SEM4005 - CAPSTONE PROJECT 2

FINAL REPORT

Smart Smith Machine (V.2.0)

Academic Supervisors

Dr Paw Yew Chai
Associate Professor
Singapore Institute of Technology

Dr Ng Kian Ann
Assistant Professor
DigiPen Institute of Technology Singapore

Industrial Supervisor

Dr Tan Chin Hiong
Senior Systems Engineer
GovTech Singapore

Project Members

Ang Ming Hui Rachel	(1802123)
Ong Yew Lee Eugene	(1800899)
Vishnuvarthan S/O S Radha Krishnan	(1801308)
Yong Zhining Jasper	(1802696)

Date of Submission : 10th April 2022

Acknowledgement

We would like to express our gratitude and appreciation to the following people who have supported and guided us throughout our CAPSTONE project.

Firstly, we would like to thank our project supervisors, Dr Paw Yew Chai, Programme Leader for SIT-DP BEng (Hons) Systems Engineering (Electromechanical Systems), and Dr Ng Kian Ann, Assistant Professor at DigiPen Institute of Technology, for their continuous support and guidance over the past 10 months. Their belief and support pushed us to perform well above our expectations.

Secondly, we would like to take this opportunity to thank the Module Coordinator, Dr Tang Liang, for ensuring that we had timely access to the relevant information and resources for our CAPSTONE project.

Lastly, we would like to express our gratitude to SIT Student Life for allowing us to access the gym at the SIT@Dover campus to work on our CAPSTONE project, despite the COVID-19 pandemic.

Abstract

As technology advances exponentially over time, people find ways to incorporate technology into their daily activities to improve quality of life through convenience and efficiency. Smart systems have made their way into the global market in areas such as home, transportation, dining, and many other sectors over the last decade or so, with an ever-increasing demand. For the most part, fitness has remained largely mechanical in nature for centuries, particularly when it comes to weight lifting exercises. The smart smith machine concept aims to improve exercise monitoring and tracking by incorporating smart technology. With image-detecting cameras, sensors, and computers, new systems for detecting weights and counting repetitions, among other things, may well be implemented.

This concept was tested and proven plausible to the appealing public in the initial version of the smart smith machine. However, it faced several issues such as poor latency time and a less efficient method for weight detection by means of color-coding. The project team hopes to improve on this concept further in this iteration of the project, bringing it closer to being a complete product ready for public use.

The project team decided to use the NVIDIA Jetson Nano as the main driver for the project with the Intel RealSense D435 Depth Camera acting as the sensor node providing both computer vision and distance data. To enhance the thermal performance of the Jetson Nano board, a 40 mm fan was mounted onto the heat sink, and an Edimax EW-7811UTC USB Wi-Fi Dongle was utilized to provide the board with wireless connectivity. These parts are to be mounted on or close to the Smith Machine to reduce the need for extensive cable management.

The board has been configured with the Jetson Nano Developer Kit SD Card Image running Ubuntu. Several project critical software that are critical for the project, such as Python 3, Intel RealSense Viewer, PyRealSense2 modules, Darknet YOLOv3, LabelIMG, OpenCV, and Visual Studio Code, were installed onto the board successfully. Additional quality-of-life software were installed, such as the Jetson Thermal Monitor, GPU Activity Monitor, and XDRP Server for Remote Control.

This report will elaborate on the work done by the project team in accordance with the Gantt Chart, in terms of the development of the software, hardware and other aspects of the smart smith machine system concept.

Table of Contents

List of Figures

List of Tables

Glossary of Terms

Term	Abbreviation
BOM	Bill of Material
FOV	Field-of-View
GPU	Graphics Processing Unit
MCU	Microcontroller Unit
OS	Operating System
PLA	Polylactic Acid
RGB	Red, Green, Blue
TOF	Time-of-Flight
YOLO	You Only Look Once

1. Introduction

1.1 Project Overview & Objectives

Many gym-goers currently find it difficult to digitally track their gym workout progress. They would typically track by means of a) recording their workout progress with paper and pen, which is cumbersome, unstructured, and unsearchable; b) manually keying in their workout progress into fitness tracking applications; or c) using specific gym equipment vendor applications, which does not allow for cross-platform/branding compatibility.

SmartGym is a vision whereby any individual can work out at any community gym across the island and have their workouts seamlessly recorded and easily accessible on a cloud.

SmartGym aims to be "Every citizen's #1 fitness-lifestyle companion", a one-stop fitness hub whereby everyone can work out at any community gym across the island and have their workouts seamlessly recorded and easily accessible on the cloud. It is designed to be interoperable with various gym equipment vendors. This way, SmartGym enhances the individual's overall awareness of their fitness lifestyle and paves the way for uplifting the population's wellness through personalized fitness services.

These services could include real-time repetition counting, personalized calories computation, body composition measurements, anomaly detection for injury prevention, posture correction, workout routine recommendation, community building through neighbourhood challenges and gamification through achievement challenges. SmartGym currently supports popular fitness equipment such as weight stack, exercise bike, and treadmill. The list of supported fitness equipment and activities would be progressively expanded.

The project aims to bring the smith machine onto the SmartGym platform. The smith machine is known to be a versatile equipment allowing users to perform different kinds of workouts. The variables interacting with the smith machine involve its different applicable weight plates and user positioning. The project would benefit the large user base while also allowing the project team to perform research and development for future implementations on more challenging projects such as free weights.

Gradually, the project will be continuous and extended to several teams of students to work on. The following table will document the project versions; with the previous, current or future teams that worked on it respectively. On the next page, Table 1.1.1 shows the teams and corresponding members who have worked on the SmartGym project, as well as the current team working on the project.

Table 1.1.1 - SmartGym Project Teams

Version	Team
1	Team 1: (Aw Kang Jie, Shawn Loo, Nitro Low, Ahmad Syakir, Wilren Foo)
2 [Current Team]	Team 2: (Ang Ming Hui Rachel, Ong Yew Lee Eugene, Vishnuvarthan S/O S Radha Krishnan, Yong Zhining Jasper)

1.2 Project Scope

As this project is expected to span across several capstone project groups, Table 1.2.1 contains the stakeholder requirements checklist of what has been done, what is going to be done in this project, and what can be expected to be done in subsequent batches. Subsequently, Table 1.2.2 states the project constraints.

Table 1.2.1 – Stakeholder Requirements Checklist

Code	Stakeholder Requirements	Status	Done By:	Remarks
SH_N1	The total weights of the standard weight plates used by the user on the Smith Machine should be recorded.	Partially done, to be improved on in this project.	Done by Team 1; To be improved by Team 2.	Currently identified using coloured tapes. Aim to identify without coloured tapes.
SH_N2	The system should be able to be modularly mounted across different Smith Machines by changing the system mounts.	Partially done, to be improved on in this project.	Done by Team 1; To be improved by Team 2.	Currently using raspberry pi and pi cameras, resulting in a high chance of changing components resulting in different mounts.
SH_N3	The system should not damage the Smith Machine such that it voids the warranty of the machine.	Done, to take note for subsequent implementations.	Done by Team 1; Team 2 to take note.	Any changes to the system should not damage the Smith Machine.
SH_N4	The system should not pose any risk to obstructing normal operations of the Smith Machine.	Done, to take note for subsequent implementations.	Done by Team 1; Team 2 to take note.	Any changes to the system should not pose any risk to the normal operations of the Smith Machine.

SH_N5	The number of repetitions of an exercise done by the user should be recorded.	Partially done, to be improved on in this project.	Done by Team 1; To improve by Team 2.	High latency between detection of repetitions, to improve the system and reduce said latency.
SH_N6	The system should be able to detect the number of sets of an exercise done by the user.	To be done in this project.	To be done by Team 2.	
SH_N7	The system should be able to export the data gathered.	To be done by future batches	TBC	
SH_N8	The system should be able to detect the type of exercise done.	To be done by future batches.	TBC.	
SH_N9	The system should be able to integrate with the existing GovTech Smart Gym application.	To be done by future batches.	TBC.	
SH_N10	The system should be able to detect whether the weights are balanced across the barbell.	To be done by future batches.	TBC.	

Table 1.2.2 – Project Constraints

Code	Constraints
PC_N1	System must not modify the smith machine
PC_N2	Cost of project must be within SGD 1000

Based on the specified stakeholder requirements, the project team aims to design and develop a system to:

1. Detect the total weight of the weight plates in use
2. Detect the movement of weights (distance, etc.)
3. Count repetitions and sets
4. Improve latency of data transmission for computation

1.3 System Requirements

This section describes the translated stakeholder requirements and project constraints into system requirements. The methods for verifying and validating system requirements are also elaborated. Table 1.3.1 and Table 1.3.2 reflect the system requirements and the methods for verification and validation of said system requirements respectively.

Table 1.3.1 – System Requirements

Acronyms	Systems Requirements	Source
SYS_R1	System should be able to be retrofitted on different smith machines	SH_N2, PC_N1, SH_N3, SH_N4
SYS_R2	The system should be able to identify the total weights of the standard weight plates on the Smith Machine.	SH_N1
SYS_R3	The system should be able to record the number of repetitions of an exercise done by the user.	SH_N5
SYS_R4	The system should be able to record the sets of repetitions of an exercise done by the user.	SH_N6
SYS_R5	System should cost under SGD 1000	PC_N2

Table 1.3.2 – System Requirements Verification and Validation

Ref No.	System Requirements	Validation & Verification Method	Description
SYS_R1	System should be able to be retrofitted on different smith machines	Demo	The system should be retrofitted onto the smith machine
SYS_R2	The system should be able to identify the total weights of the standard weight plates on the Smith Machine	Test	Test the system to identify specific weight plates
SYS_R3	The system should be able to record the number of repetitions of an exercise done by the user	Test	Test the system to record the number of repetitions of an exercise done by the user
SYS_R4	The system should be able to record the sets of repetitions of an exercise done by the user	Test	Test the system to record the sets of repetitions of an exercise done by the user
SYS_R5	System should cost under SGD 1000	Analysis	Have to ensure that the total cost of the components does not exceed SGD 1000

2. Technical Specifications

This section will elaborate on the technical aspects of the project. The emphasis will be on the selection of components, supported with appropriate justification and technical data. The MCU, Depth Camera and Wi-Fi Dongle are among the components required for this project.

2.1 Component Selection

Component selection is an important aspect of any project. The components chosen for implementation will predominantly define or pave the way for the overall system. For this project, the required components have been thoroughly researched and chosen to achieve the system outcomes.

2.1.1 MCU Selection

A Microcontroller Unit (MCU) is an embedded controller that comprises a processor unit, memory modules, communication interfaces, and other peripherals. There are various kinds of MCU boards that are available on the market, and they can be used for a variety of applications.

Given the features to be implemented, such as weight detection and distance-based repetition counting utilizing image tracking, the MCU would require immense computational power. Table 2.1.1.1 depicts a component comparison table for the selection of the MCU. Based on the research conducted [1, 2, 3], three suitable options have been elaborated in the table.

Table 2.1.1.1 – MCU Module Comparison Table

MCU	Raspberry Pi 4 MODBP - 4GB	Rock Pi N10 - RK3399Pro 4GB	NVIDIA Jetson Nano Developer Kit - 4GB
Ports & I/O Pins	• 2 x USB 3.0 ports • 2 x USB 2.0 ports • 1 x USB C port for power • 3.5-mm analog audio-video jack • 2 x Micro-HDMI ports • Camera Serial Interface (CSI) • Display Serial Interface (DSI) • 40-pin GPIO	• 1 x USB 3.0 • 2 x USB 2.0 • 1 x HDMI 2.0 port • 1 x USB C port for power • 40-pin GPIO, I2C, SPI, UART, etc.	• 4 x USB 3.0 ports • 1 x USB 2.0 Micro-B port • 2 x MIPI CSI-2 DPHY lanes • 1 x HDMI 2.0 port • DisplayPort • 40-pin GPIO, I2C, I2S, SPI, UART, etc.
Processor	• Broadcom BCM2711 system-on-chip • 1.5 GHz Quad-Core 64-bit ARM Cortex-A72 CPU @ 1.5 GHz	• Dual Cortex-A72 @ 1.8GHz • Quad Cortex-A53 @ 1.4GHz • 3-TOPs NPU with 8/16-bit compute	• Quad-Core ARM Cortex-A57 64-bit @ 1.42 GHz
Memory	• 4-GB LPDDR4-2400 SDRAM	• 4-GB LPDDR3	• 4-GB 64-bit LPDDR4
Display	• 4K 60 fps with Dual-Screen functionality	• Supports HDMI 2.0	• Supports HDMI 2.0 and DisplayPort (eDP 1.4)
Ethernet	• Gigabit Ethernet • Built-in Wi-Fi & Bluetooth 5.0 functionality	• Gigabit Ethernet	• Gigabit Ethernet • Wi-Fi (M.2 Key E)
Operating System	• Raspberry Pi OS (Raspbian) • Ubuntu, OSMC, RetroPie, etc.	• Debian / Android 8.1	• Linux4Tegra, based on Ubuntu 18.04
Cost	• Approximately = $55	• Approximately = $99	• Approximately = $100 / $169

Based on the generated module comparison table, each of the MCU modules has its own advantages and disadvantages. The Rock Pi N10 RK3399Pro is more suited for AI purposes due to its deep learning capabilities and powerful performance of data processing speeds. The Raspberry Pi 4 SBC, known to be the most versatile module, may be utilized for basic deep learning and AI tasks such as object recognition and motion detection. And the Jetson Nano with its NVIDIA Maxwell w/ 128 CUDA cores @ 921 MHz is suited for deep learning, heavy image, and graphics processing. After much consideration, the team decided to acquire the NVIDIA Jetson Nano Developer Kit, as it will be more suitable and versatile as compared to the other two modules. Figure 2.1.1.1 shows an image of the procured NVIDIA Jetson Nano Developer Kit.

Figure 2.1.1.1: NVIDIA Jetson Nano Developer Kit - 4GB

2.1.2 Depth Camera Selection

To improve on the previous implementation of the smart smith machine system, which utilized multiple components for separate functions, the project team aims to reduce the workload and data traffic on the MCU by utilizing a single multi-capable component such as depth cameras. Depth cameras provide both the ability to track and detect motion, as well as read the depth/distance of objects with the use of stereoscopic or time-of-flight (TOF) technology. This single component reduces the need for another, and it also boasts superior specifications such as higher depth resolution and field of view among several others. Table 2.1.2.1 depicts a component comparison table for the selection of the depth camera. Based on the research conducted [4], three suitable options have been elaborated in the table.

Table 2.1.2.1 – Depth Camera Comparison Table

Sensor	Azure Kinect	ASUS Xtion 2	Intel® RealSense™ D435
Technology	• Time-of-Flight	• Time-of-Flight	• Infrared Coded Structured Light combined with stereo RGB matching
Depth FOV	• NFOV: 65° • WFOV: 120°	• 52° (can be mounted in vertical position for 74°)	• 65.5° (only 42° for RGB)
Depth Resolution	• NFOV: 640 x 576 • WFOV: 512 x 512	• 640 x 480	• Up to 1280 x 720
Cost	• Approximately = USD$399	• Approximately = USD$430	• Approximately = USD$299

Based on the available options, the Intel® RealSense™ depth cameras were by far identified as the easiest to procure. Furthermore, their depth camera product line includes a variety of specifications and price points. Given the scope of the project, the team decided that the most suitable variant to incorporate would be the D435. Figure 2.1.2.1 shows an image of the procured D435 model Intel® RealSense™ depth camera.

Figure 2.1.2.1: Intel® RealSense™ Depth Camera D435

2.1.3 Wi-Fi Dongle

The Jetson Nano has an ethernet connector for network connectivity. This is one of the most important features of the board since it will need to be able to receive updates and software drivers from the internet, as well as allow for remote control and monitoring. Having an ethernet cable, on the other hand, would be less optimal because it would necessitate a physical wire connection from the board to the internet access port, which is not always within range or practicable to connect. This leads us to the idea of using a wireless Wi-Fi adapter that is compatible with the board, is normally compact, and only uses one USB connection. Table 2.1.3.1 depicts a component comparison table for the selection of the Wi-Fi dongles. With the research conducted [5, 6, 7], and taking into account the compatibility of the dongle with NVIDIA Jetson Nano board, three suitable options have been identified and elaborated in the table below.

Table 2.1.3.1 – Wi-Fi Dongle Comparison Table

Wi-Fi Dongle	Edimax EW-7811UTC AC600 Dual-Band USB Adapter	StarTech.com N150 Wi-Fi USB 2.0 Dongle	TL-WN722N
Interface	• USB 2.0	• USB 2.0	• USB 2.0
Wi-Fi Standard	• 802.11a, 802.11b, 802.11g, 802.11n, 802.11ac	• 802.11b, 802.11g, 802.11n	• 802.11b, 802.11g, 802.11n
Wi-Fi Band	• 2.4GHz + 5GHz	• 2.4GHz	• 2.4GHz
Wireless Data Rate	• Up to 433Mbps (5Ghz) • or 150Mbps (2.4Ghz)	• 150Mbps	• 150Mbps
Disc Installation?	• Not necessary, but need download driver from Edimax Website	• Yes	• Yes
Cost	• Approximately = SGD$26.00	• Approximately = SGD$30.77	• Approximately = SGD$15.12

Out of the specified options above, the only Wi-Fi dongle that does not require a disc installation was the Edimax EW-7811UTC AC600 Dual-Band USB Adapter. This simplifies the process of installing the Wi-Fi adapter on the Jetson Nano and eliminates the chance of installation issues. Apart from its ease of use, the Edimax also offers greater data speeds compared to the others and it is the most recommended adapter to be used with the Jetson Nano board. Figure 2.1.3.1 shows an image of the procured Edimax Wi-Fi Dongle.

Figure 2.1.3.1: Edimax EW-7811UTC AC600 Dual-Band USB Adapter

3. Camera Mount Concept Design

Designing a modular system that can be adaptable and mounted onto various Smith Machines may be approached in a variety of ways. The project team conducted research to identify existing camera mounts that would align with the system's requirements. Given that a single depth camera is the predominant component for performing the system's application, an interface between the camera and its mount was to be designed. Furthermore, given that the depth camera has a minimum field-of-view (FOV) distance required for image recognition, the camera must be elevated at a certain height from its mounting point. As a result, a cantilever was required to facilitate the camera elevation and to serve as the interface between the mount and the camera.

The most common forms of camera mounts identified were mainly tripod mounts, gimbal grip mounts, and Go-Pro camera mounts. Mounts that were available on the market were considered for their structural built and dynamic panning capabilities, however, they had constraints in terms of design and cost. After much consideration and conducting a thorough evaluation of the physical smith machine, the project team decided to incorporate a C-clamp design for the camera mount. Following the selected C-clamp idea, further research led to the discovery of several distinct ways to implement the design [8], such as the one depicted in Figure 3.1, on the following page. Concept sketches were also created as shown in Figure 3.2.

Figure 3.1: C-Clamp Camera Mount Design

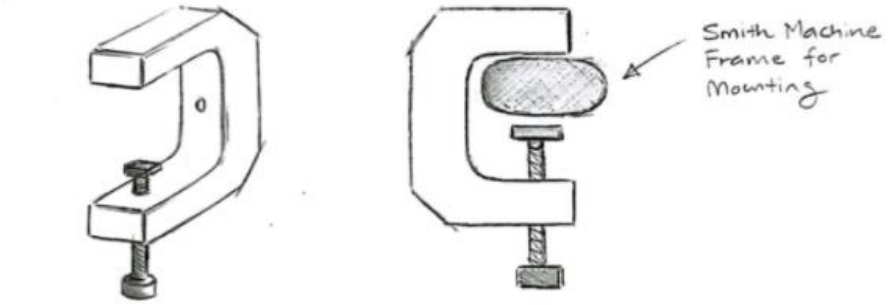


Figure 3.2: Concept Sketch for C-Clamp Camera Mount

Next, an interface was designed for the camera in order to secure it to the system. Taking into consideration that the camera must be allowed to pan vertically and adjusted horizontally, the following design was sketched as shown in Figure 3.3. The depth camera is to be mounted on the L-shaped interface, as shown in the illustration below.

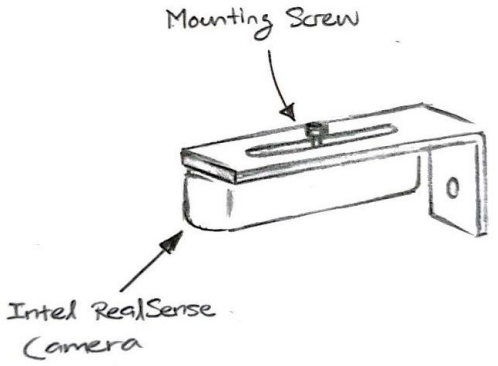


Figure 3.3: Concept Sketch for Camera Interface

Using only a single depth camera, the angular positioning of the device to obtain accurate readings of the barbell's vertical movement and distance, along with a top-down view of the weight plates loaded onto the barbell is critical and pertinent for the software implementation of the system. Essentially the idea is to position the camera interface at the desired angle before fastening it onto the cantilever. As previously stated, the cantilever will serve as the connecting component between the C-clamp camera mount and the camera interface. Figure 3.4 depicts an illustration of the overall design concept for the camera mount system.

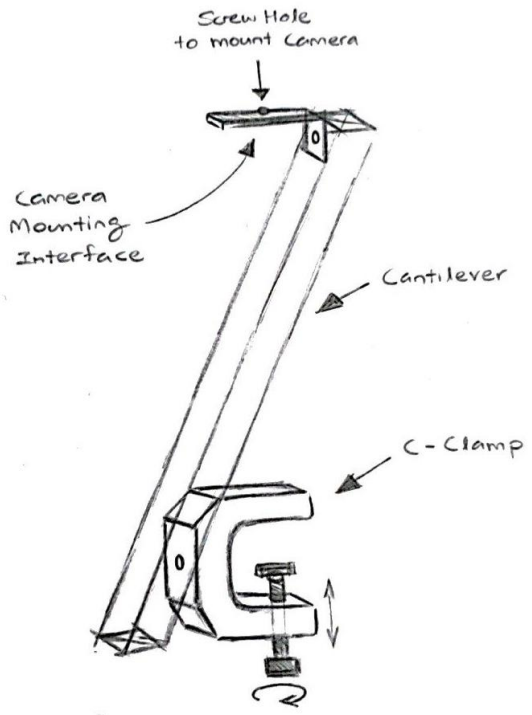


Figure 3.4: Concept Sketch for the Camera Mount System

Based on the specified stakeholder requirements, it is important to ensure that the entire system should be versatile enough to be applied on most or any smith machine without requiring permanent alterations or modifications to the machine itself, such as boring holes or sticking adhesives. Prior to the current team taking on this project, the previous iteration of the camera mount system was designed to be fitted for a specific smith machine and was not adaptable to work on other machines. The design was considerably large in size and required the use of various heavy materials. As a result, the project team opted to re-evaluate and rework the camera mount system design for the Smart Smith Machine.

4. Prototype Development

4.1 First Iteration

As soon as the team was allowed access to the gym at the SIT@Dover campus, measurements were taken and recorded for every required component of the smith machine as shown in Figure 4.1.1. The dimensions of the weight plates were also measured in order to help with software programming. The smith machine's structural frame consists mostly of oval tubings and curved cushions. Having said that, the mounting point for the C-clamp mount has to be carefully chosen for the entire system. After measuring the frames perpendicular to the portion where the weight plates will be put onto the barbell, a suitable flat-top position was selected as the system's mounting point.

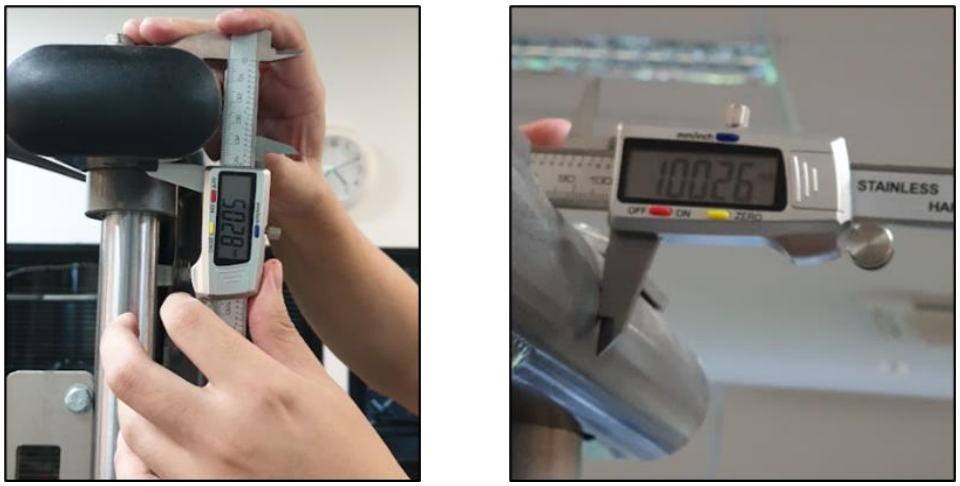


Figure 4.1.1: Measurements of the Smith Machine Frames

Several components, including the camera mount and the camera panning interface, were 3D printed for this first iteration because they were complex in design and were still in the development stage. The Fusion 360 program and Cura 3D Printing Software were used to develop and manufacture the CAD designs. Figure 4.1.2 displays CAD model and the initial prototype for the C-clamp mount.

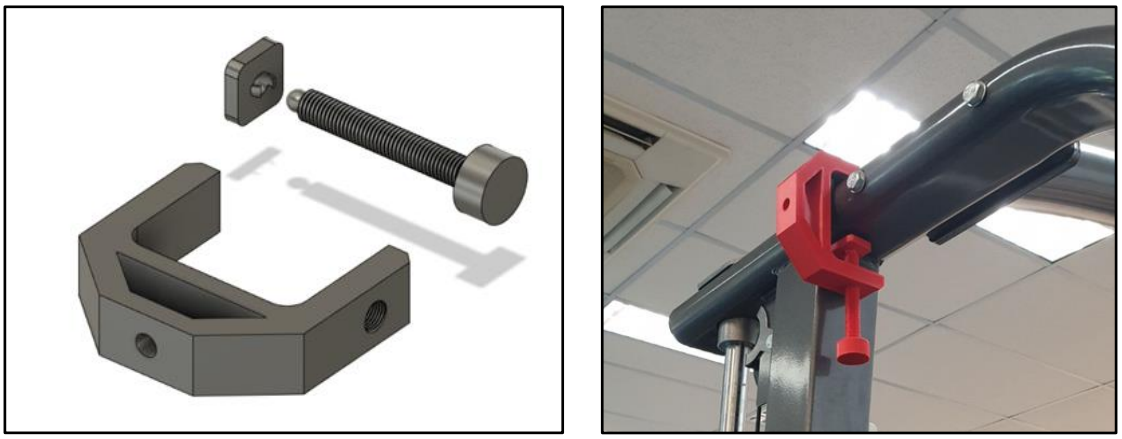


Figure 4.1.2: CAD Model (Left) and the 3D Printed C-Clamp Mount (Right)

The C-clamp mount was revealed to be structurally fragile due to its low density after the initial print. As a result, the infill percentage for printing the component was significantly raised. The threaded hole region of the C-clamp handle, for example, was strengthened with an infill percentage ranging from 50% to 100%. The specifications for the infill percentage are shown in Figure 4.1.3.

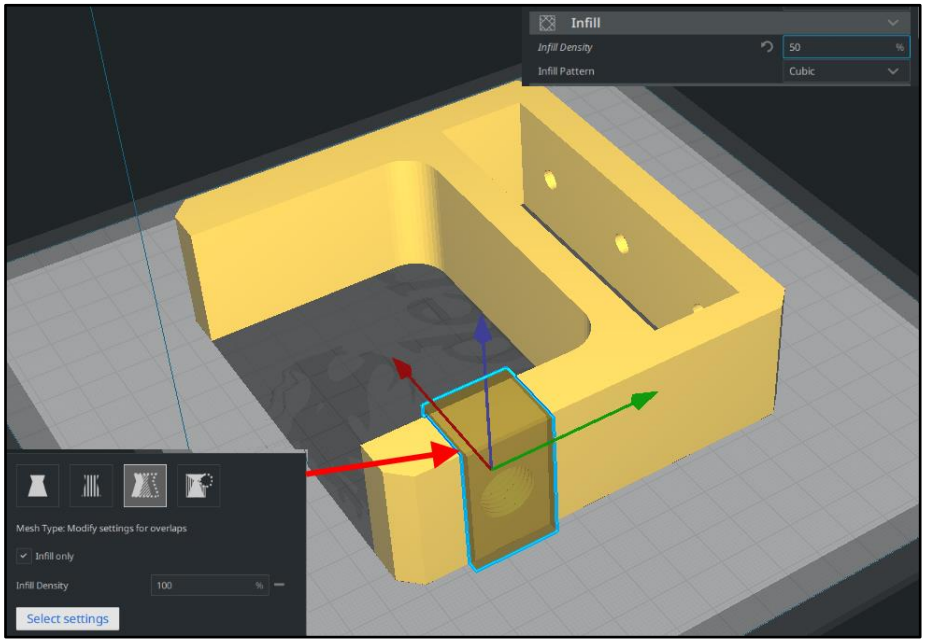


Figure 4.1.3: Infill Settings for the C-Clamp on the Cura 3D Printing Software

Adhesive rubber pads were also included and placed on several contact points of the C-clamp mount, such as the movable jaw and the C-clamp handle, to protect the surface of the smith machine. Furthermore, the paddings provide more grip for the C-clamp mount, preventing it from slipping off the smith machine's smooth metal surface. Figure 4.1.4 shows the C-clamp mount with the adhesive rubber pads attached.

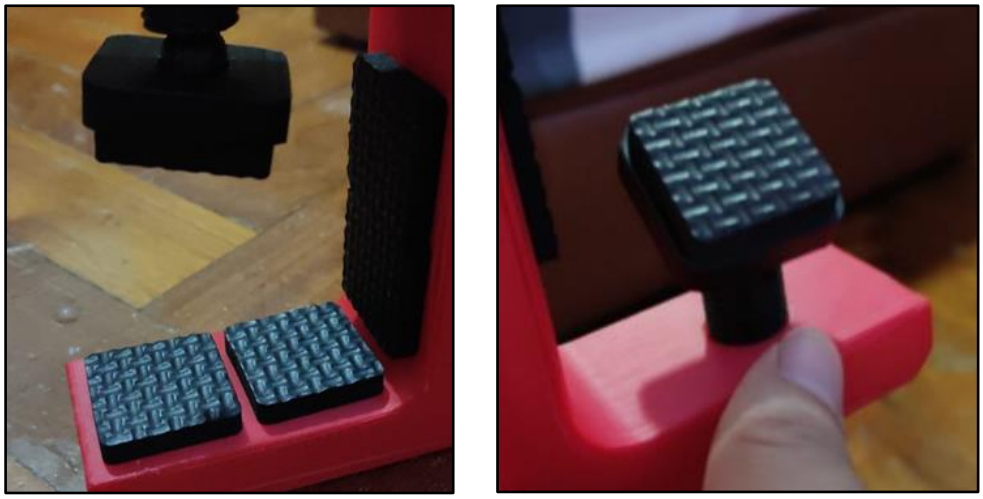


Figure 4.1.4: C-Clamp Mount with Adhesive Rubber Pads

To accurately perform distance tracking, the depth camera must be positioned high enough to accommodate its minimum field of view (FOV) distance. The FOV distance for the Intel RealSense D435 Depth Camera was measured to be approximately 30 cm. To facilitate the camera elevation, an aluminium T-slot measuring 20 mm by 20 mm and 400 mm in length was fabricated. Given that the system is to be modularly fitted across several smith machines, the grooves in the T-slot allow for position adjustments throughout its length. The fabricated T-slot cantilever with the C-clamp mount is shown in Figure 4.1.5. Drop-in fasteners and screws were used to assemble the cantilever arm and C-clamp mount.

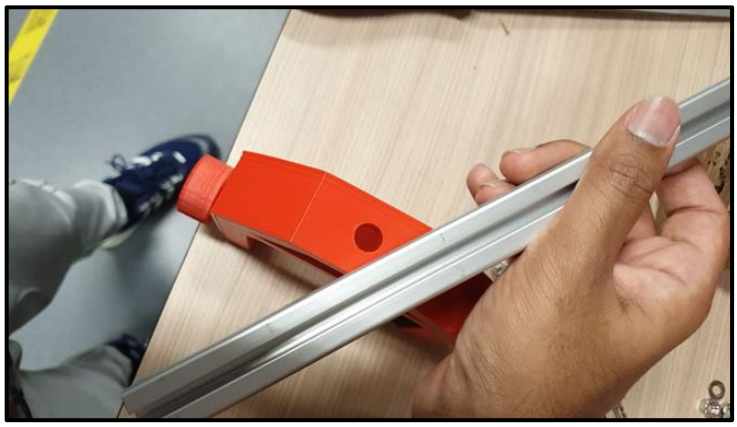


Figure 4.1.5: T-Slot Cantilever Arm

The cantilever arm was designed to function as a single axis arm, allowing the entire body of the system to incline freely in a full 360 degree angle. The depth camera was connected to the camera interface component, which was located at the top of the T-slot, where the camera will be able to pan left and right as well as tilt up and down. Figure 4.1.6 shows the underside of the Intel RealSense D435 depth camera with a screw attachment point for installation.

Figure 4.1.6: Attachment Point of the Intel RealSense Depth Camera

As previously indicated, a simple camera interface L-bracket was designed and fabricated to fit the specifications of the Intel RealSense depth camera. The L-bracket allows the camera to pan while placing the least amount of strain on the cantilever arm. The camera may be allowed to traverse horizontally using the M6 sized slot for adjustment reasons as illustrated on the CAD model in Figure 4.1.7, on the following page.

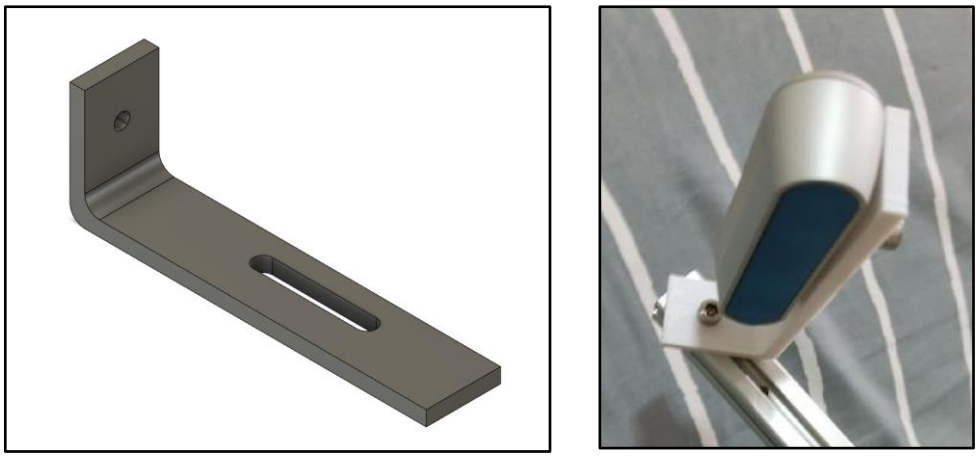


Figure 4.1.7: CAD Model (Left) and the 3D Printed L-Bracket (Right)

When the camera mount components were assembled, it was discovered that the weight of the T-slot was too much for the C-clamp to handle and that it could not be tightened at an angle. As a result, tightening the screw at the mount to secure the camera at the opposite end of the T-slot did not retain it at the required angle. Hence, the team found a workaround by designing and implementing a basic angle locking mechanism. The 3D printed double angle locking mechanism holds the cantilever arm at a fixed angle as shown in Figure 4.1.8.

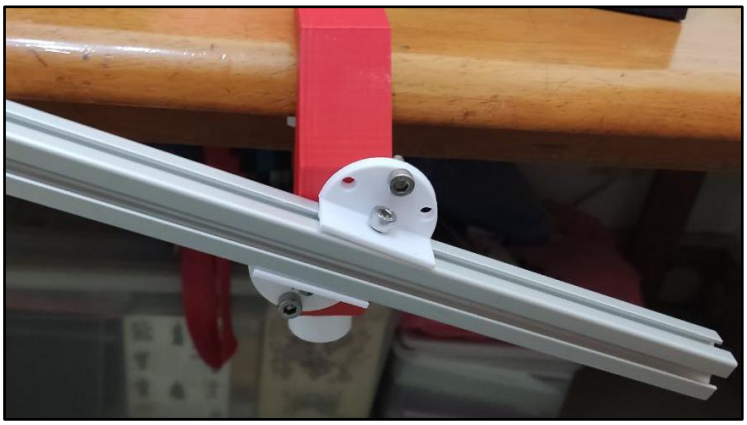


Figure 4.1.8.: Double Angle Locking Mechanism

On the following page, Figure 4.1.9 depicts the complete first iteration of the camera mount system. Given the glaring issue such as the inability to retain the cantilever at its desired angle for the camera, further design development and fabrication is required for the next design iteration.

Figure 4.1.9: First Iteration of the Camera Mount System

4.2 Second Iteration

After conducting a physical test on the first iteration of the camera mount system at the gym, the project team identified the following issues with the current design:

1. The T-slot cantilever arm was too heavy.
2. There was no proper mechanism to secure the cantilever at its desired position and angle.

As a result, it was evident that the T-slot cantilever needed to be replaced with a significantly lighter alternative. Aluminium was unquestionably the material of choice for the cantilever since it is both lightweight and sturdy. After much consideration on the particular grade of aluminium that would be suitable, the team settled on the AA6061 aluminium grade for its exceptional strength and easy machinability. Subsequently, an AA6061 aluminum plate with dimensions of 400 mm x 50 mm (L x B) and a thickness of 1.5 mm was procured and fabricated to produce an improved cantilever arm. Several holes were drilled on the plate for it to be assembled with the C-clamp mount and the camera interface.

In addition, an appropriate angle locking mechanism was incorporated to secure the cantilever arm at its desired position. A rotary dial was designed to facilitate the angle locking of the cantilever arm. The rotary dial was 3D printed, along with a redesigned C-clamp mount, to allow the rotary dial to be

integrated into the design. The rotary dial comprises multiple holes around its radial axis for M5 screws to secure the cantilever at the desired angle. Figure 4.2.1 displays the CAD models of the modified C-clamp mount and the rotary dial respectively. Following that Figure 4.2.2 depicts the physical assembly of the new components.

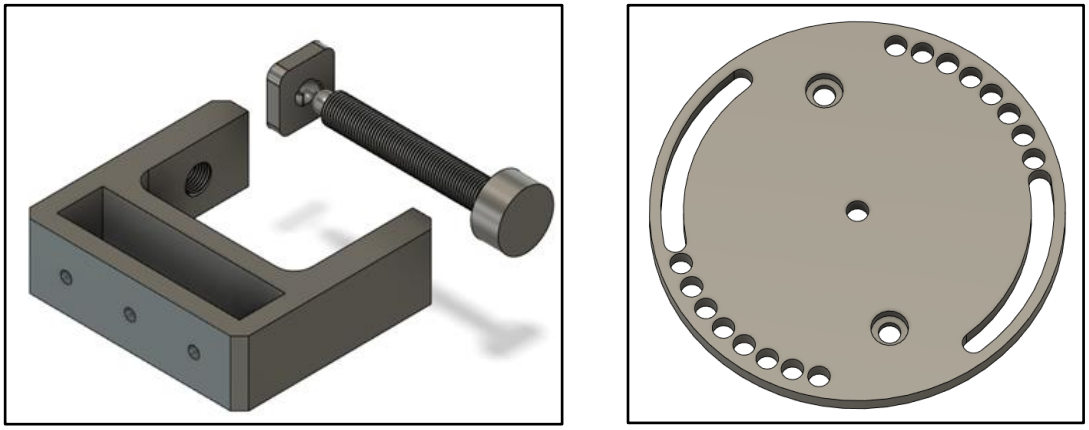


Figure 4.2.1: CAD Models of the C-Clamp Mount and Rotary Dial

Figure 4.2.2: Rotary Dial and C-Clamp Mount Assembly

On the following page, Figure 4.2.3 shows the fabricated aluminum plate assembled with the newly added rotary dial and C-clamp mount. The rotary dial functioned as expected and offered additional security for the cantilever arm, considering that it will be carrying an expensive depth camera on the other end.

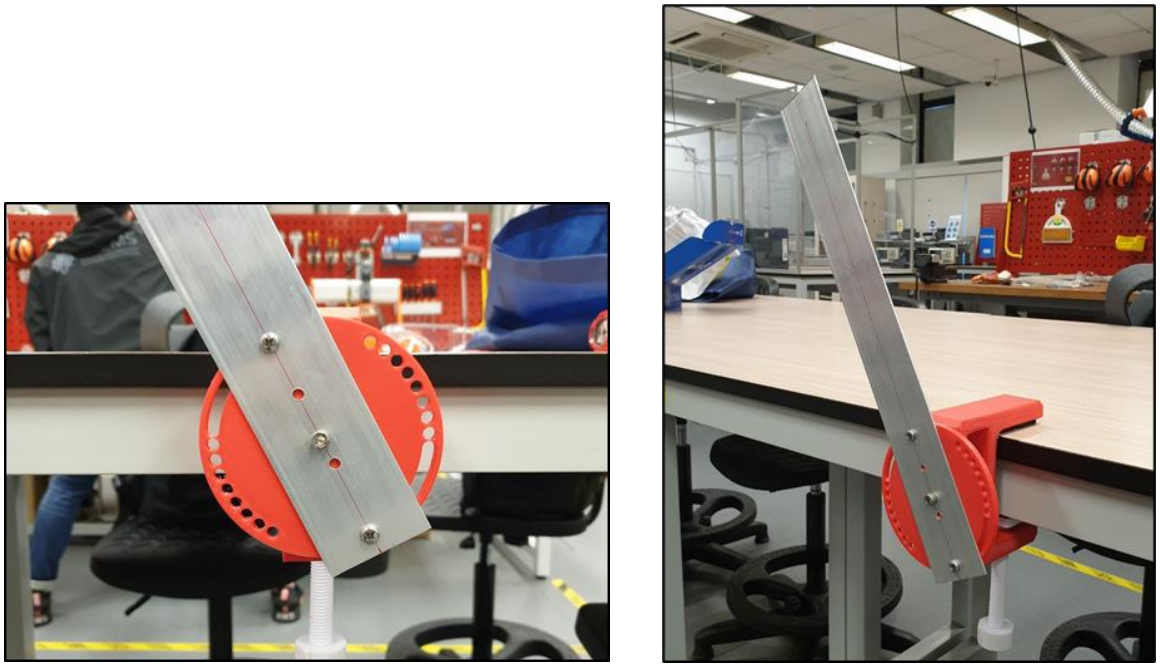


Figure 4.2.3: Aluminium Plate Cantilever Arm with Angle Locking Rotary Dial

Despite being able to support the weight of the camera, a potential problem was discovered while testing the new cantilever. Although the material was strong, the length and thickness of the plate did not prevent it from wobbling when an external force was introduced. Given that most smith machine users in the gym do not always gently place the barbell back onto the machine, the factor of vibration must be considered. Therefore, to reduce and possibly eliminate vibrations from traveling up the cantilever, the cantilever was bent into an L-shape as shown in Figure 4.2.4.

Figure 4.2.4: Fabricated L-Shaped Cantilever Arm

To further improve the existing camera interface, the same concept utilized for the cantilever angular positioning was adopted. A semi-rotary mounting interface was designed and fabricated for the camera. Figure 4.2.5 shows the CAD model for the new camera interface L-bracket and rotary dial, as well as the assembly of the components with the camera, respectively.

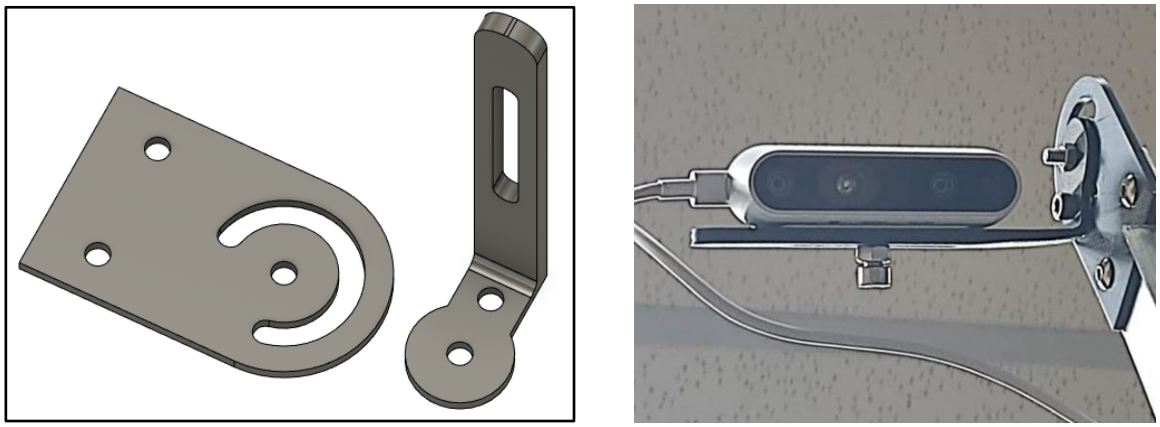


Figure 4.2.5: CAD Model (Left) and the 3D Printed Camera Interface in Application (Right)

Figure 4.2.6 depicts the complete second iteration of the camera mount system, including all the new and modified components. The new method for angular positioning has proven to be successful and has considerably improved the entire hardware of the system from the previous iteration.

Figure 4.2.6: Second Iteration of the Camera Mount System

The project team aims to develop ways to further reduce vibration travel to the camera in the next iteration, as well as improve other features of the camera interface component, such as designing a proper housing for the camera to prevent it from being tampered with or damaged by any foreign object.

4.3 Third Iteration

For the third and final iteration of the camera mount system, several modifications were done to tackle the complications discovered during software testing. The following are the problems that were identified by the project team:

1. The L-shaped cantilever did not entirely reduce the vibration caused when using the machine.
2. The L-bracket was compromised due to the weight of the camera.
3. The camera placement had to be changed from horizontal to vertical.

After much discussion on the necessary modifications to assist the software segment, the project team decided to enhance the designs for the 3D printed components in order to tackle the challenges encountered.

In order to reduce the vibration traversing across the cantilever arm, a prefabricated C-channel was procured as shown in Figure 4.3.1. Given that the C-channel has a more robust construction, the effects of machine vibration would be considerably minimized, preventing the camera from moving at its fixed location. The aluminium grade chosen for the C-channel was AA6063. To interface the C-channel to the other components, the same holes fabricated on the previous aluminum plate were replicated. The sharp edges of the C-channel were also chamfered for safety and to avoid causing damage to any external object.

Figure 4.3.1: Fabricated C-Channel Cantilever Arm

Next, the L-bracket for the camera was modified to be more structurally stable to prevent it from warping due to the weight of the camera. To resolve this, the thickness of the L-bracket and other camera interface components was increased, hence improving the component's overall design. Furthermore, the shorter side of the L-bracket was reinforced for rigidity and to keep the camera horizontally constrained. The design of the rotary dial was also modified to suit the other interfacing components.

Additionally, a housing for the depth camera was designed and 3D printed to protect the camera from being damaged or harmed by any foreign object. Figure 4.3.2 displays the CAD models of the redesigned L-bracket and camera housing.

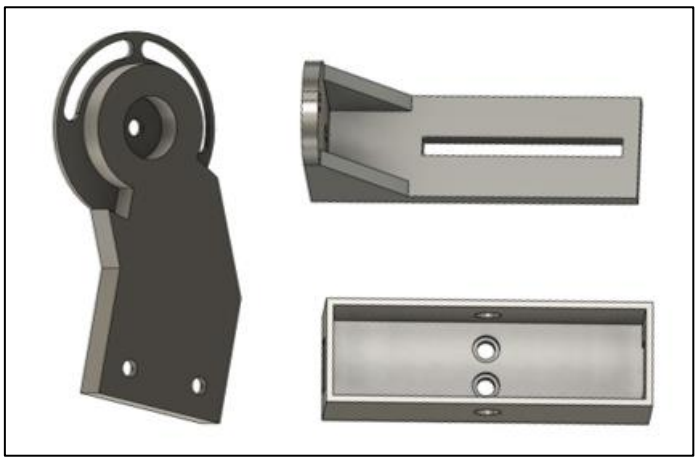


Figure 4.3.2: CAD Models of Modified L-Bracket and Camera Housing

Figure 4.3.3 depicts the assembled camera interface with the modified L-bracket, rotary dial and camera housing unit. With the modified camera interface, the camera can now be positioned vertically and horizontally with more structural integrity compared to the previous iterations.

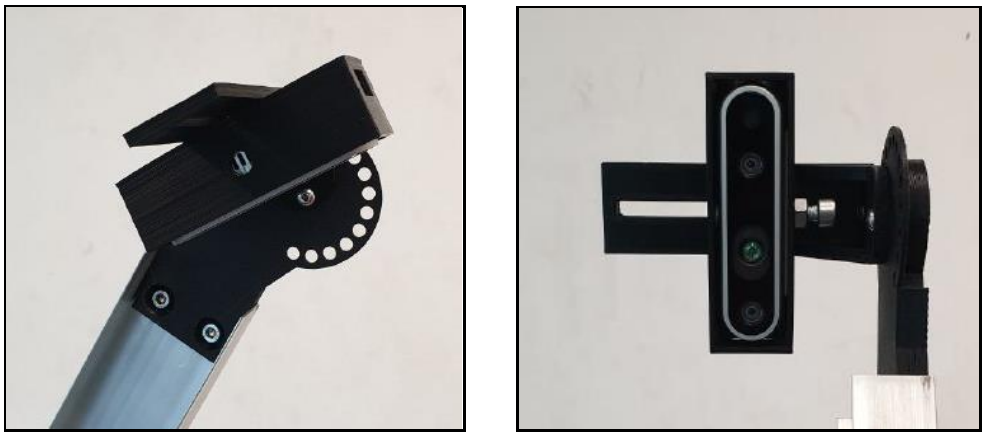


Figure 4.3.3: Fully Assembled Camera Interface

After assembling all the new components for the final iteration of the system, the project team realised that there was an absence of cable management. Hence, several small slide-on cable guides were fabricated and attached C-channel cantilever arm. On the following page, Figures 4.3.4 depicts the 3D printed slide-on cable guides and their installation on the cantilever arm respectively.

Figure 4.3.4: 3D Printed Slide-On Cable Guides (Left) Fitted onto Cantilever Arm (Right)

Figures 4.3.5 and 4.3.6 depict different profiles of the final iteration of the camera mount system, including all the new and modified components. This system is able to be mounted onto and dismounted from the smith machine with ease to perform its various function. Given that most of the components are 3D printed the weight of the entire system is manageable as compared to a fully metal fabricated system.

Figure 4.3.5: Final Iteration of the Camera Mount System (Side)

Figure 4.3.6: Final Iteration of the Camera Mount System (Front)

5. Software Development

5.1 Set-Up

The first step is to create the operating system that the Jetson Nano would use. Following the step-by-step installation guide available on NVIDIA's website, the project team downloaded the Jetson Nano Developer Kit SD Card Image and flashed it onto a Micro SD Card. This allows the SD card to act as the main storage driver and operating system of the board. The Jetson Nano operates using an ARM-based Ubuntu operating system [9].

After successfully installing the operating system, the team proceeded to update all the drivers and firmware for the board. This is to ensure that all the software and firmware that the board has are up to date before proceeding to develop anything. Several additional software were acquired to enhance the working process of the team, such as the XRDP Server for remote desktop connection, a GPU Utilization Monitor, as well as a python-based thermal monitor that displays the board's thermal readings onto a graph.

The XRDP Server allows the team to remotely access the board from another computer [10]. This allows the team the flexibility to work on the board from a separate device, alleviating the need to always have access to a spare keyboard, mouse, and display monitor. This is important as such devices would not be readily available in the gym when the team conducts on-site testing. Although this method of remotely accessing the board is good, it has issues with regards to latency and reduced frame rates depending on the network that the board and the computer are connected to.

The GPU Activity Monitor is a python based tool that displays the GPU activity onto a graph. There are methods to observe the GPU activity directly through the terminal. However, this displays the values onto a constantly updating command-line interface and is not very user-friendly. As such, the team has installed the GPU Activity Monitor to get a visual graphical representation of its utilization. This would be used further on in the project when testing the machine learning algorithms to observe the amount of load the GPU is undergoing, giving feedback on the efficiency of the software. A snapshot of the GPU Activity Monitor is shown in Figure 5.1.1.

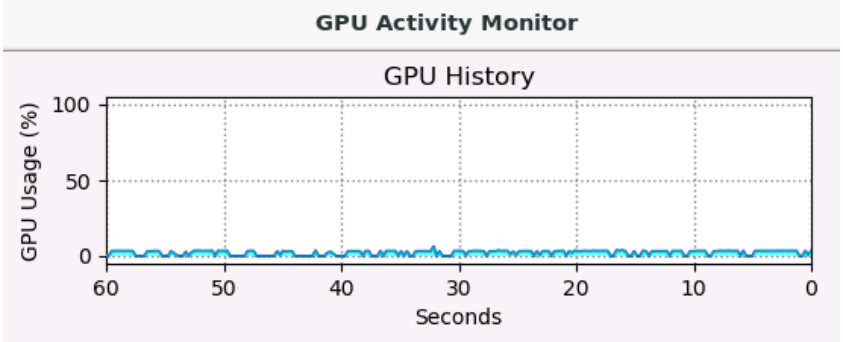


Figure 5.1.1: Snapshot of the GPU Activity Monitor

The python based thermal monitor was installed to allow the team to easily track the temperature of the board's components. This is important as the board may thermal throttle itself to prevent itself from burning out due to overheating when running many complicated tasks, such as streaming live video feed and running machine learning algorithms. Another point of concern is that the project team has used 3D printed parts, such as the chassis for the board, that were made using PLA. PLA can start flexing under pressure from as low as 60 degrees Celsius [11]. As such, the project team needs to ensure that the board is cool enough to avoid this possibility of causing the PLA to flex. A snapshot of the thermal monitor is shown in Figure 5.1.2.

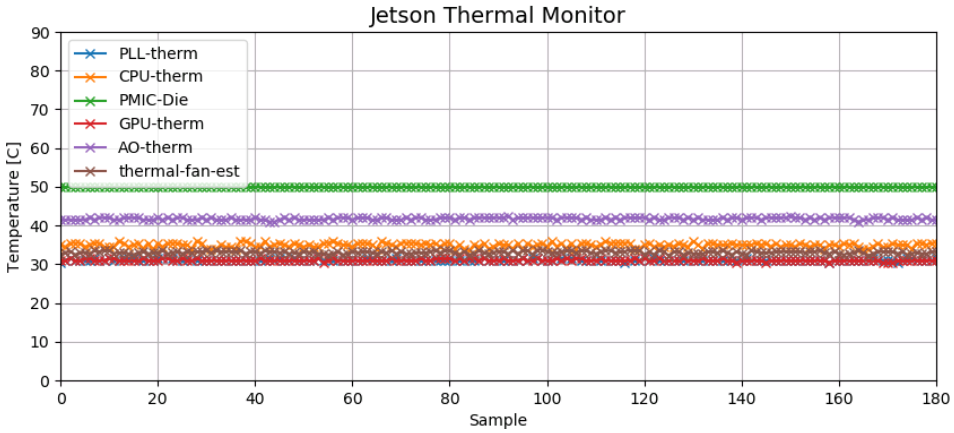


Figure 5.1.2: Snapshot of the Jetson Thermal Monitor

Apart from the quality of life softwares, the project has also installed several key modules that would help the development of the project. The softwares installed were Intel RealSense Viewer, Visual Studio Code, Python 3, OpenCV.

Intel RealSense Viewer is the software required to run the Intel RealSense D435 depth camera that the team used [12]. This was among the first software installed to test the functionality of the procured D435 camera. Aside from the Viewer itself, the team was also in the process of installing its python module PySense 2.0. This would allow the team to utilize the camera in customized python software. However, due to the lack of experience with Ubuntu and Python, the project team was facing difficulties installing the PySense 2.0 module onto the board. On the following page, Figure 5.1.3 depicts a snapshot of the Intel RealSense Viewer.

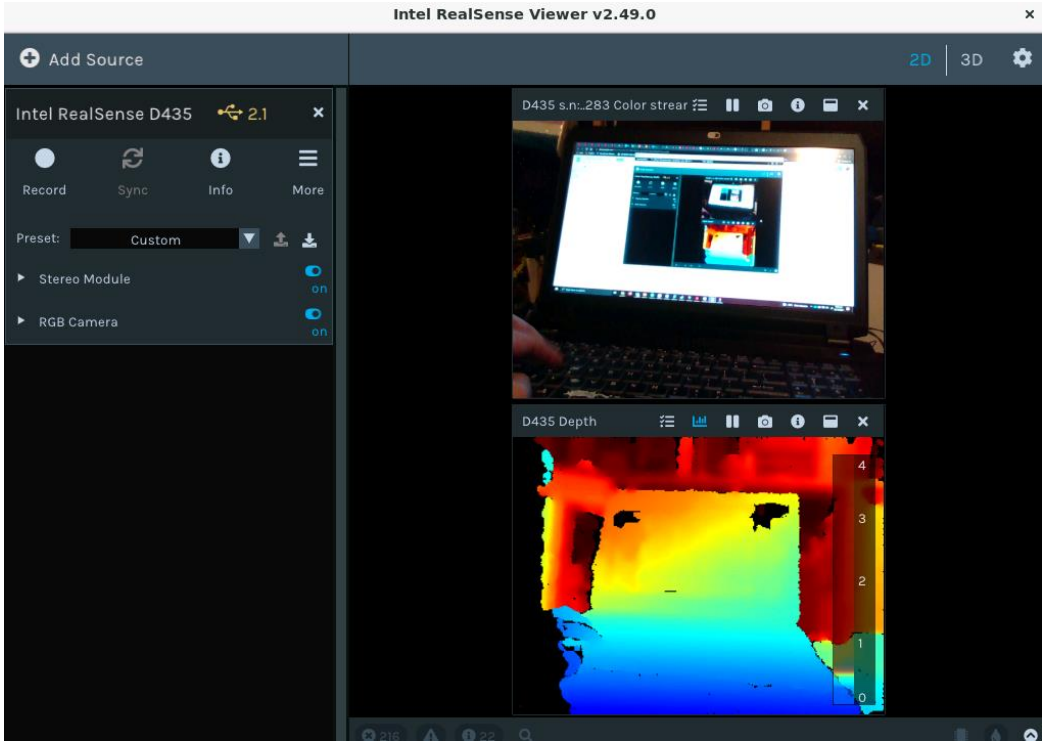


Figure 5.1.3: Snapshot of the Intel RealSense Viewer

Visual Studio Code was installed as the main coding platform for the project. This is a very comprehensive code editor that has many features to allow the smooth development of software projects. Considering that the team would have to spend lots of time programming, this would help the project team be more efficient in developing the code required for the project.

Python 3 was installed as most of the softwares available online were Python-based. There are also many libraries available in Python that the project team would eventually be using to create the project. On top of Python 3, the project team also installed PyTorch. This is an open-source machine learning framework that was included in the Jetson Nano' Developer Kit SD Card Image.

Another software installed was OpenCV. OpenCV is an open-source computer vision and machine learning software library [13]. This software was chosen for the project as the project would need to use both computer vision and machine learning to identify the weights on the Smith Machine.

The last software to install is the Darknet YOLOv3. This is to allow the system to implement object detection to detect the weight plates. This is part of the process to identify the weights on the machine. This software has several requirements, namely CMake, Powershell, CUDA, OpenCV, and cuDNN. All the steps required to install the software can be found on their GitHub page; https://github.com/AlexeyAB/darknet.

5.2 Repetition Counting

In order to create a custom project to use the RealSense camera's distance reading capabilities, the module pyrealsense2 is needed. To install this module, the team followed the steps stated in GitHub's librealsense page; https://github.com/IntelRealSense/librealsense/tree/master/wrappers/python.

The project team faced issues getting python to import the RealSense module and followed the steps shown in Figure 5.2.1 retrieved from https://github.com/IntelRealSense/librealsense/issues/7722.

```
# Installs librealsense and pyrealsense2 on the Jetson NX running Ubuntu 18.04
# and using Python 3
# Tested on a Jetson NX running Ubuntu 18.04 and Python 3.6.9 on 2020-11-04

sudo apt-get update && sudo apt-get -y upgrade
sudo apt-get install -y --no-install-recommends \
    python3 \
    python3-setuptools \
    python3-pip \
    python3-dev

# Install the core packages required to build librealsense libs
sudo apt-get install -y git libssl-dev libusb-1.0-0-dev pkg-config libgtk-3-dev
# Install Distribution-specific packages for Ubuntu 18
sudo apt-get install -y libglfw3-dev libgl1-mesa-dev libglu1-mesa-dev

# Install LibRealSense from source
# We need to build from source because
# the PyPi pip packages are not compatible with Arm processors.
# See link [here](https://github.com/IntelRealSense/librealsense/issues/6964).

# First clone the repository
git clone https://github.com/IntelRealSense/librealsense.git
cd ./librealsense

# Make sure that your RealSense cameras are disconnected at this point
# Run the Intel Realsense permissions script
./scripts/setup_udev_rules.sh

# Now the build
mkdir build && cd build
## Install CMake with Python bindings (that's what the -DBUILD flag is for)
## see link: https://github.com/IntelRealSense/librealsense/tree/master/wrappers/python#building-from-source
cmake ../ -DBUILD_PYTHON_BINDINGS:bool=true
## Recompile and install librealsense binaries
## This is gonna take a while! The -j4 flag means to use 4 cores in parallel
## but you can remove it and simply run `sudo make` instead, which will take longer
sudo make uninstall && sudo make clean && sudo make -j4 && sudo make install

## Export pyrealsense2 to your PYTHONPATH so `import pyrealsense2` works
export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python3.6/pyrealsense2
```

Figure 5.2.1: Script to Install Pyrealsense2 Libraries for Jetson Nano

After successfully installing the module, the project team followed pysource's guide on creating a real-time distance detection software using python and the realsense camera; https://pysource.com/2021/03/11/distance-detection-with-depth-camera-intel-realsense-d435i/.

The initial version of the code only has a check for the escape key after the camera is initialized. If the escape key is pressed, the system will exit. Otherwise, it will keep retrieving the distance at a particular pixel of the distance frame and display it onto the colour frame which contains the real-time image. Taking advantage of this starting point, the team designed a flow chart that allows the system to count the number of repetitions done per set and the number of sets in total. The flow chart is illustrated in Figure 5.2.2.

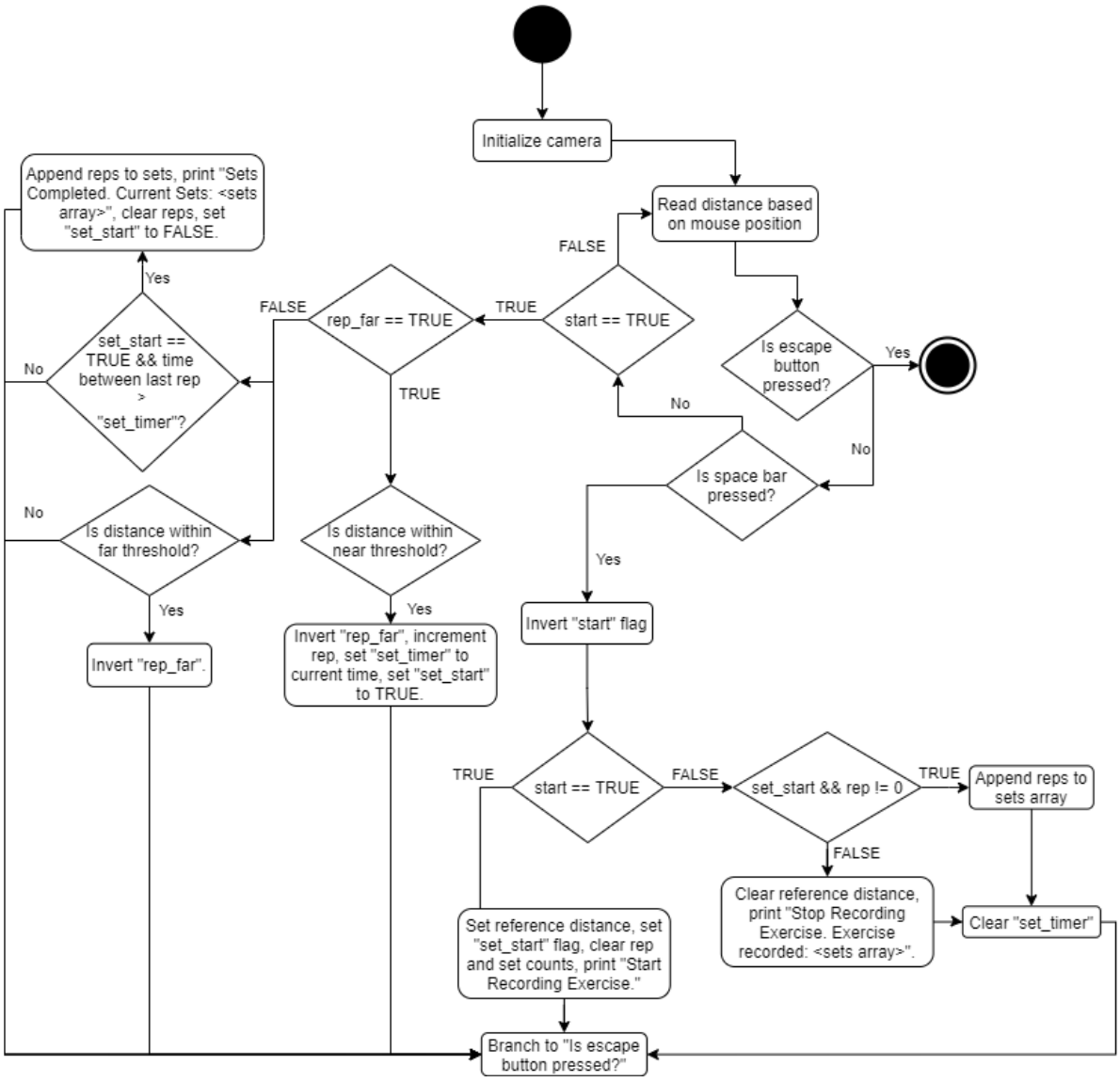


Figure 5.2.2: Repetition and Sets Counting Flowchart

To start recording the exercise, the user needs to press a button to allow the system to know to take reference of the current distance to the weights. For the current prototype system, the spacebar key is used to signal when the recording is supposed to start and stop. In the final implementation, this button will signal when the user wishes to start tracking his exercise and stop tracking his exercise upon completion of all sets.

As the current system is unable to use computer vision to identify the position of the weights within the image frame, the project team opted to maintain the utilization of the mouse to adjust the distance measuring position and adjust the distance reading position to the weights. In the final implementation, the approximate centre of the weights will be used as the position to measure the distance.

To maintain the accuracy of the reference position, the system is programmed to stop moving the distance reading position once the exercise has begun. This is to prevent users from accidentally shifting the distance reading position as this may affect the relative displacement of the weights.

Once the spacebar key is pressed for the first time, the "start" flag will be inverted to be true. This means that the exercise recording has commenced. This will clear existing repetition and set variables to track the new series of exercises. At this instance, the system will also take note of the current distance of the weight from the camera. This will act as its reference home position. As long as the displacement of the weights exceeds a predefined distance away from its reference position, the flag "rep_far" will toggle to be true, signalling that the weights are currently far enough away from the reference position to count as half a repetition. The repetition will be counted as completed when it comes within a nearer threshold. For the initial development, the thresholds were defined to be 30cm for the far threshold and 10cm for the near threshold.

When the weights complete one full repetition, the system will take note of the time that the repetition was completed in monotonic time. The system will compare the time that the last repetition was completed against the current monotonic time. If it exceeds a predefined threshold, the system will take it as having completed a set and append the number of repetitions to the sets array before clearing the repetitions counter. The current time threshold is set to five seconds for testing purposes, but this value can be adjusted depending on the user's preference.

Upon completion, the user can end the recording of the exercise by pressing the spacebar again. If any repetition was done and the timing threshold to append the value to the set has yet to be reached, the system will automatically append the repetition counts to the sets array and display the array. In the final implementation of this project, the sets array will be exported via a .json file for further analysis in subsequent project developments.

5.3 Image Recognition

To train the neural network of the system, a large variety of weight plate images are essential. These images comprised of either the same or different weight plates with varying backgrounds. Once these images are collected, the aspect ratio of the images are to be set to 3:4, and the size of the images should be 640 x 480 pixels. Since it is a repetitive process, a script was written to automate the resizing process for over 200 images.

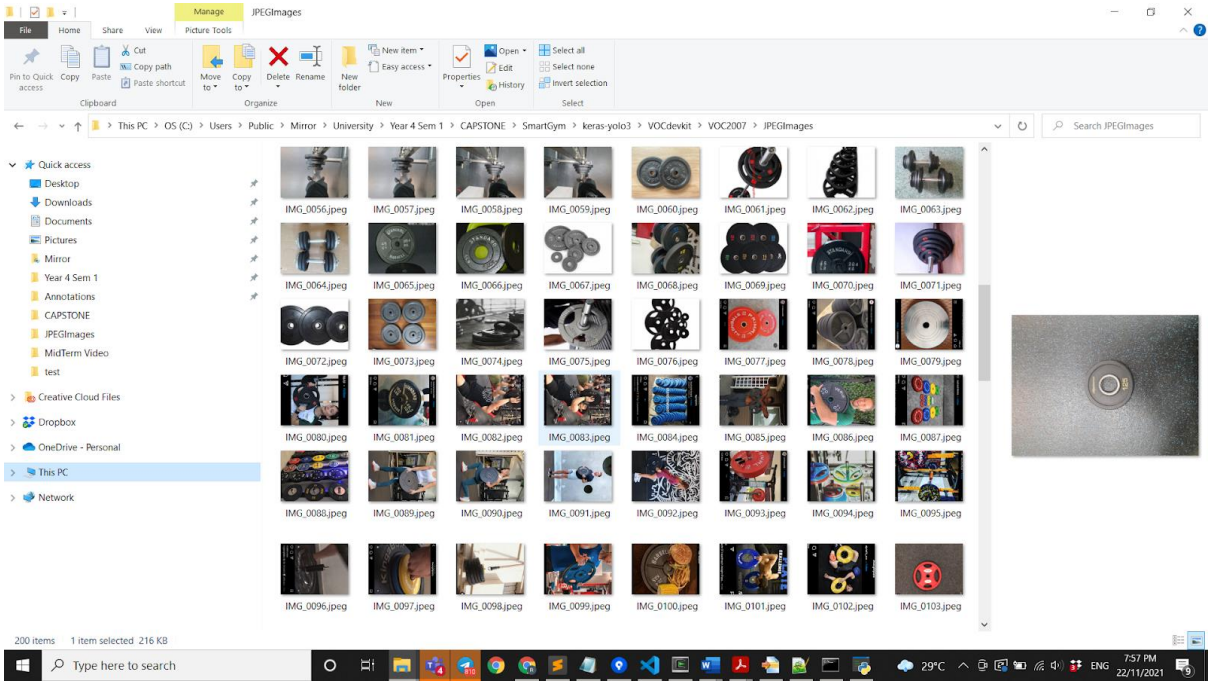


Figure 5.3.1: Images Collected for Training the Neural Network

Apart from resizing the images, it is recommended to rename all the files into a specific naming convention for easy identification. To do so, the team created a python script to rename all the files inside a particular folder to the following name convention: “IMG_XXXX.jpeg”, where XXXX is its indexed position within the folder. The snapshot of the script is shown in Figure 5.3.2.

```
import os
path = 'test'
files = os.listdir(path)

for index, file in enumerate(files):
    os.rename(os.path.join(path, file), os.path.join(path, "".join(['IMG_',str(index+200).zfill(4),'.jpeg'])))
```

Figure 5.3.2: Python Script for Renaming of Image Files

After the images have been processed, the images were labelled by running the open-sourced tool name labelling retrieved from GitHub at https://github.com/tzutalin/labelImg. The first step is to adjust the predefined_classes.txt under the data folder of labelling main directory. The predefined_classes.txt is supposed to contain the list of classes that needs to be identified, which in this case only contains weights. Boxes had to be drawn around the object that is to be labelled, which in this case is the weight

plates, as shown in Figure 5.3.3 below. As a result, it generates .txt files containing the notations that show the desired object to learn within the image of the same file name.

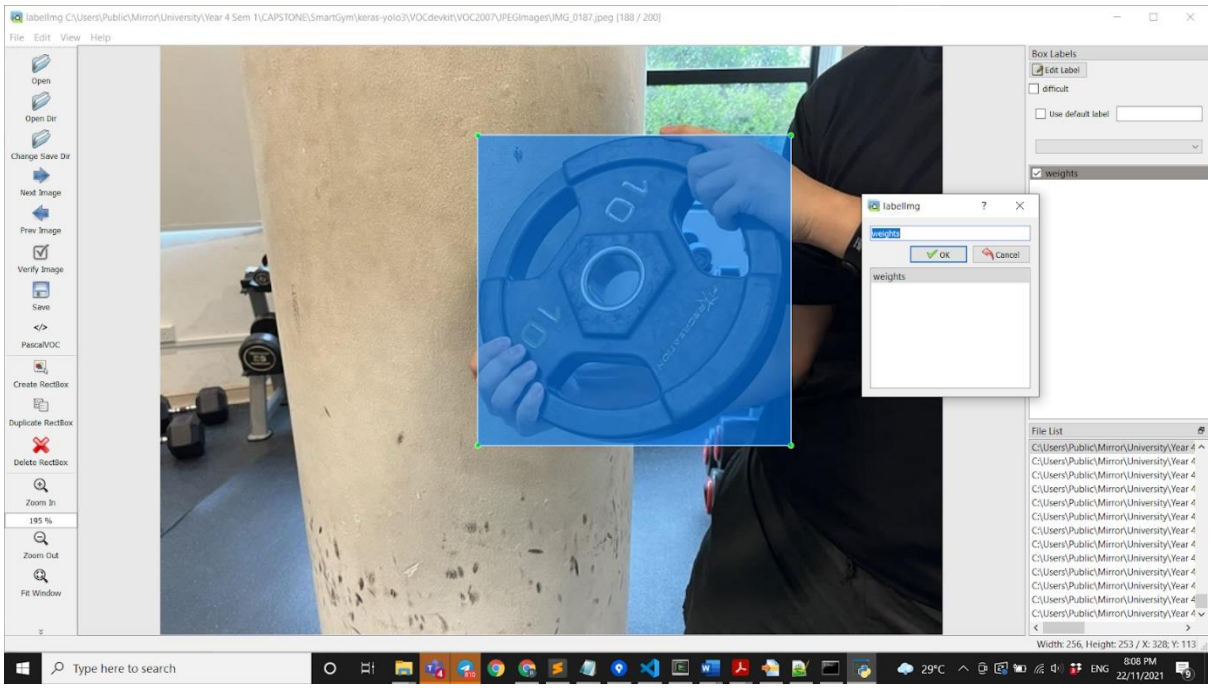


Figure 5.3.3: A Box Drawn Over the Image of the Object (Weight Plate) for Labelling

Once the notations are generated, the system needs to be able to link the images to the notations for the sake of training, testing, and validation of the computer vision system. The system that the project team chose to use was Keras You Only Look Once (YOLO) version 3 to train the computer vision system followed by YOLO Tiny to be run on the board. YOLO Tiny is a faster version of YOLOv3 with approximately 442% increased efficiency.

With all the images annotated, the next step is to create a text file containing the list of images to use to train the images, test and validate the trained system. This was done by creating a python script to compile the names of all the images within a folder. The python script is shown in Figure 5.3.4.

```
1   import os
2
3   a = open("test.txt", "w")
4   for path, subdirs, files in os.walk(r'data\obj\test'):
5       for filename in files:
6           f = "data/obj/"+filename+"\n"
7           a.write(str(f))
```

Figure 5.3.4: Python Script to Compile the Names of All the Images Within a Folder

With the text files created, training of the custom dataset can begin. For this process, the project team used Google Colab to train the custom dataset. The instructions were created by PySource from https://pysource.com/2020/04/02/train-yolo-to-detect-a-custom-object-online-with-free-gpu/.

The project team archived the images and the annotations into a single .zip archive named images.zip and uploaded the archive onto Google Drive. From there, the team created a new Google Colab notebook created by PySource and linked the notebook to the Google Drive account with the images archive by running the second line of code from the notebook. The whole notebook was then executed to begin the training process.

The notebook first clones the Darknet version of YOLOv3 from GitHub before compiling it using the allocated NVIDIA GPU. The notebook then configures the Darknet network for training based on YOLOv3 by setting the number of classes to train to one and sets the number of filters to 18. These are parameters that would be used when training the new dataset. The next instruction will be to create the name of the object to detect, in this case weights. It also creates the data file of the object which includes the number of classes to train, the file containing the name of the files to train, test and validate the trained system, the file that contains the object names, and the backup location of the output files. The notebook then unzips the images and starts the training process.

Once the training has completed or the process has reached a sufficiently low loss rate, the latest weight file will be stored in the link Google Drive account under the YOLOv3 folder. After downloading the latest folder, the file can be downloaded and put into the same folder as the file yolo_object_detection.py that is part of the PySource project documents. The next few changes are to change a few lines in yolo_object_detection.py. The first change is line 11 to change the name of the class to weights. The second change is to change line 14 to link images_path to the directory that contains several test or validation images. If there is an error saying “IndexError: invalid index to scalar variable” when trying to execute the python file, change line 19 to “output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]” from “output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]”. With all the changes made to yolo_object_detection.py, the next step is to run it and test that the custom dataset has been trained sufficiently to detect weights. A sample image of the output from running the file is shown in Figure 5.3.5, on the following page.

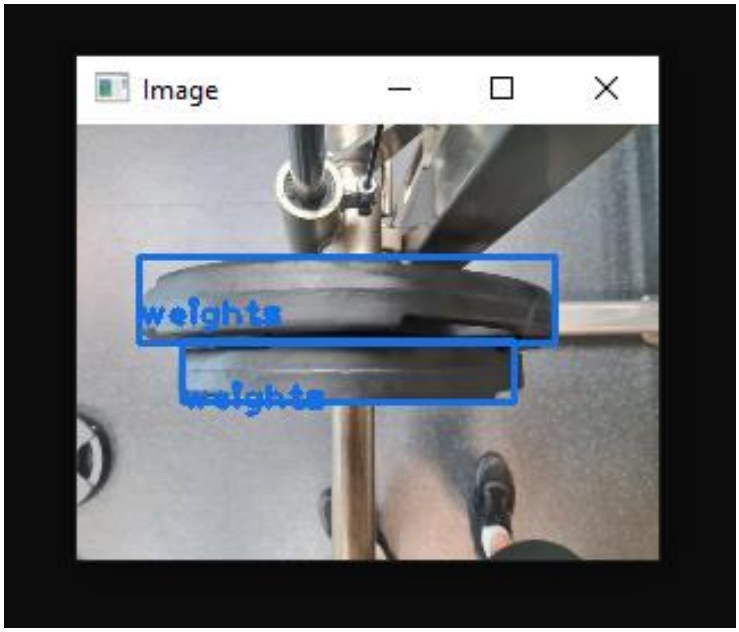


Figure 5.3.5: Initial Sample Image of Object Detection in Action

The next thing to verify is to test it with the Darknet YOLOv3. To test it with the live webcam feed, the project team used the original Darknet YOLOv3 files and made some modifications. The first file to change is the yolov3.cfg. To ensure successful identification of the different files, a copy was made and named yolov3-obj.cfg.

The first change to be made is to change line 20's max_batches to 2000 and line 22 steps to be 1600,1800. The next change is to change all three occurrences where classes = 80 to classes = 1 and all three occurrences where filters = 255 to filters = 18.0. The next step is to make two new files, obj.names and obj.data to be stored in the directory build\darknet\x64\data\ from the main darknet folder. The file obj.names will contain the name of the class trained, namely weights. The file obj.data is supposed to contain the data as shown in Figure 5.3.6.

```
1  classes = 1
2  train  = data/train.txt
3  valid  = data/test.txt
4  names = data/obj.names
5  backup = backup/
```

Figure 5.3.6: Files for Darknet

The files train.txt and test.txt are the same as generated previously and are to be shifted to build\darknet\x64\data\. The image files and their annotations are all to be shifted to build\darknet\x64\data\obj\. To test the program on Windows, the following command was executed "darknet.exe detector demo data/obj.data yolov3-obj.cfg yolo-obj.weights -c 0". This is running the

darknet YOLOv3 on detector demonstration mode using the obj.data file with YOLOv3 using the custom dataset in yolo-obj.weights from the first webcam device. By running the above command, the project team then tested the weights identification using several images that were captured. A sample of the output is shown in Figure 5.3.7.

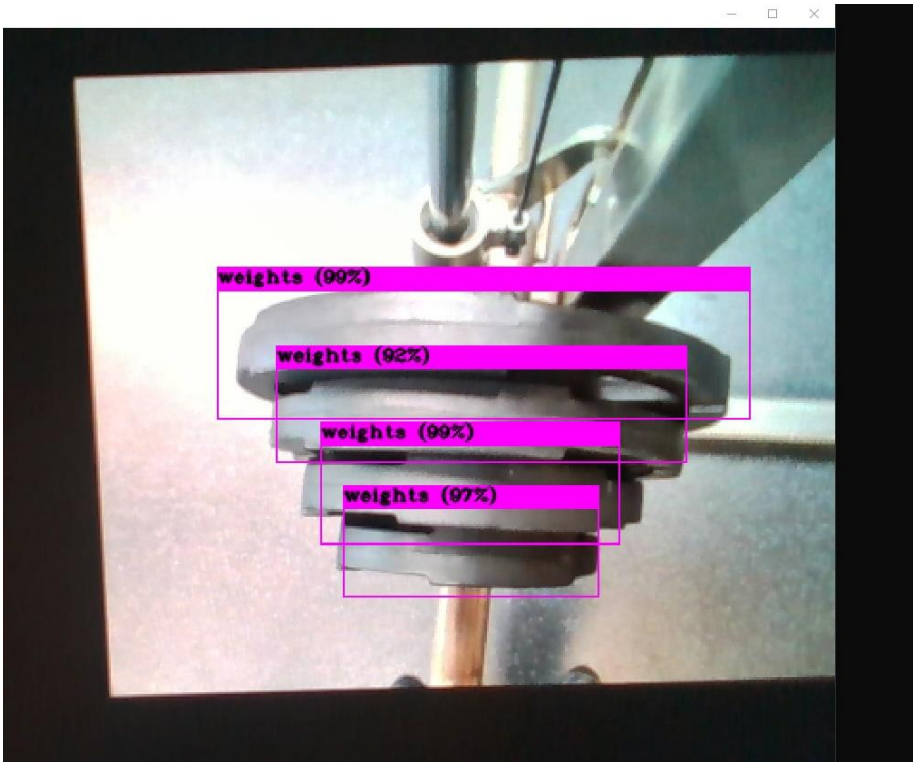


Figure 5.3.7: Sample Image of Object Detection in Action with Accuracy Output in Percentage

After successfully identifying the weight plates, the next step is to identify the mass of said weight plates.

5.4 Weight Identification

To identify the weights, the project team would need to combine both the object detection from YOLO and the depth reading from the RealSense camera. The object detection would be able to give two dimensions of the object, in particular the x-axis and the y-axis, while the depth sensor would provide the third dimension, namely the z-axis, required to accurately size the object.

The main problem is to identify how many millimetres (mm) is represented within one pixel of the image frame. If the weight is near the camera, the weight will occupy a larger number of pixels of the image frame than when the weight is further away from the camera. Therefore, the depth camera will help to allow the system to identify how far the object is. To do so, the system would first have to be calibrated by identifying the number of pixels a certain weight occupies at a certain distance. The weight would then be moved to a different distance and the number of pixels would then be recorded. With this data, it is possible to identify the change in the number of mm per pixel (mm/pixel) according to

the change in the distance. This is done by plotting a graph and finding the best fit line, followed by finding the equation of the best fit line. The data gathered is shown in Figure 5.4.1 when the camera is set to the resolution of 640 pixels by 480 pixels.

Size of weight (mm)	left	right	Depth (mm)	Pixels	mm/pixel
330	44	468	488	424	0.778302
	65	430	565	365	0.90411
	82	405	640	323	1.021672
	95	375	740	280	1.178571
	100	346	860	246	1.341463
	128	340	1000	212	1.556604
	220	404	1160	184	1.793478
	324	450	1680	126	2.619048

Figure 5.4.1: Tested mm/pixel Values at various Depths for 640 by 480 Pixels Resolution

With the data shown in Figure X, it is possible to plot a graph of mm/pixel against the depth to observe the change in the mm/pixel as the depth changes. The graph that has the best fit line and the equation of the line is shown in Figure 5.4.2

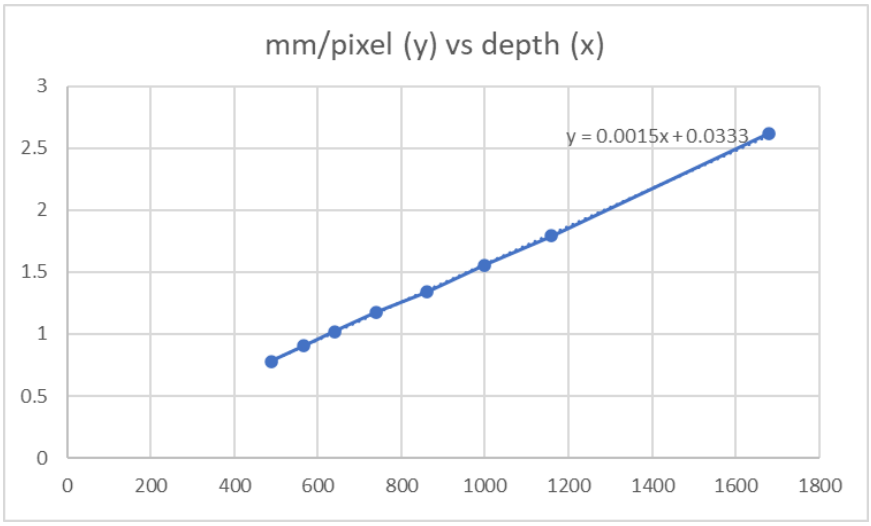


Figure 5.4.2: mm/pixel Graph

From Figure 5.4.2, it can be observed that the equation of the mm per pixel against depth is y = 0.0015x + 0.0333, where y is the mm/pixel and x is the depth. With the mm/pixel, the actual size of the weights detected can be found simply by multiplying the mm/pixel by the number of pixels the weight occupies.

For this project, the project team was unable to integrate the depth sensor together with the object detection program due to a lack of time. As such, the project team developed the weight identification as part of the Darknet YOLOv3 project and defined the weight that is at the edge of the barbell to be a 2.5kg weight with a diameter of 19cm to act as a point of reference to identify the mm/pixel despite the lack of the depth sensor.

With the method of identifying the actual size of the weights done, the next step is to design the software to implement this calculation. Figure 5.4.3 depicts the flowchart of the software for weight identification.

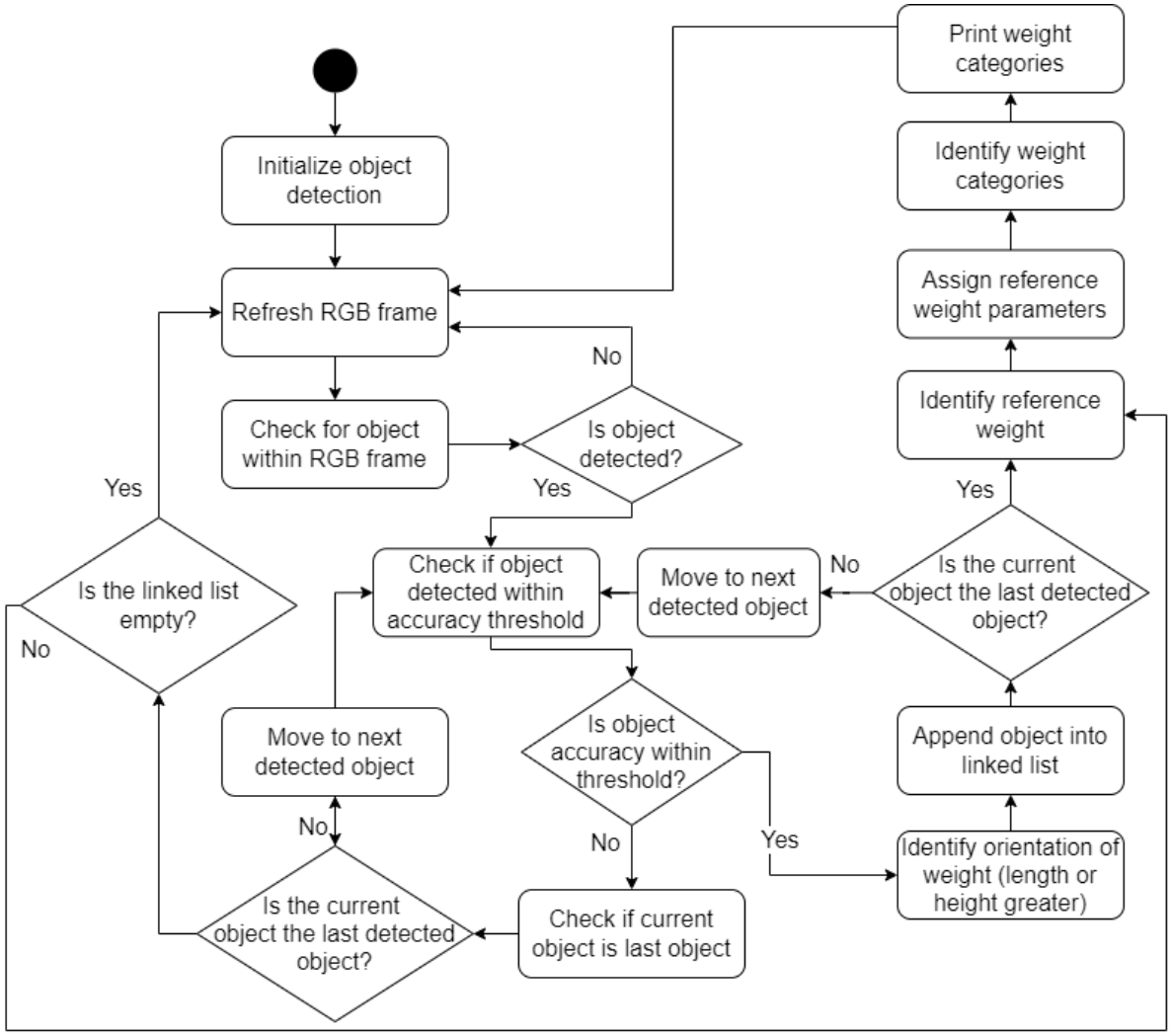


Figure 5.4.3: Weight Identification Software Flowchart

The software first initializes the object detection program. Once the program is running, the program will constantly refresh the Red, Green, Blue (RGB) frame that contains the image captured by the camera. The software will then check whether there is an object in the RGB frame. If no object is detected, the program will keep refreshing the RGB frame. Once an object is detected, the software will check whether the likelihood that the object detected is actually a weight plate is above a threshold. If it does not meet the threshold, the program will check whether this is the last object that was detected in this frame. If it is, the system will loop back to the refresh RGB frame step. However, if this object is not the last object that was detected, it loops back to checking the threshold of the accuracy of the next object.

If the object detected is above the accuracy threshold, the system will then identify the orientation of the weight by checking whether its length or height is greater. This is to determine which of the dimensions is the diameter of the plate. This is because all the weight plates are identified based on their diameters. Once the orientation of the plate is identified, the number of pixels representing the diameter is stored in the struct containing all the information pertaining to the weights. Other information, such as the borders of the weight, the pixel length and height, category of weight (referring to the type of weight plate), and the actual size of the plate, are stored in the struct as well.

Once the struct has been created, it is then appended to the linked list containing all the information required to identify the weights. If the current object is not the last object detected, it will loop back to check whether the next object is within the accuracy threshold. Otherwise, the program will move on to identify the reference weight. The program will just cycle through the linked list and identify the top most weight and note it as the reference weight. The flow chart for this sequence is described in Figure 5.4.4.

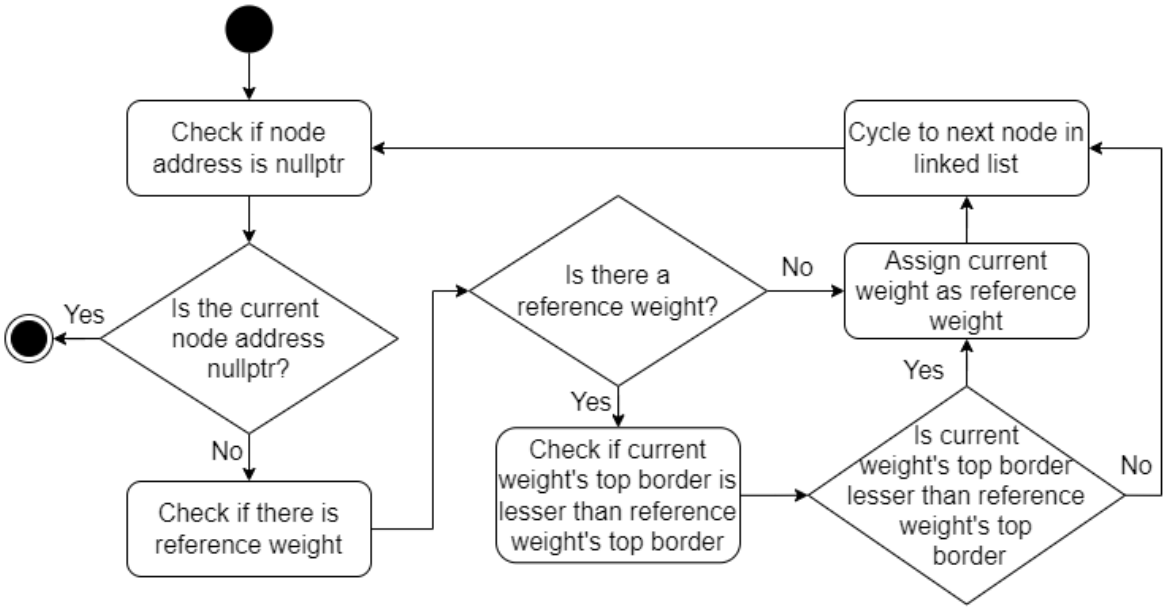


Figure 5.4.4: Reference Weight Identification Flowchart

Once the reference weight is identified, it is assigned the index of category 2.5kg weight plate and the actual size of 19 cm. This will then be used to identify the other weights by calculating the actual size of each individual plate and comparing them against the data set of the diameters of the weight plates.

The program will then assign the closest weight category's index to the object. The weight categories are as follows:

1. 1.25kg (16cm)
2. 2.5kg (19cm)
3. 5kg (23cm)
4. 10kg (30cm)
5. 15kg (36.5cm)
6. 20kg (43cm)

Once the weights have been identified, the software will then print the weight of the plates in the RGB frame.

To implement the above, two files were altered: demo.c and image_opencv.cpp. The file demo.c is executed when using the webcam to display real time object detection, while the image_opencv.cpp is used to detect the objects and identify the boundaries of the object. The changes made to demo.c are shown in the following figures.

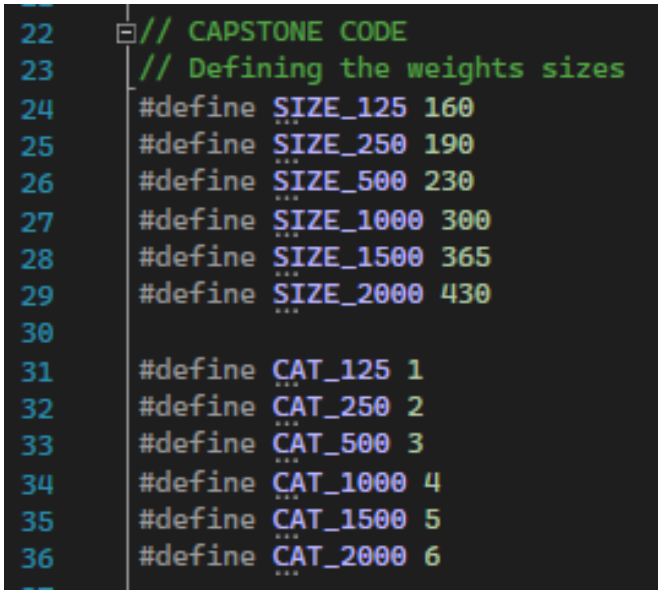


Figure 5.4.5: Weight Defined Values in Demo.c File

In Figure 5.4.5, the sizes of the weights in mm were defined and the index of the categories were declared.

```
//CAPSTONE CODE

struct Weight_Plate {
    // index is the position in the detection linked list
    // weight_cat = type of weight plate.
    // 0 = unknown, 1 = 1.25, 2 = 2.5, 3 = 5, 4 = 10, 5 = 15, 6 = 20 kg.
    // left/right/top/bot are pixel borders of weight.
    // Pixels is the pixel length of greater value.
    // larger_value = 1 means length is greater, 2 means height is greater, 0 means undefined.
    // length/height is pixel length/height of weight.
    int index, weight_cat, left, right, top, bot, pixels, larger_value, length, height;
    // size = actual size of the plate (calculated based on image)
    float size;
};
```

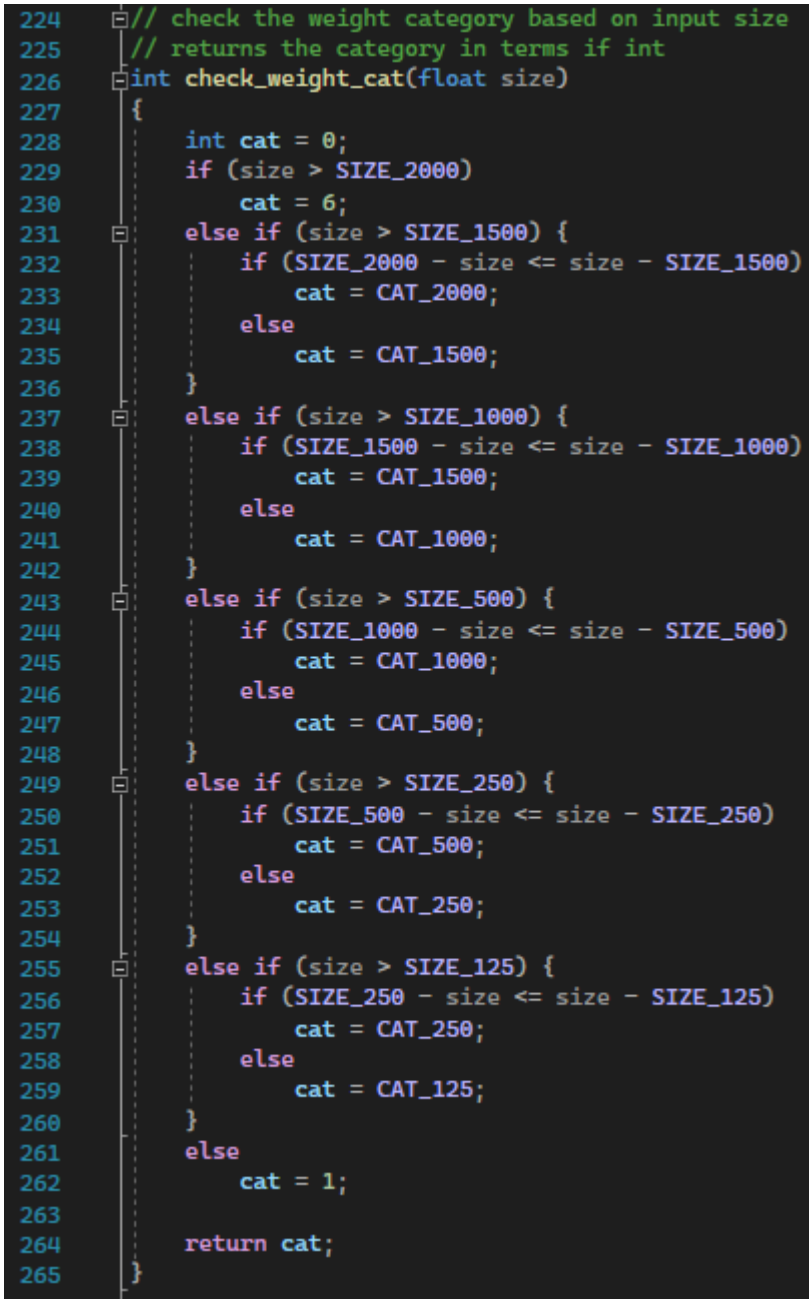
Figure 5.4.6: Demo.c File struct for Weight Plates

In Figure 5.4.6, the struct of the weight plate is declared. It contains the index in the detected objects' linked list, the weight category, the borders of the object, the pixel length of the greater length, a flag named "larger_value" to determine whether the length or height is greater, the pixel length and pixel height. These variables are stored as integers. There is another float variable called "size" containing the actual size of the weight. This is done by multiplying the pixels by the mm/pixel.

```
// linked list code
struct Node {
    struct Weight_Plate weight_plate;
    struct Node* next;
};

void append(struct Node** head_ref, struct Weight_Plate new_data)
{
    /* 1. allocate node */
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

    struct Node* last = *head_ref;  /* used in step 5*/

    /* 2. put in the data  */
    new_node->weight_plate = new_data;

    /* 3. This new node is going to be the last node, so make next of
          it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new node as head */
    if (*head_ref == NULL)
    {
        *head_ref = new_node;
        return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
    last->next = new_node;
    return;
}

void printList(struct Node* node)
{
    // cycling through linked list
    while (node != NULL)
    {
        // printing all values of the struct Weight_Plate
        printf("Print Weights: Cat = %d, Size = %f, Left = %d, Top=%d, Right=%d, Bottom=%d, Pixels=%d, Larger=%d, Length=%d, Height=%d\n",
            node->weight_plate.weight_cat, node->weight_plate.size, node->weight_plate.left, node->weight_plate.top,
            node->weight_plate.right, node->weight_plate.bot, node->weight_plate.pixels, node->weight_plate.larger_value,
            node->weight_plate.length, node->weight_plate.height);
        node = node->next;
    }
}
```

Figure 5.4.7: Demo.c Linked List Functions

In Figure 5.4.7, the code required to create a linked list is created. The code to print all the contents of the linked list is also included.



```
224  // check the weight category based on input size
225  // returns the category in terms if int
226  int check_weight_cat(float size)
227  {
228      int cat = 0;
229      if (size > SIZE_2000)
230          cat = 6;
231      else if (size > SIZE_1500) {
232          if (SIZE_2000 - size <= size - SIZE_1500)
233              cat = CAT_2000;
234          else
235              cat = CAT_1500;
236      }
237      else if (size > SIZE_1000) {
238          if (SIZE_1500 - size <= size - SIZE_1000)
239              cat = CAT_1500;
240          else
241              cat = CAT_1000;
242      }
243      else if (size > SIZE_500) {
244          if (SIZE_1000 - size <= size - SIZE_500)
245              cat = CAT_1000;
246          else
247              cat = CAT_500;
248      }
249      else if (size > SIZE_250) {
250          if (SIZE_500 - size <= size - SIZE_250)
251              cat = CAT_500;
252          else
253              cat = CAT_250;
254      }
255      else if (size > SIZE_125) {
256          if (SIZE_250 - size <= size - SIZE_125)
257              cat = CAT_250;
258          else
259              cat = CAT_125;
260      }
261      else
262          cat = 1;
263
264      return cat;
265  }
```

Figure 5.4.8: Demo.c Function to Identify Weight Category

In Figure 5.4.8, the code to identify the weight category once its actual size has been computed is shown.

```
void print_weight_size(mat_cv* mat, detection* dets, int num, float thresh) {
    mat_cv* show_img = (mat_cv*)mat;
    int i, j = 0;
    struct Node* head = NULL;
    float mm_per_pixel = 0;
    // looping through all the detected objects
    for (i = 0; i < num; ++i) {
        // checking if the probability the object detected actually matches the trained object is above the threshold
        if (dets[i].prob[0] > thresh) {
            // creating the new struct for weight plate with the appropriate values
            struct Weight_Plate New_Weight;
            New_Weight.index = i;
            New_Weight.weight_cat = 0;
            New_Weight.size = 0;
            New_Weight.left = get_left(mat, dets, i);
            New_Weight.right = get_right(mat, dets, i);
            New_Weight.top = get_top(mat, dets, i);
            New_Weight.bot = get_bot(mat, dets, i);
            New_Weight.length = New_Weight.right - New_Weight.left;
            New_Weight.height = New_Weight.bot - New_Weight.top;
            // determining which of the length or height is greater
            if (New_Weight.length >= New_Weight.height) {
                New_Weight.pixels = New_Weight.length;
                New_Weight.larger_value = 1;
            }
            else {
                New_Weight.pixels = New_Weight.height;
                New_Weight.larger_value = 2;
            }
            // append to linked list
            append(&head, New_Weight);
        }
    }
    // checking the created linked list
    // printf("\n Created Linked list is:\n");
    // printList(head);
    struct Node* current = head;
    struct Node* ref = head;

    // Finding the reference plate (furthest right or down)
    while (current != NULL) {
        // determining the orientation of the plates
        // if larger_value == 1, length is greater and weights are horizontal, hence check the bottom most plate for its reference size
        // if larger_value == 2, height is greater and weights are vertical, hence check right most plate for its reference size
        if (current->weight_plate.larger_value == 1) {
            // check if current iteration of linked list is further up (ref plate is furthest up)
            if (current->weight_plate.bot < ref->weight_plate.bot) {
                ref = current;
            }
        }
        else {
            // check if current iteration of linked list is further left (ref plate is furthest left)
            if (current->weight_plate.right < ref->weight_plate.right) {
                ref = current;
            }
        }
        current = current->next;
    }
```

Figure 5.4.9: Demo.c of Weight Identification Function (Part 1)

In Figure 5.4.9, the first section of the main function (print_weight_size) is defined. The inputs are the OpenCV matrix containing data of the image frame, the first node of the linked list containing all the detected objects, the number of objects detected, and the accuracy threshold of the objects.

The program then cycles through the objects detected that meet the accuracy threshold and extracts the information required by the struct. Once the struct is created, it is appended into the linked list of the weights. The program then cycles through all the detected weight plates and identifies which of the weight plates is the top most or left most plate (depending on the orientation of the weights). and defines it as the reference weight.

```
    // Calculating the number of mm each pixel represents
    if (head != NULL) {
        mm_per_pixel = (float)SIZE_250 / (float)ref->weight_plate.pixels;
        ref->weight_plate.size = SIZE_250;
        ref->weight_plate.weight_cat = 2;
        printf("mm per pixel: %f\n", mm_per_pixel);
        printf("Reference Weight: Cat = %d, Size = %f, Left = %d, Top=%d, Right=%d, Bottom=%d, Pixels=%d, Larger=%d, Length=%d, Height=%d\n",
            ref->weight_plate.weight_cat, ref->weight_plate.size, ref->weight_plate.left, ref->weight_plate.top,
            ref->weight_plate.right, ref->weight_plate.bot, ref->weight_plate.pixels, ref->weight_plate.larger_value,
            ref->weight_plate.length, ref->weight_plate.height);
    }

    // Calculates the sizes of each weight plate and the category of the weight plate
    current = head;
    while (current != NULL) {
        current->weight_plate.size = current->weight_plate.pixels * mm_per_pixel;
        //printf("Size of Plate: %f\n", current->weight_plate.pixels * mm_per_pixel);
        current->weight_plate.weight_cat = check_weight_cat(current->weight_plate.size);
        //printf("Cat of Plate: %d\n", check_weight_cat(current->weight_plate.size));
        current = current->next;
    }
    // checking the created linked list
    printf("\nCalculated Linked list is:\n");
    printList(head);
    // Draw the labels on the box
    current = head;
    while (current != NULL) {
        draw_box_name(mat, current);
        current = current->next;
    }
}

// END OF CAPSTONE CODE
```

Figure 5.4.10: Demo.c of Weight Identification Function (Part 2)

In Figure 5.4.10, the program then calculates the mm/pixel based on the reference weight. With the calculated mm/pixel, the program then calculates the size of each weight and assigns the identified category to the object. The program then prints all the data in the linked list and adds the label containing the identified weight to the image frame by calling the function “draw_box_name”.

```
            if (!benchmark && !dontdraw_bbox) {
                draw_detections_cv_v3(show_img, local_dets, local_nboxes, demo_thresh, demo_names, demo_alphabet, demo_classes, demo_ext_output);
                print_weight_size(show_img, local_dets, local_nboxes, demo_thresh);
            }
            free_detections(local_dets, local_nboxes);
```

Figure 5.4.11: Demo.c Calling of Weight Identification Function

The last change to demo.c is shown in Figure 5.4.11, where the function “print_weight_size” is called after the function “draw_detections_cv_v3” is called. The function “draw_detections_cv_v3” is to draw the bounding boxes of the object that was detected. The changes made to image_opencv.cpp will be shown in the following images.

```
// CAPSTONE CODE
// Defining the weight categories
#define CAT_125 1
#define CAT_250 2
#define CAT_500 3
#define CAT_1000 4
#define CAT_1500 5
#define CAT_2000 6
// END OF CAPSTONE CODE
```

Figure 5.4.12: “image_opencv.cpp” Weight Category Definitions

In Figure 5.4.12, the index of the weight categories are assigned.

```
903  //CAPSTONE CODE
904
905  extern "C" int get_left(mat_cv* mat, detection * dets, int i) {
906      cv::Mat* show_img = (cv::Mat*)mat;
907      box b = dets[i].bbox;
908      if (isnan(b.w) || isinf(b.w)) b.w = 0.5;
909      if (isnan(b.x) || isinf(b.x)) b.x = 0.5;
910      b.w = (b.w < 1) ? b.w : 1;
911      b.x = (b.x < 1) ? b.x : 1;
912      int left = (b.x - b.w / 2.) * show_img->cols;
913      return left;
914  }
915
916  extern "C" int get_right(mat_cv* mat, detection * dets, int i) {
917      cv::Mat* show_img = (cv::Mat*)mat;
918      box b = dets[i].bbox;
919      if (isnan(b.w) || isinf(b.w)) b.w = 0.5;
920      if (isnan(b.x) || isinf(b.x)) b.x = 0.5;
921      b.w = (b.w < 1) ? b.w : 1;
922      b.x = (b.x < 1) ? b.x : 1;
923      int right = (b.x + b.w / 2.) * show_img->cols;
924      return right;
925  }
926
927  extern "C" int get_top(mat_cv* mat, detection * dets, int i) {
928      cv::Mat* show_img = (cv::Mat*)mat;
929      box b = dets[i].bbox;
930      if (isnan(b.h) || isinf(b.h)) b.h = 0.5;
931      if (isnan(b.y) || isinf(b.y)) b.y = 0.5;
932      b.h = (b.h < 1) ? b.h : 1;
933      b.y = (b.y < 1) ? b.y : 1;
934      int top = (b.y - b.h / 2.) * show_img->rows;
935      return top;
936  }
937
938  extern "C" int get_bot(mat_cv* mat, detection * dets, int i) {
939      cv::Mat* show_img = (cv::Mat*)mat;
940      box b = dets[i].bbox;
941      if (isnan(b.h) || isinf(b.h)) b.h = 0.5;
942      if (isnan(b.y) || isinf(b.y)) b.y = 0.5;
943      b.h = (b.h < 1) ? b.h : 1;
944      b.y = (b.y < 1) ? b.y : 1;
945      int bot = (b.y + b.h / 2.) * show_img->rows;
946      return bot;
947  }
```

Figure 5.4.13: “image_opencv.cpp” Border Calculation Functions

In Figure 5.4.13, the functions to get the left, right, top, and bottom borders of the object are declared. The actual implementations are extracted from the function “draw_detections_cv_v3”.

```
949  struct Weight_Plate {
950      // index is the position in the detection linked list
951      // weight_cat = type of weight plate.
952      // 0 = unknown, 1 = 1.25, 2 = 2.5, 3 = 5, 4 = 10, 5 = 15, 6 = 20 kg.
953      // left/right/top/bot are pixel borders of weight.
954      // Pixels is the pixel length of greater value.
955      // larger_value = 1 means length is greater, 2 means height is greater, 0 means undefined.
956      // length/height is pixel length/height of weight.
957      int index, weight_cat, left, right, top, bot, pixels, larger_value, length, height;
958      // size = actual size of the plate (calculated based on image)
959      float size;
960  };
961
962  struct Node {
963      struct Weight_Plate weight_plate;
964      struct Node* next;
965  };
```

Figure 5.4.14: “image_opencv.cpp” Weight Plate struct and Linked List Node Definition

In Figure 5.4.14, the struct “Weight_Plate” and “Node” are defined.

```
967  extern "C" void draw_box_name(mat_cv* mat, Node* node)
968  {
969      cv::Mat* show_img = (cv::Mat*)mat;
970      cv::Point pt_text;
971      std::string labelstr;
972      float const font_size = show_img->rows / 1000.F;
973      cv::Scalar black_color = CV_RGB(0, 0, 0);
974      switch (node->weight_plate.weight_cat) {
975      case CAT_125:
976          labelstr = "1.25kg";
977          break;
978      case CAT_250:
979          labelstr = "2.5kg";
980          break;
981      case CAT_500:
982          labelstr = "5kg";
983          break;
984      case CAT_1000:
985          labelstr = "10kg";
986          break;
987      case CAT_1500:
988          labelstr = "15kg";
989          break;
990      case CAT_2000:
991          labelstr = "20kg";
992          break;
993      default:
994          labelstr = "Unknown";
995          break;
996      }
997      pt_text.x = node->weight_plate.left;
998      pt_text.y = node->weight_plate.top - 4;// 12;
999      cv::putText(*show_img, labelstr, pt_text, cv::FONT_HERSHEY_COMPLEX_SMALL, font_size, black_color, 2 * font_size, CV_AA);
1000 }
1001
1002 //END OF CAPSTONE CODE
```

Figure 5.4.15: “image_opencv.cpp” Function to Draw Bounding Box Labels onto Colour Frame

In Figure 5.4.15, the function “draw_box_name” was defined. This function displays the weight of the plate according to the index of the weight category onto the RGB frame. The actual implementation to identify the location to draw the text is extracted from “draw_detections_cv_v3”.

```
1138
1139                  cv::rectangle(*show_img, pt_text_bg1, pt_text_bg2, color, width, 8, 0);
1140                  cv::rectangle(*show_img, pt_text_bg1, pt_text_bg2, color, CV_FILLED, 8, 0);    // filled
1141                  cv::Scalar black_color = CV_RGB(0, 0, 0);
1142                  //cv::putText(*show_img, labelstr, pt_text, cv::FONT_HERSHEY_COMPLEX_SMALL, font_size, black_color, 2 * font_size, CV_AA);
1143                  // cv::FONT_HERSHEY_COMPLEX_SMALL, cv::FONT_HERSHEY_SIMPLEX
1144              }
```

Figure 5.4.16: “image_opencv.cpp” with Removed Bounding Box Labels

The final change to image_opencv.cpp is shown in Figure 5.4.16, where line 1142 was commented. This is to remove the label that contained the type of object that was detected.

6. Project Findings

This section will discuss the project's findings, lessons learned, and possible improvements.

Training of Image Recognition using Google Collab

Training of image recognition is more compatible with Linux-based operating systems (OS) as compared to Windows OS. The project team considered the alternative to run the code on the Google Collab notebook to train the YOLOv3 system to identify the weight plates. Given the exact same CUDA, CUDNN, and Python versions, the code was able to be executed flawlessly on the Linux-based Google Collab, as compared to the Windows OS.

Angle Placement of Camera

The project team conducted considerable testing to determine which angle positioning of the camera will still allow it to perform object detection before it stops detecting/recognizing objects. The angle was measured to be between 18 to 20 degrees.

Orientation of Camera

The YOLO object detection system was only able to detect the weights when they are horizontally relative to the camera's point of view. As the majority of the images utilized in the training dataset comprised of weights stacked in the horizontal position, the project team suspects this to have caused the training algorithm to only detect weights stacked horizontally.

Utilisation of RGB Camera using a Point of Reference vs Utilising Depth Sensor for Weight Plate Identification

There were two ways of implementing the weight plate identification. Table 6.1 shows the pros and cons of each possible solution.

Table 6.1 – Pros and Cons for RGB vs Depth Camera

	Utilisation of RGB Camera using a Point of Reference	**Using Depth Sensor**
Pros	• Easier to implement • With reference, weight plate detection is simpler	• More dynamic • Able to have second layer of verification to identify weights
Cons	• Not dynamic • Purely reliant on image recognition • Needs to have a reference recognized by the YOLO system	• Need to interface the depth sensor into the YOLO system.

Limitations from Distance between Camera and Weight Plate

When the distance between the camera and the weight plate(s) is too near or too far, the system will either not recognise the weight plate, or it will tag the wrong label for the weight plate. Possible causes are likely due to insufficient training data, or a lack of duration to run the training algorithm.

Using Newer Version of YOLO

The currently used YOLOv3 was only able to run the object detection on the Jetson Nano at about 2.3 frames per second. A possible way to improve the system is to utilize a newer version of YOLO (either v4 or v5), as the object detection function would perform better with the newer version of YOLO.

Camera Mount Reliability vs Flexibility

The team originally envisioned the camera mount as a system that would focus on flexibility in terms of its capacity to pan, extend, and tilt with a suitable amount of force. One of the inspirations is flexible joint camera mounts, such as the one shown in Figure 6.1 below.

Figure 6.1: Flexible Joint Camera Mount

However, when the team refined the design for the camera mount, it became evident that there was a flaw in the concept owing to several elements that were not anticipated prior to physical testing. When using the smith machine, it is fairly usual for the weighted barbell to bang into the base or one of the levels along the guide rails, causing vibration to resonate through the machine's construction. Because this camera mount is designed to be mounted on the structure itself, the vibrations affected the camera by causing it to deviate from its initial fixed position and even rattling the camera, creating interference in its accuracy to read the weight plates or conduct the necessary repetition counting.

This issue created a conflict between reliability and flexibility, where if the system is to be more reliable, some flexibility may need to be sacrificed in order to accomplish it. The choice to incorporate a rotary dial angle locking mechanism and a solid structural cantilever into the design reduced the flexibility of

adjusting the camera position on the fly, necessitating the use of specific tools to even make minor adjustments to the camera's position. The project team was unable to accomplish the level of flexibility that was previously hoped for, but considering that this adjustment was essential to assure the fulfilment of stakeholder needs, it was judged necessary in the end.

The camera mount's modular architecture retains some degree of freedom in that the camera mount, cantilever, and camera interface modules are interchangeable to meet the demands of various smith machines. As demonstrated in this study, design modifications on the module may be made individually, reducing the requirement to rethink and remake the entire system to meet the needs of a new machine.

Identification of Weight Plates using Plate Thickness

In this project the diameter of the weight plates has been used to differentiate the different types of weight plate and their corresponding weights. However, if the weight plates all share the same diameter, their thickness will have to be the defining parameter.

However, the idea of using the thickness of the weight plate may not be ideal. Referring to Figure 6.2, the idea would only work only if the camera were placed perfectly parallel above the weight plate in pos 1, but in reality, the camera can never be positioned perfectly above all weight plates mounted on the barbell.

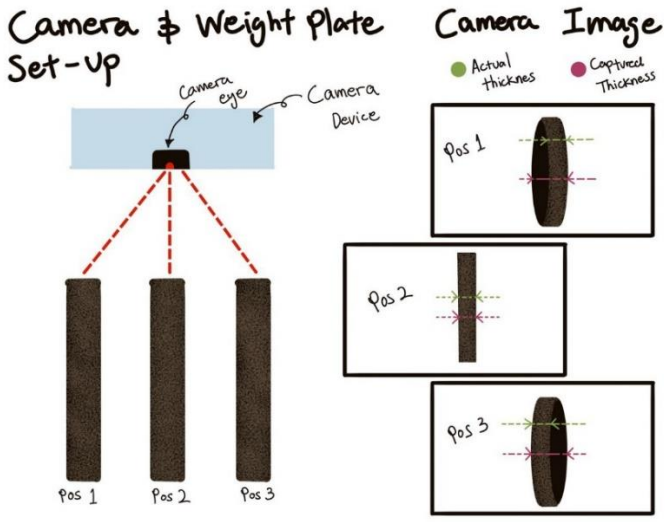


Figure 6.2: Visualisation on How Weight Plate Identification Works Using Plate Thickness

There are also cases where the weight plate is placed like in pos 2 and 3. The actual weight thickness is labelled in green, which would correspond to a certain weight value. However, the system would take the thickness, which is labelled in purple, which is the thickness of another type of weight plate, hence returning the wrong label/value of the weight plate. Therefore, utilizing the thickness for weight plate identification is not encouraged.

7. Project Timeline

Figures 7.1 and 7.2 depict the Gantt Chart generated for the entire project timeline for CAPSTONE 1 and 2, respectively. The Gantt Chart reflects all of the project tasks that were completed within the specified timeframe.

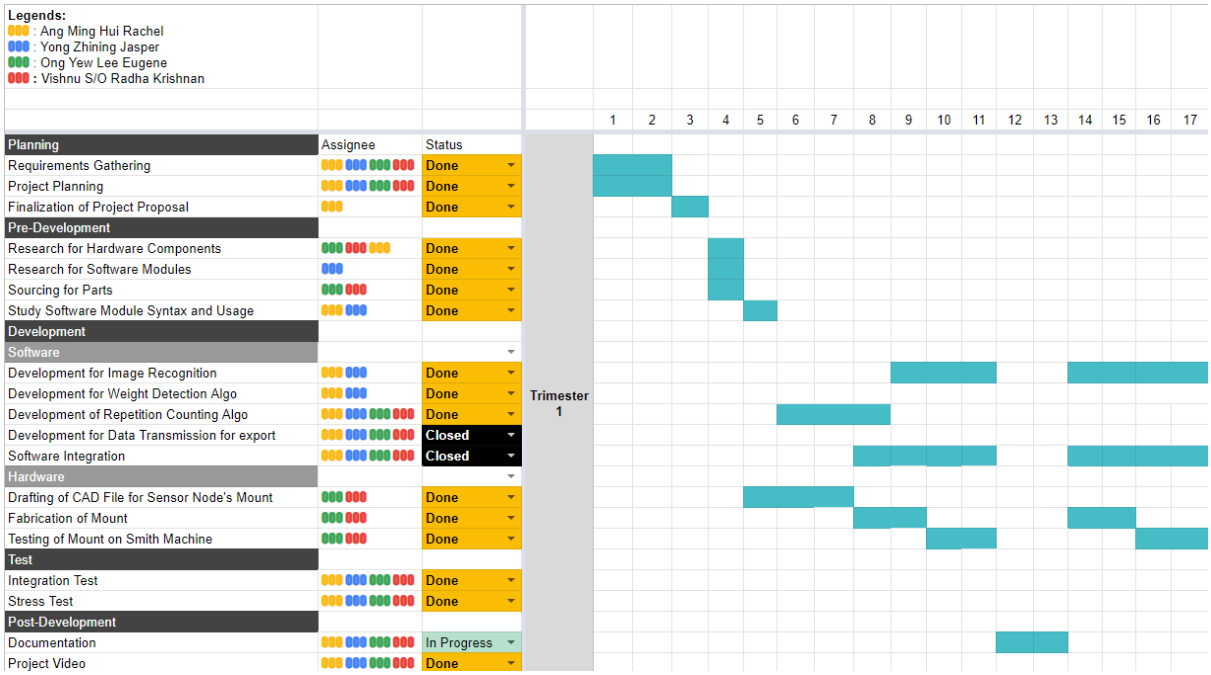


Figure 7.1: Project Gantt Chart for CAPSTONE 1

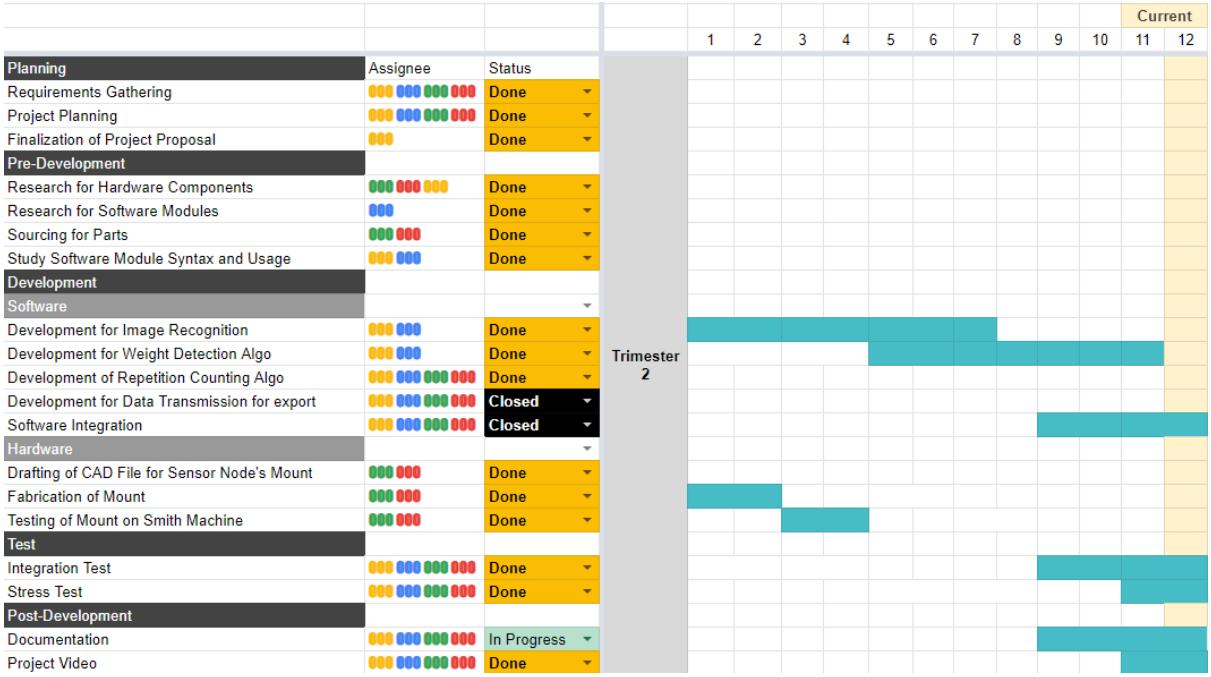


Figure 7.2: Project Gantt Chart for CAPSTONE 2

8. Budgeting

As shown in Table 8.1, the project was allocated a fund of SGD1000, and the remaining balance as of 1st March 2022 is SGD350.35.

Table 8.1 – Project Cost Management

Date	Debit/Credit (SGD)	Balance (SGD)	Description
06/09/2021	1000	1000	Initial Project Funds
06/10/2021	-199	801	Jetson Nano & Power Brick
08/10/2021	-200	601	Intel RealSense D435
21/10/2021	-8.3	592.7	40mm CPU Fan
21/10/2021	-22	570.7	SANDISK 64GB Micro SD Card
21/10/2021	-28	542.7	Edimax Ac600 Wireless Mini Dual-Band USB Adapter EW-7811UTC
21/10/2021	-1	541.7	Hardware Screws
05/11/2021	-23	518.7	Power Plug for Jetson Nano
19/11/2021	-74.9	443.8	ALU Plate
30/12/2021	-56	387.8	Edimax Ac600 Wireless Mini Dual-Band USB Adapter EW-7811UTC
01/03/2022	-37.45	350.35	C-Channel ALU Plate 1mm Thickness

9. Conclusion

Over the past two trimesters, the project team conducted immense research on both the theory and algorithms required to bring both the hardware and software components to life. The project team was able to demonstrate the two main features the project team set out to improve on, namely the weight identification by image recognition and the counting of repetitions and sets by an integrated depth camera. These features were accomplished by a single integrated modular system consisting of a depth camera, an MCU, and a modular camera mounting system. Despite the challenges posed by the COVID-19 pandemic throughout the project's duration, the project team was able to push through and complete the second iteration of the Smart Smith Machine. The project outcomes were documented to be handed over to the next project team to learn from the challenges faced and how the project could be improved. Every iteration that is completed brings the project closer to its end goal of being deployed at gyms all around Singapore.

References

[1] "Jetson Nano vs Raspberry Pi 4: The Differences," All3dp.com. [Online]. Available: https://all3dp.com/2/raspberry-pi-vs-jetson-nano-differences/. [Accessed: 22-Oct-2021].

[2] yida, "Rock pi N10 vs raspberry pi 4 vs Jetson nano - AI and Deep learning capabilities - latest open tech from seeed," Seeedstudio.com, 05-Dec-2019. [Online]. Available: https://www.seeedstudio.com/blog/2019/12/05/rk3399pro-vs-raspberry-pi-4-vs-jetson-nano-ai-and-deep-learning-capabilities/. [Accessed: 22-Oct-2021].

[3] E. Brown, "Rock Pi N10 SBC delivers AI-enhanced RK3399Pro starting at $99 - LinuxGizmos.com," Linuxgizmos.com, 04-Dec-2019. [Online]. Available: https://linuxgizmos.com/rock-pi-n10-sbc-delivers-ai-enhanced-rk3399pro-starting-at-99/. [Accessed: 22-Oct-2021].

[4] "Depth Sensors Comparison," Ipisoft.com. [Online]. Available: https://docs.ipisoft.com/Depth_Sensors_Comparison. [Accessed: 22-Oct-2021].

[5] "StarTech.com N150 WiFi USB 2.0 Dongle," Rs-online.com. [Online]. Available: https://sg.rs-online.com/web/p/wireless-adapters-wifi-dongles/8988585/. [Accessed: 22-Oct-2021].

[6] "TL-WN722N," Tp-link.com. [Online]. Available: https://www.tp-link.com/us/home-networking/usb-adapter/tl-wn722n/. [Accessed: 22-Oct-2021].

[7] MaxWeb Multimedia Design, "EDIMAX - Wireless Adapters - AC600 Dual-Band - AC600 Wireless Dual-Band Mini USB Adapter," Edimax.com. [Online]. Available: https://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/au/wireless_adapters_ac600_dual-band/ew-7811utc. [Accessed: 22-Oct-2021].

[8] "CAMVATE Camera Clamp Mount for DSLR Camera Led Video Light and Binoculars Holder Mount," Amazon.com. [Online]. Available: https://www.amazon.com/CAMVATE-C-Clamp-Desktop-Holder-Camera/dp/B01N64SZ4G. [Accessed: 22-Oct-2021].

[9] "Jetson Software," Nvidia.com, 14-Oct-2015. [Online]. Available: https://developer.nvidia.com/embedded/develop/software. [Accessed: 22-Oct-2021].

[10] "How to Install Xrdp Server (Remote Desktop) on Ubuntu 20.04," Linuxize.com. [Online]. Available: https://linuxize.com/post/how-to-install-xrdp-on-ubuntu-20-04/. [Accessed: 22-Oct-2021].

[11] "3D Printing Materials – Keene Village Plastics," Villageplastics.com. [Online]. Available: https://www.villageplastics.com/3d-printing-materials/. [Accessed: 22-Oct-2021].

[12] “Developing depth sensing applications - collision avoidance, object detection, volumetric capture and more,” Intelrealsense.com, 23-Feb-2021. [Online]. Available: https://www.intelrealsense.com/sdk-2/. [Accessed: 22-Oct-2021].

[13] “About OpenCV,” Opencv.org, 06-Mar-2019. [Online]. Available: https://opencv.org/about/. [Accessed: 22-Oct-2021].

~ End of References ~

Appendix A

The following tables depict the project's Bill of Material (BOM). The BOM List is segmented into the various hardware modules of the system.

Project Bill Of Materials							
Item	Name	Quantity	Description	Type	Supplier/Brand	Cost (SGD)	Source/Place of Purchase
Electronics Module							
1	Jetson Nano Developer Kit	1	Primary GPU for Project	GPU	Nvidia	$ 176.00	Sim Lim Square - Continental Electronics Pte Ltd
2	Power Brick	1	For powering Jetson Nano	Power Unit	NIL	$ 23.00	Sim Lim Square - Continental Electronics Pte Ltd
3	RealSense D435	1	Primary Sensory unit for Project	Depth Camera	Intel	$ 200.00	Carousell - @vivahate
4	40mm CPU Fan	1	For cooling Jetson Nano Board	Cooling unit	NIL	$ 8.30	Sim Lim Tower - Kaichin Computer Systems Pte Ltd
5	64GB Micro SD Card	1	Memory unit for Jetson Nano	Memory card	SANDISK	$ 22.00	Sim Lim Square - Win Micro Pte Ltd
6	Ac600 Wireless Mini Dual-Band USB Adapter EW-7811UTC	1	WiFi unit for Jetson Nano	WiFi Adapter	Edimax	$ 28.00	Sim Lim Square - Bizgram Asia Pte Ltd
7	M3 Steel Phillips Round Pan Head Screw 25mm	4	To secure CPU fan to Jetson Nano	Hardware Screws	NIL	$ 1.00	Lazada
8	Type-C USB Cable 3m	1	Connect Depth Camera to GPU	USB Cable	NIL	$ 8.00	Shopee Singapore
9	Velcro Straps	3	For cable management and mounting GPU	Straps	NIL	$ 3.00	Shopee Singapore
C-Clamp Module							
10	C-Clamp Handle	1	Base of camera mount (105g)	3D-Print	self-made	$ 4.27	Eugene
11	C-Clamp Adjusing Screw	1	For clamping of camera mount (35g)	3D-Print	self-made	$ 1.41	Eugene
12	C-Clamp Movable Jaw	1	For clamping of camera mount (5g)	3D-Print	self-made	$ 0.22	Eugene
13	C-Clamp Rotary Dial	1	For adjusting angle of cantilever arm(42g)	3D-Print	self-made	$ 1.72	Eugene
14	M5 Steel Button Head Screw 16mm	2	Integrate rotary dial to C-clamp handle	Hardware Screws	NIL	$ 0.20	Lazada
15	M5 Steel Nut	2	For M5 Screws	Hardware Screws	NIL	$ 0.20	Lazada
16	Non-slip Self Adhesive Furniture Rubber Pads	5	Cushioning to protect smith machine mounting surface	Rubber Pads	NIL	$ 2.00	Lazada
Cantilever Module							
17	C-Channel ALU Plate 1mm Thickness	1	Camera Mount Cantilever Arm	Aluminium Plate	NIL	$ 37.45	Kian Huat Hardware Company
18	Cable Guide	4	Cable Management for USB cable(4g)	3D-Print	self-made	$ 0.64	Eugene
19	M5 Steel Cap Head Screw 10mm	2	Locking screw for cantilever	Hardware Screws	NIL	$ 0.20	Lazada
20	M5 Steel Cap Head Screw 10mm	2	Mounting screws for Camera Interface Rotary Dial	Hardware Screws	NIL	$ 0.20	Lazada
21	M5 Steel Button Head Screw 20mm	1	Axis screw for cantilever	Hardware Screws	NIL	$ 0.10	Lazada
22	M5 Steel Nut	5	Screw nuts for cantilever	Hardware Screws	NIL	$ 0.50	Lazada
Camera Interface Module							
23	Camera Interface Rotary Dial	1	For adjusting angle of depth camera(34g)	3D-Print	self-made	$ 1.38	Eugene
24	Camera Interface L-Shape Bracket	1	Horizontal Adjustment for depth camera(23g)	3D-Print	self-made	$ 0.95	Eugene
25	Camera Interface Housing unit	1	Protective frame for depth camera(30g)	3D-Print	self-made	$ 1.23	Eugene
26	M6 Steel Cap Head Screw 15mm	1	Mounting depth camera to housing unit	Hardware Screws	NIL	$ 0.15	Lazada
27	M6 Steel Nut	1	Mounting depth camera to housing unit	Hardware Screws	NIL	$ 0.15	Lazada
28	M5 Steel Button Head Screw 12mm	1	Axis screw for L-Shape Bracket	Hardware Screws	NIL	$ 0.10	Lazada
29	M5 Steel Nut	1	Axis screw nut for L-Shape Bracket	Hardware Screws	NIL	$ 0.10	Lazada
30	M3 Steel Button Head Screw 12mm	1	locking screw for L-Shape Bracket	Hardware Screws	NIL	$ 0.10	Lazada
31	M3 Steel Button Head Screw 12mm	2	interfacing housing unit to L-shape bracket	Hardware Screws	NIL	$ 0.20	Lazada
32	M3 Steel Nut	3	Screw nuts for L-Shape Bracket	Hardware Screws	NIL	$ 0.10	Lazada