Graduate Certificate Intelligent Robotic Systems Practice Module Report: An Intelligent Cellphone Adaptive Smart Holder (CASH) With Always The Perfect Viewing Angle

Xin Cheng GU, Pei Feng LI, Xuan Bo XU, Yi Zhou YANG, Wee Ping YEONG
Institute of Systems Science, National University of Singapore, Singapore 119615

***Abstract*— The motivation of our project is to address the many situations where people just wish our ever useful cellphone can intelligently follow whatever direction we are looking at when our hands are inconvenienced or just tired. This is what drives our team to develop an adaptive cellphone holder that utilizes a robotic arm (Tinkerkit Braccio) coupled with depth camera (RealSense) which can do just that. To achieve this aim, we explored using various face and hand detection and landmark tracking models. We had also experimented with hand gesture recognition algorithms that will provide the option to switch between face or hand tracking or even freeze mode. In the movement of the robotic arm, it is often desired for the end-effectors to reach the target destination in a smooth trajectory. It is this inclination that our team decided to employ PID control feedback system to help reach this objective. On the robotic arm driver, the team had also designed our own command encoding methodology which can encode all commands with only 5 bytes thus reducing transmission delay and increasing response sensitivity. The innovation of this drive enabled us to simulate direction and speed with precision and to execute several commands simultaneously.**

***Index Terms*—robotic arm driver, tinkerkit braccio, face tracking, hand gesture tracking, posture alert, mediapipe, adaptive smart phone holder, PID control**

I. Introduction

With the ubiquitous cellphone getting more and more involved in every aspect of our life, people are spending a lot of time looking at their cellphones for various activities. However, sometimes people's experience with their cellphones is not particularly good. For example, when in the kitchen preparing food, it is common to see people using their phones to view recipes or videos demonstrating how a dish is being made. However, people often find it very difficult to keep up with the steps in the recipe or videos, because they often need to constantly move around and monitor many things thus it is impossible to be always keeping an eye on the screen.

One option is to hold the phone with them in one hand and work with the other hand, but the problem is that not everyone has the flexibility to use only one hand for tasks. The same embarrassment also occurs when making handicrafts and repairing equipment, etc. At that moment, people often want to have a "third hand". Another issue often occurs when people look at their phones for long periods of time for reading e-books or even enjoying an online movie. Not only will their hands get tired holding the device, a prolong incorrect neck posture will result in long term neck and spinal issues that is detrimental to a person's well-being.

To address these problems, we propose a smart and novel robotic cellphone holder system called Cellphone Adaptive Smart Holder (CASH) that tracks the user's movements and intelligently adjusts its position to ensure the user maintain a good viewing angle. This system integrates cutting-edge computer vision models such as Face Detection, Face Pose Estimation, Hand Detection and Gesture Recognition models, as well as the classic PID control system to achieve accurate and smooth real-time tracking of targets. To apply the PID algorithm on the Braccio manipulator, a special driver is proposed. In addition, it also features an alert system that will notify the user whenever an undesirable neck posture is detected. In this work, we also conducted experimental comparisons of some existing methods. We believe that these explorations will help to create a truly practical and feasible industrial-grade product in this direction.

A. System Demonstration

Video demonstrations can be viewed via the following YouTube links:

- Up, Down, Left, Right tracking demo url: https://www.youtube.com/watch?v=DVr2x93KlOQ
- Forward, Backward tracking demo url: https://www.youtube.com/watch?v=_69mIraG3G8
- Bad neck posture alert demo url: https://www.youtube.com/watch?v=Mf7oWQOTHns&t=3s

II. System

A. Overall System Architecture

The whole system can be divided into four layers: Data Collection layer, Data processing Layer, Application Layer and Driver layer. There four core sub-modules: Process Control Module, Arm Control Module, Face Detection Module, Hand Detection Module and drivers are distributed in different layers to complete various functions of the system. Figure 1 shows the overall system architecture.

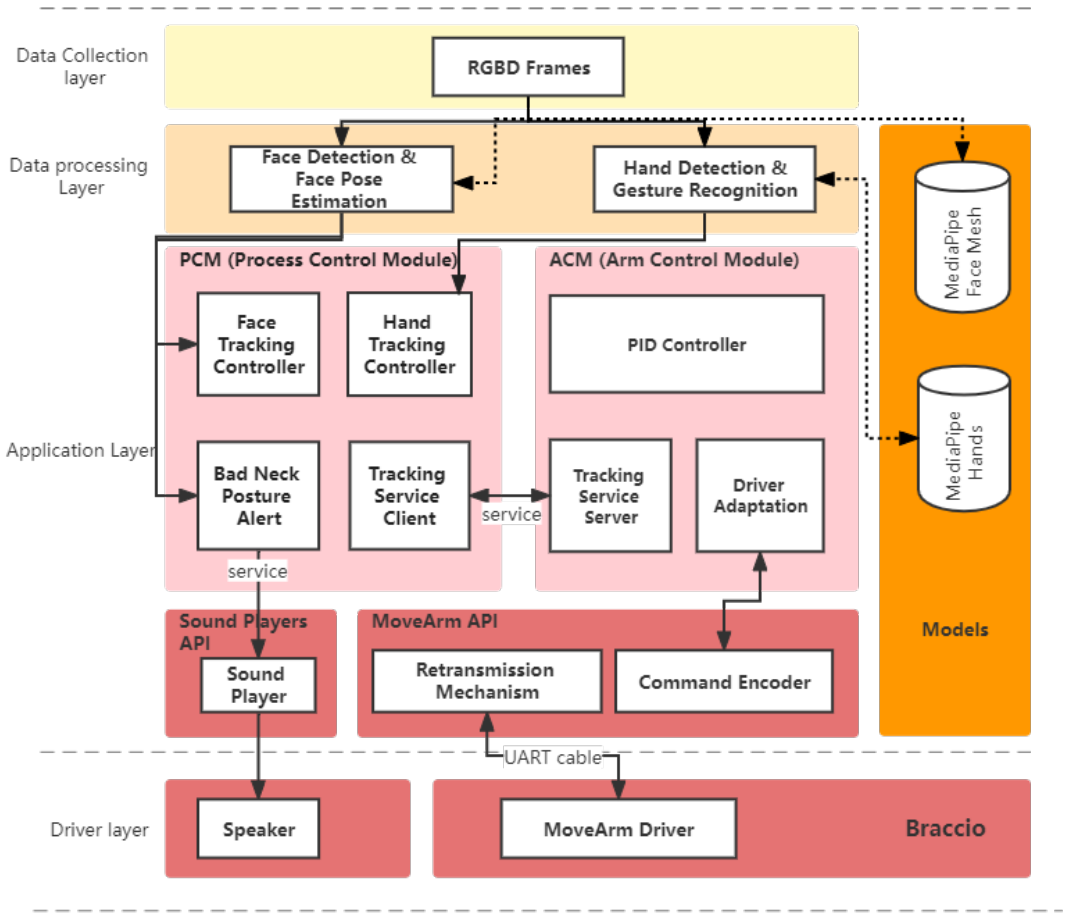

Fig. 1. System Architecture

At the data collection layer, a RealSense D415 camera[1] is employed to capture both image information and depth information in real time. Face Detection, Face Pose Estimation, Hand Detection and Gesture Recognition are performed based on this information at the data processing layer. The results of the model will be sent to the Application Layer, where the Process Control Module will collect all the results information to make decisions and give instructions to Arm Control Module. The Arm Control Module will then further instruct the driver through the PID algorithm. Finally, our own innovative robotic arm driver that was designed to communicate efficiently with the Braccio through the serial port would drive the Braccio manipulator arm to the specified position.

B. ROS architecture

The whole system is integrated based on Robot Operating System (ROS) [1]. As shown in the Figure 2, there are a total of 7 nodes, 4 topics and 2 services in the whole system, which has two advantages: (1) High fault tolerance. Failure of some nodes does not affect other nodes. (2) Efficiency. Asynchronous parallel execution between nodes can greatly improve the operating efficiency of the system, especially for systems like CASH with multiple intelligent models. However, using multiple nodes will cause messages belonging to the same moment to arrive at a node at different times due to different execution speeds between different nodes. For example, the recognition results of face and hand models belonging to the same time may arrive at different times due to the difference in model processing speed when transmitted to the Process Control Module. To address this issue, the ApproximateTimeSynchronizer is applied to align the messages at each moment according to the timestamp.

C. Process Control Module

The Process Control Module (PCM) is the main logic program of CASH, which oversees decision making, errors handling and instruction. Figure 3 shows the workflow of the PCM. The PCM will first obtain the current target information from the detection and recognition module. Then it will determine which mode the current system is in, whether it

[1]RealSense Camera is a kind of depth camera. Products are available at https://store.intelrealsense.com/

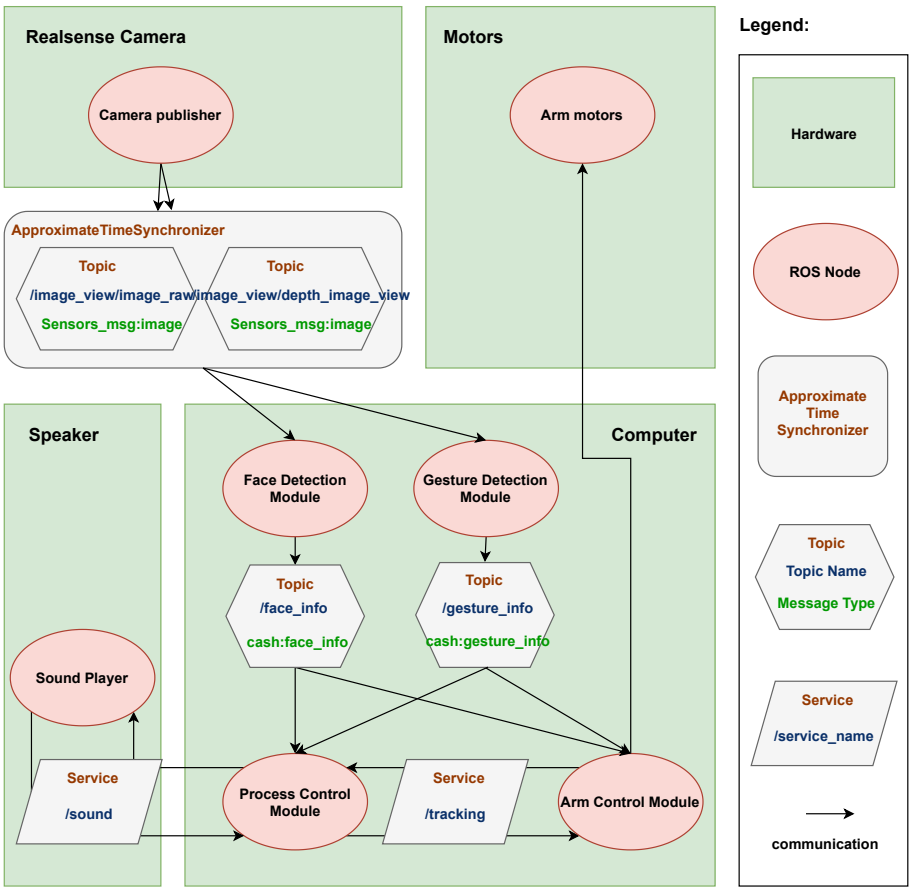

Fig. 2. Ros Architecture

is necessary to start CASH to track the target, and whether it is necessary to perform bad neck posture alert to the user. Finally, the PCM will call different services to take action and PCM will also handle errors in the whole process. The entire workflow is designed to take care of user experience and improve fault tolerance as much as possible.

It should be noted that the PCM will send different commands to the ACM under different circumstances. More specifically, PCM sets up a virtual center boundary box, which is in the center of the frame and occupies one-ninth of the frame. The PCM determines what instructions to send the ACM by observing whether the center of the target is within the center boundary box or not. When a detected face or hand target is outside the center box, it means that the mobile phone is in a relatively biased position for the user. In this case, the PCM will send the command to the ACM to move the holder (Up/Down/Left/Right) towards the center of the frame using PID Controller. Once the target center enters the center bounding box, the PCM will instruct the ACM to perform Front/Back movement using depth data until the optimal distance is maintained. This strategy is simple but effective, and works well with the PID controller.

III. Face Detection

A. Purpose and Related Work

The face detection module has two main purposes, the first is to generate the face position coordinate relative to the camera, which is used to later control the robot arm to track the user face, the second is to generate the face rotation angle (yaw and pitch) relative to the camera, we will use this rotation angle to determine whether the current head pose is correct, if not correct and the user continues to hold the bad posture for more than a certain period, an alarm will be issued to the user.

According to our desired functionality and the purpose of face detection, our face detection module can be divided into three parts, which are Face Mesh (face detection and face landmarks mapping), Head Pose Estimation and Bad Posture Alert. Fig. 4 shows the workflow of the face detection module.

There are many ways to generate face landmarks, currently the more popular methods are Google MediaPipe Face Mesh [2] and Dlib [3]. We have done some research to compare the two methods, where we discovered the biggest difference between these two methods is that they work on different pipelines. For Dlib, each frame needs to perform face detection and facial landmarks, so the inference speed is slow, and it is not smooth in real-time video, especially for lightweight

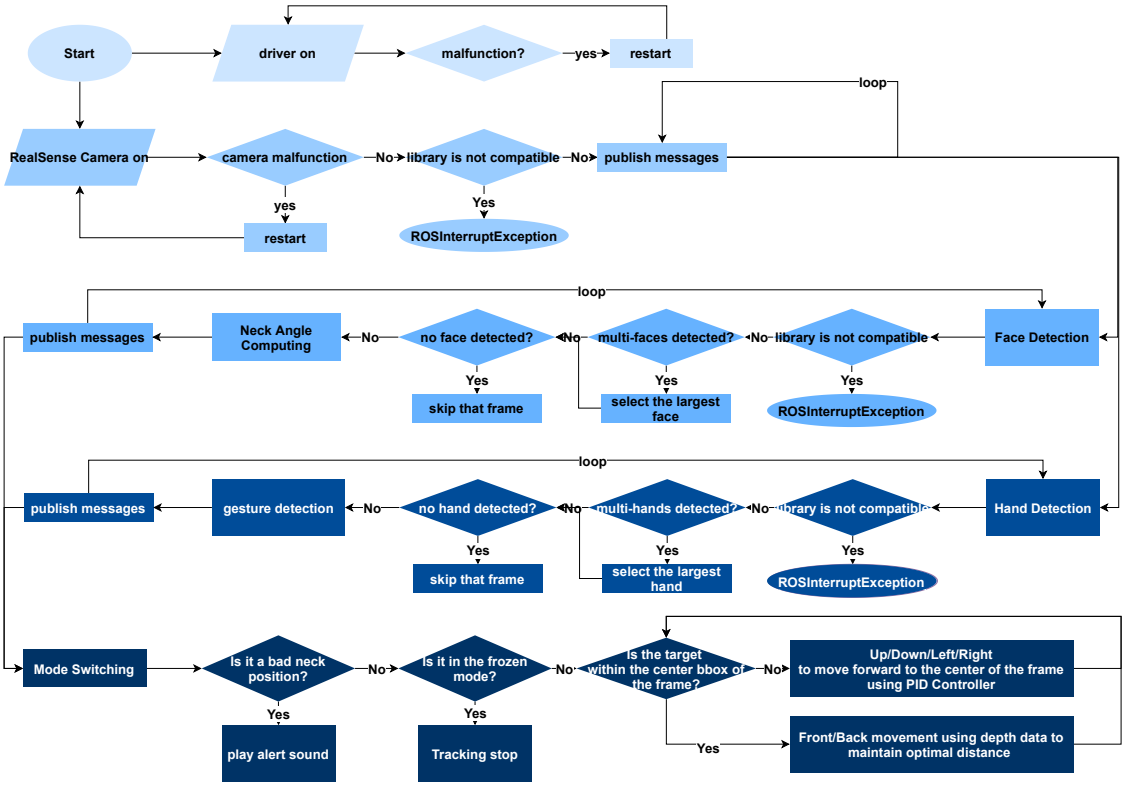

Fig. 3. PCM workflow

portable devices. But for MediaPipe, face detection and facial landmarks detection is only performed once in the beginning, the remaining process is just landmarks tracking across frames, potentially much faster inference time as tracking can be less costly computationally than detection, so MediaPipe has much faster speed.

In this project, we obtain the rotation angle of the head in order to detect if the user is in a bad posture, but in reality we need to know how the head is tilted with respect to a camera in many applications. In a virtual reality application, for example, one can use the pose of the head to render the right view of the scene. In a driver assistance system, a camera looking at a driver's face in a vehicle can use head pose estimation to see if the driver is paying attention to the road. Of course one can also use head pose based gestures to control a hands-free application or game.

B. Face Mesh

Face mesh mainly generates face landmarks of points in the frames. We used the Google MediaPipe Face Mesh library in this project, which is a solution that estimates 468 3D face landmarks in real-time even on mobile devices. It employs machine learning (ML) to infer the 3D facial surface, requiring only a single camera input without the need for a dedicated depth sensor. Utilizing lightweight model architectures together with GPU acceleration throughout the pipeline, the solution delivers real-time performance critical for live experiences. Fig. 5 illustrates the detected landmarks in the face. The red box indicates the cropped area as input to the landmark model, the red dots represent the 468 landmarks in 3D, and the green lines connecting landmarks illustrate the contours around the eyes, eyebrows, lips and the entire face. The detected landmarks will further be used in head pose estimation part. Our team had also considered the situation of having multiple faces in a single frame, in which case we set a rule to output the landmarks of the face closest to the camera. Meanwhile, Face Mesh will also output the landmark coordinates of nose bridge point as the target coordinates for the robotic arm to track later.

C. Head Pose Estimation

Face pose estimation part is used to generate the rotation angles yaw and pitch of the detected face relative to the camera. Based on the landmarks we can then estimate the head pose, which is considered a Perspective-n-Point [4] pose computation [5] problem. The pose computation problem consists of solving for the rotation and translation that minimizes the reprojection error from 3D-2D point correspondences. We used the function *solvePnP* [6] and related functions from OpenCV [7] library to estimate the face pose. To calculate the pose of a face in an image we need the 2D coordinates of a few points and corresponding 3D locations of the same points, as well as the camera intrinsic matrix and the distortion coefficients (see Fig.6, more precisely the X-axis of the camera frame is pointing to the right, the Y-axis downward and the Z-axis forward).

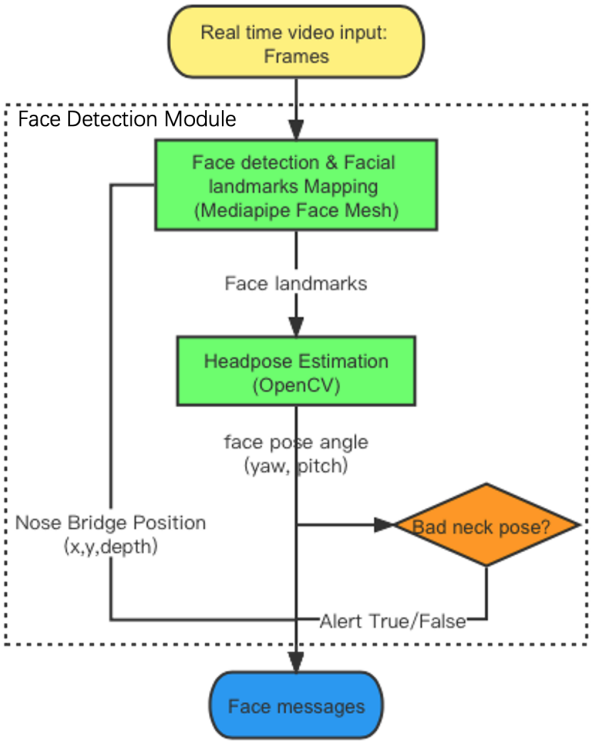

Fig. 4. Face Detection Module Workflow

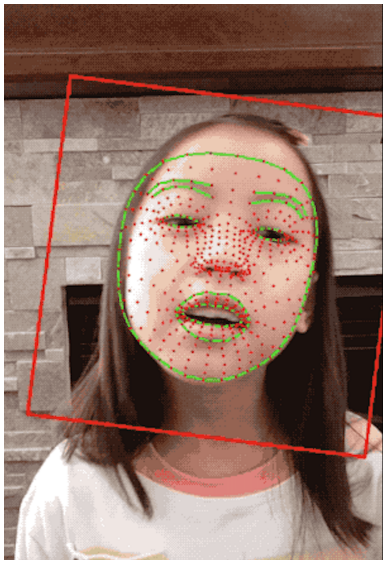

Fig. 5. 468 3D Facial Landmarks

According to the instruction of OpenCV*solvePnP*, the number of input landmarks must be more than 4 points and object points can be in any configuration [5], and we also referred to some tutorials on the Internet [8] where the author had also used only a small number of points to complete the head pose estimation. So in our project, we used just 6 landmarks as inputs instead of the entire 468 points to do the pose estimation, as this would reduce calculations. We used these key points: tip of the nose, the chin, the left corner of the left

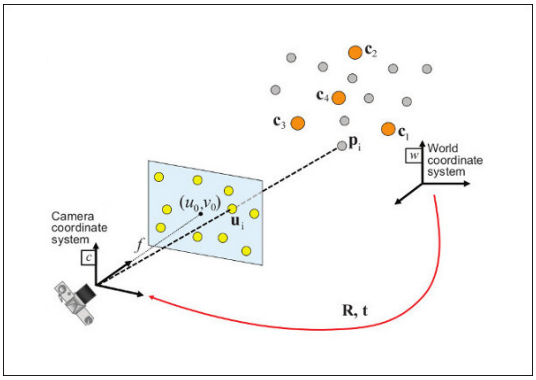

Fig. 6. 468 3D Facial Landmarks

eye, the right corner of the right eye, the left corner of the mouth, and the right corner of the mouth. Fig.7 illustrates the location of these key points that we used. After the processing of head pose estimation, the result of head rotation angle will be used for further bad posture alert.

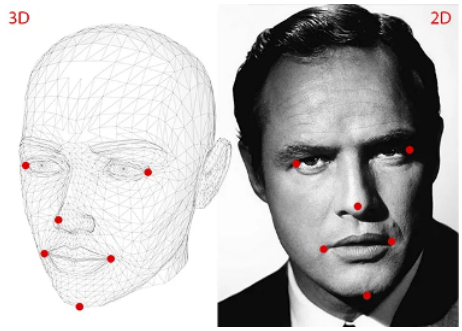

Fig. 7. Key Input Points of the Landmarks

D. Bad Posture Alert

The Bad Posture Alert part is used to judge the rotation angle of the head, and to send out sound and screen display alarms to the user when the user is detected to be in a bad posture for a prolong period of time. After we get the head rotation angle yaw and pitch from Head Pose Estimation part, we just set a rotation threshold to determine if the user is in a bad viewing posture. The threshold for yaw is 30 degrees and minus 30 degrees, and the threshold for pitch is 15 degrees and minus 30 degrees, which means that if the user turns left or right more than 30 degrees, or tilts up more than 15 degrees or tilts down more than 30 degrees, then we consider the user to be in a bad posture after a defined period of time. We chose these threshold because these seems to be the optimal head viewing angle based some studies [9] we found on the internet.

Postural discomfort and the consequent aches and pains result from the effort to view the monitor when it is set at the wrong place in relation to the operator's position. A poor angle leads to postural (neck and shoulders) discomfort. According to the research, for comfortable viewing on a screen it is probably reasonable to place the monitor at about 15 degrees

(or slightly lower) below the horizontal line. Such a location creates a preferable visual zone of 30 degrees (see Fig.8) which consists of +15 degrees to -15 degrees from the normal line of sight. Numerous field studies among people doing intense visual work indicate that looking upwards (above the horizontal) is tiring. On the other hand, looking downwards, that is, lower than 15 degrees below the horizontal, was not reported as particularly fatiguing. This finding allows one to extend the visual zone downward by another 15 degrees, so the acceptable visual zone is from +15 to -30 degrees.

For practical use the time threshold can be set to 10 minutes or something else, but for the demo we only set it to 5 seconds in our tests. When the user maintains a bad posture for more than the time threshold, an alarm will be triggered which composed of an alarm sound that will be issued through the speaker, as well as a red alarm box that will be displayed on the screen.

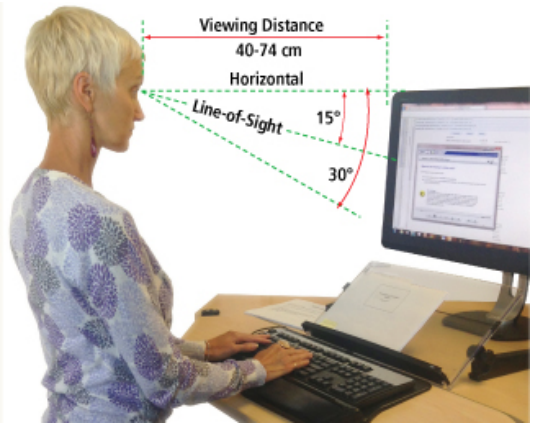

Fig. 8. Monitor Position

IV. Gesture Recognition

A. Purpose and Related Work

The purpose of this module is to recognize gestures representing 0 to 5 in real-time video captured by the RealSense camera, enabling users to control the holder and switch between different modes with gesture.

As to how it is utilized in this project, Gesture Recogntion is a widely used design in human-computer interaction (HCI) because it provides a natural and flexible way to input signals [10]. To recognize gestures, glove-based method using sensors to acquire the physical position of hands, as well as vision-based method reasoning directly from video inputs, have been developed by researchers [11]. In our project's case, vision-based method was selected considering the hardware we have. Currently, there are a lot of methods for vision-based gesture recognition, and the majority of them depends on hand detection (or called hand localization) to first extract hands from the input frame [12]. Feature-based methods like segmentation using color [13], depth [14], HoG [15], as well as learning-based methods using CNN [16], have been proposed to solve this detection task [11]. Based on localized hands, feature extraction and classification can be performed to recognize gestures [17]. Depth information is commonly used since it not only helps localize hands, but also reflects posture of hands [18]. However, for out project's use cases, many of the existing methods are either not robust enough, or incapable of real-time processing. Therefore, we proposed a Gesture Recognition solution, which uses a real-time Hand Pose Estimation pipeline to localize hands and estimate coordinates of hand landmarks as features, and a classification method to classify the gesture based on features extracted. It is also worth mentioning that depth frames are not used in our solution because RGB frames alone are accurate enough for our purpose.

B. Hand Pose Estimation

Hand Pose Estimation refers to the task that locates hands in input frames and predicts coordinates of hand landmarks. Google MediaPipe Hands, an end-to-end pipeline for Hand Pose Estimation, is used in our project [19]. It consists of a palm detection model for hand localization and a hand landmark model for coordinate prediction. Hands are first located by the former model and then the cropped regions of hands are sent to the latter model to predict coordinates of hand landmarks. Besides, the pipeline can also utilize hand landmarks identified in the previous frame to locate hands faster. Fig. 9 shows 21 hand landmarks that can be predicted by Google MediaPipe Hands.

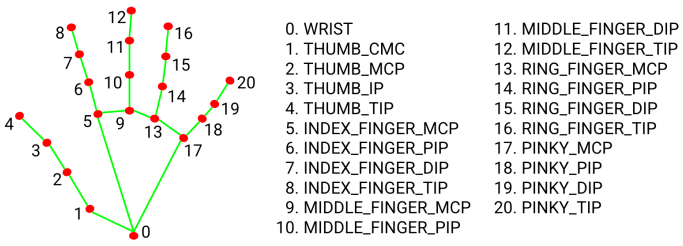

Fig. 9. 21 hand landmarks

To evaluate the pipeline's performance, an experiment was conducted. The pipeline was made to perform Hand Pose Estimation on a test set, after which the predicted hands would be matched with ground truths and several metrics would be calculated to show the pipeline's capabilities. The matching process and calculations of metrics are shown below.

1) Set a threshold T.
2) For each image, calculate joints error between each predicted hand and each ground truth. The joints error is calculated by:

$$err = \sum_{i=0}^{20} ||p_i - g_i||$$

in which i is the index of landmark (see Fig. 9); p refers to coordinates of predicted hand's landmarks and g refers to coordinates of landmarks in the ground truth. This metric indicates how close to the ground truth the predicted hand is.

3) Select the 'prediction-ground truth' pair with the smallest joints error. If the joints error is smaller than the threshold T, the pair will be regarded as a successful match. A True Positive will be counted and remaining pairs containing the predicted hand and ground truth in this successful match will be removed from the checklist.
4) Continue checking until the checklist is empty.
5) Repeat 2) until all images are checked.
6) Calculate $MPJPE$ (Mean per joint position error) by averaging joints errors of all successful matches. This metric indicates generally how well the pipeline estimates hand landmarks under the threshold T.
7) Calculate False Positive by $FP = pred - TP$, where $pred$ indicates the number of all predicted hands. Also False Negative can be calculated by $FN = gt - TP$, where gt is the number of hands in ground truth. These two metrics show the accuracy of hand localization under the threshold T.

The Rendered Hand Pose (RHD) dataset [20] was used to build the test set. It is a widely used dataset consisting of 350 * 350 pixels RGB images. Each image includes a background and a rendered 3D character, either one or both hands of whom can be seen. Illumination and character's skin color vary by image. Labels of an image are coordinates of landmarks of the visible hand(s). The first 1,000 images was selected as our test set and Fig. 10 shows some of samples.

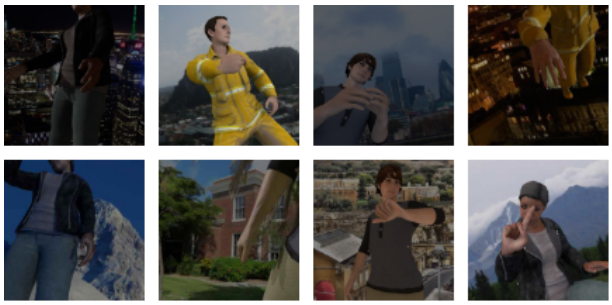

Fig. 10. Samples in the test set

Fig. 11 shows the metrics obtained by performing Hand Pose Estimation on the test set with Google MediaPipe Hands.

Confusion matrix			
	Pred. hands	Pred. BG	
Actual hands	866	493	
Actual BG	44	-	
Metrics			
Precision	Recall	MPJPE	FPS (on CPU)
95.2%	63.7%	37 pixels	12

Fig. 11. Metrics from the experiment

As shown, the pipeline has a 95.2% Precision of and a 63.7% Recall in hand localization. The slightly low Recall is still acceptable considering that not single images, but continuous frames from the input video that the pipeline is going to process in actual deployment where results from previous frames can be utilized to improve the performance. For landmark prediction, the $MPJPE$ is 37 pixels, meaning that the total error of all 21 landmarks in a correctly predicted hand is 37 pixels when the frame size is 350 * 350 pixels, which is relatively good. Besides, 12 FPS on CPU is enough for real-time processing. The results prove that the pipeline is capable for the Hand Pose Estimation task.

C. Gesture Classification

Based on estimated joints, we proposed a rule-based method and a learning-based method to classify the gesture.

In the rule-based method, three landmarks are selected from the thumb and each finger, then an angle is calculated and compared with 150° to determine whether the thumb or the finger is unbent. The number of unbent thumb and fingers (an integer from 0 to 5) represents the class of the gesture (class 0 to class 5). Specifically, the selected landmarks and the calculated angles are listed below (for indices of landmark, please refer to Fig. 9).

1) For thumb: $\angle p_0 p_2 p_4$
2) For index finger: $\angle p_5 p_6 p_8$
3) For middle finger: $\angle p_9 p_{10} p_{12}$
4) For ring finger: $\angle p_{13} p_{14} p_{16}$
5) For pinky: $\angle p_{17} p_{18} p_{20}$

For the learning-based method, a shallow Neural Network was trained to classify the gesture based on predicted landmarks. The input of the network is a (1, 42) vector, which is the concatenation of coordinates of all 21 landmarks from a predicted hand. Fig. 12 shows the architecture of the network.

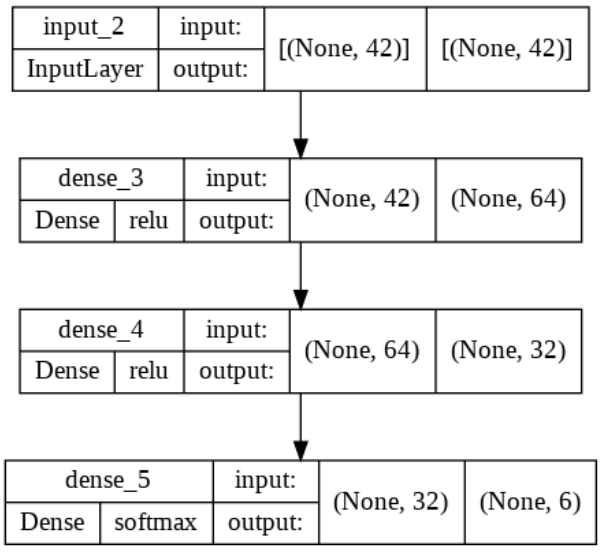

Fig. 12. Model architecture

The Sign Language Gesture Images Dataset [21] is used to train the network. It consists of 50 * 50 pixels RGB images, each of which contains only one hand with a certain gesture. Six gestures representing 0 to 5 were selected, with 1,500

Fig. 13. Samples of each gesture

images per gesture. Fig. 13 shows samples from each gesture class.

The total 9,000 images were first shuffled and then split into 60%: 20%: 20% for training, validation and testing. Images in the training and the validation set were processed by Google MediaPipe Hands to construct input vectors for model training, while the test set was untouched so it could be used to compare the two (rule-based and learning-based) methods later. X coordinates are divided by cropped region's width and y coordinates are divided by the height for normalization purpose before the concatenation. Fig. 14 shows the trends of accuracy and loss throughout the training process.

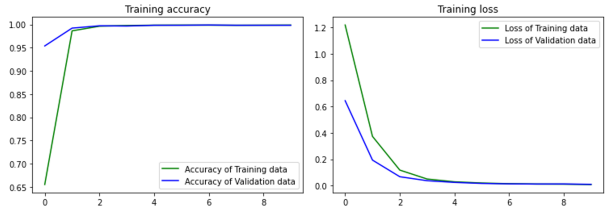

Fig. 14. Accuracy and loss in traning

Finally, the rule-based and learning-based method were concatenated with Google MediaPipe Hands respectively to build two completed pipelines for the gesture recognition task. Fig. 15 shows their performance on the test set mentioned above.

Rule-based, 35 FPS on CPU

	precision	recall	f1-score	support
0	1.00	0.73	0.84	310
1	0.80	0.99	0.88	298
2	0.84	0.96	0.90	276
3	0.77	0.89	0.83	312
4	0.89	0.72	0.80	306
5	1.00	0.93	0.96	298
accuracy			0.87	1800
macro avg	0.88	0.87	0.87	1800
weighted avg	0.88	0.87	0.87	1800

Learning-based, 32 FPS on CPU

	precision	recall	f1-score	support
0	1.00	1.00	1.00	310
1	1.00	1.00	1.00	298
2	1.00	1.00	1.00	276
3	1.00	1.00	1.00	312
4	1.00	1.00	1.00	306
5	1.00	1.00	1.00	298
accuracy			1.00	1800
macro avg	1.00	1.00	1.00	1800
weighted avg	1.00	1.00	1.00	1800

Fig. 15. Comparison of results

As shown, the rule-based method has a lower accuracy but a higher frame rate compared to the learning-based method, and it is more robust than the latter in deployment. Therefore, both methods could actually be deployed in our system but our team decided on the rule-based method which was found to be accurate enough to achieve our needs during our testing.

V. Tinkerkit Braccio Robotic Arm

A. Arm Control Module

For this project, we employed the use of Tinkerkit Braccio robotic arm (see Fig. 16) with 6 degrees of freedom (DOF) to produce complex movements and operations. There are a total of 6 servo motors with 1 at each joint, namely

Fig. 16. Tinkerkit Braccio Assembled

1) Base: rotate from left to right
2) Shoulder: helps lower arm move up/down
3) Elbow: helps upper arm move forward/backward
4) Wrist: helps wrist move forward/backward
5) Wrist Rotate: rotates wrist in circular motion
6) Gripper: open or close the gripper claws

The Arm Control Module (ACM) is where the mechanics of the tracking tasks are performed. We will only require the Base joint for left/right, the Elbow joint for up/down and also the Shoulder joint for front/back movements. For tracking operations, both speed and direction are crucial aspects in the ACM.

Braccio uses a servo motor which is controlled via Pulse Width Modulation (PWM) to precisely control directions through the rotational angle of motor at the various joints. PID circuit is utilized as a closed loop feedback controller by generating error signal which can be used to control speed.

B. How Servo Motor Works

A servo motor is a rotary actuator that allows for accurate control of angular position, acceleration and speed. Servo motors are used in numerous places like robotics, industrial application as well as manufacturing. These servo motors are small in size but pack a punch with energy-efficiency.

Servo motors are a part of a closed-loop system [22] and consist of a few parts which are namely: control circuit, shaft, potentiometer, amplifier, and gears. A servo motor is an electrical device and it is self-contained to turn a piece of a machine with great precision and high efficiency. The shaft output of the motor can be moved to a specific angle and speed that an ordinary motor cannot achieve. The servo motor uses a regular motor with a sensor for positional feedback. The controller of the servo motor is significant and is thus structured explicitly for this reason.

Servo Motors are controlled by Pulse Width Modulation (PWM) which is an electrical pulse with different pulse width sent via wire. Instead of varying the voltage level, PWM is used to precisely control the rotational speed and angle of a

DC Motor. The basic idea of PWM is that it uses a square wave with a fixed voltage value for logic 1 (high) and 0 (low) but different duty cycles to control the rotation [23]. The 6 servo motors (4x SR431 and 2x SR311) used in the Braccio robotic arm for this project use such PWM control.

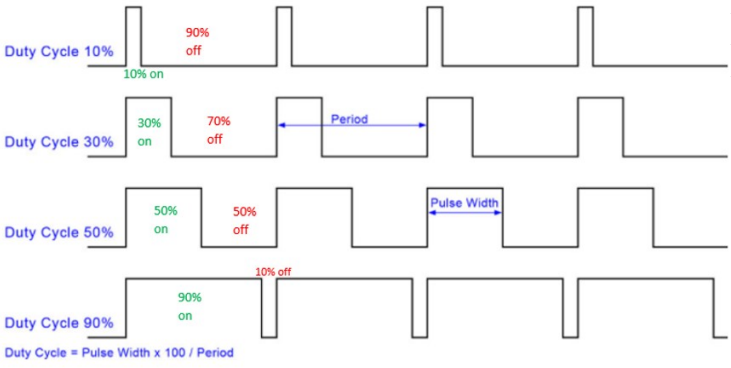

Fig. 17. PWM signals with different duty cycles

The duty cycle is defined as the proportion of time for logic 1 in the whole period of the square wave. Only when the PWM signal is in logic 1 state, the DC motor would perform rotation. In addition, the duty cycle reflects how often the PWM signal is in logic 1 state (see Fig. 17). In other words the extent of rotation is proportional to how high the duty cycle is in a period. However, duty cycle is not the only factor to decide rotation. The upper speed limit of DC Motor can also be proportional to the voltage value of logic 1, which means this value also has impact on the rotation speed when the duty cycle is not 100%.

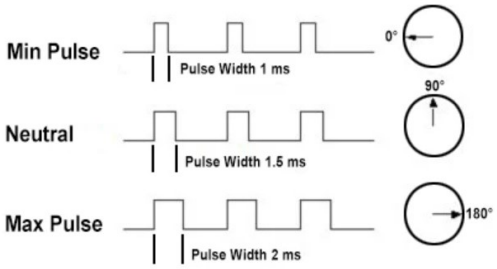

Fig. 18. Example showing how PWM controls servo motor rotation

A servo motor can usually only turn 90° in either direction for a total of 180° movement. To illustrate a simple and easy to understand example (see Fig. 18), a servo motor anticipates a pulse every 5milliseconds(ms) period and the pulse length will determine how far the motor turns. Here, a 1.5ms pulse (30% duty cycle) will make the motor turn to the neutral 90° position. A signal shorter than 1.5ms will move it in the counter clockwise direction towards the 0° position, and any longer than 1.5ms will conversely turn the servo in a clockwise direction towards the 180° position [24].

C. PID Controller for Smooth Movements

The PID circuit is often utilized as a closed loop feedback controller and is very commonly used for many forms of servo circuits. The purpose of any servo circuit is to hold the system at a predetermined value (set point) for long periods of time. The PID circuit actively controls the system so as to hold it at the set point by generating an error signal that is essentially the difference between the set point and the current value of the control device (see Fig. 19).

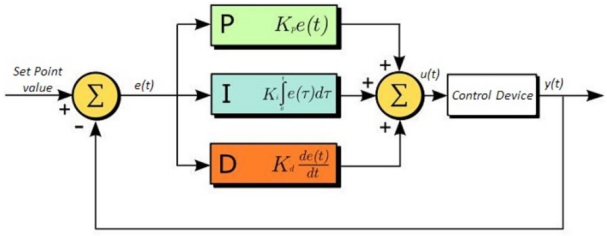

Fig. 19. PID diagram as a closed control loop feedback mechanism

In the movement of the robotic arm, it is often desired for the end-effectors to reach the target destination in a smooth trajectory. It is this motivation that our team decided to explore using PID method to help achieve this aim.

The PID controller is composed of three different actions, the proportional (P) action, the integral (I) action and the derivative (D) action. Each action has a different functionality that can help the process to improve its performance. Depending on the system requirements, a certain combination of these actions can be used. The possible combinations are: only proportional (P), proportional + derivative (PD), proportional + integral (PI) and proportional + integral + derivative (PID). The most common and effective one is the PID action, although sometimes it is enough to have a P, PI or PD. The sum of all three parts contribute to the control mechanism such as speed control of a motor in where

- P value depends upon current error,
- I on the accumulation of previous error, and
- D predict future error based on the current rate of change.

To understand each of these control actions and its application, first of all it is important to define the signal properties that are going to be affected by the control actions. The signal responses have certain properties that define the behaviour of themselves (see Fig. 20). These properties are

- Rise time: the necessary time for a signal (in our case, distance error) to change from a very low value to a very high value. Generally, it measures the time that it takes the signal to go from its 10% to its 90%. Clearly, it is preferred to have a short rise time so as to have a faster response.
- Overshoot: measures how much the signal exceeds its target value. It represents the difference between the maximum signal value and the stabilised value.
- Peak time: time it takes the signal to reach its first peak.

- Settling time: time that it takes the signal to become steady. This occurs when the signal is in the range of its 2% and 5% final value.
- Steady-state error: represents the difference between the final value of the signal and the value it has when it already becomes steady.

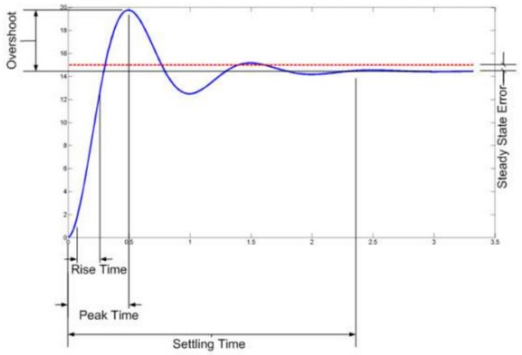

Fig. 20. Properties of a typical signal

1) PROPORTIONAL GAIN: The proportional action equation is:

$$u(t) = K_p.e(t)$$

The proportional gain is Kp, and it is so called because it is proportional to the error signal e(t). Larger proportional gain results in larger changes in response to the error, and thus affects the speed at which the controller can respond to changes in the system. While a high proportional gain can cause a circuit to respond swiftly, too high a value can cause oscillations and instability in the output of the system. Besides, when the gain is increased, the rise time decreases and the overshoot increases.

2) INTEGRAL GAIN: The integral action equation is:

$$u(t) = K_i \int_0^t e(t').dt'$$

The integral gain is Ki and this action changes the inputs of the process based on the accumulated error, therefore it is said that this action "looks" at the past. The integral action usually eliminates or reduces a lot the steady-state error thus improving the steady-state behaviour if the system is stable. However, if the gain is too small it will not make any effect, and if it is too big, it may cause oscillations. Similar to the proportional gain, when the integral gain increases, the rise time decreases and the overshoot increases.

3) DERIVATIVE GAIN: The derivative action equation is:

$$u(t) = K_d \frac{de(t)}{dt}$$

The derivative action is Kd, and this action is sometimes called "anticipatory" control because it "looks" at the future error. When the gain is increased, it decreases the overshoot and the settling time, and sometimes it also improves the stability if its value is small. On the other hand, it hardly has an effect on the rise time and the steady-state error.

As can be seen, the three control gains has both its advantages and its disadvantages, so an equilibrium must be found. What is already clear is that the combination of these three, the PID control action, is the best choice when it is desired to implement a controller in a system. However, it is not easy to find the values of the three gains. There are some methods that help the developer to reach these values, which are called tuning methods [25] and will be discussed in the following sections.

D. PID Controller Tuning

1) MANUAL TUNING: This method is carried out by looking at the system response, analysing it and applying the necessary gains. Depending on the response, the designer has to know which action the system needs to improve its behaviour. After applying the control action, the designer has to look again at the response and come up with the new changes in the control action to improve again the response. This process has to be repeated several times until the requirements are satisfied or until reaching the best system response possible. Normally, we start with the three gains Kp, Ki and Kd set to zero. Then, Kp is slowly increased until observing some oscillations in the output. At this moment, Kp is set to its half to have a "quarter amplitude decay" type response. This means that the period is reduced to its fourth part and the transient response becomes faster. However, a little overshoot may occur, so the Kp should be even smaller than its half. Then, Ki has to be increased little by little until offset is corrected and the steady-state error is eliminated. It is important to take into account that a high integral gain could cause instability. After doing this, the controller will usually be well defined, but if it is desired to have a faster response, Kd is increased. However, it will introduce overshoot if this gain is too big. If these steps are followed carefully, it is likely that the three gains together will help improve the system response. Nevertheless, the gains might not be really accurate since this is a rudimentary way of obtaining the PID parameters. As this is a "trial and error" method, it is very important to know what does each gain exactly do. Therefore, the following table shows the effects of each gain into the system behaviour when the respective gains are increased.

Parameter	Rise Time	Overshoot	Settling Time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor Change	Decrease	Decrease	No effect	Improve if K_d is small

Fig. 21. Effects of the signal characteristics when the gain parameters increase

2) ZIEGLER-NICHOLS METHOD: This method was introduced in the 1940s by John G. Ziegler and Nathaniel B. Nichols. It starts with Ki and Kd being set to zero. Kp is then increased until the system starts oscillating constantly.

The value of this proportional gain is called Ku, and the period of this oscillations is called Tu. These two values have to be obtained with the maximum precision because of their criticality in the process. This is because from the two values, the following table that defines the values of the three gains can then be calculated.

Control Type	K_p	K_i	K_d
P	0.5 K_u	-	-
PI	0.45 K_u	0.54 K_u/ T_u	-
PID	0.6 K_u	1.2 K_u/ T_u	3 K_u T_u/40

Fig. 22. Ziegler-Nichols table for obtaining the parameters of the control action

3) SOFTWARE TOOLS: Currently there are several software that can help ease the process of setting up the PID controller. These software collects all the required data and then a model is created and suggest a certain tuning based on the data. They have a mathematical model that induces an impulse to the system and based on the response, they apply the necessary gains. Most of the companies prefer to use this method instead because it is faster and much more efficient. Some of these software are Matlab, BESTune and Expertune.

E. PID Controller Implementation

Having gained a deeper understanding on how PID control works in a system, our team implemented the concepts via Python codes which are integrated into our ROS architecture.

We used the manual tuning method as discussed in the earlier section by starting with a value of 0.5 for all three gains and gradually tune the values one at a time. We then observed how our ACM worked each time and further tuned the gains by referencing the very useful table which shows the effects of each gain into the system behaviour when they are increased (see Fig. 21), until we are satisfied with the smoothness of the arm tracking movements.

Fig. 23 serves as an example on how we implemented PID control for the left and right movement. As can be observed, we ended up with a very much smaller Integral Gain, Ki value compared to the other 2 because of the tendency of the robotic arm to overshoot and settle down despite having reached the target center frame. Line 7 and 15 shows how our code continuously obtain the distance error signal by taking the difference between target coordinates and the video frame center. Line 8 and 16 is how the 3 errors sum up in our PID closed loop system that serves to control the arm tracking speed. We then do an integer divide by 10 and constrained the speed output within the desirable range to prevent whipping motion which may result in the cellphone being thrown out or even damage the robotic arm with the added load.

All these enabled our robotic arm to move faster in a controlled manner when distance error is bigger but slows down once it nears the target center, achieving our aim for a smooth and accurate tracking trajectory.

F. Why Not Fuzzy Logic Controller

A fuzzy logic controller (FLC) is actually a PID controller that has variable gains. So FLC can be viewed as an approximation of experienced rules that is mostly used to transform neural networks or statistical interpolations into PID tuning. It tends to be more useful in nonlinear processes where we do not fully understand the process or it is just too complicated.

```
1   # PID controller gain variables
2   KP=0.4  #proportional
3   KD=0.1  #derivative
4   KI=0.00001 #integral
5
6   if(center_t[0] - center[0] > 0):##########pid for Left and right
7       error_x = abs(center[0] - center_t[0])
8       speed_x =((error_x*KP) +(prev_error_x*KD) +(sum_error_x*KI))//10
9       speed_x = max(min(10, speed_x), 1)  # make sure speed don't go past max(prevent whipping motion) or below min
10      di_x = 'L'
11      prev_error_x = error_x
12      sum_error_x += error_x
13
14  elif(center_t[0] - center[0] < 0):
15      error_x = abs(center[0] - center_t[0])
16      speed_x =((error_x*KP) +(prev_error_x*KD) +(sum_error_x*KI))//10
17      speed_x = max(min(10, speed_x), 1)  # make sure speed don't go past max(prevent whipping motion) or below min
18      di_x = 'R'
19      prev_error_x = error_x
20      sum_error_x += error_x
21
22  else:
23      di_x = 'R'
24      speed_x = 0
```

Fig. 23. Python code snippet to show PID control implementation

For example if you are dealing with several interacting spring-mass damper systems, a direct mathematical solution might be a bit too complex. This is where fuzzy approximations might be useful, whereby using variable gains is naturally better than fixed ones [26]. For our project, we are just dealing with robotic arm movement task which is a linear and direct process, hence using PID controller is sufficient and proven to be effective in removing excess oscillations.

G. Braccio Retrofitting

In order for the Braccio to transform into CASH, our team need to modify the original configuration by removing the grippers, and replace it with a stand-alone cellphone holder as an end-effector. We managed to source for a holder that is similar in style and colour to the RealSense camera which was also mounted behind on the upper arm. This enabled us to produce an overall aesthetic appearance that is still pleasing to the eye despite being a first prototype (see Fig. 24).

Fig. 24. Side and front view of CASH

VI. BRACCIO HARDWARE DRIVER

As described in the previous section, the PID needs to control two variables, one is direction and the other is speed. This section will discuss how to reproduce PID control in the

real world. Which means how to drive the Braccio robotic arm. First, we will discuss what are the limits of the native driver and then we will introduce the new driver that we proposed called "MoveArm" for Braccio.

A. The Limits of Native Driver

The Braccio has a set of native drivers. The driver provides only two interfaces: one for initializing the Braccio and the other for controlling the angle of each joint (This API is called movejoint() for short). However, only movejoint() is functional, and we ignore the first initialization function, and focus on the movejoint(). The movejoint() has 6 input parameters, the first parameter indicates the time when a action is completed, and the other 6 parameters indicate the angle of each joint. So for this movejoint() interface there are 3 limits.

- inability to set the speed precisely and move continuously.
- the command cannot be stopped after being executed.
- the command cannot be executed consecutively.

B. The MoveArm Driver

The MoveArm driver is designed for the Braccio to control the speed and directions.

1) Components: The MoveArm driver has 3 components, one set of API in host side, one firmware on Arduino board and one UART interface (also called serial port). The set of API is coded in Python where there is one command encoder and one re-transmission function in this code. The command encoder can generate a single command into 5 bytes string, and we defined this simple protocol as follows:

Byte in command	Description
0: Start Flag	Firmware uses this byte to find the start of one command
1: Direction	Firmware uses this byte to select the joint entity
2: Speed	Firmware uses this byte to set the joint speed
3: Reserve	Reserve for future use
4: End Flag	Firmware uses this byte to finish parsing one command

2) Function – Control Direction: We defined six directions for PID, they are left/right, up/down and front/back. In reality, they are not the absolute direction in the coordinate system, but the direction relative to the axis of the Braccio base.

This is how we define the relative direction:

- Left/right is defined as a constant rotation of the BASE to 0 / 180 degrees.
- Up/down is defined as a constant rotation of the ELBOW to 0 / 180 degrees.
- Front/back is defined as a constant rotation of the SHOULDER to 0 / 180 degrees.

So the direction that we defined is not an absolute direction, but a relative one which corresponds to a degree of freedom (DOF). For tracking tasks, controlling the direction on DOF is faster and more efficient than the absolute direction.

3) Function – Save State and Control Speed: The native driver executes commands based on interrupts. Therefore, one command must be executed before the next command can be executed. For the MoveArm driver, the interrupt is only used to save the command, and each command relates to each status. We designed a main loop to process these status. For this driver we have 5 status array for the 5 joints, where each status will be updated in every loop. Due to the speed of UART interface, we set the interval for a loop to 20ms. This means that we can control the speed that starts from 1 step per 20ms, and through this is how we could manage to control the speed precisely.

VII. Conclusion

Integrating all of the above designs, CASH was able to perform all of the functions that our team had proposed. In this part we will discuss the Human Factor Engineering (HFE) considerations in CASH and the future work.

A. HFE in CASH

1) Physical HFE: For the physical HFE, we replaced a stand-alone cellphone holder with a camera mounted just behind and designed an algorithm which allows the cellphone to autonomously face the user and maintain an optimal viewing angle. It means that the user can watch the screen without needing to constantly adjust his head up and down. It allows users to use this function easily and does not cause extra pain or damage to the neck and cervical spine.

2) Cognitive HFE: Cognitive HFE is primarily in our gesture interaction design. We defined 3 different hand gestures to provide the flexibility to switch between the modes according to the user's needs or preferences. The hand gestures with two fingers up indicates mode 2 (hand tracking), three fingers means mode 3 (face tracking) and closed fist represents 0 (freeze). This design makes it easy for users to intuitively remember the gestures required for each mode switch, and quickly respond to the wrong gesture when making an operation.

3) Organizational HFE: When we were developing this design, we found that if CASH is too sensitive the robotic arm will move a lot which will make it difficult for the user to focus on the screen. Conversely if CASH is not sensitive enough, the user will find that the robotic arm cannot perform its primary function of following their gaze. To address this we designed a detection window whereby if the target moves within the range of the detection window, CASH will not move to track the target. But once the target is out of this window, then CASH will start tracking the target and move itself to align the target to the center of camera.

Our hand gesture recognition algorithm still works very well for the other three unused cases of 1, 4 and 5. This can be easily changed in our code to allow the user to change the mode switching according to their own preference.

By designing a suitable width and height of the detection window and applying the PID control method, CASH can track

the user smoothly and quickly with an intuitive mode switch control. This allows CASH to be easily integrated in human's daily life and solve some of our problems, hence meeting the objective that our team had initially set out to achieve.

B. Future work

1) Hardware: For future work, CASH should focus on the miniaturization aspect in order to truly make it functional to be widely adopted.

- The laptop we used now will be replaced with a smaller AI application board. An AI application board like HUAWEI's Atlas board or Intel neural compute stick are possible considerations. Using such kit or board can make the robotic arm system light and portable.
- Replace the Braccio custom robotic arm to a customized robotic arm. A metallic arm can ensure durability and strength and hold most cellphones on the market. Customized robotic arms can be customized based on scenarios, and in the future we can also offer different sizes of the robotic arm to meet different market needs.
- Replace Realsense RGB-D camera to a miniature camera which is lighter, cheaper and smaller. The RGB camera is enough for CASH, we can add a ultrasonic sensor to get the deep information.
- Develop a collision detection function to protect the arm motor, joints and robotic arm itself to ensure acceptable service life.

2) Software: In this aspect for future work, CASH should focus on the scalability in software because we want our robotic arm system to be platform agnostic and can attract other developers to develop new functions.

- Provide full suite logging system to collect the run logs, run errors and provide a debug system for developers to debug.
- Encapsulate all the functions to let CASH become an independent ROS node, then it can be used in every platform with ROS environment.
- Provide full SDK for the secondary development and provide development guide to attract developers or like-minded enthusiasts.

REFERENCES

[1] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system," 2018-05-23. [Online]. Available: https://www.ros.org II-B

[2] "Mediapipe face mesh." [Online]. Available: https://google.github.io/mediapipe/solutions/face_mesh.html III-A

[3] "Dlib library." [Online]. Available: http://dlib.net III-A

[4] "Perspective-n-point." [Online]. Available: https://en.wikipedia.org/wiki/Perspective-n-Point III-C

[5] "Pnp pose computation." [Online]. Available: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html III-C, III-C

[6] "solvepnp." [Online]. Available: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html III-C

[7] "Opencv." [Online]. Available: https://opencv.org III-C

[8] S. Mallick, "Head pose estimation using opencv and dlib," 2016. [Online]. Available: https://learnopencv.com/head-pose-estimation-using-opencv-and-dlib/ III-C

[9] "Appropriate viewing angle." [Online]. Available: https://www.ccohs.ca/oshanswers/ergonomics/office/monitor_positioning.html III-D

[10] S. Yang, P. Premaratne, and P. Vial, "Hand gesture recognition: An overview," in *2013 5th IEEE International Conference on Broadband Network Multimedia Technology*, 2013, pp. 63–69. IV-A

[11] M. Oudah, A. Al-Naji, and J. Chahl, "Hand gesture recognition based on computer vision: A review of techniques," *Journal of Imaging*, vol. 6, no. 8, p. 73, Jul 2020. [Online]. Available: http://dx.doi.org/10.3390/jimaging6080073 IV-A, IV-A

[12] F. F. M. Ma'asum, S. Sulaiman, and A. Saparon, "An overview of hand gestures recognition system techniques," *IOP Conference Series: Materials Science and Engineering*, vol. 99, p. 012012, nov 2015. [Online]. Available: https://doi.org/10.1088/1757-899x/99/1/012012 IV-A

[13] K. B. Shaik, P. Ganesan, V. Kalist, B. Sathish, and J. M. M. Jenitha, "Comparative study of skin color detection and segmentation in hsv and ycbcr color space," *Procedia Computer Science*, vol. 57, pp. 41–48, 2015, 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915018918 IV-A

[14] Y. Li, "Hand gesture recognition using kinect," in *2012 IEEE International Conference on Computer Science and Automation Engineering*, 2012, pp. 196–199. IV-A

[15] V. Kulkarni, "Appearance based recognition of american sign language using gesture segmentation," 2010. IV-A

[16] J. Malik, A. Elhayek, and D. Stricker, "Structure-aware 3d hand pose regression from a single depth image," in *EuroVR*, 2018. IV-A

[17] M. Yasen and S. Jusoh, "A systematic review on hand gesture recognition techniques, challenges and applications," *PeerJ Computer Science*, vol. 5, p. e218, 2019. IV-A

[18] J. Suarez and R. R. Murphy, "Hand gesture recognition with depth images: A review," in *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, 2012, pp. 411–417. IV-A

[19] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann, "Mediapipe hands: On-device real-time hand tracking," 2020. [Online]. Available: https://arxiv.org/abs/2006.10214 IV-B

[20] C. Zimmermann and T. Brox, "Learning to estimate 3d hand pose from single rgb images," arXiv:1705.01389, Tech. Rep., 2017, https://arxiv.org/abs/1705.01389. [Online]. Available: https://lmb.informatik.uni-freiburg.de/projects/hand3d/ IV-B

[21] A. Khan, "Sign language gesture images dataset," Sep 2019. [Online]. Available: https://www.kaggle.com/datasets/ahmedkhanak1995/sign-language-gesture-images-dataset IV-C

[22] Mohamad Moshiri, "Hardware implementation of robotic arm for motion control using infrared communication," 2019. V-B

[23] Chen Dawen, "Robot motion control (pwm and pid)," 2015. [Online]. Available: https://blogs.ntu.edu.sg/scemdp-201415s2-g12/robot_motion_control/ V-B

[24] Jameco Electronics, "How servo motors work." [Online]. Available: https://www.jameco.com/Jameco/workshop/Howitworks/how-servo-motors-work.html/ V-B

[25] Mateo Ubeda Almerich, "Pid control for robotic arm," 2020. V-C3

[26] Luis Prox, "What are pros and cons of using fuzzy logic controller vs pid controller for models, such as a spring mass damper system?" 2018. [Online]. Available: https://www.quora.com/What-are-pros-and-cons-of-using-fuzzy-logic-controller-vs-PID-controller-for-models V-F