimated size compared to the actual size of the object increases as the object moves further away from the camera.

As the error increases the further the object is away from the camera, the prototype is unsuccessful at determining the actual size and distance of the object. Possible causes of this error can be due to the lack of camera calibration and stereo rectification beforehand as seen in **Figure 4.2**, which would affect the position calculation of the red contour and the resulting disparity. Hence, improved results are expected after proper camera calibration and rectification.

CHAPTER 5
RESULTS & DISCUSSION

As stated in the Methodology, a total of 4 prototypes were created to achieve objectives of the project: CalibrationAccuracy.py, which will check the accuracy of the calibration done on the stereo cameras after the rectification remapping is done. DrawBoxCrop.py will generate a disparity map using stereo matching and allow the users to measure the distance and size of objects of their selection. ProofOfConcept.py is the calibrated version of the prototype created in the preliminary results of the project, simulating the stereo depth and size measurement in a perfect scenario. Lastly, ObstacleDetector.py is an algorithm that detects objects in the stereo image that are within a user defined range. All the programs mentioned in this section can be found in Appendix B: CODE USED IN PROJECT in the form of pseudocode. This section of the report will discuss each one of the prototypes created in detail and show the results obtained from them.

5.1. Prototype 1: CalibrationAccuracy.py

The objective of this prototype is to check the accuracy of the rectification done on the stereo camera. The accuracy of the rectification of the stereo camera has a major impact on the calculation of the disparity, the generation of the disparity map and the size and distance calculation obtained in a later step. An accurately rectified camera can significantly improve the block matching process, disparity map generated, and the size and distance estimation. This is because the block matching algorithm puts a heavy emphasis on matching similar points by checking the most similar points that lie on the same y-coordinate. Knowing this, this prototype checks the accuracy of the calibration by checking each point in the chessboard and comparing its y-coordinate difference between its left and right stereo image. With this prototype, the objective 1 of the project can be tested, which is to calibrate stereo cameras for stereo vision. (See Appendix B: Code B7: CalibrationAccuracy.py for the pseudocode of the prototype)

To test the program, the user has to upload the rectification mapping MonoParam.xml file obtained from StereoCalibration.py to the same directory as this program (See Appendix B: Code B2: StereoCalibration.py). Then, take a 10x7 chessboard image, show it to the camera, and

press the ‘s’ key. This will remap the stereo camera to the provided rectification settings and test the accuracy of the rectification using the alignment of the chessboard stereo image.

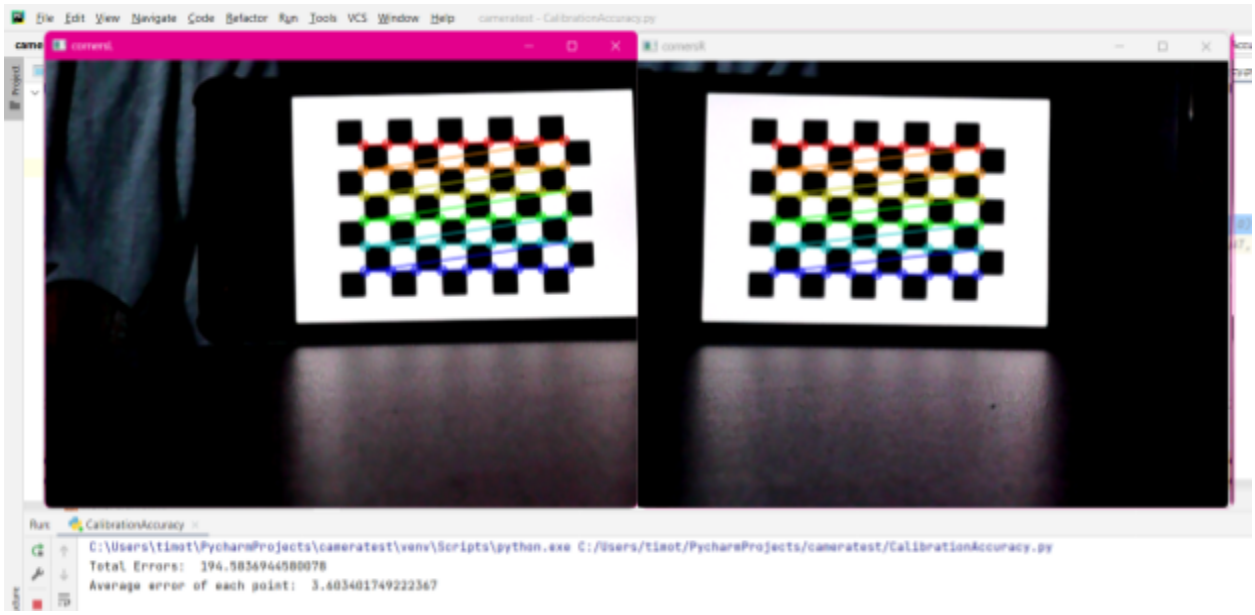

Figure 5.1:CalibrationAccuracy.py: Example of the stereo calibration checking of an uncalibrated camera

Figure 5.1 above shows the calibration checking prior to calibration. As shown in the output, the average error between each of the chessboard points in the stereo images is 3.603 which is above 1. Hence, the uncalibrated camera output is inaccurate.

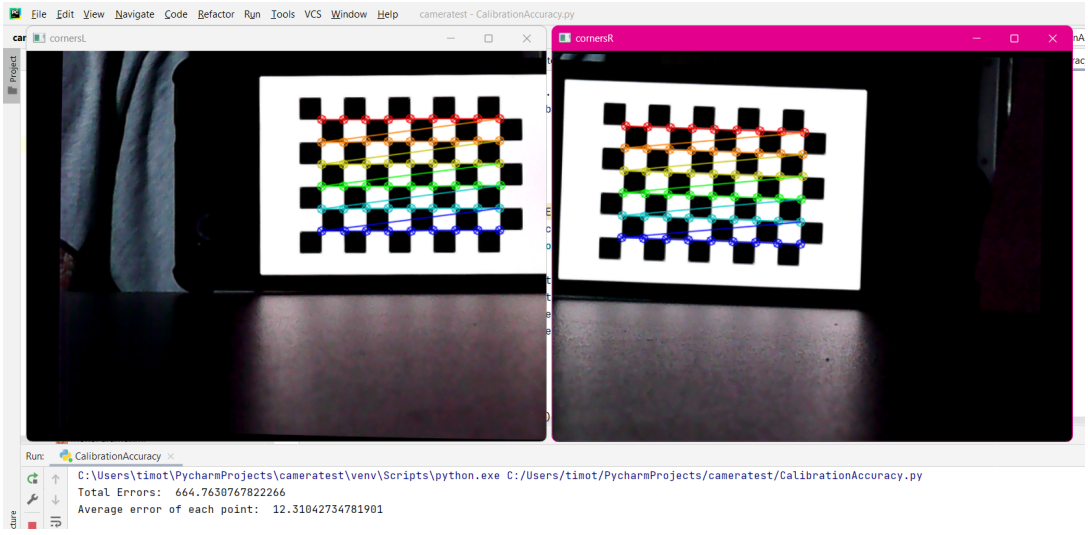

Figure 5.2:CalibrationAccuracy.py: Example of the stereo calibration checking of a poorly rectified camera

Figure 5.2 above shows the calibration checking if the calibration and rectification was done poorly. As shown in the output, the average error between each of the chessboard points in the stereo images is 12.310 which is above 1. Hence, the rectified stereo camera output is poorly rectified and should be calibrated again.

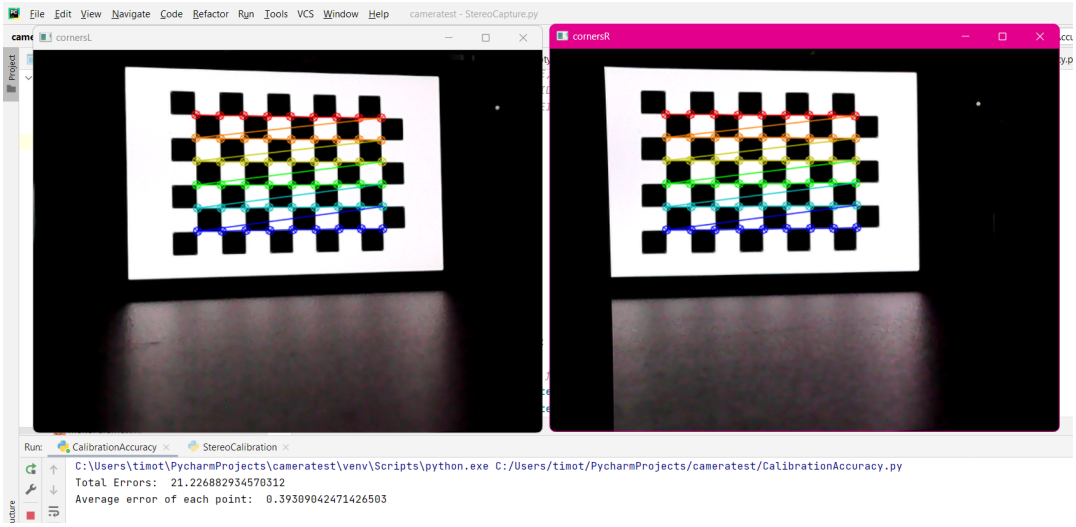

Figure 5.3:CalibrationAccuracy.py: Example of the stereo calibration checking of an accurately rectified and calibrated camera

Figure 5.3 above shows the calibration checking if the calibration and rectification was done correctly. As shown in the output, the average error between each of the chessboard points in the stereo images is less than 1 at 0.393. Hence, the rectification matrix is considered to be accurate, and the resulting disparity map obtained in the following steps will be more reliable. Additionally, the resulting disparity map generated will show a much more accurate result that more closely resembles the actual disparity map of the scene as seen in the **Figure 5.4**below.

Figure 5.4:CalibrationAccuracy.py: (left to right) The left stereo image, the resulting disparity map of a poorly calibrated stereo camera, the resulting disparity map of an accurately calibrated stereo camera.

As seen in the **Figure 5.4** above, the resulting disparity map generated when a stereo camera is properly calibrated can significantly improve the resulting disparity map. Hence, it is proven that stereo calibration and rectification can improve stereo matching and the resulting disparity map.

5.2. Prototype 2: DrawBoxCrop.py

The objective of this prototype is to give users the freedom to select any object in a stereo image and allow users to estimate the distance and size of the user selected object. The disparity map is generated using stereo semi global block matching, and the distance and size measurement is done using the principle of similar triangles. Since the prototype has no form of object recognition, the users are expected to highlight the region of object that they want to measure. With this prototype, objective 2 of the project can be tested, which is to estimate the distance and size of an object from stereo images. (See Appendix B: Code B3: DrawBoxCrop.py for the pseudocode of the prototype)

Users will input their own stereo images into the respective left and right folders as well as provide the rectification mapping .xml file into the folder. The resulting program will generate the disparity map based on the files provided and specified in the program. The users can then select the region that they want to detect the size and distance of, and the best estimated distance and size of the selected region will be shown to the user.

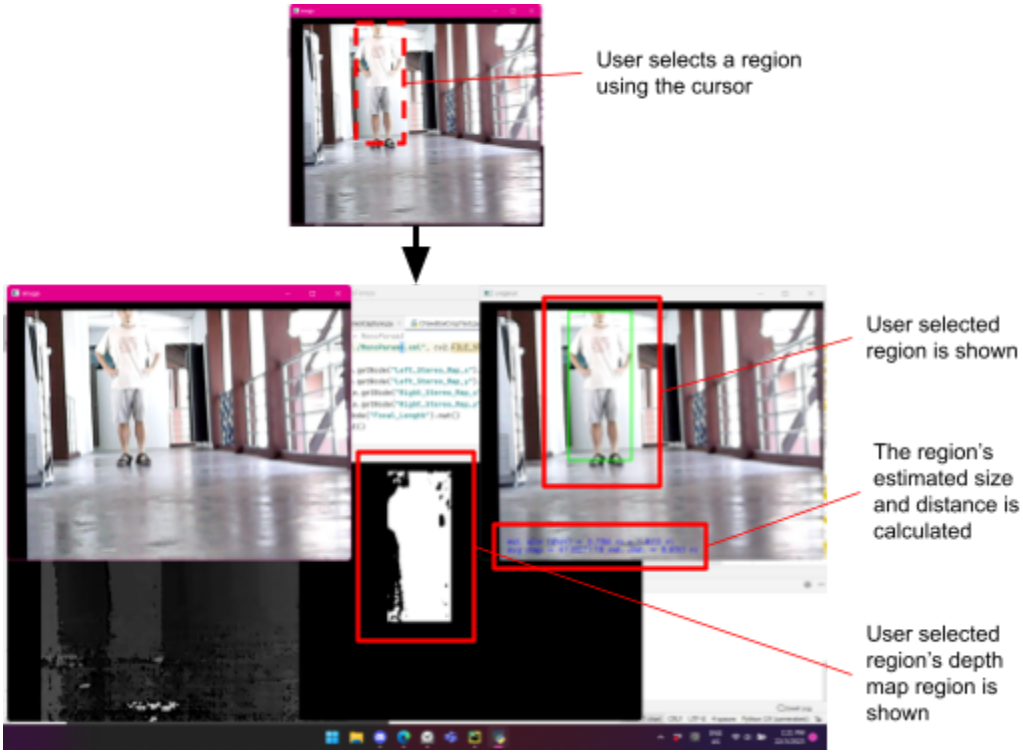

Figure 5.5:DrawBoxCrop.py: Demonstration of how the user defined region size and distance estimation works and how to read the resulting output

Figure 5.5 above shows the example output when the program runs. The program will first show users the left stereo image to the user, and the user is allowed to select a region of the image that they would like to calculate the distance and size of. Then, 2 new windows will pop out, one window to show the users the resulting mask created based on the highest disparity in the user's selected region, and another window will show a bounding box drawn on the left stereo image showing the region of mask created and the estimated distance and size of the mask.

Using the prototype created, 12 stereo images were taken from distances 1 to 12 meters. The results of the experiment are tabulated in the following **Table 5.1**:

Table 5.1: Results of the DrawBoxCrop.py prototype

Disparity value	**Actual distance, m**	**Estimated Distance, m**	**Deviation of estimated distance from actual distance, m**

249.7962	1	1.489	0.489
189.3216	2	1.965	0.035
125.0340	3	2.975	0.025
93.2237	4	3.99	0.01
74.1895	5	5.014	0.014
59.8195	6	6.218	0.218
50.8740	7	7.311	0.311
44.3926	8	8.379	0.379
38.2265	9	9.73	0.73
33.8429	10	10.991	0.991
27.0000	11	13.152	2.152
24.4063	12	15.24	3.24

Table 5.1 above shows the results from the 12 images taken, each at different distances. In the following section, the data from the table are visualized with graphs to show a clearer picture of the experiment.

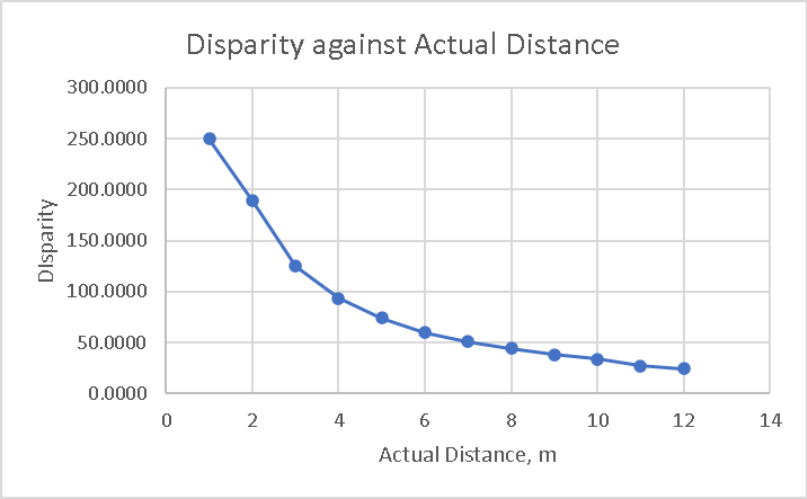

Figure 5.6:DrawBoxCrop.py: Graph of Disparity against Actual Distance

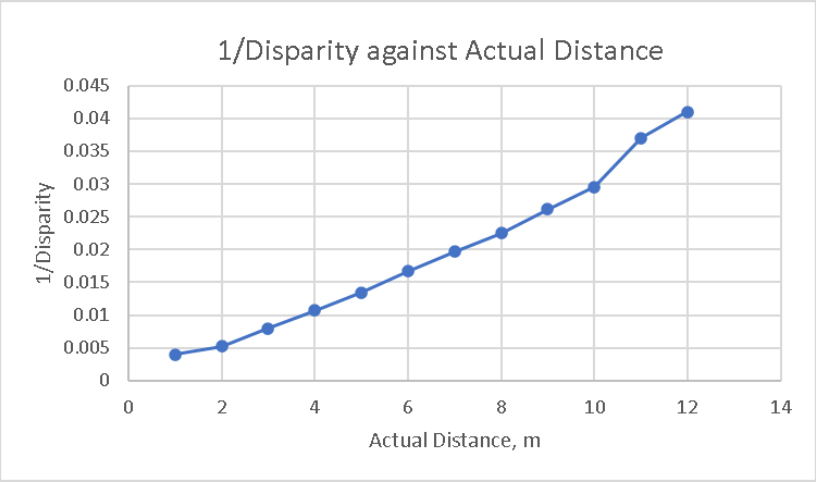

Figure 5.7:DrawBoxCrop.py: Graph of 1/Disparity against Actual Distance

The 2 figures above, **Figure 29** shows the disparity value of objects at different distances, and **Figure 30** shows the graph of **Figure 29** when the disparity values are inverted. As stated in the disparity map generation part of the literature review, the disparity value of matched points in the stereo image is inversely proportional to the distance between the object and stereo camera. The results obtained from the prototype created, when visualized in a graph, shows a similar trend as the theoretical graph shown in **Figure 14**. Hence, the hypothesis is correct.

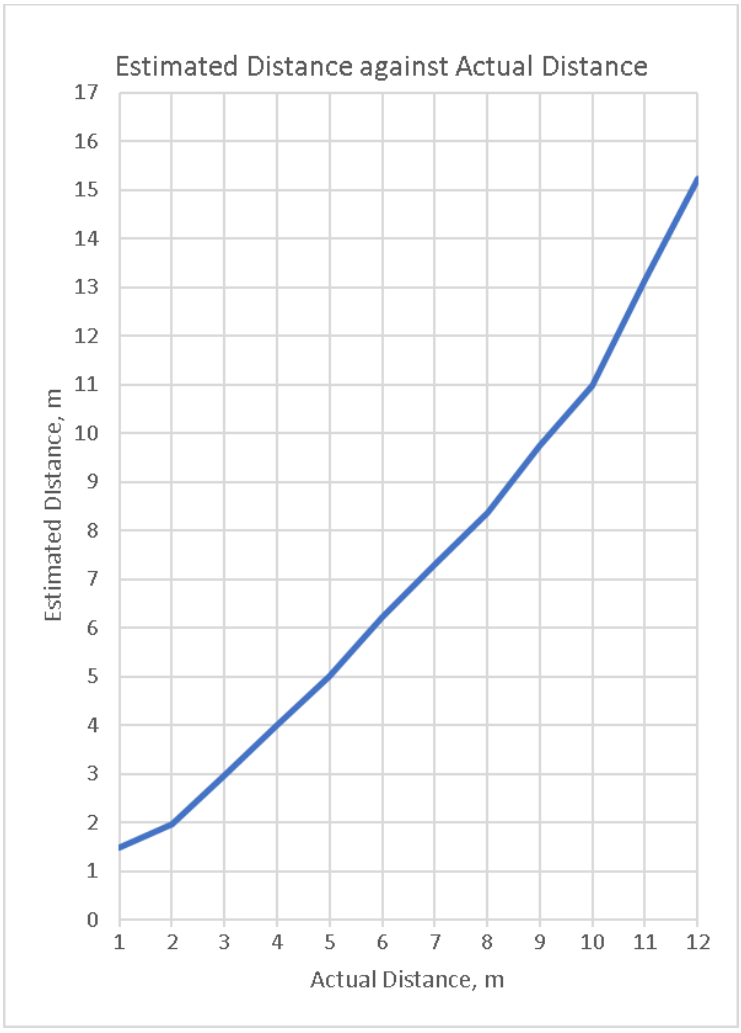

Figure 5.8:DrawBoxCrop.py: Graph of Estimated Distance against Actual Distance

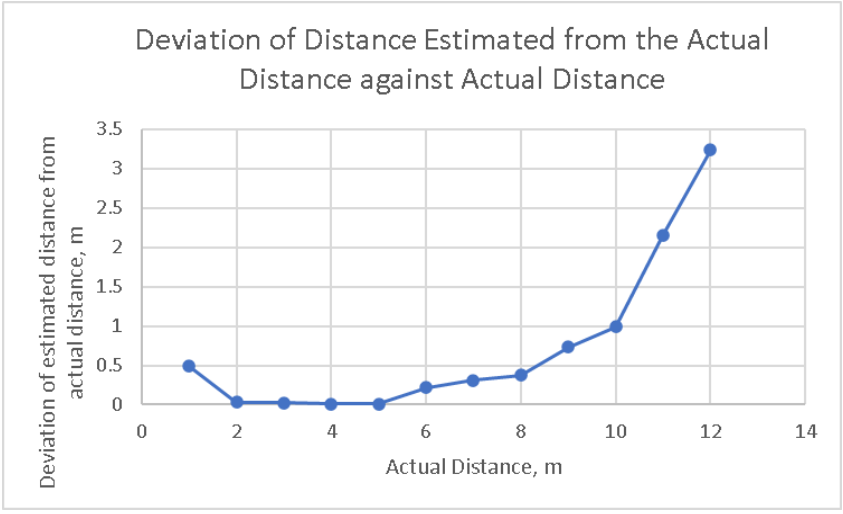

Figure 5.9:DrawBoxCrop.py: Graph of Deviation of distance estimated from the actual distance against the actual distance

In the next 2 figures above, **Figure 5.8** shows the graph of estimated distance against actual distance between object and the stereo camera, while **Figure 5.9**shows the deviation of the estimated distance against the actual distance. From **Figure 5.8**, the estimated distance increases mostly linearly with the actual distance, however from **Figure 5.9**, it is shown that the estimated distance deviates farther away from the actual distance when the distance between the object and the stereo camera is at 1 meter and between 6 to 14 meters. The deviation of the estimated distance from the actual distance is very low between 2 to 5 meters. However, when the value is not within this range, the distance calculated becomes less reliable. Between 2 to 8 meters, the estimated distance obtained does not deviate more than 0.5 meters from the actual distance.. This means that the prototype works best when measuring objects between 2 to 5 meters away from the stereo camera, has somewhat reliable results from 6 to 8 meters, and is very unreliable when the object is at 1 meter or more than 8 meters away from the stereo camera.. Additionally, because the size measurement relies heavily on the distance measurement, the size measurement is unreliable when the distance of the object being measured is not between 2 to 5 meters.

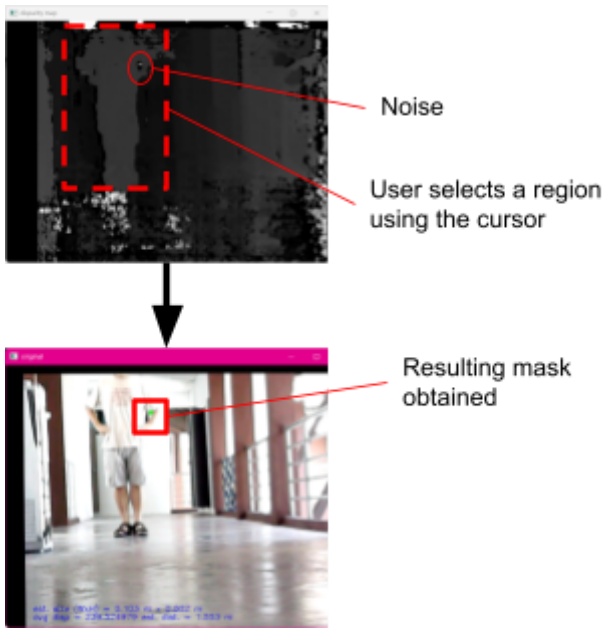

Figure 5.10: DrawBoxCrop.py: Resulting mask with noisy disparity maps

Besides that, as seen in the **Figure 5.10** above, because the algorithm selects the highest disparity value in the highlighted region and creates a mask based on the highlighted region and

a boundary set by the user, noisy disparity maps may cause the desired region to not be chosen by the algorithm. If there is a speckle of noise in the disparity map that has a higher disparity value than the desired region's disparity value (example: unexpected small white region in the disparity map like noise), the resulting region selected will appear to be the small white region. This is because the program selects the highest disparity value within the highlighted region selected by the user and creates a mask around it, which makes it prone to select regions that have the highest disparity as long as it is within the highlighted region regardless of the size, which is quite commonly caused by noise in the disparity map. If this issue occurs, the user should try to highlight the region while avoiding highlighting the noise regions until a desired result is obtained. This issue can also be fixed through further fine-tuning of the disparity map.

5.3. Prototype 3: ProofOfConcept.py

The objective of this prototype is to prove the feasibility of the project objective to estimate distance and size of objects from stereo images. The prototype was created during the preliminary stages of the project without any form of camera calibration. This prototype is an expansion of the preliminary results, where the prototype uses a non calibrated and rectified stereo camera, this prototype uses a calibrated and rectified stereo camera. The program detects the largest red object in the stereo image, calculates the disparity using the difference in x coordinates between the red object detected in the left and right stereo image, and uses triangulation to detect the distance between the stereo camera and the red object and the size of the red object. This prototype is done to simulate a perfect scenario in stereo vision, where the object detected in the stereo camera is set to a constant to fix the stereo matching consistency. With this prototype, the objective 2 of the project can be tested, which is to estimate the distance and size of an object from stereo images. (See Appendix B: Code B5: ProofOfConcept.py for the pseudocode of the prototype, see Appendix A: Image A2: Red Object for the red object specified in the report)

To test the prototype, use the red image provided in the appendix of the report, the program will search for an object within the specified hue, saturation and brightness values. The

coordinates of the red object detected on the stereo camera is then computed to obtain the disparity, and the disparity is used to estimate the distance and size of the detected red object.

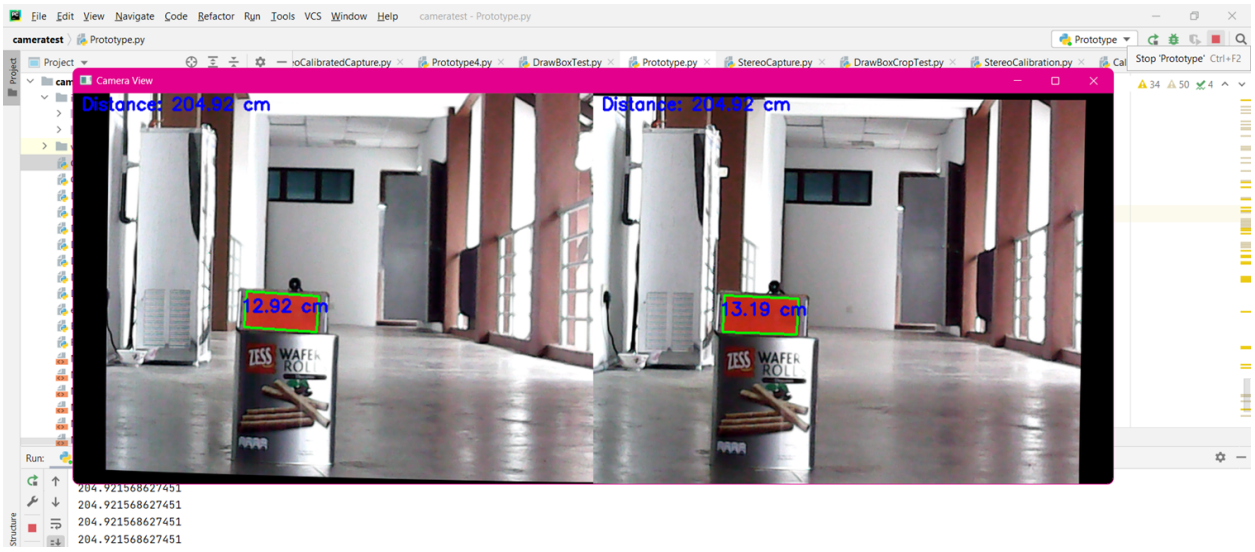

Figure 5.11:ProofOfConcept.py: Example output

Figure 5.11 above shows an example output when the program is run. On top of the red object, a green bounding box is drawn, and the estimated size of the object drawn in the box. And on the top left corner of the image, the estimated distance of the object is drawn.

Using the prototype created, the red object was placed between 1 to 7 meters of distance away from the stereo camera, and the average of the first 500 results of the experiment were recorded in the following **Table 5.2**.

Table 5.2: Results of the ProofOfConcept.py prototype

Estimated Distance, m	**Actual Distance, m**	**Estimated width, cm**	**Deviation of estimated distance from actual distance, m**	**Deviation of estimated size and actual size**
1.036755	1	13.01416	0.036755	0.014156
2.013102	2	13.01944	0.013102	0.019435
2.948482	3	12.88726	0.051518	0.112739

3.919172	4	12.89837	0.080828	0.101634
5.035406	5	13.00947	0.035406	0.009471
6.563228	6	15.35868	0.563228	2.358679
7.65753	7	15.17799	0.65753	2.177994

The results from **Table 5.2** are visualized with graphs in the following section to show a clearer picture of the results of the experiment.

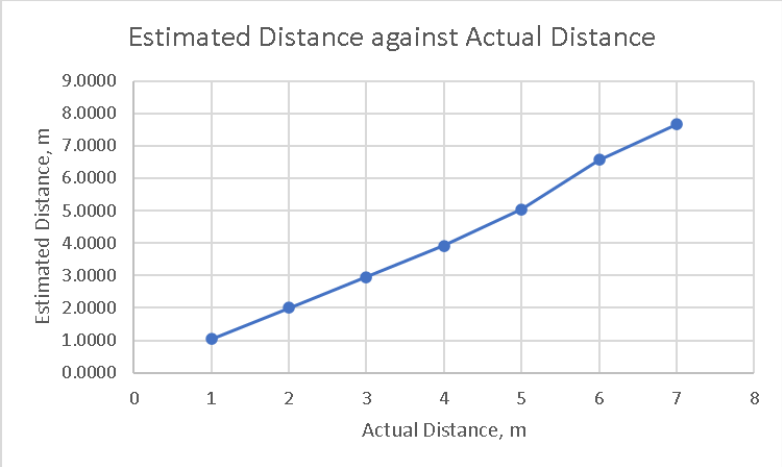

Figure 5.12:ProofOfConcept.py: Graph of Estimated Distance against Actual Distance

Figure 5.12 above shows the graph of the estimated distance against the actual distance between the object and the stereo camera, where the estimated distance increases linearly with the actual distance until about 5 meters. However, from 6 meters onwards, the estimated distance starts to deviate from the actual distance.

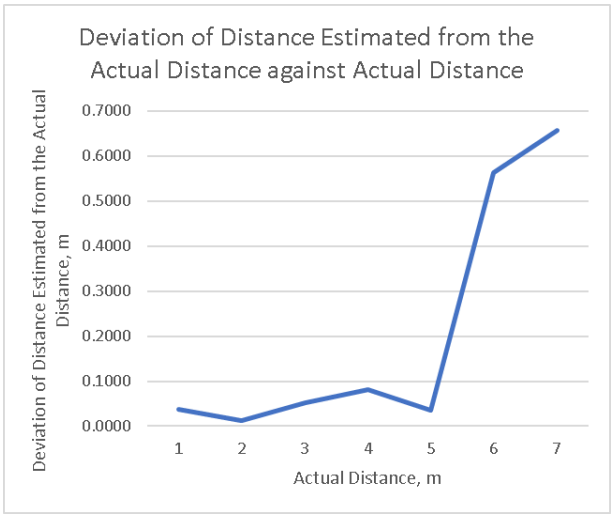

Figure 5.13:ProofOfConcept.py: Graph of Deviation of Distance Estimated from the Actual Distance against Actual Distance

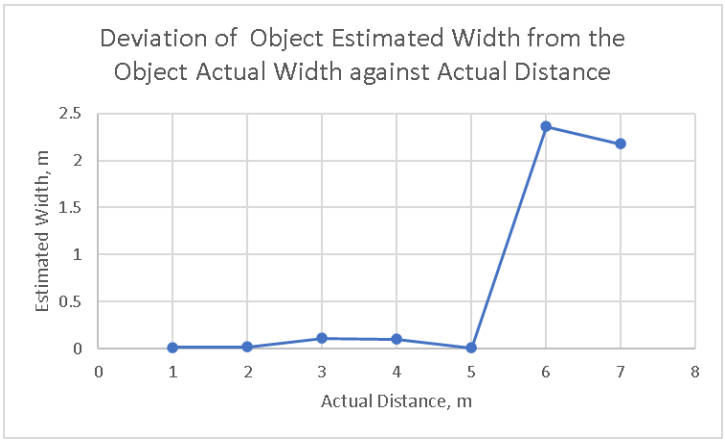

Figure 5.14:ProofOfConcept.py: Graph of Deviation of Object Estimated Width from the Object Actual Width against Actual Distance

As seen in **Figure 5.13**and **Figure 5.14** above, the deviation of the estimated distance and size from the actual distance and size starts to deviate a lot from the actual distance and size respectively when the red object is placed more than 5 meters away from the stereo camera. However, where in the preliminary results, the distance estimated started to deviate when the red object was placed more than 1 meter away from the stereo camera, the new prototype's distance estimation only starts to deviate when the object is placed more than 5 meters away from the stereo camera after stereo calibration. This proves that stereo calibration can significantly improve the stereo distance estimation of a camera.

5.4. Comparing DrawBoxCrop.py and ProofOfConcept.py

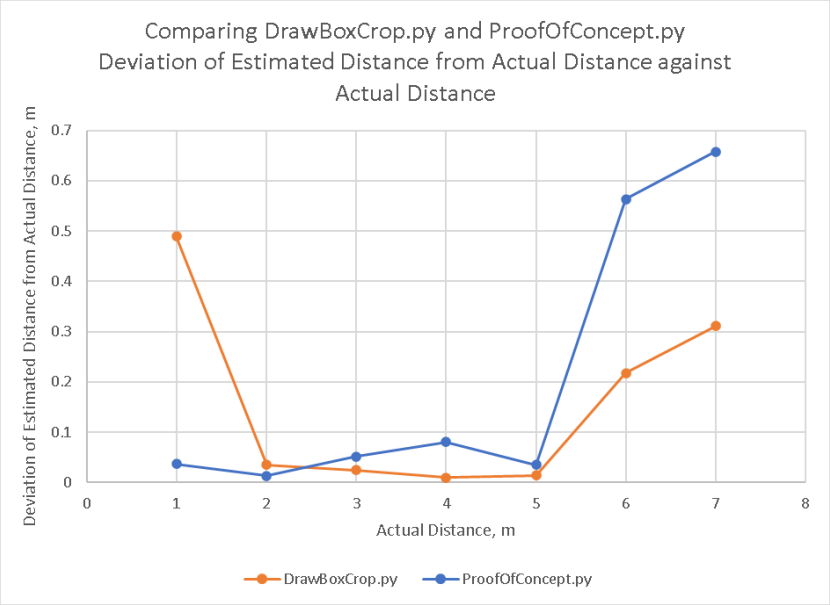

Figure 5.15:Comparison: Graph Comparing DrawBoxCrop.py and ProofOfConcept.py, Deviation of Estimated Distance from Actual Distance against Actual Distance.

Figure 5.15 shows graphs of the deviation of the estimated distance from the actual distance against the actual distance for the prototypes DrawCropBox.py and ProofOfConcept.py overlaid on top of each other. When comparing the results of distance estimation using a disparity map (DrawBoxCrop.py) and the results using a simple object detection, it is observed that at closer distances, the distance estimation using disparity mapping generates a less accurate result compared to the object detection method. Between 2 to 5 meters, the distance estimation with disparity map and simple object detection yield similar results. However, as the object moves more than 5 meters away from the camera, the results obtained by both methods start to deviate more from the actual distance, where the disparity map method shows lesser deviation compared to the simple object detection method. This proves that both methods of distance and size measurement have their own strengths and weaknesses, where the disparity map method has better results when the object is farther away from the stereo camera, while the simple object detection method has better results when the object is closer to the stereo camera. Additionally, it is observed that both methods estimated distance start to deviate from the actual distance when placed farther away than 5 meters from the camera. It can be hypothesized that the camera setup

used for the prototypes has a falloff range of 6 meters, where if the detected disparity object falls outside of this range, the accuracy of the distance measurement is inaccurate.

5.5. Prototype 4: ObstacleDetector.py

The objective of this code is to implement stereo vision in a real-world use case scenario. A 3D view and awareness of the scene and distance and size estimation of objects is a very important part of machines that need to avoid obstacles, most commonly in modern vehicles and robotics. Using the disparity map of the scene, the distance of every point in the disparity map can be obtained. By filtering out far objects in the disparity map, only close objects will be detected, and the contour size of the object will be able to determine the size of the objects. Additionally, to provide more customizability, the user is able to specify the range of objects that will be detected as an obstacle. With this prototype, the objective 2 of the project can be tested, which is to estimate the distance and size of an object from stereo images. (See Appendix B: Code B4: ObstacleDetector.py for the pseudocode of the prototype)

The program captures the live stereo scene and generates a disparity map of the scene. A window with 2 trackbars is also created to allow users to specify the minimum distance and maximum distance between objects from the stereo camera that the program will detect. Using boundaries and the disparity map obtained, the program filters out the disparity values that do not appear within the range specified, and the remaining contour is bound in a rectangular box, and the box's size and distance is estimated.

To test the program, a scene of a park at Universiti Teknologi PETRONAS was captured with different objects at different distances. The main object of the scene includes the floor edge at about 3 meters away from the camera, some bushes at 5 to 6 meters away from the camera, and a tree trunk at about 8 meters away from the camera> the background of the image contains more trees at distances farther than 10 meters. The following tests are run to test the obstacle detector program.

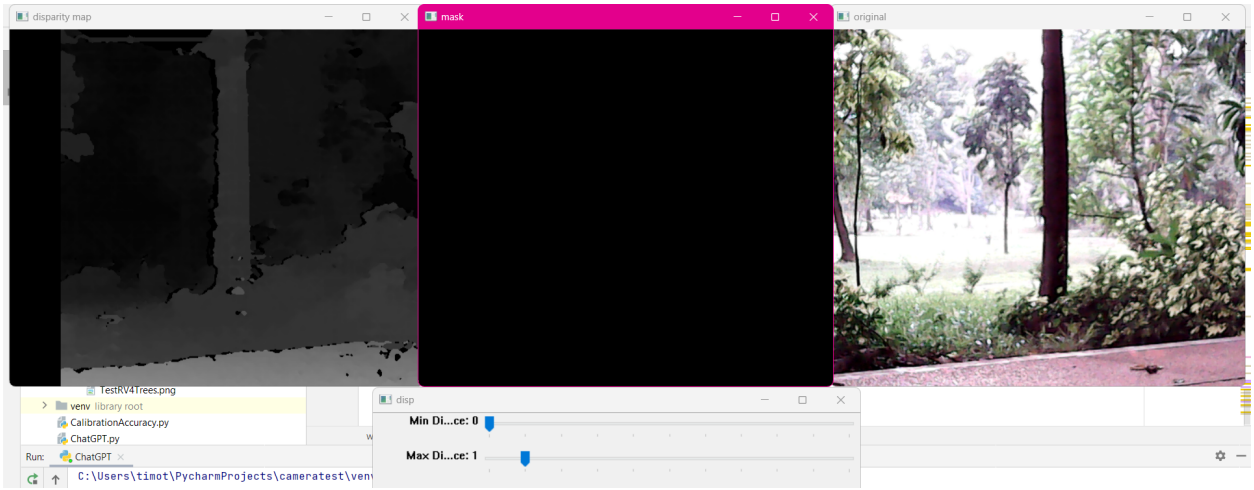

Figure 5.16:ObstacleDetector.py: Result of the program when user sets the minimum distance at 0 meters maximum distance at 1 meters

Figure 5.16 above shows the resulting disparity map and the detected object when the user sets the range of the detection to be between 0 to 1 meters. As nothing in the disparity map lies in this range, the program does not detect anything. To put it in an obstacle detection in vehicles analogy, this would be a case where no obstacles are detected.

Figure 5.17:ObstacleDetector.py: Result of the program when user sets the minimum distance at 0 maximum distance at 10

Figure 5.17 above shows the resulting disparity map and the detected object when the user sets the range of the detection to be between 0 to 10 meters. In this case, a few objects are detected within this range such as the floor, the bushes and the tree trunk, hence they are detected. In the mask, the contour of objects that are detected within this range is shown, where the white parts indicate that objects are within the given range, and the black parts indicate that

those objects do not. In this case, the trees in the background lie further than 10 meters away from the stereo camera, hence it is not detected. To put it in an obstacle detection in vehicles analogy, the settings will show the road that the vehicle is traveling on and the nearby objects such as cars, lamp posts and passerby.

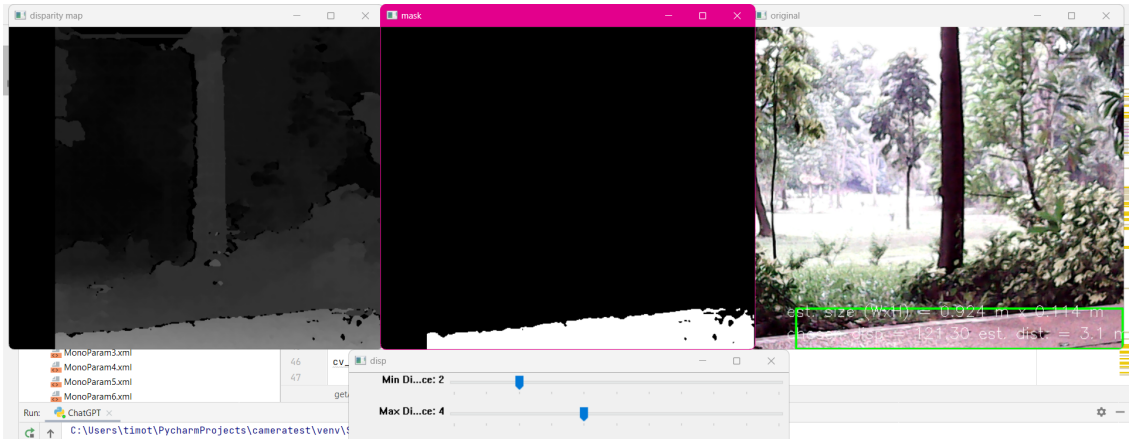

Figure 5.18:ObstacleDetector.py: Result of the program when user sets the minimum distance at 2 meters maximum distance at 4 meters

Figure 5.18 above shows the resulting disparity map and the detected object when the user sets the range of the detection to be between 2 to 4 meters. The edge of the floor is detected as lying between 2-4 meters from the stereo camera, hence it is detected by the program and shows up in the mask. Because the bushes and the tree trunk lie farther away than the specified range, they are not detected by the program. Additionally, a bounding box is drawn around the detected floor, and its estimated distance and size is also shown to the user in text form. In this case, the width of the detected floor area is about 3.1 meters away from the stereo camera and about 1 meter in width. To put it in an obstacle detection in vehicles analogy, the settings will show the road that the vehicle is traveling on.

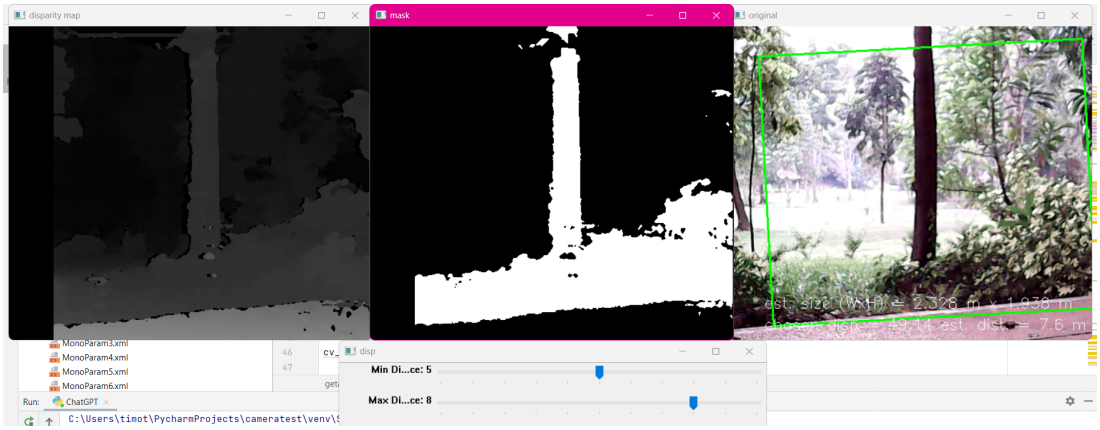

Figure 5.19:ObstacleDetector.py: Result of the program when user sets the minimum distance at 5 meters maximum distance at 8 meters

Figure 5.19above shows the resulting disparity map and the detected object when the user sets the range of the detection to be between 5 to 8 meters. The bushes and tree trunk is detected to be lying between 5 to 8 meters away from the stereo camera, hence it is detected by the program and shows up in the mask. Because the floor edge lies farther away than the specified range, it is not detected by the program. Additionally, a bounding box is drawn around the bush and the tree trunk, and its estimated distance and size is also shown to the user in text form. In the above case, the bushes and tree trunk is detected to be about 7.5 meters away from the stereo camera, the bushes is detected to be about 2.3 meters in width and the tree trunk is detected to be about 2 meters tall. To put it in an obstacle detection in vehicles analogy, this setting would allow the vehicle to detect objects on the road such as other cars, lamp posts and passerby, while the road will not be detected as an obstacle.

CHAPTER 6
CONCLUSION

Stereo vision is a distance estimation technique for measuring distances and sizes of objects using a pair of stereo cameras. The technique uses the disparity values of any point in the left stereo image and its corresponding point in the right stereo image to estimate the distance of objects using the principle of similar triangles. To obtain the disparity values, the cameras have to be calibrated and rectified in such a way that the stereo cameras have almost identical output. Then, for every pixel in the stereo image, a block matching algorithm runs through the left stereo images to determine the most identical block in the corresponding right stereo image, and the difference in x coordinates between the point in the left stereo image and its correspondent in the right stereo image is the disparity value. Using the principle of similar triangles, the distance of that point as well as its size can be estimated. The further away the object is from the stereo camera, the lower the disparity value becomes.

Some of the advantages of stereo vision for distance measurement include the wide availability of cameras and its simple implementation method as well as the ability to sense distance in real time, making it suitable for real-time applications such as robotics and obstacle detection.

This project aims to create a complete stereo camera setup for distance and size measurement, which includes understanding the processes involved. As part of my FYP 1, a prototype of the stereo vision has been made to test the feasibility of the distance and size measurement process without any calibration done. As part of my FYP 2, a simple algorithm was created to calibrate the stereo camera to improve the stereo block matching (see Appendix B: Code B2: StereoCalibration.py), and another algorithm was created to test the accuracy of the stereo calibration (see Appendix B: Code B7: CalibrationAccuracy.py). To allow users to adjust the disparity map result until a satisfactory disparity map is generated, an algorithm was written that creates a simple user interface for the user to adjust and see the changes of the parameters and its effects on the disparity map in real-time (see Appendix B: Code B6: DMapSlider.py). Then, a prototype was created that allows the user to select a region in the disparity map generated, and the algorithm will compute the distance and size of the selected region (see

Appendix B: Code B3: DrawBoxCrop.py). Lastly, a simple obstacle detection algorithm was created that allows the user to specify the distance range of objects that will be detected by the algorithm (see Appendix B: Code B4: ObstacleDetector.py). Additionally, stereo calibration was done to the initial prototype from FYP 1 to generate a more accurate result (see Appendix B: Code B5: ProofOfConcept.py).

From all the prototypes created, there is no clear best prototype at distance estimation. All prototypes created have their own use cases for different scenarios. However, the similar limitation for all prototypes is that they all have a limited range where they work best in. In the case of this project setup, the best range to estimate the size and distance of objects is within 1 to 5 meters, as the estimation starts to lose accuracy when the object is farther than 5 meters away from the stereo camera. Based on the results obtained from the preliminary results and the disparity map, proper stereo calibration can significantly improve the accuracy of the distance and size measurement. However, based on the results obtained from DrawBoxCrop.py and ProofOfConcept.py, it is observed that the current stereo camera setup is unable to properly estimate the distance of objects more than 5 meters away even with proper stereo calibration. These limitations depend on the technology and implementation of the stereo vision system, and the best way to implement the stereo vision depends on the requirements of the application and limitations of the camera setup. Factors that could affect the range that the stereo vision distance and size measurement is able to work in include the camera's resolution and parameters, the accuracy of the disparity map generated as well as the distance between the cameras.

6.1. FUTURE IMPROVEMENTS

As demonstrated in the preliminary results and the rectified version of it, the distance and size measurement of the fixed red object has an accurate estimation. Hence, the algorithm can be improved if the detection of the red object was changed to recognize different objects such as people and trees. A similar method would be used where the object will be bound in a rectangular box, and based on the box's coordinates and area, the disparity, distance and size can be calculated. However, implementing object recognition may make the algorithm slower.

In the initial stages of the project, a Raspberry Pi was used as the computer that handles the stereo computation due to its portable and multipurpose nature and its ease of implementation

in robotics projects. However, as the prototypes became more complex, the algorithm started to run slower on the Pi than on a conventional computer. As a future improvement, the algorithm should be streamlined and optimized to improve the performance of the algorithm on a Raspberry Pi.

Additionally, there are instances where the stereo images do not fully match one another even when in the same environment. This is due to the camera's lack of ability to turn off auto-exposure and auto-white balancing. Exposure in cameras refers to how much light reaches the camera sensors, which will affect the brightness of the image, while white balancing refers to the camera's ability to adjust the output image's color, which will affect its temperature (warmer, colder, natural image).

Figure 6.1: Example when the right camera takes a darker image than the left image despite being in the same environment

Figure 6.1 above shows the resulting stereo image taken under certain conditions that would cause the left and right stereo image to appear different from one another, which would affect the resulting disparity map generated. Modern cameras have the option for users to turn off the auto-exposure and auto-white balancing, however, the cameras used for this project lacked the ability to do so. There exists algorithms to fix the stereo images to more closely match one another, however, from the little experiment done, the resulting images do not match the expected output. Hence as a future improvement, cameras that allow users to manually adjust and fix the value of the white balance and exposure should be used.

Furthermore, the method of getting a usable disparity map can be improved. At the moment, the users have to adjust the parameters of the stereoSGBM manually until a desired result is obtained. However, this process can be tedious, and sometimes the resulting disparity

map will not be the best result obtainable. As an improvement, machine learning can be done to streamline the disparity map generation process, which can improve the process of getting the best parameters for the disparity map for different scenarios. Additionally, current state-of-the-art machines are able to compute disparity maps using deep learning, which includes methods such as Convolutional Neural Networks (CNN) [20]. While deep learning is proven to yield better disparity map results with the ability to adapt to different lighting changes, such methods require a large amount of data and time.

As seen with the results observed in Figure 5.15, the graph comparing distance measurement based on the disparity map and triangulation were shown, where both methods show a huge drop-off in accuracy when estimating the distance of objects more than 5 meters away. It is suspected that this range where distance can be accurately measured is dependent on the distance between the cameras, where the further the distance between the cameras are, the farther the range of distance between the camera and object where the distance can be estimated accurately. Further experimentation should be done to test this hypothesis.

Potential future improvements for each prototype:

1. CalibrationAccuracy.py:

   In addition to calculating the difference in y coordinate between each chessboard points in the stereo left image and its correspondent on the stereo right image, as a future improvement, the program should also check the difference in width of the chessboard to determine if any stretching done to the images during the stereo rectification process is done accurately. Hence, the program can check if the calibration and rectification of the stereo images on both the x and y coordinates of the points are done accurately.
2. DrawBoxCrop.py:

   At the moment, the prototype is highly sensitive to noise in the disparity map, especially noise that shows up in the disparity map as very close objects when there is nothing there as seen in **Figure 5.10**. Since the mask generation process relies on the closest object highlighted, noise will cause the region highlighting of the user to yield undesired results. As a future improvement, the region highlighting and mask generation algorithm should be changed. Additionally, the disparity map should be improved to

reduce the amount of noise in the disparity map. The prototype should undergo more testing and tweaking for better results.

3. ProofOfConcept.py:

   The current prototype detects the object based on the color of the object on screen, and selects the largest red color region in the image. As a future improvement, potential object detection can be done to allow it to work on more general objects and not just red colored objects. Additionally, since this prototype does not require users to tinker with stereo matching, it is the best method to test the effect of the distance between the cameras and its ability to estimate distance of objects at different distances.

4. ObstacleDetector.py:

   The obstacle detection of the result currently works in a user defined range. Additionally, the current obstacle detection will detect any largest object within the specified range regardless of its actual size. This can cause insignificant small objects to also be detected as obstacles. A possible improvement to the project is to allow users to specify a minimum size of an object as well. Furthermore, object detection can help segment parts of the images to better classify the distance of images (example: segment **Figure 5.18** and **Figure 5.19** to separate distance estimation of road, tree trunk and bushes in the same range).

In conclusion, stereo vision is proven to be a viable method of estimating real life distance and size of objects at limited ranges based on the implementation. This can be helpful for users who want to estimate the distance and size of an object without physically measuring it. Stereo vision is also proven to be useful in obstacle detection, which has many practical use case scenarios such as in self-driving vehicles and robotics. The steps for a complete stereo camera setup are proper camera calibration, stereo rectification, stereo matching and disparity map generation, and through the inaccurate results obtained from the preliminary results and improved results after stereo calibration, it is clear that stereo calibration plays a huge role in proper distance and size estimation as well. Hence, the objectives of the project are achieved. Further improvements to the project can be expected from machine learning, optimization of algorithms and better equipment.

REFERENCES

1. “Subaru's EyeSight Cameras Will Soon Help You See at Intersections,” CarGuide.ph, December 27, 2019. [Online]. Available: https://www.carguide.ph/2019/12/subarus-eyesight-cameras-will-soon-help.html?m=1. [Accessed November 27, 2022].

2. D. Biswas, “Stereo Camera Calibration and Depth Estimation from Stereo Images,” Medium, June 28, 2021. [Online]. Available: https://dibyendu-biswas.medium.com/stereo-camera-calibration-and-depth-estimation-from-stereo-images-29d87bc702f3. [Accessed October 19, 2022].

3. W. Li, T. Gee, P. Delmas and H. Friedrich, “A practical comparison between Zhang's and Tsai's calibration approaches,” (n.d.) [Online]. Available: https://cpb-ap-se2.wpmucdn.com/blogs.auckland.ac.nz/dist/7/206/files/2018/08/c036-1t2cjae.pdf. [Accessed October 19, 2022].

4. W. Burger, “Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation,” ResearchGate, May, 2016. [Online]. Available: https://www.researchgate.net/publication/303233579_Zhang's_Camera_Calibration_Algorithm_In-Depth_Tutorial_and_Implementation. [Accessed November 27, 2022].

5. P. Dias, “Tsai Camera Calibration,” INFORMATICS HOMEPAGES SERVER, November 5, 2003. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/DIAS1/. [Accessed November 27, 2022].

6. “What Is Camera Calibration? - MATLAB & Simulink,” MathWorks, (n.d.). [Online]. Available: https://www.mathworks.com/help/vision/ug/camera-calibration.html. [Accessed November 27, 2022].

7. G. Hu, Z. Zhou, J. Cao and H. Huang, “Highly accurate 3D reconstruction based on a precise and robust binocular camera calibration method,” The Institution of Engineering and Technology - Wiley Online Library, October 21, 2020. [Online]. Available:

https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-ipr.2019.1525. [Accessed October 19, 2022].

8. H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," Nature, September 10, 1981. [Online]. Available:https://www.nature.com/articles/293133a0. [Accessed November 27, 2022].

9. R. Hartley, "Estimation of relative camera positions for uncalibrated cameras," SpringerLink, May 28, 2005. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-55426-2_62. [Accessed November 27, 2022].

10. M. Hornáček, "Tutorial: Calibrated Rectification Using OpenCV(Bouguet's Algorithm)," documen.site, 2013. [Online]. Available: https://documen.site/download/document-446_pdf. [Accessed November 27, 2022].

11. D. Paz, "Drawbacks of widespread stereo matching techniques," Wooptix, (n.d.). [Online]. Available: https://wooptix.com/drawbacks-of-widespread-stereo-matching-techniques/. [Accessed October 19, 2022].

12. A. Spizhevoy and A. Rybnikov, "How to do it - OpenCV 3 Computer Vision with Python Cookbook," O'Reilly, March, 2018. [Online]. Available: https://www.oreilly.com/library/view/opencv-3-computer/9781788474443/698585cf-debd-4d28-8abf-86aa233141ec.xhtml. [Accessed November 27, 2022].

13. J. Corse, "Basic Stereo & Epipolar Geometry," Electrical Engineering and Computer Science at the University of Michigan, October 22, 2014. [Online]. Available: https://web.eecs.umich.edu/~jjcorso/t/598F14/files/lecture_1022_epipolar.pdf. [Accessed October 19, 2022].

14. Y. Dawood, K. R. Ku-Mahamud and E. Kamioka, "Distance Measurement for Self-Driving Cars Using Stereo Camera," ResearchGate, January, 2017. [Online]. Available:

https://www.researchgate.net/publication/320336266_Distance_Measurement_for_Self-Driving_Cars_Using_Stereo_Camera. [Accessed October 19, 2022].

15. K. Sadekar, "Making A Low-Cost Stereo Camera Using OpenCV," LearnOpenCV, January 11, 2021. [Online]. Available: https://learnopencv.com/making-a-low-cost-stereo-camera-using-opencv/. [Accessed October 19, 2022].

16. F. Souza, "3 Ways To Calibrate Your Camera Using OpenCV and Python," Medium, March 29, 2021. [Online]. Available: https://medium.com/vacatronics/3-ways-to-calibrate-your-camera-using-opencv-and-python-395528a51615. [Accessed October 19, 2022].

17. O. Padierna, "Stereo 3D reconstruction with OpenCV using an iPhone camera. Part III," Medium, January 3, 2019. [Online]. Available: https://medium.com/@omar.ps16/stereo-3d-reconstruction-with-opencv-using-an-iphone-camera-part-iii-95460d3eddf0. [Accessed October 19, 2022].

18. A. Rosebrock, "Find distance from camera to object using Python and OpenCV," PyImageSearch, 19 January, 2015. [Online]. Available: https://pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/. [Accessed Retrieved October 19, 2022].

19. "Congruence and similarity | Lesson (article)," Khan Academy, (n.d.). [Online]. Available: https://www.khanacademy.org/test-prep/praxis-math/praxis-math-lessons/gtp--praxis-math--lessons--geometry/a/gtp--praxis-math--article--congruence-and-similarity--lesson. [Accessed Retrieved 24 March, 2024].

20. A. Sahu, and K. Sadekar, "Disparity Estimation Using Deep Learning," LearnOpenCV, 22 February, 2022. [Online]. Available: https://learnopencv.com/disparity-estimation-using-deep-learning/. [Access Retrieved 24 March, 2023].

APPENDIX

APPENDIX A: IMAGES USED IN PROJECT

Image A1: 10x7 Chessboard

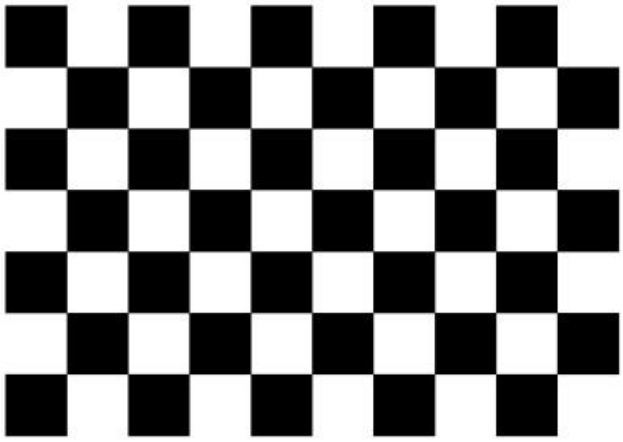

Figure 7.1: 10x7 chessboard used for stereo calibration and rectification

Image A2: Red Object

Figure 45: Red object detected in ProofOfConcept.py (See Appendix B: Code B5: ProofOfConcept.py). Image is 130mm x 65mm when used in ProofOfConcept.py.

APPENDIX B: CODE USED IN PROJECT

Code B1: StereoCapture.py

Pseudocode:

1. while Program is running
    1.1. Capture and show stereo image frame
    1.2. if user presses 's' key
        1.2.1. Save left image to './images/stereoLeft/' as img*N*.png; //N = the number of times the 's' key has been pressed
        1.2.2. Save left image to './images/stereoRight/' as img*N*.png;
        1.2.3. Increase *N* by 1

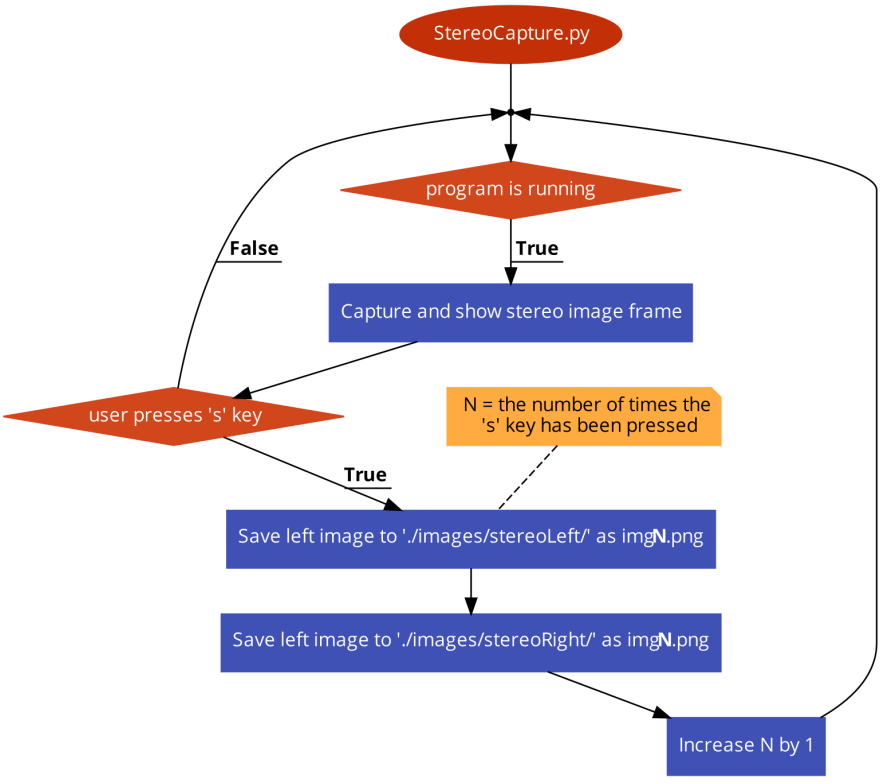

Figure 7.3: Flow Diagram of StereoCapture.py Pseudocode

Code B2: StereoCalibration.py

Pseudocode:

1. Define chessboard as 10 by 7
2. for i=0;i<5;i++;
    2.1. Load left and right stereo image
    2.2. Convert stereo image to grayscale
    2.3. Find and save chessboard corners in stereo image
    2.4. Draw corners onto stereo image
    2.5. Wait for user to press 'space' key
3. Calibrate individual cameras //To get camera matrix
4. Stereo calibrate the stereo camera //To get transformation Matrix
5. Stereo rectify the stereo camera //To get rotational matrix
6. Convert rectification matrix into rectification map // rotation + transformation = rectification
7. Store rectification into .xml file
8. Remap the stereo images using the rectification map
9. Show users the unrectified and rectified stereo images

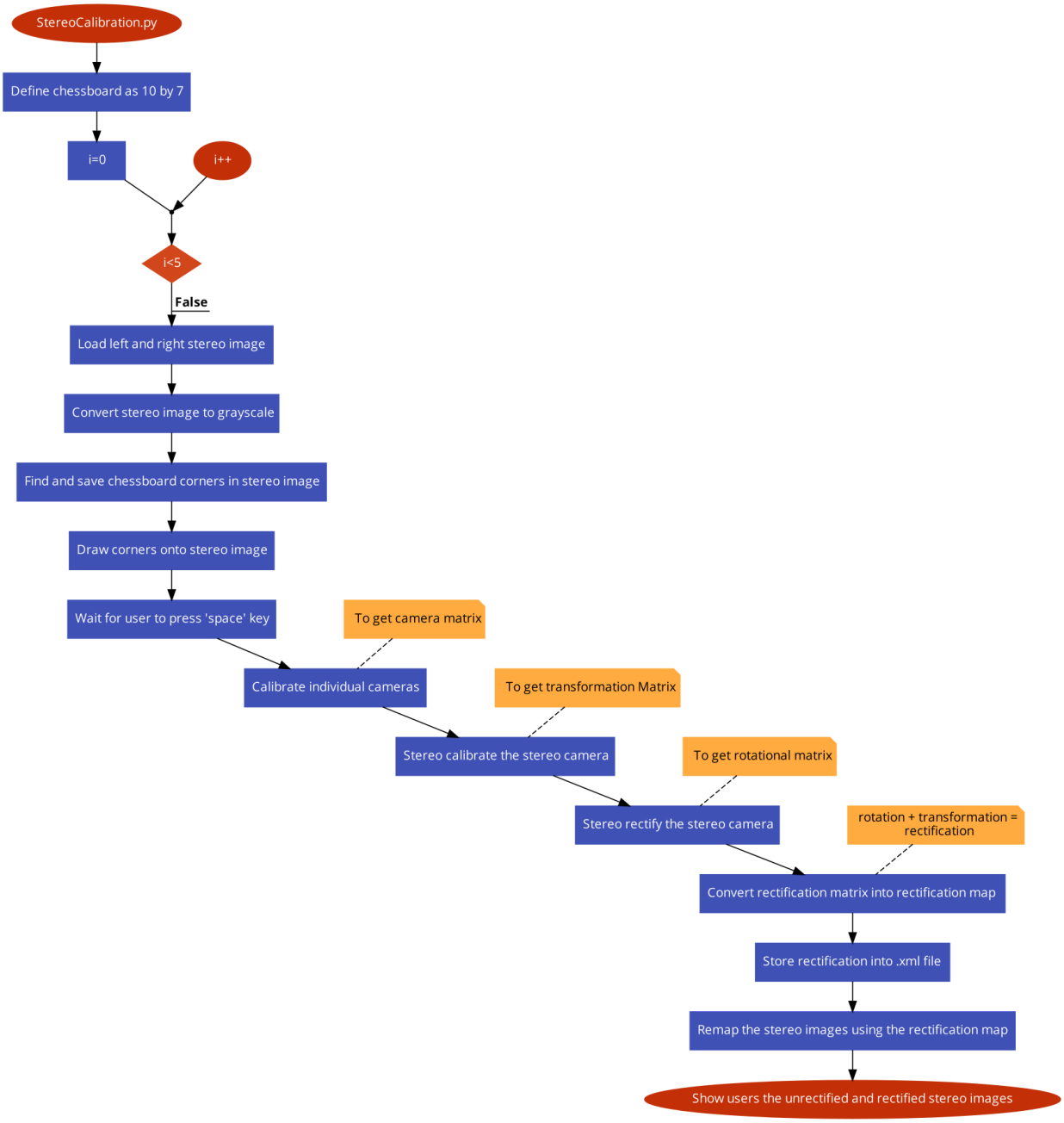

Figure 7.4: Flow Diagram of StereoCalibration.py Pseudocode

Code B3: DrawBoxCrop.py

Pseudocode:

1. Read rectification matrix file
2. Read stereo images
3. Create stereo semi-global block matching object
4. While the program is running
    4.1. Rectify Stereo image
    4.2. Convert image to black and white
    4.3. Compute the disparity map
    4.4. If the user highlights a region in the disparity map
        4.4.1. Create mask based on the highest disparity value in the highlighted region
        4.4.2. Bind mask in rectangular box
        4.4.3. Calculate average disparity of mask box
        4.4.4. Calculate distance of stereo camera to mask box
        4.4.5. Calculate width and height of mask box
        4.4.6. Draw bounding box and estimated distance and size on image

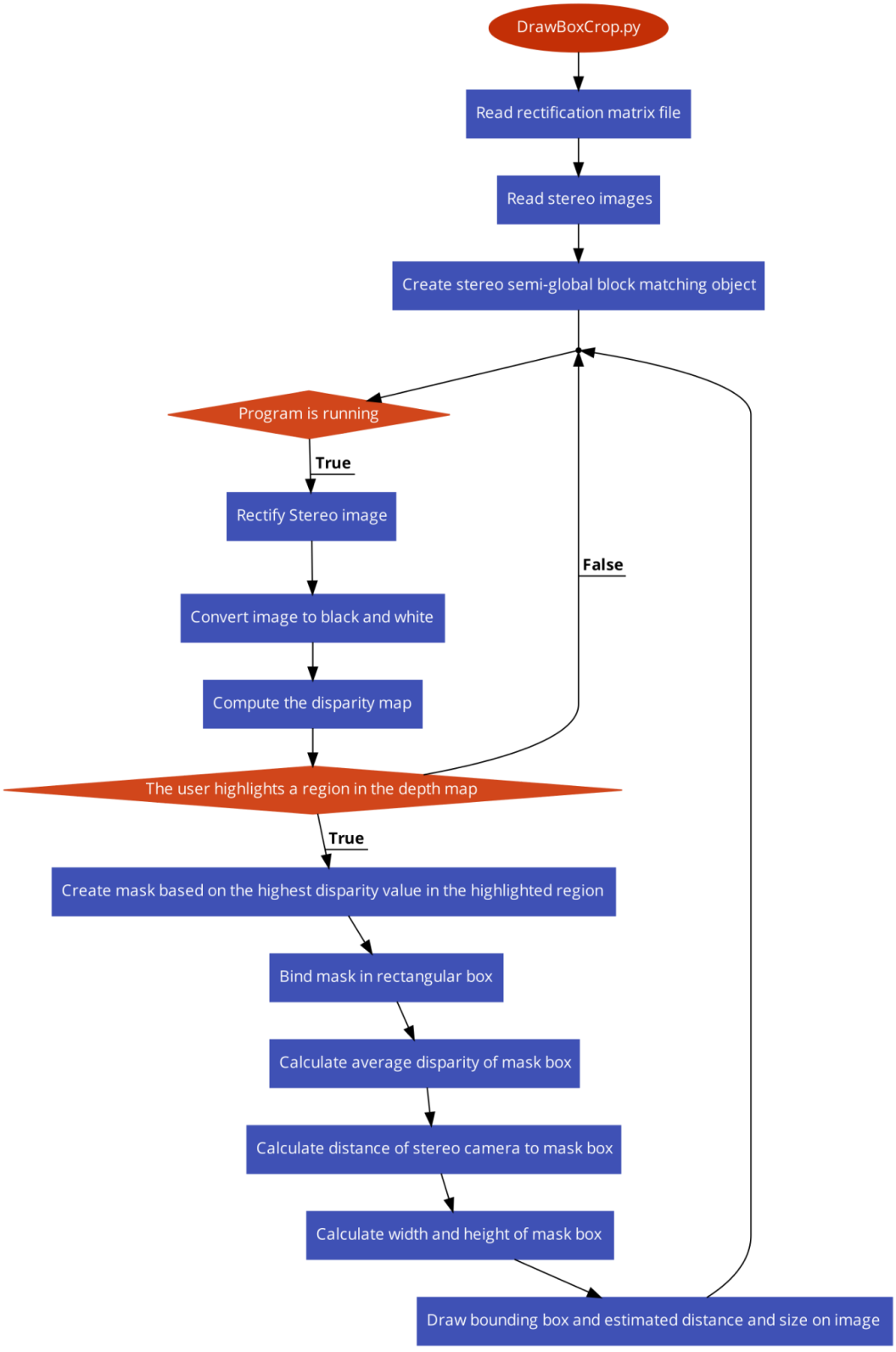

Figure 7.5: Flow diagram of DrawBoxCrop.py pseudocode

Code B4: ObstacleDetector.py

Pseudocode:

1. Read rectification matrix file
2. Read stereo images
3. Create stereo semi-global block matching object
4. while Program is running
    4.1. Rectify Stereo image
    4.2. Crop Stereo Image
    4.3. Convert image to black and white
    4.4. Compute the disparity map
    4.5. User selects maximum distance
    4.6. User selects minimum distance
    4.7. Create mask that filters out disparities in the disparity map that is not in the range selected by the user
    4.8. Bind mask in rectangular box
    4.9. Calculate average disparity of mask box
    4.10. Calculate distance of stereo camera to mask box
    4.11. Calculate width and height of mask box
    4.12. Draw bounding box and estimated distance and size on image

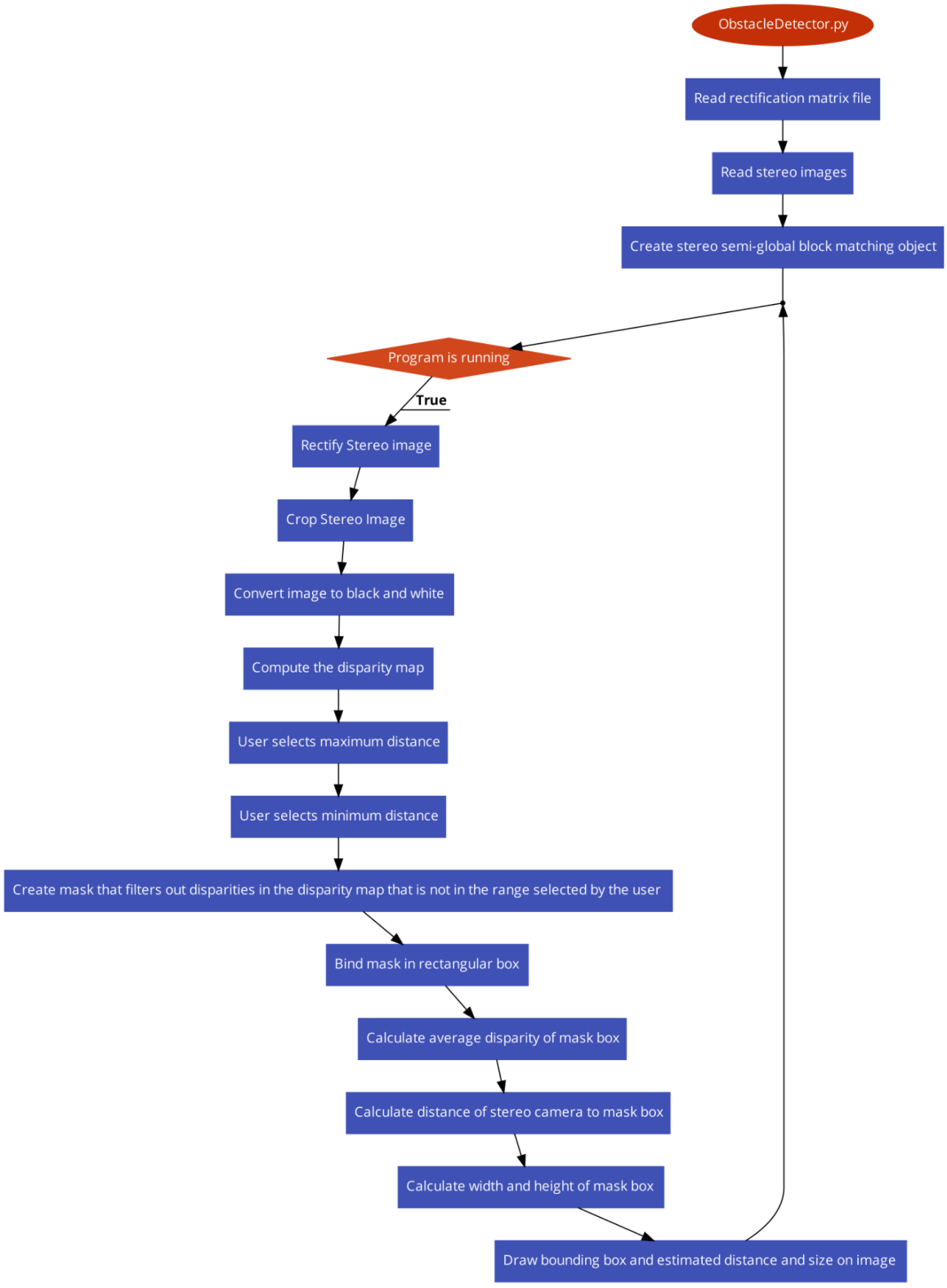

Figure 7.6: Flow diagram of ObstacleDetector.py pseudocode

Code B5: ProofOfConcept.py

Pseudocode:

1. Read rectification matrix file
2. while program is running
    2.1. Rectify Cameras
    2.2. Capture stereo image frame
    2.3. Look for largest red object in frame
    2.4. Draw bounding box around the red object detected
    2.5. Calculate difference in x position of the red object between the left and right stereo image to get disparity
    2.6. Calculate distance from stereo camera to the red object using disparity
    2.7. Calculate width of the red object using estimated distance
    2.8. Draw bounding box and estimated size and distance on the stereo image

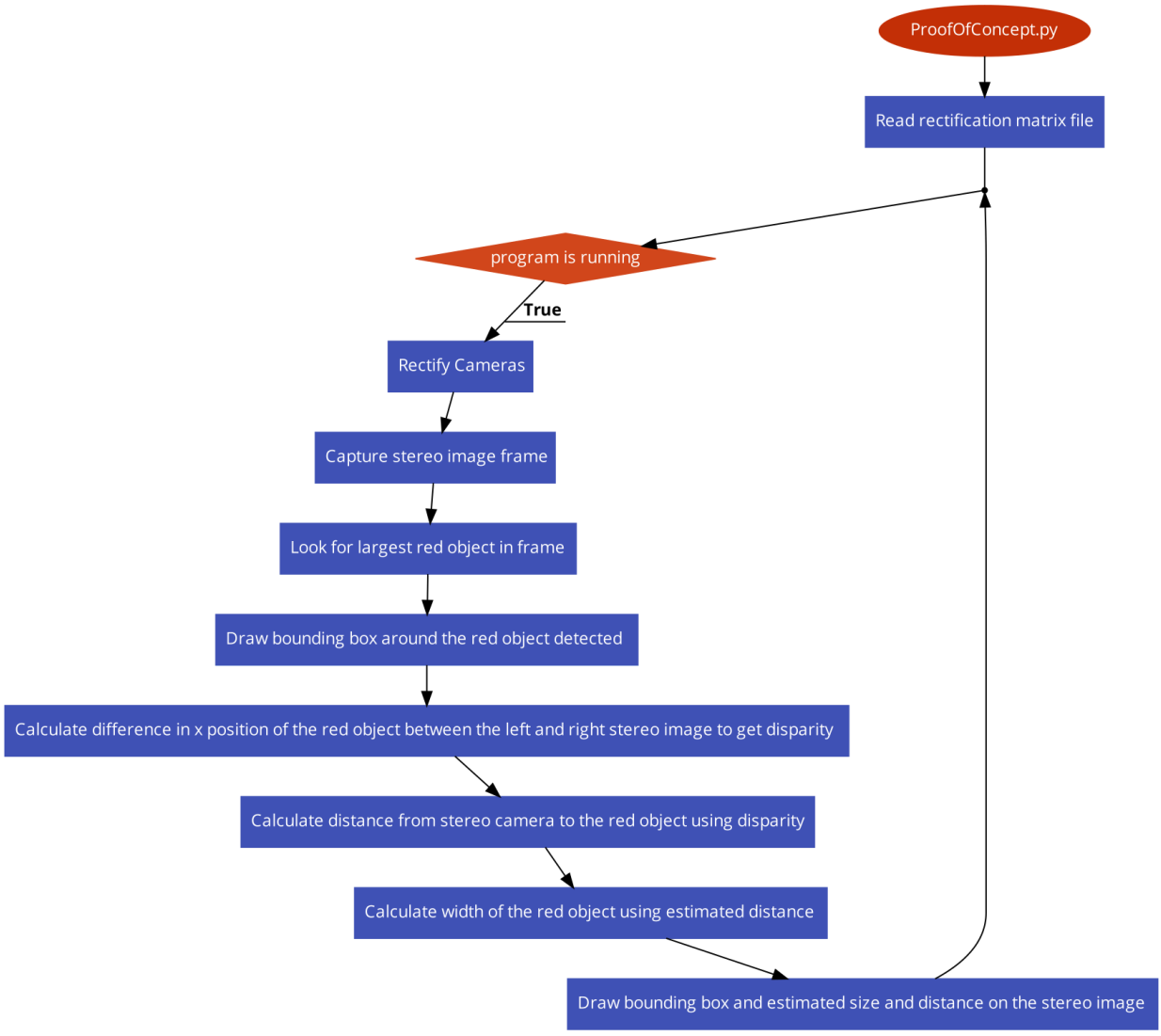

Figure 7.7:Flow diagram of ProofOfConcept.py

Code B6: DMapSliderTest.py

Pseudocode:

1. Load in stereo images
2. Read rectification matrix file
3. Create window for the trackbar
4. Create trackbar for each parameter for StereoSGBM
5. Create StereoSGBM object
6. Convert stereo images to grayscale
7. Rectify stereo images
8. while Program is running
    8.1. Track and save the trackbar values into respective variables
    8.2. Set the parameters of the StereoSGBM as the trackbar values
    8.3. Compute stereoSGBM to generate disparity map
    8.4. Show disparity map to users

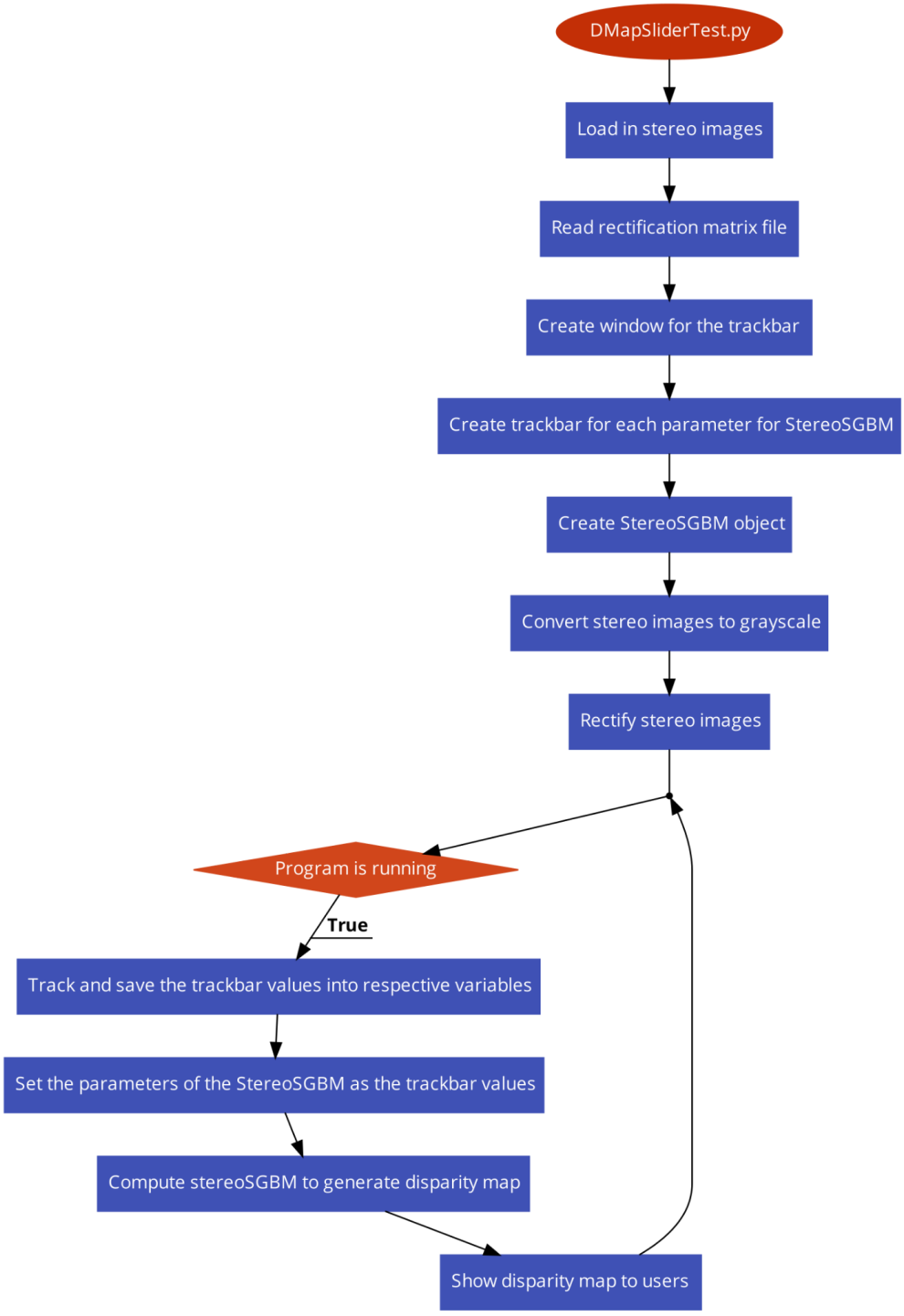

Figure 7.8: Flow Diagram of DMapSliderTest.py Pseudocode

Code B7: CalibrationAccuracy.py

Pseudocode:

1. Read rectification matrix file
2. While the user does not press 's' key
    2.1. Rectify Cameras;
    2.2. Capture stereo image frame // When the user presses ‘s’ key,
3. Find chessboard points in image
4. Draw chessboard points onto image // starting from the first chessboard point,
5. While not all chessboard points checked
    5.1. Get chessboard point on left stereo image
    5.2. Calculate difference between that point and its correspondent point in the right stereo image
    5.3. Save that difference in an array
    5.4. Go to next point
6. Calculate total difference of all points
7. Calculate average difference of all points
8. if Average error less than 1
    8.1. Calibration is considered accurate
9. else
    9.1. Stereo cameras need to be calibrated again

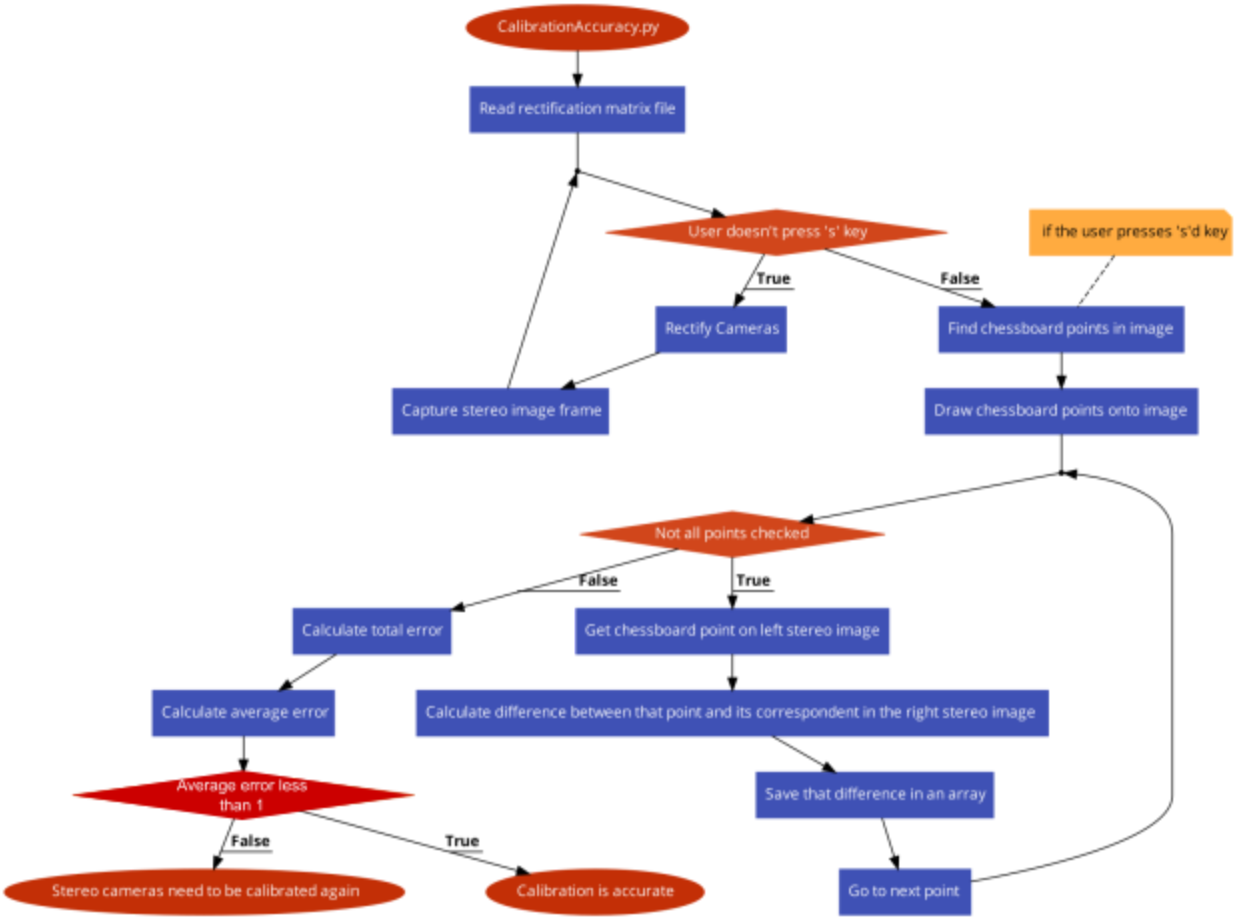

Figure 7.9:Flow diagram of CalibrationAccuracy.py pseudocode

APPENDIX C: VALUE OF PARAMETERS FOR STEREO BLOCK MATCHING

The parameters for the stereo semi-global block matching algorithm are the following.

1. numDisparities - sets a range of disparities to be considered. Higher value means more disparities to be considered but increases computation time. Must be a multiple of 16 as StereoSGBM computes disparities in 16-pixel increments. Should be set to a value appropriate for the dimensions of the stereo image.

2. minDisparity - sets the minimum difference in pixel values between left and right images to consider as a match

3. blockSize - sets the window size used for the matching. Larger window improves quality but increases computation time. Should be an odd number as the window size needs a center pixel. Level of detail in the input image should be considered.

4. preFilterType - controls the pre-filtering step in StereoSGBM, used to reduce noise and improve the quality of the input images. sets the filter type used (Gaussian or median filter).

5. preFilterSize - sets the Gaussian filter size used during the pre-filtering. Recommended to be an odd number to avoid offset.

6. preFilterCap - sets the maximum intensity value that the filter can produce. Recommended to be a value between 1 and 63 as setting it too high can cause the filter to saturate.

7. textureThreshold - controls minimum texture threshold in the stereo images.

8. uniquenessRatio - sets threshold for uniqueness. A match is only considered valid when the ratio of the best match to the second best match is greater than the uniqueness ratio.

9. speckleRange - sets the maximum disparity variation in a contour.

10. speckleWindowSize - Sets the window size for detecting and removing small regions of disparity commonly caused by noise in the image. Should be smaller than block size as speckles are smaller than the matching window.

11. disp12MaxDiff - sets maximum allowable difference in disparity between the left and right stereo image. Disparities above this threshold are invalid. Used for filtering out large disparities caused by factors such as occlusion.

12. P1 - controls penalty for small difference in disparity between neighboring pixels. Higher P1 means that small differences in disparity will be penalized more heavily. Should be set lower than P1.

13. P2 - controls penalty for larger differences in disparity between neighboring pixels. Higher P2 means that larger differences in disparity will be penalized more heavily. Higher P1 and P2 values can cause over-smoothing, while lower P1 and P2 values will output sharper images.

Through trial and error, the parameters are set to the following values for this project:

1. numDisparities = 64
2. minDisparity = -1
3. blockSize = 5
4. preFilterType = 0
5. preFilterSize = 3
6. preFilterCap = 5
7. textureThreshold = 10
8. uniquenessRatio = 5
9. speckleRange = 5
10. speckleWindowSize = 5
11. disp12MaxDiff = 1
12. P1 = 600

13. P2 = 2400