Assistive Robot for Carrying Out Household Activities

A. Chan, T. Lu, Z.J. Mohamadi, A. Osmani, E. Young

4th Year Project Final Report

Department of Electronic &
Electrical Engineering

UCL

Supervisor: Chow Yin Lai

24 April 2024

We have read and understood UCL's and the Department's statements and guidelines concerning plagiarism.

We declare that all material described in this report is my own work except where explicitly and individually indicated in the text. This includes ideas described in the text, figures and computer programs.

We declare the use of the generative AI system, Chat-GPT-3.5, to proof-read our final draft.

This report contains 74 pages (excluding this page, references and appendices) and $\sim$ 13100 words. (12000 text + 900 in Figure captions & tables + 200 in Section titles).

Signed: *Edward Young* (Student) Date: 23/04/24

Signed: *Arlend Osmani* (Student) Date: 23/04/24

Signed: *Alex Chan* (Student) Date: 23/04/24

Signed: *Zabih Jan Mohamadi* (Student) Date: 23/04/24

Signed: *Tianchen Lu* (Student) Date: 23/04/24

Assistive Robot for Carrying Out Household Activities

A. Chan, T. Lu, Z.J. Mohamadi, A. Osmani, E. Young

Executive Summary

This report outlines the design, development, and testing of an assistive robot, and highlights the challenges encountered during this process.

We have successfully implemented an object retrieval pipeline, capable of autonomously navigating to an object identified via an object detection model called YOLOv5, picking up the object, and returning it to the user, see Section 5.7. This has been integrated as part of the ROS ecosystem. As part of this, a custom controller was developed for an omnidirectional base, something which didn't exist before, thereby providing a potentially useful contribution to the open source community.

We have successfully implemented means of controlling the robot via intuitive hand gestures, videos of this are available online at [1, 2].

We have successfully implemented a form of voice control for our robot, something that was lacking in existing literature, see Section 1.2.1. The report also highlights the potential of assistive robots aiding the elderly and disabled with simple activities, serving as a compelling proof-of-concept. It shows the feasibility of the robots as solutions to problems billions are facing, helping with tasks that may otherwise be difficult for them.

Future work on improving the implemented kinematics, increasing arm manipulation, utilising a more robust recognition model and providing a more comprehensive wireless implementation would result in a more significant impact, as highlighted in see Section 6.2. Continued development holds promise for aiding the disabled and elderly while also alleviating tasks from the understaffed social care sector.

Contents

1 Introduction and Literature Review

1.1 Introduction

The World Health Organisation (WHO) estimates that by 2030, 1 in 6 people in the world will be aged 60 years or over and will double in number by 2050 [3]. Elderly adults with mobility issues were also shown to have household activity challenges [4]. Robotics have been successful at transforming various industries and processes. One area in which robotics is yet to impact profoundly is assistive care robots, as identified in a recent review paper [5]. With this in mind, the project's main objective is to create a proof-of-concept autonomous, assistive robot for household activities. In making such a robot, areas explored include computer vision, voice recognition, mapping, collision-free navigation, and an application, all via wireless connectivity. This project aims to help people who lack mobility, have reduced eyesight, or, for various other reasons, struggle with performing household activities.

1.2 Literature Review

Before embarking on the project a thorough literature review was undertaken. The primary reason is to identify the gaps in the literature. This step carries increased importance since this is a self-proposed project. This way, we can best prepare ourselves to improve existing work rather than repeating it.

1.2.1 Similar Robots

Some similar robot works incorporating mapping, path planning, and computer vision are Ali *et al* and Luo *et al* [6, 7]. Ali *et al* showcase an autonomous mobile manipulator using ROS, SLAM is implemented using 2-dimensional LiDAR sensor data allowing for mapping and path planning. A depth camera is used to manoeuvre a 5-degrees-of-freedom manipulator and a library is used for camera vision capabilities. Autonomous mobile manipulation in challenging environments was proven with high accuracy. Luo *et al* showcase a ROS-based mobile, an autonomous industrial robot with a manipulator. The SLAM cartographer algorithm is utilised to map, and Dijkstra is used for path planning using LiDAR information. A depth camera is used with an MNSSMB detector for object detection and allows for path-planning manipulation of the arm to pick up the object. The robots, although close in ensemble to our proposed robot, lacked in a few areas due to no utilisation of hand gestures or a holistic model for human interaction. There was also no voice recognition option to control the robot and an 'application' allowing for switching between modes and visualisation. Czygier *et al* [8] created an autonomous, continuously searching robot with object recognition using YOLO. The ROS-based robot capable of travelling in difficult terrain uses a pre-trained YOLO model to identify objects in an area. The area is mapped using LiDAR data, GPS to manoeuvre in unknown areas and SLAM to move in known areas. This robot did not use voice to find specific objects or even scan to find an object nor did it have an arm but intended to in future work. Also, there was no self-trained YOLO model with select objects rather a pre-trained model was used.

1.2.2 Hand Gesture Recognition

Phuong *et al* [9] make use of hand gestures by using a machine learning pipeline, MediaPipe, to control the movement of a SCARA robot. There is the use of the 21 joints recognised in the hand with 2-dimensional recognition of hand gestures when within robot

operating range to open and close the joints via information sent to motors. Allena *et al* [10] show 2-dimensional hand gesture control of a ROS-car for surveillance purposes utilising Mediapipe. The joints in the hand were used to get different actions to occur, the spacing between the end joints of the thumb and index finger was used for speed of movement, the tilt of the hand for directional steering and the back of the hand for reversing. Wameed *et al* [11] also skipped the complexity of training a model for a convolutional neural network and used MediaPipe for hand gestures to control a 'tank robot'. 2-dimensional recognition of hand gestures was used for the robot movement and showed high accuracy results of 97%. Both papers showed a lack of 3-dimensional hand gesture recognition, restricting the accuracy of the number of hand gestures that can be done and natural hand positions that can be used. Also, there was no dual-hand recognition for multiple-part control such as a robotic arm. The past examples have shown control of robot base movements but not of a manipulator using hand gestures. Chatterjee [12] implemented a powerchair for healthcare purposes to have robotic arm control and powerchair movement. There is usage of embedded transmitters on gloves on each hand, each controlling the movement of the chair or the arm. Small hand movements were used to facilitate the lack of movement in users' hands.

1.2.3 Holistic Model

Shaif *et al* [13] worked on a humanoid robot with vision and voice recognition. A holistic model was utilised for human body pose recognition allowing for different pose recognition and potential actions to occur with the humanoid robots. There was also voice recognition and hand gesture recognition which allowed for different commands of motion to occur like 'forward' and gestures to be recognised to do the same. The hand gesture recognition used 3-dimensional hand gesture recognition allowing for more hand gestures to be recognised and more natural hand positioning that might be hard with 2-dimensional recognition. One gap that was not addressed was using 2 hands in hand gesture recognition which would allow for control of more things. The holistic model implementation did not allow for the robot to follow the human, based on the distance from the camera. The voice recognition was also limited to simple movement commands.

1.2.4 Voice Recognition

Voice recognition can be used in assistive robots to move to different locations within a mapped area as shown by Sharan *et al* [14]. 'ROSWITHA' and 'Volksbot' are used via voice recognition by mapping done with SLAM to get an RTAB-MAP whereby voice commands allow the robot to go to different routes. An automatic speech recognition module is used to understand the spoken commands for them to be done. A gap here is found as voice is not used to do other things like mapping and navigating a menu of other potential modes that may be present and rather just for movement to locations. Djinko *et al* [15] have shown voice recognition for video-based object detection using YOLOv7. Google text-to-speech is used to command the robot to find an object and the YOLO module goes through past videos recorded where the object's bounding box was present, and the object location was realised. This was only done with previous videos and not with live feed, so there was no real-time finding of an object which would help.

1.2.5 Novelty

From the gaps identified in the literature, there are some points of novelty with our robot. The first point of novelty is the use of 3-dimensional hand gestures particularly both hands allowing us to control the base of the robot and the arm simultaneously. Another point is the hand gesture to use pixels on the screen to allow for other modes to activate such as opening the gripper. The novelty in using a holistic model to perform acts such as having the robot follow of which there was a lack in the literature [14]. Also, the voice recognition to operate in different modes like grabbing an object and navigating through modes will be new. Another point of novelty is using our own trained dataset for the object recognition using YOLO which was not done before. The last point of novelty is bringing all of this together, Ali *et al* and Luo *et al* [6, 7] have some parts working but missing other parts such as voice.

2 Goals and Objectives

The goals and objectives of the project have been altered from those set out in the project proposal. The updated goals and objectives are more reflective of the work achieved by the group.

1. Create a robot that can perform basic tasks using Arduino code and robotics algorithm for picking/placing.
    (a) Implement forward, inverse kinematics, and arm-path planning.
    (b) Re-design and program the base of the robot.
        i. Create brackets to accommodate the robot arm.
        ii. Complete the circuitry and program algorithm necessary for controlling wheels driving the robot.
    (c) Allow for wireless connectivity for commands between the robot and computer.
2. Integrate Computer Vision framework for object and gesture recognition.
    (a) Apply YOLO algorithm (Object Recognition) on our own dataset to compute training and testing. (Achieve mAP 80% on 10+fps).
    (b) From Mediapipe algorithm, create, design and define control algorithms for the robot (Hand-Gesture and Holistic control)
3. Integrate LiDAR for SLAM & create a labelled map in UI/app.
    (a) Install LiDAR sensor on the robot, get data from the sensor and use particle filter SLAM for mapping.
    (b) Create user interface/application.
    (c) Integrate framework that allows for robot to label and add locations of objects detected via computer vision to this map.
4. Integrate voice commands to control robot.
    (a) Used existing recognition models for speech commands.
        i. Enable voice commands to control the Object and Gesture recognition subsystem.
    (b) Program and create tasks for robot when trigger words and phrases such as 'Manual' & 'Grab the orange' are communicated.

3 Background Theory

In this section, we provide an overview of the techniques used in our implementation.

3.1 MediaPipe

MediaPipe is an open-source framework developed by Google for anyone to build upon and integrate machine learning models into their own purposes.

3.1.1 Hands model

The MediaPipe Hand-Landmarker model is a model that can detect hand landmarks such as joints in images and world coordinates of hands in the visual frame. These landmarks can be seen in Figure 1.

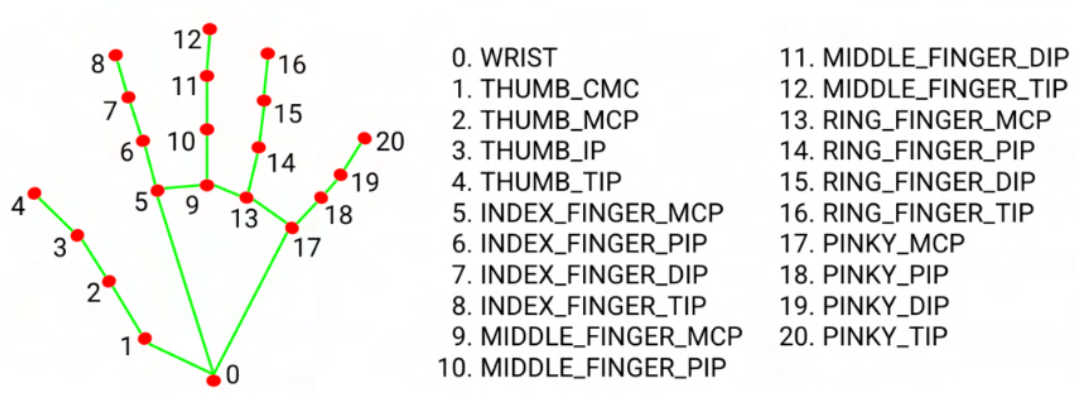


Figure 1: Hand Landmarks

3.1.2 Holistic model

The MediaPipe Holistic model is a model that can detect pose, face and hand landmarks in the image and world coordinates to create whole-body analysis in the given visual frame.

Figure 2: Hollistic model [16]

3.2 Object Recognition

With technology advancing rapidly, object recognition has become a critical component in the realm of artificial intelligence and robotics, often being used in applications such as medical imaging, facial recognition or even video tracking [17]. When an object recognition model is selected, several factors must be considered, such as training/validation/testing dataset partitions, losses, precision/recall curve, number of epochs, number of images, etc. All of these factors were considered when selecting the object recognition model for this project. The model that was chosen was YOLOv5, and the reason for this selection can be seen in Section 4.2.

Dataset partitions: Datasets should be divided into three main sets: training, validation, and testing. The training set is used to train the model, the validation set is used to monitor the model's performance and avoid overfitting, and the testing set is used to check the performance of the model on new data [18]. Depending on the size of the dataset and complexity of the model, the splits for the three sets can be altered, however, the common splits are 80/10/10, 70/15/15 or 60/20/20 [19].

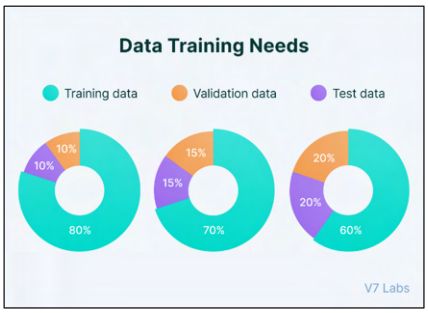
Losses: During training and validation, various types of losses are monitored, as these will determine the model's ability to correctly label objects and distinguish them from the background. Minimising these losses is necessary for improving the performance of the model, however, minimising these losses too much could result in overfitting [20].

Precision/Recall Curve: These two parameters are monitored to evaluate the model's ability to predict all true positives (precision) whilst also being able to detect all relevant instances (recall) [21]. An example of a precision/recall curve can be seen in Figure 3b. Ideally, a balance between precision and recall is necessary, as a large area under the curve represents high values for both. This is desirable as it shows the model's ability to correctly identify objects without a high rate of false positives [22].

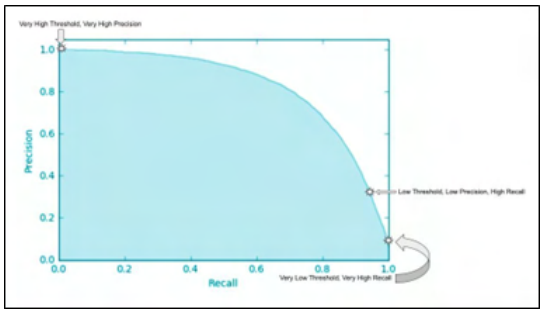
Precision and recall are calculated using the following formulas:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{1}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2}$$



(a) Dataset Partitions [19]



(b) Example Precision/Recall Curve [22]

Figure 3: Model Parameters

Epochs: An epoch is defined as one complete pass through a dataset, therefore deciding the number of epochs that will be used during training is of great importance. Too few epochs would lead to underfitting, thus the model fails to learn the data sufficiently, whereas too many epochs would lead to overfitting, thus the model has learned unnecessary details for generalisation of objects [23].

Number of images: The greater the number of images, the greater the accuracy and robustness of the model. When selecting the images that would be used in training, it is necessary to diversify the images, for example, taking the images in different lighting, orientations, and backgrounds. This helps the model to generalise objects in real-life scenarios.

All of these factors were considered when selecting the object recognition model for this project. The model that was selected was YOLOv5, and the reason for this selection can be seen in Section 4.2.

3.3 SpeechRecognition

Hand gesture recognition was built using Python. As a result, we believed it sensible to also implement the voice control using Python. In this way, it would be simpler to fuse the two functionalities. To this end, the Python library, 'SpeechRecognition' was used [24]. As part of the SpeechRecognition package, we are using Google's speech-to-text API for text-to-speech translation.

3.4 Robot Arm

The robot arm we are using was used in Year 3 Robotics module. It is a 3-dof robot arm, as shown in Figure 4. Joint 1 controls the rotation of the black cylinder at the base of the arm. Joint 2 controls the yellow arm, and Joint 3 controls the blue arm. The range of Joint 1 and Joint 3 is from 0 - 180°. Joint 2 has a range of 0 - 130°. Joint 4 (the gripper) is also controlled by a servo motor. The gripper is open when Joint 4 is at 160° and closed when Joint 4 is at 100°.

Figure 4: Robot Arm

3.5 Robot Base

We are using the same robot base as the one used in the maze solver scenario. Figure 5 outlines the wheel definitions of our robot. In addition to the wheel definitions, we can also see the mounting holes of the LiDAR sensor. The mounting of the LiDAR serves as the reference point for our robots direction of travel.

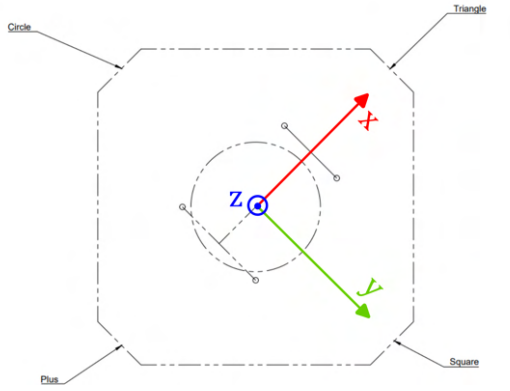


Figure 5: Robot Base Layout, the annotated frame of reference is that of the LiDAR (base_link), x direction is forwards/backwards.

3.6 ROS2

Robot Operating System 2 (ROS2) is a set of libraries and tools that allow for faster development of robotics applications. More complex robots, such as ours, often have multiple subsystems. The benefit of using ROS2 is that it allows for the discretization of a complex robot.

3.6.1 Nodes

ROS2 separates a robot's subsystems into 'nodes'; often one subsystem consists of multiple ROS2 nodes. In doing so we can create a layer of abstraction between the different subsystems of our robot. This allows for easier debugging and, critically, a more accomplished implementation. ROS2 nodes publish information, known as 'messages', to locations, known as 'topics', other nodes can then subscribe to these topics to receive the information published. A graph of nodal interconnectivity can be seen in Figure 6.

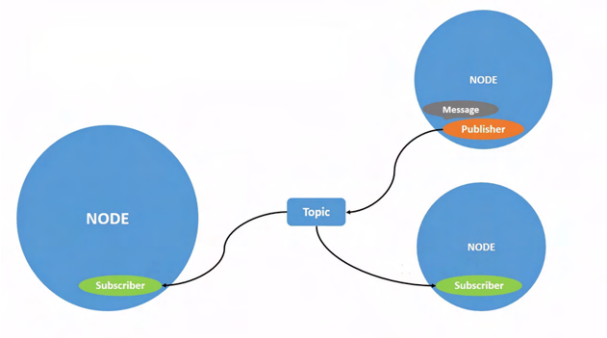


Figure 6: ROS2 Node Interconnectivity (Reproduced from [25])

3.6.1.1 ROS2 Control

One such ROS2 node is the ROS2 Control Node. A primary feature of ROS2 control is that it simplifies the process of integrating new hardware into an existing robot. Although this feature was not utilised by us it means that should we wish to make changes in the future, this process will be simpler. ROS2 control achieves this through abstraction of the hardware and control framework [26]. ROS2 Control consists of 3 parts, the controller, the controller manager and the hardware interface, as seen in Figure 7.

The role of the controller is to translate body velocities into wheel velocities. Body velocities can be provided by a person, commonly via a keyboard/gamepad or a navigation stack, the official navigation stack of ROS2 is 'Nav2' [27]. Controllers are specific to the wheel layout of the robot, this is because different layouts have different kinematics. There exist several controllers for commonly occurring wheel layouts, differential drive, Ackermann and bicycle [28, 29, 30]. There does not exist, however, a controller for an omnidirectional base, meaning one had to be written in line with the ROS2 specifications for custom controllers [31]. The controller interface takes the form of a ROS2 package, a format that allows for it to be shared with other ROS2 users [32].

The controller manager manages the lifecycle of controllers, and their access to the hardware interfaces and offers services to the ROS-world [33]. It is not specific to any controller or hardware interface and therefore needs no alterations for our use.

The hardware interface is how we describe our hardware to ROS2 Control. To tell ROS2 Control the physical constructs of our robot we provide it with a 'URDF' file [34]. This URDF file specifies the geometry and organization of robots in ROS2. An example of something included in a URDF file is the type and Cartesian position of all robot joints. Another aspect of the hardware interface is that it is used to define the command interface and state interfaces of the robot joints. These two variables describe the current state of the joint and the next command of the joint. For our robot, this is simply the angular velocity of the joint at the previous time step and the angular velocity of the joint at the current time step respectively. Another key job of the hardware interface is to convert the angular wheel velocities from the controller into values that can be passed to actual hardware. Commonly this involves a conversion of the angular wheel velocity into a PWM value. The hardware interface also then handles passing this converted value to the hardware. Commonly this is done via serial communication. Hardware interfaces are unique to all robots, as a result, it was required that one be written. The hardware interface also takes the form of a ROS2 package.

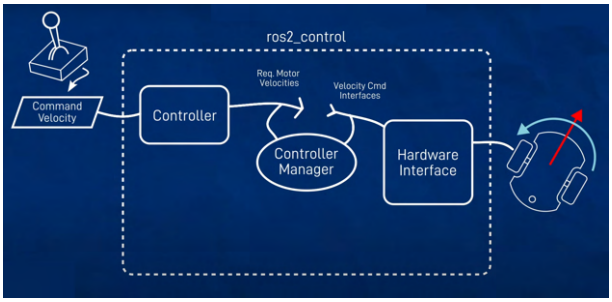
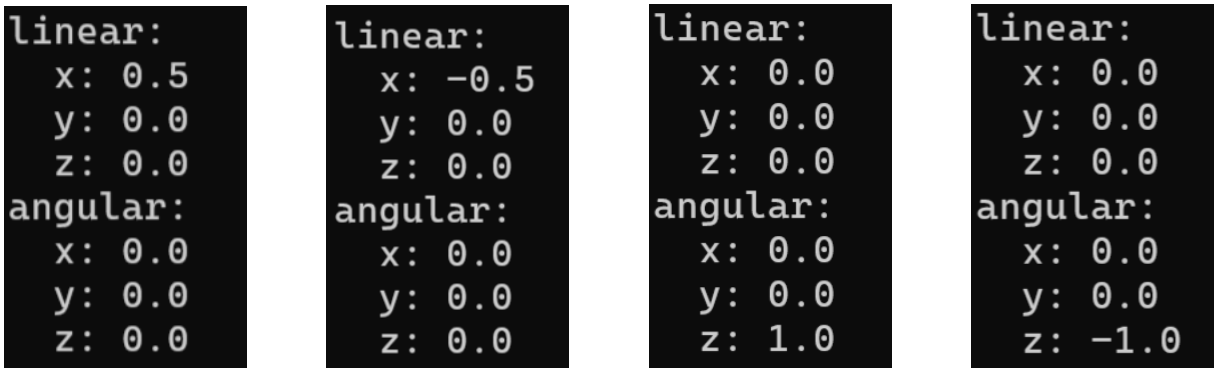


Figure 7: ROS2 Control Node (Derived from [35])

3.6.1.2 Teleop Twist Keyboard

Another example of a ROS2 node is Teleop Twist Keyboard [36]. This node takes keyboard inputs from a user and outputs the corresponding velocity. The 9 accepted keystrokes can be seen in Figure 9. The output body velocity is published as a message on the topic 'cmd_vel', short for command velocity. The aforementioned custom controller subscribes to this topic, and as such receives the desired body velocity. The body velocity outputted by the Teleop Twist Keyboard contains 6 values. Our robot can move forwards/backwards, left/right, and about its centre. This corresponds to velocities of linear x, linear y and angular z. Taking another look at Figure 9 after reading Section 4.9.1 will provide an insight as to how pressing each key will move the robot.



(a) Pressing 'i' **(b)** Pressing ',' **(c)** Pressing 'j' **(d)** Pressing 'l'

Figure 8: Body Velocities for different keyboard inputs

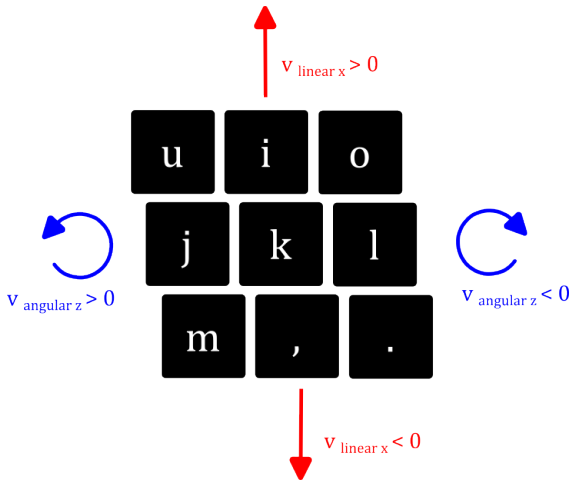
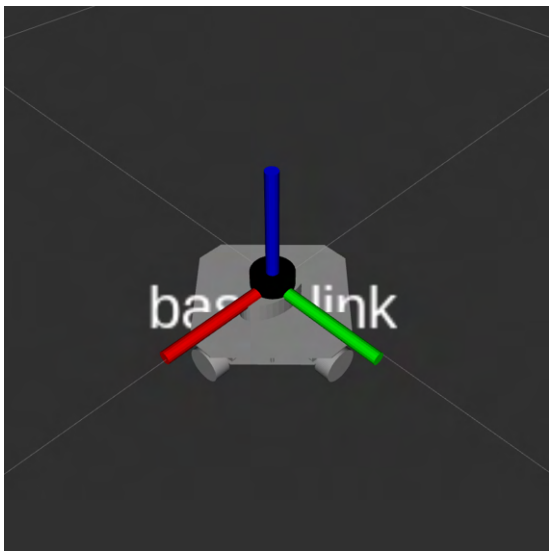


Figure 9: Accepted keystrokes

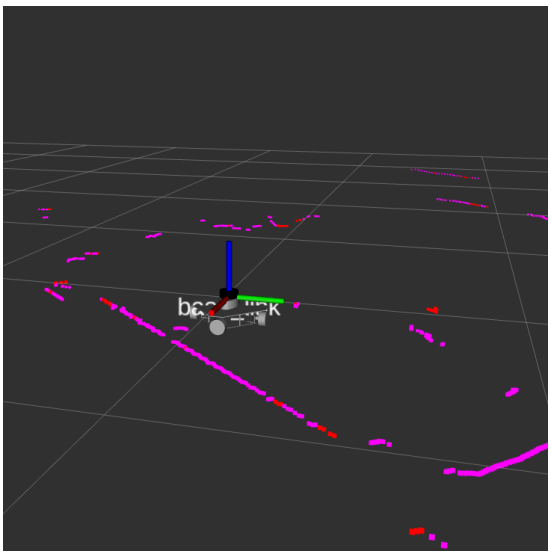
3.6.2 RViz2

RViz2 is a 3D visualizer for the ROS2 framework. We are using it to provide a visualisation of our robot to the user. In this way, it is clear how the robot is interacting with its environment. Utilising RViz2 is essential to implementing SLAM and Navigation in ROS2. A non-exhaustive list of visualisations provided by RViz2 is as follows:

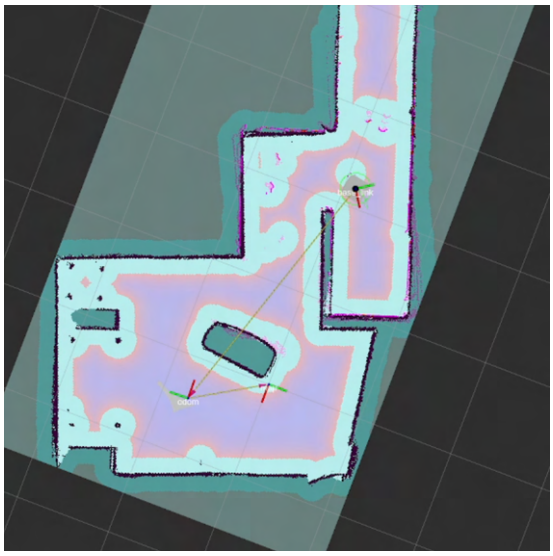
- Robot Model, TFs, Axes
- Received Laser Scans
- SLAM Maps, Navigational Cost Maps, Planned Paths



(a) Robot Model



(b) Laser Scan Visualisation



(c) Cost Map Visualisation

Figure 10: RViz2 Overview

3.6.3 TF Tree

A TF (transform) tree defines the relationships between coordinate frames in a robotic system. Transformations between the separate coordinate frames can either be static or variable. In the transform tree below the base_link → base_footprint is static. This is because the LiDAR (base_link) sits on top of the robot (base_footprint) and does not move. Variable transforms are provided by various robot subsystems. The map → odom transform is provided by a SLAM system, intuitively this makes sense as we know the L in SLAM stands for localisation - the SLAM system localises the robot inside of the global map frame. The odom → base_link transform is provided by an odometry subsystem i.e. wheel encoders or an imu. Wheel encoders-based odometry systems are prone to drift over time, this is commonly due to imperfect contact with the floor, causing wheel slippage. Other implementations use canonical scan matcher. It has advantages over a typical encoder-based wheel odometry system in that it is not susceptible to drift over time. The following is the required transform tree for our robot.

$$map \rightarrow odom \rightarrow base\_link \rightarrow base\_footprint$$

The listed coordinate frames are defined as follows [37]:

- map: a fixed world frame, pose of a mobile platform, relative to the map frame, should not significantly drift over time.
- odom: a fixed world frame, the pose of a mobile platform in the odom frame can drift over time.
- base_link: rigidly attached to the mobile robot base, for our robot this is the frame of the LiDAR sensor
- base_footprint: rigidly attached to the mobile robot base, for our robot this is located directly below base_link frame ($\Delta z =$ 10cm)

3.6.4 Distributed Computing

ROS2 is designed with distributed computing in mind. A ROS2 system can comprise dozens, even hundreds of nodes, spread across multiple machines. Depending on how the system is configured, any node may need to communicate with any other node, at any time [38, 39].

A schematic showing how we are using the distributed nature of ROS2 in our implementation can be seen in Figure 11. Common implementations use a base station, where tasks requiring high processing power are performed, and a Remote PC, which handles the sending of information to the robot wheels. We have adopted this approach.

ROS2 nodes can communicate with each other as long as they are on the same subnet mask of the same WiFi network. The environment variable 'ROS_DOMAIN_ID' exists to segregate up this subnet mask of the WiFi network. By setting the same 'ROS_DOMAIN_ID' on both the Remote PC (Pi) and our Host Machine we can ensure that ROS2 messages can be passed between the two with no interference from other users on the network.

Figure 11: How we are using the distributed nature of ROS2 - laser scans are registered by a ROS Node on Raspberry Pi, these are published as a message on the topic '/scan', this is used on the Host PC for SLAM. Messages about robot velocity are published by the Nav2 node or Teleop Twist Keyboard on the '/cmd_vel' topic, this is used by the controller on the Raspberry Pi to work out which wheels to move.

3.7 LiDAR

LiDAR (Light detection and ranging) sensors can be categorized into various schemes two of the most common classifications are pulsed time of flight (TOF) and triangulation [40]. In this project we are using the RPLiDAR A1, This is a triangulation-based LiDAR sensor [41], the working principles of which are as follows:

- An optical pulse it emitted from the laser
- The reflected pulse is detected at the sensor
- The position of the light detected at the sensor array is used to calculate the distance of the object it reflected off of.

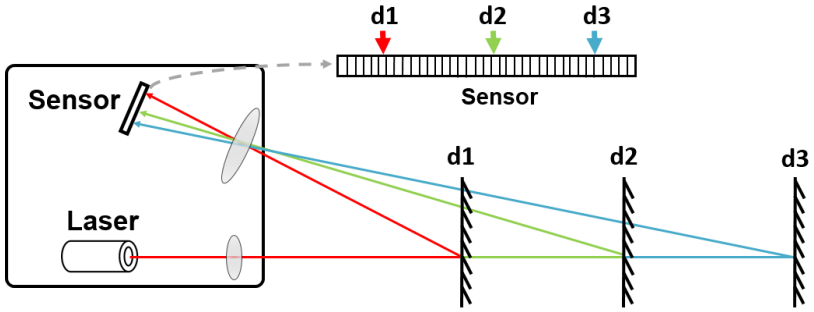


Figure 12: Illustration of Triangulation LiDAR (Reproduced from [40])

4 Experimental Methods

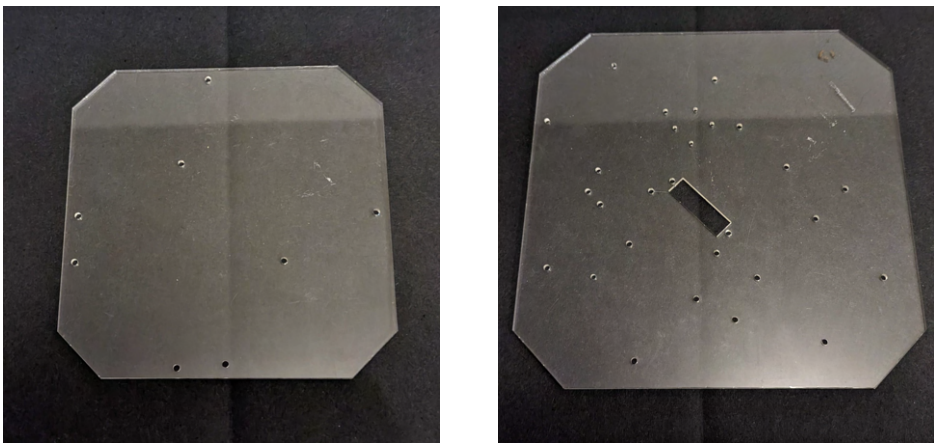
In this section, we explain how we used the techniques, outlined in Section 3, to solve our problem statement.

4.1 Robot Redesign

At the beginning of the project, we were provided with a robot base and robot arm. This robot base was too small to accommodate our robot's many components, and as a result, we needed to redesign the robot base.

4.1.1 Base Plate

When redesigning the base plate, we first considered the overall layout of the robot, since one of the most important components, the LiDAR, requires no obstacles whilst scanning the room. Consequently, the LiDAR is placed on the bottom level of the robot, making sure that no other components occlude its view of the x, y plane in which the LiDAR laser sits. A second layer is created to accommodate the robot arm. After the overall layout of the robot is settled, we expand the base plate from the original plate on the robot. Then, a CAD file was generated from Fusion 360 with all mounting points for each component. Finally, two 3mm acrylic base plates were made by the laser cutter from the workshop. Figure 13 shows this.



(a) Bottom Plate **(b)** Top Plate

Figure 13: Robot Plates

4.1.2 Micro-controllers and Wiring

4.1.2.1 Arduino Mega

Our robot is required to move in the x and y directions and rotate z. We need 4 independent control signals for this to be possible. The Adafruit motor shield v2, shown in Figure 14, gives us this functionality. In addition, it has additional soldering pads for further modifications. We utilised these soldering pads to allow for both the arm and base to be controlled via the same Arduino. To do this we needed to solder 8 additional pins (4 5V pins and 4 GND pins) - a pair for each of the servos in the arm. These new pins were connected up to the motor shield existing 5V and GND pins as shown in Figure 14, the red wire connects the pin headers to the Adafruit's 5V and the black wire connects them to its GND. Figure 14 also shows the wire connection of the 4 motors.

4.1.2.2 Adafruit Motor Shield

As mentioned in the previous section, we decided to use the Adafruit motor shield, which occupies all digital pins from 0 to 12 on Arduino UNO, and we also need four extra digital pins to control the arm. As a result, we decided to use the Arduino Mega which has more digital pins. As Figure 14 shows digital pins from 13-16 are connected to signal pins of the robot arm through male-to-male jumper wires.

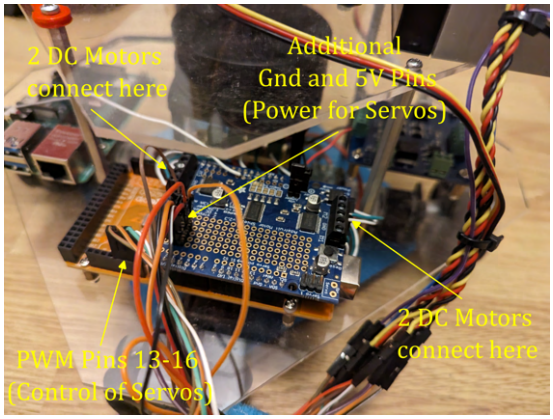


Figure 14: Adafruit Motor Shield

4.1.3 Serial Communication with Arduino

As our main algorithm is written in Python and the robot base requires Arduino, we decided to use serial communication to connect our Python code with Arduino. As shown in Figure 15, through serial, a set of signals is sent to the Arduino mega. The first integer indicates whether the signal is controlling the base or the arm.

```
void loop()
{
  if (Serial.available()>0)
  {
   String message = Serial.readStringUntil('\n'); // Read the message until a newline character is received
    // Split the message into individual parts
    if (message.length() <= 20)
    {
      int parts = sscanf(message.c_str(), "%d %d %d %d %d %d", &mode, &Joint1Angle,  &Joint2Angle,  &Joint3Angle, &Joint4Angle, &Speed);
      if (mode==0)
      {
        Joint1.write(Joint1Angle+Joint1Offset);
        Joint2.write(Joint2Angle+Joint2Offset);
        Joint3.write(Joint3Angle+Joint3Offset);
        Gripper.write(Joint4Angle);
      }


     if (mode==1){
       Circle_Dir=Joint1Angle;
       Square_Dir=Joint2Angle;
       Plus_Dir=Joint3Angle;
       Triangle_Dir=Joint4Angle;
       Speed=Speed;
```

Figure 15: Arduino code for serial communication

4.1.4 3D Design

In addition to the base plate micro-controller modifications, we 3D printed a few components for our robot.

4.1.4.1 Gripper Extension

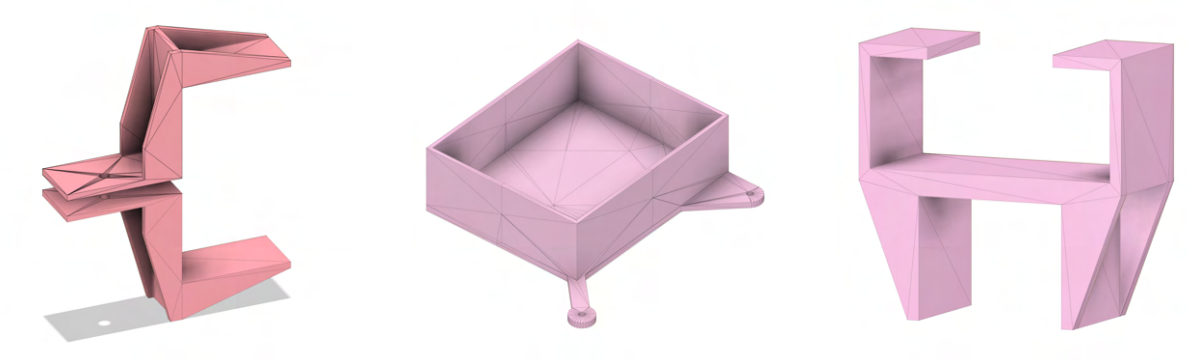
The gripper that comes with the robot arm is designed for grabbing small objects. When we were testing the pick-up object mode, instead of grabbing the object, the gripper sometimes squeezed and pushed away the object. Therefore, it is crucial to design a gripper extension for our robot arm. The gripper extension on each side is 2 centimetres, as shown in Figure 16a.

4.1.4.2 Object Holder

During testing, we found that the battery drains quickly when the robot finishes the pick-up task and returns to its original position. Besides, the gripper cannot hold the object hold the object long enough. Consequently, we designed an object holder to hold the object after our robot completed the pick-up task. This can be seen in Figure 16b.

4.1.4.3 Camera Holder

The camera we used for our robot is a Logitech webcam, which has a flexible joint. As we are going to mention in Section 4.7, our camera is placed on the top of the robot arm, and the approach angle and joint angles are determined by the centre and the bounding box of the object. As a result, a camera holder is needed for aligning the camera with the robot arm. Figure 16c shows this.



(a) Gripper Extension **(b)** Object Holder **(c)** Camera Holder

Figure 16: 3D Models

4.2 Object Detection (YOLOv5)

To produce an effective and efficient object detector model, that would allow for various household objects to be picked and placed in different locations, a suitable model architecture was necessary. For this project, the model architecture that was selected was YOLOv5 due to its high performance in both speed and accuracy, which are critical for real-time object detection tasks. When comparing YOLOv5 to other object detection models such as EfficientDet, there are clear advantages to using this model as seen in the figure below:

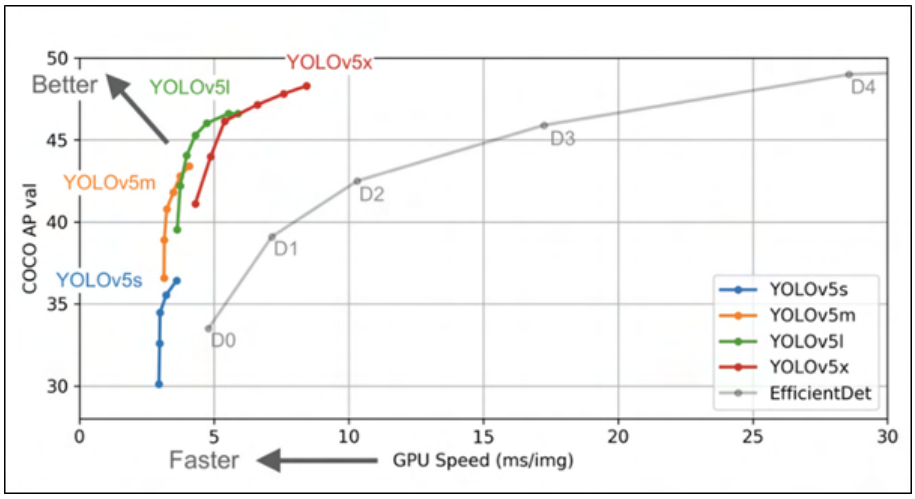


Figure 17: YOLOv5 and EfficientDet comparison [42]

Figure 17 shows that YOLOv5 has four variations; YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large) and YOLOv5x (extra-large). Depending on the variation chosen, there is a trade-off between speed and accuracy, with YOLOv5x having the most accuracy whilst having the longest training speed, and the opposite being true for YOLOv5s. It can also be seen that in terms of training speed, all variants of YOLOv5 far surpass the training speeds of EfficientDet. Furthermore, even though the training speeds of YOLOv5 are much faster, the most accurate YOLOv5 model has a similar degree of accuracy compared to the EfficinetDet D4 model [42]. Due to these advantages, YOLOv5 was selected.

4.2.1 Dataset Preparation

As the eventual goal for this project is to be able to pick and place various household objects, a custom dataset must be produced as the pre-existing YOLOv5 models contain classes that do not align with our unique requirements. For this project, five common yet specific household objects were used for the training of the model: an orange, a fork, a phone, a can, and a mouse. Our dataset comprised 500 distinct images focusing on the five classes, with each object being photographed 100 times under various conditions to ensure that the model would be able to generalise well across different scenarios. These conditions included changes in background, lighting, and orientation to simulate changes in the environment when using object detection in real-life scenarios. Having taken the photographs, it was then necessary to annotate each image, outlining the precise boundaries of each object. Examples of the annotations can be seen in Figure 18.

(a) Image of Can **(b)** Can Annotated

(c) Image of Mouse **(d)** Mouse Annotated

Figure 18: Training Images

To ensure the accuracy of the model, these annotations must have precise, edge-to-edge boundaries as the YOLOv5 model relies on these annotations to characterise each object. Once each image has been annotated, a corresponding bounding box and label is assigned to that specific image. For example, for Figure 18b, the corresponding label was:

"3 0.497000 0.448125 0.442000 0.533750"

Where:
'3': class ID of the object.
'0.497000': normalised x-coordinate of the centre of the bounding box.
'0.448125': normalised y-coordinate of the centre of the bounding box.
'0.442000': normalised width of the bounding box.
'0.533750': normalised height of the bounding box.

These values are used by the YOLOv5 software to learn where objects are located within images and what those objects represent.

4.2.2 Software and Training Process

The training for the object detection model was conducted on Python, and three libraries were used to produce the code needed for the training: Pytorch, Numpy and OpenCV. Furthermore, the dataset was partitioned into three sets: training, validation, and testing, in a ratio of 70:15:15. Having completed the prerequisites, the training could now begin using the Python code below:

```
!cd yolov5 && python train.py --img 320 --batch 16 --epochs 400 --data dataset.yml --weights yolov5s.pt --workers 8
```

Figure 19: Training Python Code

Where:
'python train.py': Executes the training script.
'- - img 320': Sets the image size to 320x320 pixels.
'- - batch 16': Specifies the batch size as 16.
'- - epochs 400': Defines the number of epochs as 400.
'- - data dataset.yml': Points to the yml file that contains the paths to the training, validation, and testing datasets, as well as class definitions.
'- - weights yolov5s.pt': Initialises the training using the pre-trained weights of YOLOv5s.
'- - workers 8':This allocates eight worker threads to load the data.

Once the training has concluded for the custom dataset, it could be loaded and evaluated using the following Python code:

```
model = torch.hub.load('ultralytics/yolov5', 'custom', path='yolov5/runs/train/exp5/weights/best.pt', force_reload=True)
```

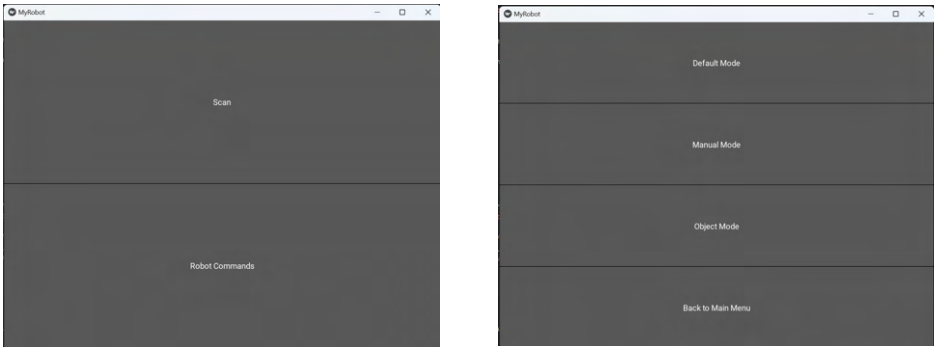
Figure 20: Loading YOLOv5 Model

Each training run results in a corresponding file called 'exp (number)' which contains a 'results' and 'weights' folder. The 'weights' folder contains two files that refer to the weights of the 'last' and 'best' performing models. For a more efficient and accurate model, the 'best' weight is used when loading the model. During the training process, the model is saved periodically, and after each save, the performance is evaluated. If the performance of the model is better than the previous models, in terms of loss and accuracy, the current model is saved as the 'best' model. Therefore, once the 400 epochs have been completed, the 'best' file within the weights folder corresponds to the most accurate model. The performance of the model we have used in this project can be seen in Section 5.4.

4.3 Application and UI

4.3.1 Initial approach with module

Initially, we tried using a Python library called Kivy because it allows us to build user interfaces on multiple platforms. As shown in Figure 24, the front menu has two options for users: Scan, which controls the robot to scan the room and generate a map, and robot commands, which asks users to select between different modes. However, both Kivy and OpenCV have their own event loop, and they interfere with each other, causing freezing and unresponsiveness in the user interface. As a result, we abandoned this method.

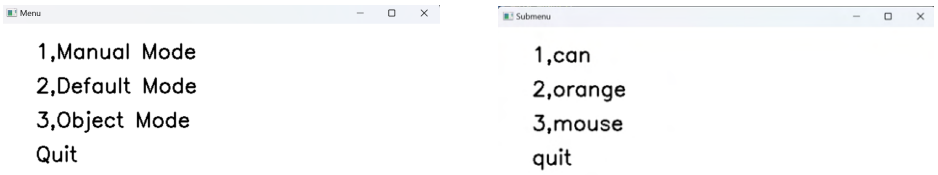


(a) Front Menu **(b)** Robot Menu

Figure 21: UI using Kivy

4.3.2 CV approach and implementation

Because we were unable to display video using KIVY, we tried using OpenCV(cv2) for our user interface development. While the OpenCV library is known for its computer vision application, it can also be used to create simple user interfaces. When utilizing OpenCV for UI development, the cornerstone is the function 'imshow()', which allows us to display images, video frames as well as words. First of all, we created a window using 'imshow()' function with several options. Then we recorded the keyboard input from our user. Once the correct input is received, the previous window will be destroyed, and another widow will appear with options for the next menu. As shown in Figure 22, our user interface has two layers. The first layer of the menu lets users select between different modes (mentioned in Section 4.7). The second menu appears when the user has selected the mode 'Object'.



(a) Front Menu **(b)** Object Menu

Figure 22: UI using OpenCV

4.4 Voice implementation

Below we describe the approach taken to solving the problem of voice control.

4.4.1 Transcribe_microphone function

As mentioned in Section 3 we are using the SpeechRecognition library to turn speech into text. Therefore, the following describes the process of extracting the command from transcribed speech.

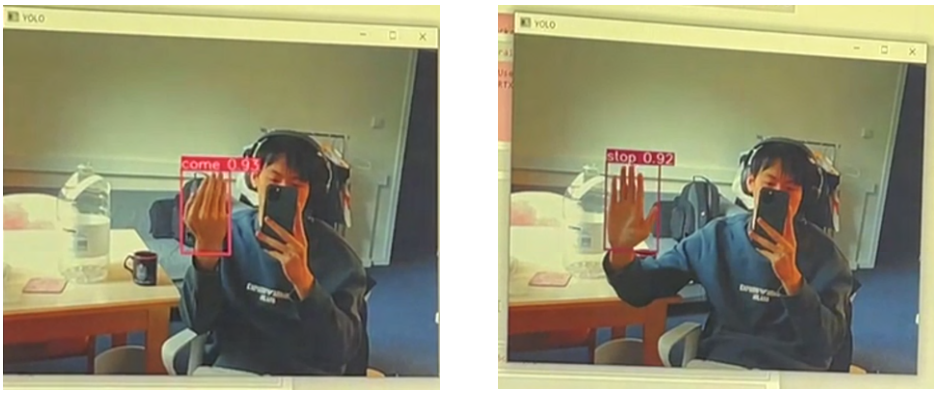
Figure 23: Speech to command function

4.5 Hand Gesture Control

Below we describe the approach taken to solving the problem of hand gesture implementation.

4.5.1 Initial YOLO approach

Initially, we had decided to use a custom YOLO model for object detection. Two classes were created to test our model's capability. These were 'come' and 'stop', seen in Figure 24. The steps followed in creating this model follow those outlined in Section 4.2.



(a) Come **(b)** Stop

Figure 24: Initial Hand Gesture Commands

Due to the limited control capabilities of using YOLO, other approaches were explored.

4.5.2 Final MediaPipe approach

MediaPipe packages have pre-trained models which can identify hands and locate joints in the 2D screen-space, the information provided by these models can then be utilised to develop more sophisticated custom controls for the robot.

4.5.3 Hands model

To aid explanation and understanding, Figure 1, outlining hand landmarks, has been reproduced below.

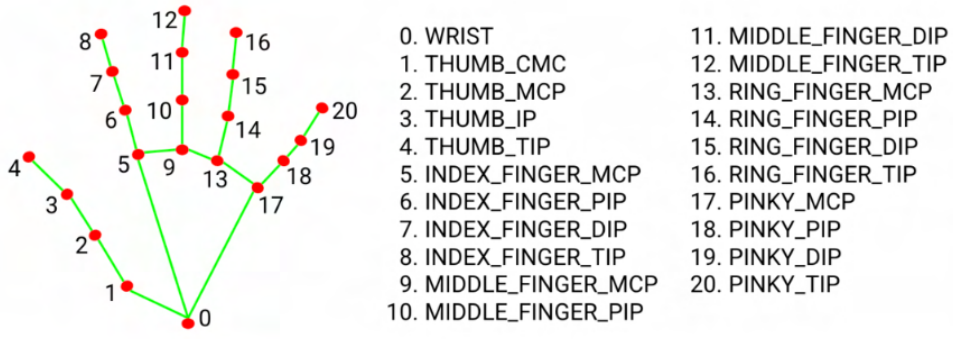


Figure 25: Hand Landmarks

This MediaPipe hands model identifies hands in the frame, but to do further processing, it is essential to identify and label left and right hands. The official labelling function is

known to have issues differentiating between the left and right hands [1]. The function to label the left/right hand 'get label', is obtained from a Youtube comment, and was used instead, see Appendix 9.7.

4.5.4 Joint Creation and Definitions

The first robot joint is programmed by using data from finger joints 5, 0 and 17 in Figure 1. An easy way to imagine this is to think of the hand position before giving another person a high-five, with fingers pointing to the sky, wrist up-right and palm facing the person to receive the high-five. Changing the value of robot Joint 1 can be imagined by rotating the wrist from facing the person, to facing yourself. This movement gives the range from 180° to 0° of robot Joint 1.

From the left drawing of Figure 26, x value of landmark 5 $<$ x value of landmark 17 ($a[0] < c[0]$), also, z value of landmark 5 $>$ z value of landmark 17 ($a[2] > c[2]$). These conditions meant the palm was facing the camera.

To measure the angle, two points are created. Firstly, the mid-point between landmark 5 and 17 is created as M, Secondly, point J is at the same xy plane as landmark 5 but parallel with M on the z axis. From this point, ∠JM5 can be calculated using $\arctan(a[0] - J[0])/(J[2] - M[2])$.

Then, ∠JM5 is added to or subtracted from 90 depending on whether the palm faces the person or the camera to create an angle of joint 1.

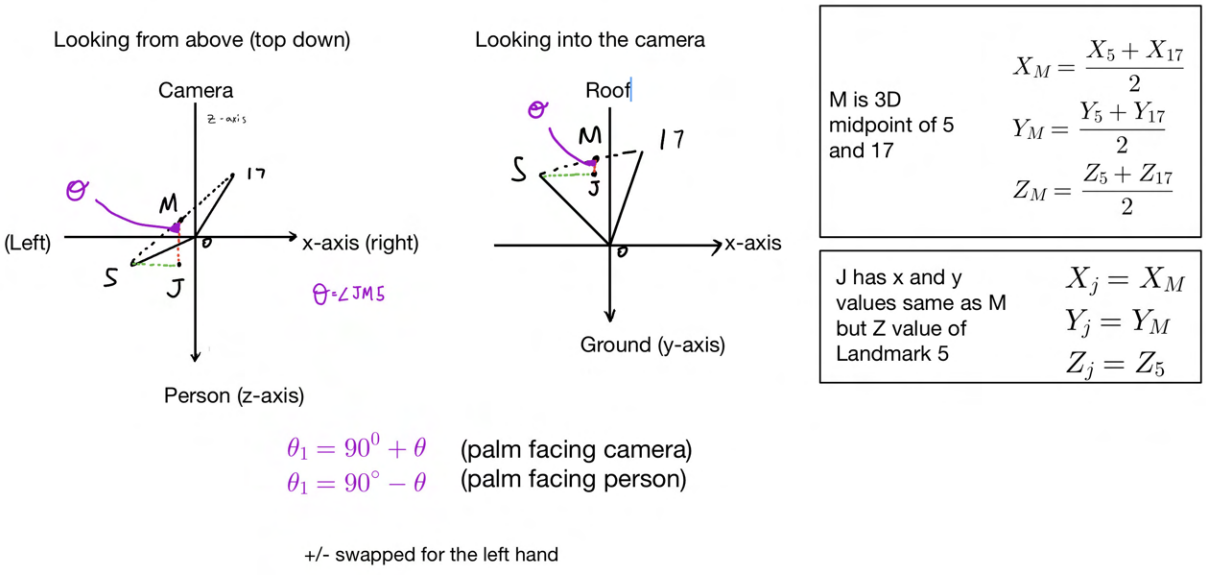


Figure 26: Definition of Joint 1

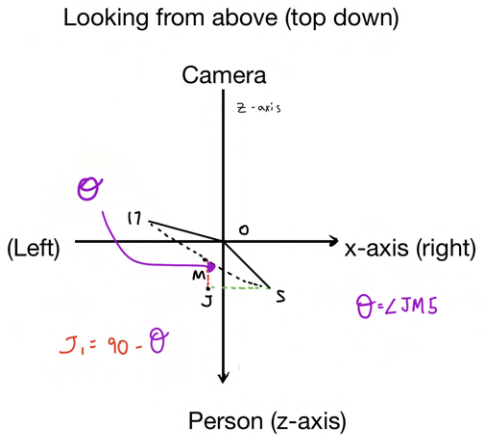


Figure 27: j1 when palm facing person

To imagine the geometry of Joint 2, the starting position can be thought of as the high-five position. However, instead of the wrist turning around left and right, the wrist curls forward and backwards for the range of motion. The 3 main points to calculate the angle of the second robot joint are the mid-point of landmarks 5 and 17 (M), the hand landmark 0 (wrist), and a created point K which always sits at a distance behind the wrist-like how satellites orbit the Earth. This point K always has the same y value as landmark 0 (meaning always sits at the same height as the wrist). See equations in Figure 28 for the definition of K. Note: θ_1 is the Joint 1 angle calculated previously.

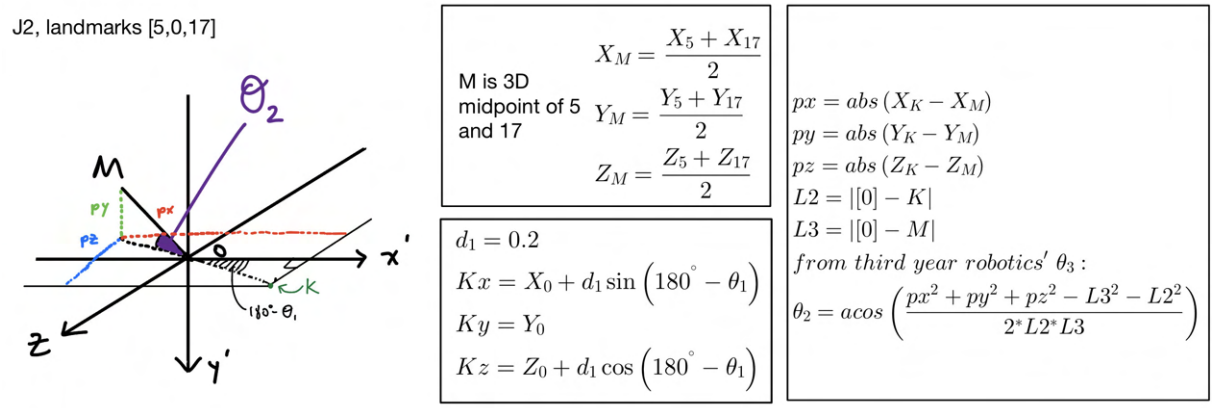
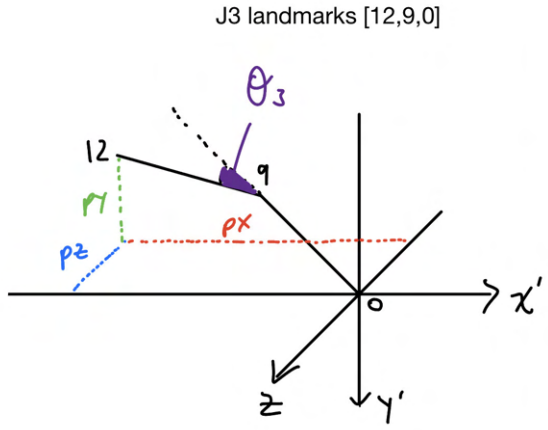


Figure 28: Definition of Joint 2

The starting pointing of joint 3 control can be imagined by the starting position of a high-five. However, the range of motion to control joint 3 is by curling the fingers down. The full range of motion 0-150 is done from the fingers being straight, to fingers curling in. The landmarks used are 12 (tip of middle finger), 9 (middle finger knuckle) and 0 (wrist).



$$px = abs\,(X_{12} - X_0)$$
$$py = abs\,(Y_{12} - Y_0)$$
$$pz = abs\,(Z_{12} - Z_0)$$
$$L2 = |[0] - [9]|$$
$$L3 = |[9] - [12]|$$
$$from\ third\ year\ robotics'\ \theta_3:$$
$$\theta_3 = acos\left(\frac{px^2 + py^2 + pz^2 - L3^2 - L2^2}{2{*}L2{*}L3}\right)$$

Figure 29: Definition of Joint 3

4.5.5 Implementation of hand gesture control

extract_ abcM: This function, extracts the 3D coordinates of any given three hand landmarks e.g. 12 (tip of middle finger), 9 (middle finger knuckle) and 0(wrist) into A, B, and C respectively. Each landmark contains three values, e.g. $a[0], a[1], a[2]$, which are the x, y, z values of A (landmark 12 in this case). Additionally, it extracts M and J, where M is the 3D coordinate of the midpoint between A and C, and where J is the point J as described in Figure 26.

```
def extract_abcM(joint):
    a = np.array([hand.landmark[joint[0]].x, hand.landmark[joint[0]].y, hand.landmark[joint[0]].z]) # First coord
    b = np.array([hand.landmark[joint[1]].x, hand.landmark[joint[1]].y, hand.landmark[joint[1]].z]) # Second coord
    c = np.array([hand.landmark[joint[2]].x, hand.landmark[joint[2]].y, hand.landmark[joint[2]].z]) # Third coord
    M = np.array([(hand.landmark[joint[0]].x + hand.landmark[joint[2]].x)/2,
                  (hand.landmark[joint[0]].y + hand.landmark[joint[2]].y)/2,
                  (hand.landmark[joint[0]].z + hand.landmark[joint[2]].z)/2]) # mid point coord
    J = np.array([M[0], M[1], a[2]]) # J = point on xy plane with 'a', paralell z-axis to M

    return a,b,c,M,J
```

Figure 30: Code of extract abcM

fold_angle: This function calculates the difference between 180 and the angle bounded by the three given landmarks ($180 - \angle abc$).

```
def fold_angle(a,b,c):
    px=abs(c[0]-a[0])
    py=abs(c[1]-a[1])
    pz=abs(c[2]-a[2])
    L2=np.linalg.norm(b - c)
    L3=np.linalg.norm(b - a)
    fold_angle=np.arccos((px**2+py**2+pz**2-L3**2-L2**2)/(2*L3*L2))
    fold_angle= fold_angle*180/np.pi

    return fold_angle
```

Figure 31: Code of fold_angle

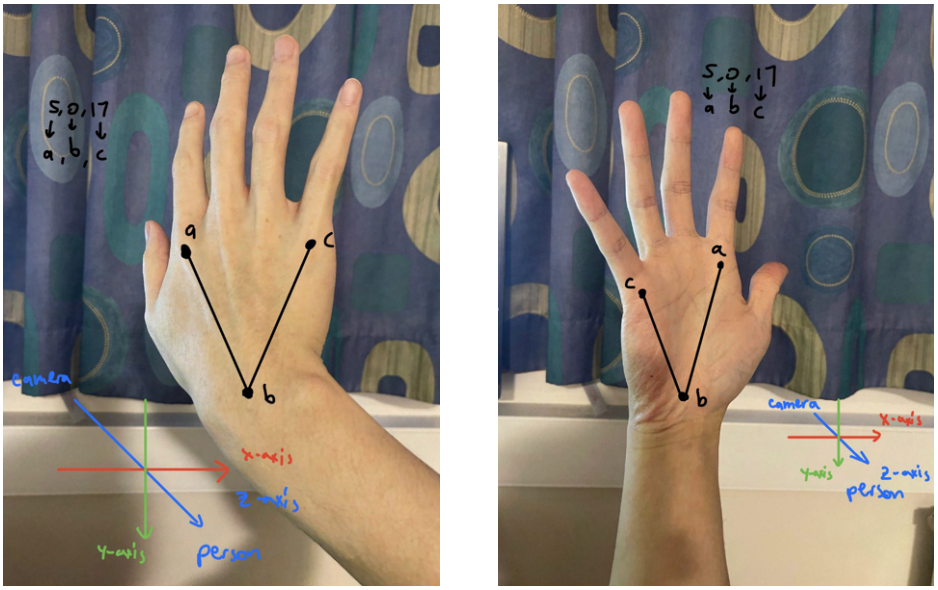
wrist_vert_r: This function calculates the angle of Joint 1 as explained in Figure 26, from landmark information 'extract abcm'.

```python
def wrist_vert_r(a,b,c,J,M):
    if a[0]< c[0] and  a[2]>c[2]:  #palm facing camera
        radians = abs(np.arctan2((a[0] - J[0]), (J[2]-M[2])))  #(x5-xJ)/(zJ-zM)
        j1 = 90+(radians*180.0/np.pi)
    elif a[0]>c[0] and a[2]>c[2]:  #palm facing person
        radians = (np.arctan2((a[0] - J[0]), (J[2]-M[2])))  #(x5-xJ)/(zJ-zM)
        j1 = 90-(radians*180.0/np.pi)
    elif a[0]< c[0] and  a[2]<c[2]:
        j1=180
    elif a[0]>c[0] and a[2]<c[2]:
        j1=0
    return j1
```

Figure 32: Code of wrist_vert_r

However, limitations are set to prevent imaginary domain:

1. Limit 1, when $a[0] < c[0]$ and $a[2] < c[2]$, angle j1 is limited to 180.
2. Limit 2, when $a[0] > c[0]$ and $a[2] < c[2]$, j1 is set to 0.



(a) $a[0] > c[0]$ and $a[2] > c[2]$, $j1 = 180$ **(b)** $a[0] > c[0]$ and $a[2] > c[2]$, $j1 = 0$

Figure 33: Over limit cases for j1

wrist_vert_l: This function calculates wrist angle for the left hand, with only the 90 +/- sign flipped compared to 'wrist_vert_r'.

```
def wrist_vert_l(a,b,c,J,M):
    if a[0]< c[0] and  a[2]>c[2]:  #palm facing camera
        radians = abs(np.arctan2((a[0] - J[0]), (J[2]-M[2])))  #(x5-xJ)/(zJ-zM)
        j1 = 90-(radians*180.0/np.pi)
    elif a[0]>c[0] and a[2]>c[2]:  #palm facing person
        radians = (np.arctan2((a[0] - J[0]), (J[2]-M[2])))  #(x5-xJ)/(zJ-zM)
        j1 = 90+(radians*180.0/np.pi)
    elif a[0]< c[0] and  a[2]<c[2]:
        j1=0
    elif a[0]>c[0] and a[2]<c[2]:
        j1=180
    return j1
```

Figure 34: Code of wrist_vert_l

j2_r: This function calculates the robot arm's joint 2 angle as explained in Figure 28.

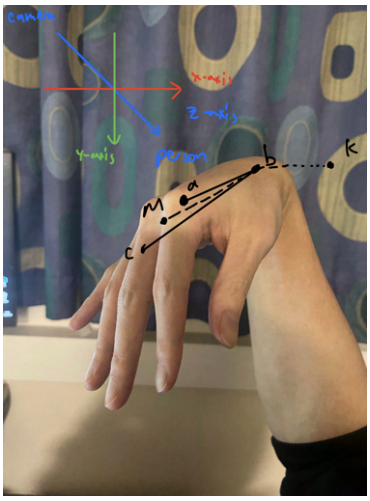
```
def j2_r(a,b,c,J,M,K):
    if M[2]>b[2] and a[0]>c[0] and M[1]>b[1]:   #back down
        j2=0
    elif M[2]<b[2] and a[0]<c[0] and M[1]>b[1]:   #front down
        j2=0
    elif M[2]>b[2] and a[0]<c[0] and M[1]>b[1]:  #back up
        j2=180
    elif M[2]<b[2] and a[0]>c[0] and M[1]>b[1]:  #front  up
        j2=180
    else:
        j2=fold_angle(M,b,K)

    return j2
```

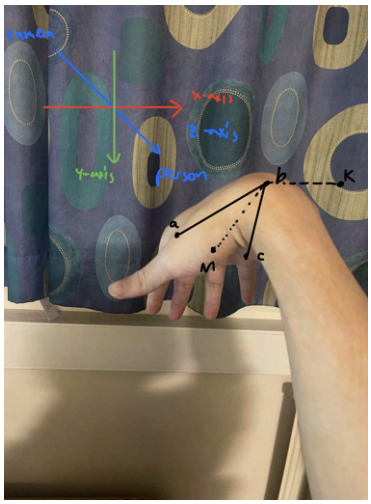
Figure 35: Code of j2_r

Again, limitations are set to prevent imaginary domain:

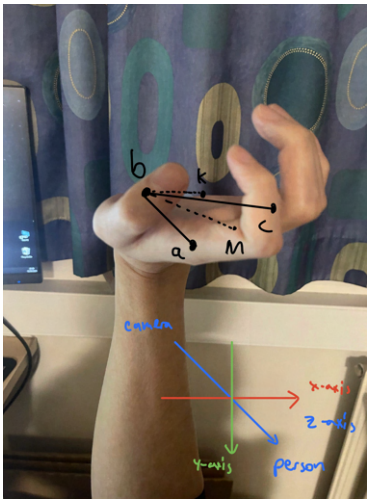
1. Limit 1, when $M[2] > b[2]$, $a[0] > c[0]$ and $M[1] > b[1]$, the hand is facing the user and ground, hence j2 should be 0.
2. Limit 2, when $M[2] > b[2]$, $a[0] > c[0]$ and $M[1] > b[1]$, the hand is facing forward and ground, hence j2 should also be 0.
3. Limit 3, when $M[2] > b[2]$, $a[0] > c[0]$ and $M[1] > b[1]$, the hand is pointing to the user with palm up, hence j2 should be 180.
4. Limit 4, when $M[2] > b[2]$, $a[0] > c[0]$ and $M[1] > b[1]$, the hand is pointed front with palm up, hence j2 should be 180.



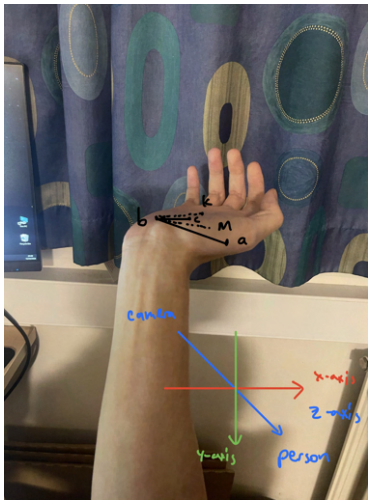
(a) palm back and down, $j2 = 0$



(b) palm front down, $j2 = 0$



(c) palm back and up, $j2 = 180$



(d) palm front and up, $j2 = 180$

Figure 36: over limit cases for $j2$

j2_l: Mostly identical to j2_r, but calculates joint 2 angle of left hand.

```
def j2_l(a,b,c,J,M,K):
    if M[2]>b[2] and a[0]<c[0] and M[1]>b[1]:  #back down
        j2=0
    elif M[2]<b[2] and a[0]>c[0] and M[1]>b[1]:  #front down
        j2=0
    elif M[2]>b[2] and a[0]>c[0] and M[1]>b[1]: #back up
        j2=180
    elif M[2]<b[2] and a[0]<c[0] and M[1]>b[1]:  #front up
        j2=180
    else:
        j2=fold_angle(M,b,K)


    return j2
```

Figure 37: Code of j2_l

Next_angle: Functions above have calculated angles from image recognition, but before these angles are transmitted to the motors, they should be filtered to prevent over-responsive jerky outputs. Hence this function is created, whose purpose is to get the image recognition's angle and find a suitable output angle for the motors.

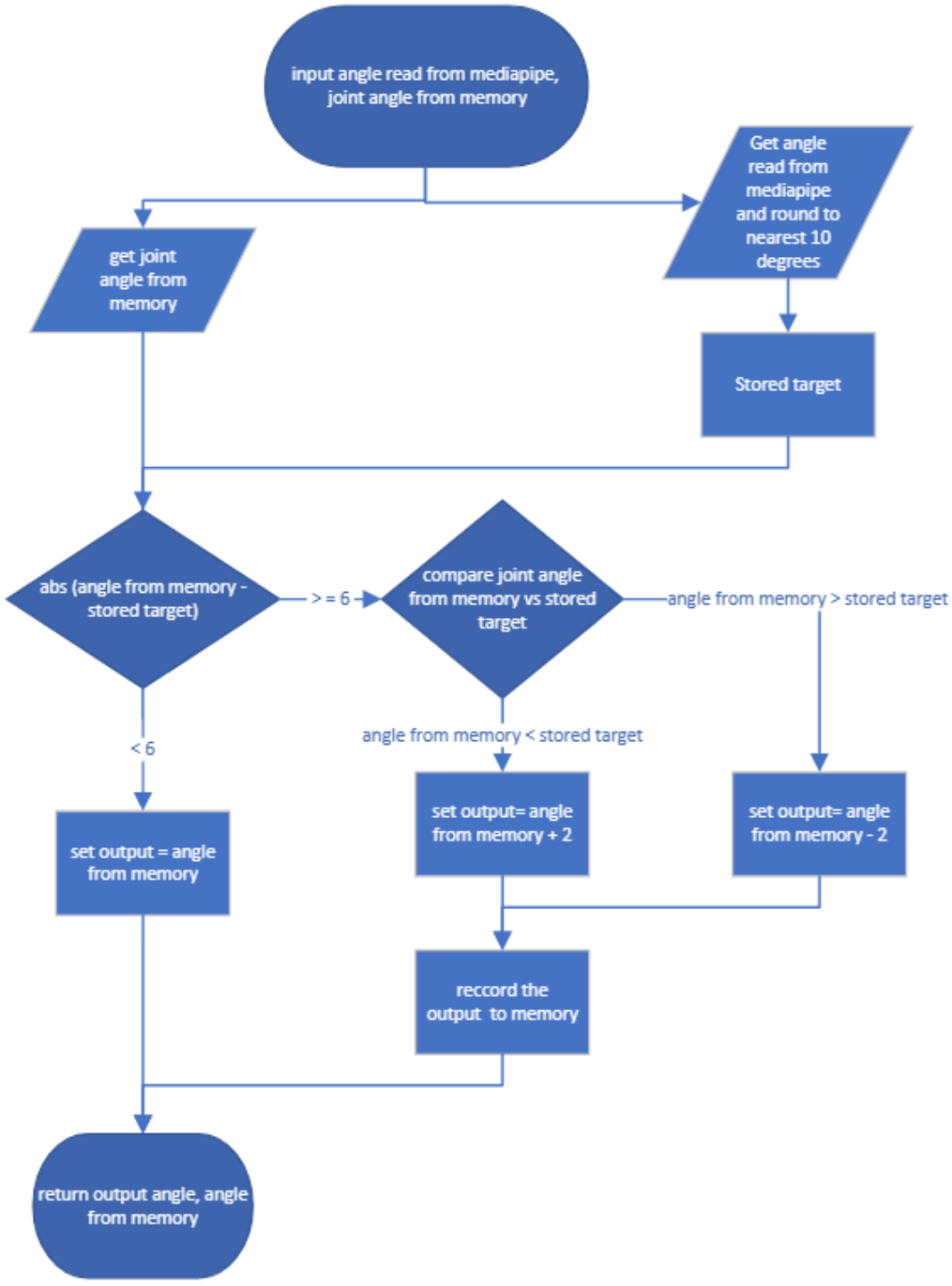


Figure 38: Next_angle Flowchart

manual_r: This function summarizes all the angle calculations of the right hand from 'wrist vert' (j1), 'j2r' (j2) and 'fold angle' (j3), renders the results on landmark positions and applies 'next_angle' and outputs arm motor angles.

```
def manual_r(image, results, previousj1, previousj2, previousj3):
    #joint 1

    joint=[5,0,17]     #5,0,17
    a,b,c,M,J= extract_abcM(joint)           #extract abcMJ
    text_position = np.array([hand.landmark[joint[1]].x, hand.landmark[joint[1]].y])# position of text , Second coord
    j1=wrist_vert_r(a,b,c,J,M)           #calculates joint angle
    outputj1,previousj1 = next_angle(j1,previousj1)    #smoothing
    render_result(image,str(round(outputj1, 0)),text_position)

    #joint 2
    joint=[5,0,17]
    a,b,c,M,J = extract_abcM(joint)
    text_position = np.array([M[0],M[1]])# Second coord
    cj1=j1*np.pi/180
    K=np.array([b[0]+0.2*np.sin(np.pi-cj1), b[1], 0.2*np.cos(np.pi-cj1)+b[2]])
    j2=j2_r(a,b,c,J,M,K)
    text_positionk=np.array([K[0],K[1]])
    outputj2,previousj2 = next_angle(j2,previousj2)
    render_result(image,str(round(outputj2, 0)),text_position)
    render_result(image,'K',text_positionk)


    #joint 3
    joint=[12,9,0]
    a,b,c,M,J = extract_abcM(joint)
    text_position = np.array([hand.landmark[joint[0]].x, hand.landmark[joint[0]].y])
    j3=fold_angle(a,b,c)
    outputj3,previousj3 = next_angle(j3,previousj3)
    render_result(image,str(round(outputj3, 0)),text_position)


    #sending
    outputj1 = round(int(outputj1))
    outputj2 = round(int(outputj2))
    outputj3 = round(int(outputj3))
    #Combine the joint angles into a single message

    return image,outputj1,outputj2,outputj3
```

Figure 39: Code of manual_r

Since the left hand is designed to control the base motors, additional functions are created to bundle with angle calculation for base controls.

1. curled_finger: A function returns a boolean stating whether the 'fold angle' of the given 3 finger landmarks is bigger (True) or smaller (false) than 100 degrees.

2. straight_finger: Opposite to curled finger, this function returns the boolean if 'fold angle' is bigger or smaller than 30.

3. fingers_straight: Returns boolean whether all fingers are straight. Its inputs are 3 finger landmarks from each finger and run 'straight_finger' on each finger. When all 'straight_finger' returns true, 'fingers_straight' returns true.

4. fingers_curled: Returns whether all fingers are curled. Its inputs are 3 finger landmarks from each finger and run 'curled_finger' on each finger. When all 'curled_finger' returns true, 'fingers_curled' returns true.

```
def fingers_straight():
    joint=[20,18,17]
    a,b,c,M,J = extract_abcM(joint)    #20,18,17
    #con3 pinky down
    con3=straight_finger(a,b,c)

    joint=[16,14,13]
    a,b,c,M,J = extract_abcM(joint) #16,14,13
    #con4 ring down
    con4=straight_finger(a,b,c)

    joint=[12,10,9]
    a,b,c,M,J = extract_abcM(joint)  #12,10,9
    #con5 middle down
    con5=straight_finger(a,b,c)

    joint=[8,6,5]
    a,b,c,M,J = extract_abcM(joint)   #8,6,5
    #con6 index down
    con6=straight_finger(a,b,c)

    if con3==True and con4==True and con5==True and con6==True:
        return True
    else:
        return False
```

Figure 40: Code of fingers straight

```
def fingers_curled():
    joint=[20,18,17]
    a,b,c,M,J = extract_abcM(joint)    #20,18,17
    #con3 pinky down
    con3=curled_finger(a,b,c)

    joint=[16,14,13]
    a,b,c,M,J = extract_abcM(joint) #16,14,13
    #con4 ring down
    con4=curled_finger(a,b,c)

    joint=[12,10,9]
    a,b,c,M,J = extract_abcM(joint)  #12,10,9
    #con5 middle down
    con5=curled_finger(a,b,c)

    joint=[8,6,5]
    a,b,c,M,J = extract_abcM(joint)   #8,6,5
    #con6 index down
    con6=curled_finger(a,b,c)

    if con3==True and con4==True and con5==True and con6==True:
        return True
    else:
        return False
```

Figure 41: Code of fingers curled

After the creation of these functions, commands are created for controlling the base motors. These conditions are tailored to when the left hand is closer to the camera than the right hand.

1. 'turn left' is sent when $j1 > 20$, $j2 > 160$ and check fingers curled returns true.
2. 'turn right' is sent when $j1 > 160$, $j2 > 50$ and check fingers curled returns true.
3. 'reverse' is sent when $j1 > 30$, $70 > j2 > 150$ and $j3 > 40$.

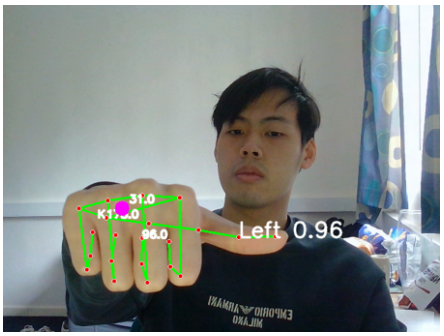
4. 'forward' is sent when $j1 > 160$, $75 > j2 > 120$, $30 > j3 > 90$ and $a[2] > c[2]$.

5. 'stop' is sent when $60 > j1 > 120$, $60 > j2 > 120$ and check_fingers_curled returns true.

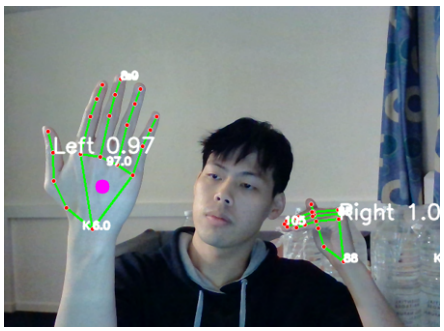
Some pictures below have the right hand in place, but this is just to show that as long as the left hand is closer to the camera, it does not affect the control of the base.

(a) Stop

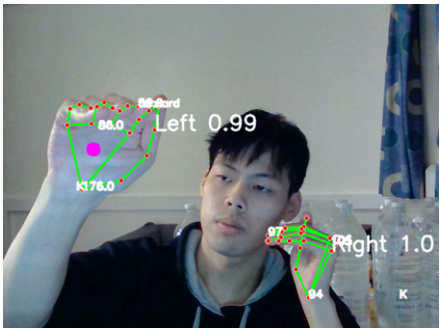
(b) Turn left



(c) Turn right



(d) Reverse



(e) Forward

Figure 42: Base control command gestures

All the required functions for left-hand control are in place, hence left left-hand summarizing function manual_l is then created, which includes command recognition and angle calculations as described above.

```
def manual_l(image, results, livespeed):

    command='none'
    #stop
    #palm has to face inside (con1)

    joint=[5,0,17]
    a,b,c,M,J = extract_abcM(joint)   #5,0,17
    text_position_j1= np.array([hand.landmark[joint[1]].x, hand.landmark[joint[1]].y])# Second coord
    j1=wrist_vert_l(a,b,c,J,M)
    render_result(image,str(round(j1, 0)),text_position_j1)

    #con2 palm upright
    joint=[5,0,17]
    a,b,c,M,J = extract_abcM(joint)   #5,0,17
    text_position = np.array([M[0],M[1]])
    cj1=j1*np.pi/180
    K=np.array([b[0]-0.2*np.sin(np.pi-cj1), b[1], b[2]+0.2*np.cos(np.pi-cj1)])
    j2=j2_l(a,b,c,J,M,K)
    text_positionk=np.array([K[0],K[1]])
    render_result(image,str(round(j2, 0)),text_position)
    render_result(image,'K',text_positionk)

    #check fingers curled
    check_fingers_curled=fingers_curled()
    check_fingers_straight=fingers_straight()

    if 60<j1<120 and 60<j2<120 and check_fingers_curled==True:
        #render the result on the image
        command='stop'
        render_result(image,'stop',text_position_j1)

    joint=[12,9,0]
    a,b,c,M,J = extract_abcM(joint)  #12,9,0
    text_position = np.array([hand.landmark[joint[0]].x, hand.landmark[joint[0]].y])
    j3=fold_angle(a,b,c)
    render_result(image,str(round(j3, 0)),text_position)

    if j1<20 and j2>160 and check_fingers_curled==True:
        command='turn left'
    if j1>160 and j2<50 and check_fingers_curled==True:
        command='turn right'


    if j1<30 and 70<j2<150 and j3<40:
        command='rev'
        render_result(image,'rev',text_position)
    elif j1>160 and 75<j2<120 and 30<j3<90 and a[2]<c[2]:
        command='forward'
        render_result(image,'forward',text_position)
    if command=='rev' or command=='stop' or command=='forward' or command=='turn left' or command=='turn right' or command=='le
        hi='hi'
    else:
        command = 'nothing'

    return image,command
```

Figure 43: Code of manual_l

4.5.6 Communication and Driving Functions

Before sending instructions to the motors, commands need to be converted into motor values, hence the creation of the 'driving' function.

driving: This function gets the commands recognised by the calculation functions and returns motor directions and speed. E.g. When the command is 'reverse', circle direction, square direction, plus direction and triangle direction are 0,1,2,2 respectively can be with the speed of 50. Where 0 and 1 are clockwise/counter-clockwise and 2 means stop/no rotation.

Since arm motor values can be directly sent to the Arduino, they don't require conversion, hence bundled with base motor values and sent to the Arduino.

pass_instruction_arm and pass_instruction_base: passes the angles and commands calculated from functions onto the Arduino.

4.6 Holistic Control and tracking functions

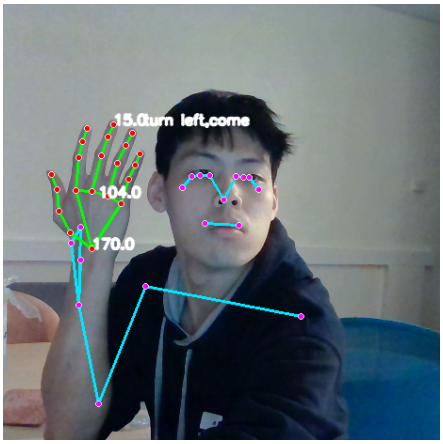
Below describes the implementation of the holistic model to another mode of control different to the manual mode.

4.6.1 Holistic Model

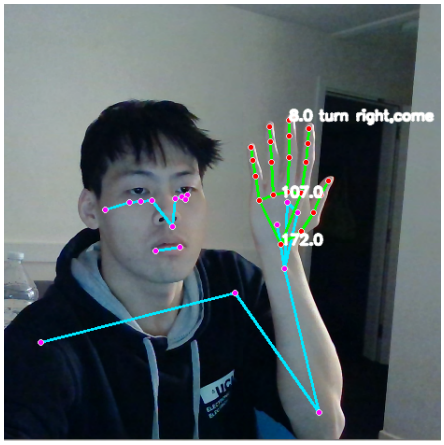
come_follow: This ambidextrous function utilises the holistic model to analyse the hands and body for command identification and target tracking. This function provides two control mechanisms, firstly, a simple 'come/stop' method and secondly, an automatic tracking system.

A simple tracking function is triggered when the user shows a 'come' command as shown in Figure 44, the robot constantly adjusts itself to approach the hand. Depending on whether the hand is on the left or right side of the frame, it adjusts the base and tries to centre the hand. Moreover, the robot will stop when the hand/target is too close to the robot. It works by analysing the dimension of the presented hand, where the height is from the tip of the middle finger to the base of the wrist and the width is the distance between the pinky finger and thumb. When either of these distances occupies half of the camera frame (e.g. 320 out of 640 on the x-axis), it triggers the 'stop' command. This function requires a 'come' command to be shown at all times.

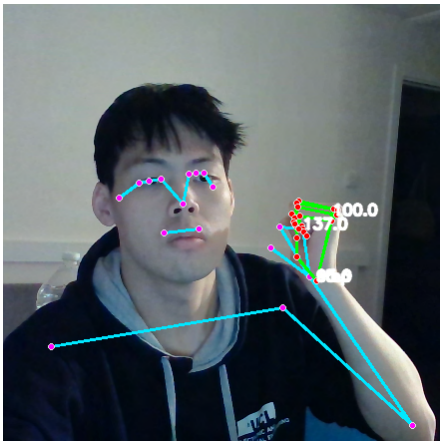
(a) Stop due to approached target



(b) turn left and come



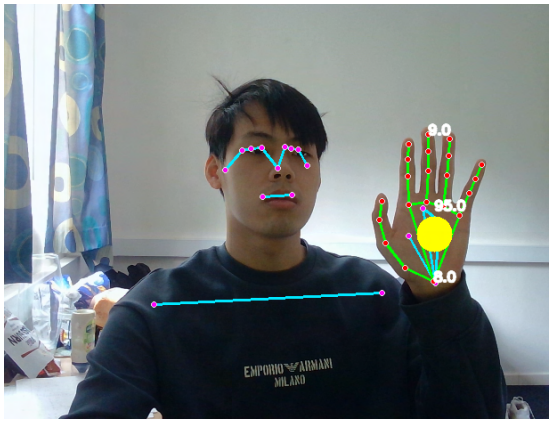
(c) Turn right and come



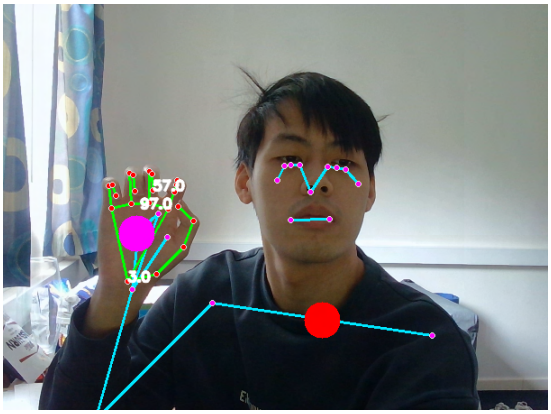
(d) Stop

Figure 44: Simple tracking

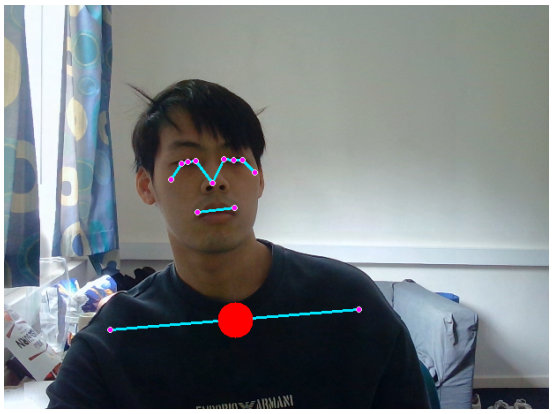
The more intelligent tracking function doesn't require a constant 'come' command in the frame. Firstly, it checks if either hand is in the high-five position (if $j1 > 30$ and $70 > j2 > 110$ and checks fingers straight returns true). When the condition is satisfied, a button is rendered in the centre of the palm (centre of landmarks 5,0,17). When this button is present, if the user grabs the button, in other words, 'check fingers curled' returns true, it will activate 'person tracking'.



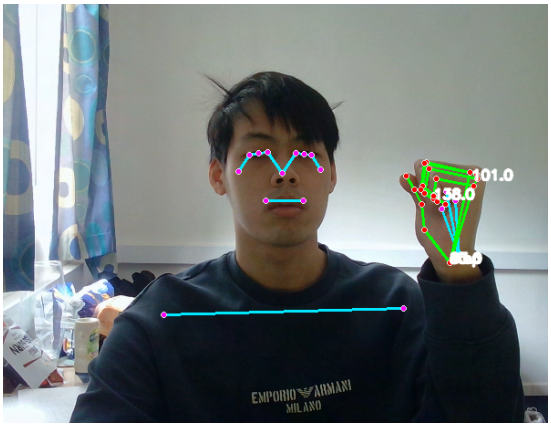
(a) Button shown



(b) Grabbed button, initiate tracking



(c) Tracking



(d) Stop cancel

Figure 45: Automatic tracking

As seen in Figure 45, the 'button' can be shown in either hand and when the user 'grabbed' the button, a red dot showing the centroid between the shoulders is shown as the target to track. Also, when tracking is initiated, hands no longer need to be shown. Furthermore, the tracking can be cancelled anytime when a stop command is shown, and the red dot disappears.

4.6.2 Person Tracking

This function controls the robot to follow the person presented in the camera frame when an intelligent tracking command is initiated, which allows the user to continue their other task e.g. cooking while the robot is approaching the user. It requires an input such as the width and height difference of each shoulder, the centroid between the shoulders, all base motor speeds, directions and arm angles.

When the y value of the midpoint between the shoulders goes above or under the

threshold, the command is sent to the arm to adjust either joint 2 or 3 depending on their current angle. Since the camera frame is 640*480, the midpoint of y axis is 240, ± 60 makes lower and upper boundaries, namely 300 and 180. If the centroid is > 300 (meaning the centroid is on the lower half of the screen), j2 is decreased to point the camera downwards. On the other hand, when the centroid is > 180 (centroid in the upper frame), either decrease j3 or increase j2 to raise the camera. When $j2 >= 120$, $j3$ should be decreased to raise the camera instead of increasing j2 (if $j2 > 120$, then j2 can be raised to achieve the same effect).

On the other hand, when the y value of the centroid is within the boundaries, the base motors are adjusted to centre the centroid to the centre of the screen (at midpoint 320), bounds are set at 340 and 300 to trigger 'turn right' or 'turn left'. When the centroid is within these boundaries, it determines whether to progress or not by comparing the height and width of the shoulders. The y distance (width) target is set to be 300 and the x distance (height) target is set to be 160.

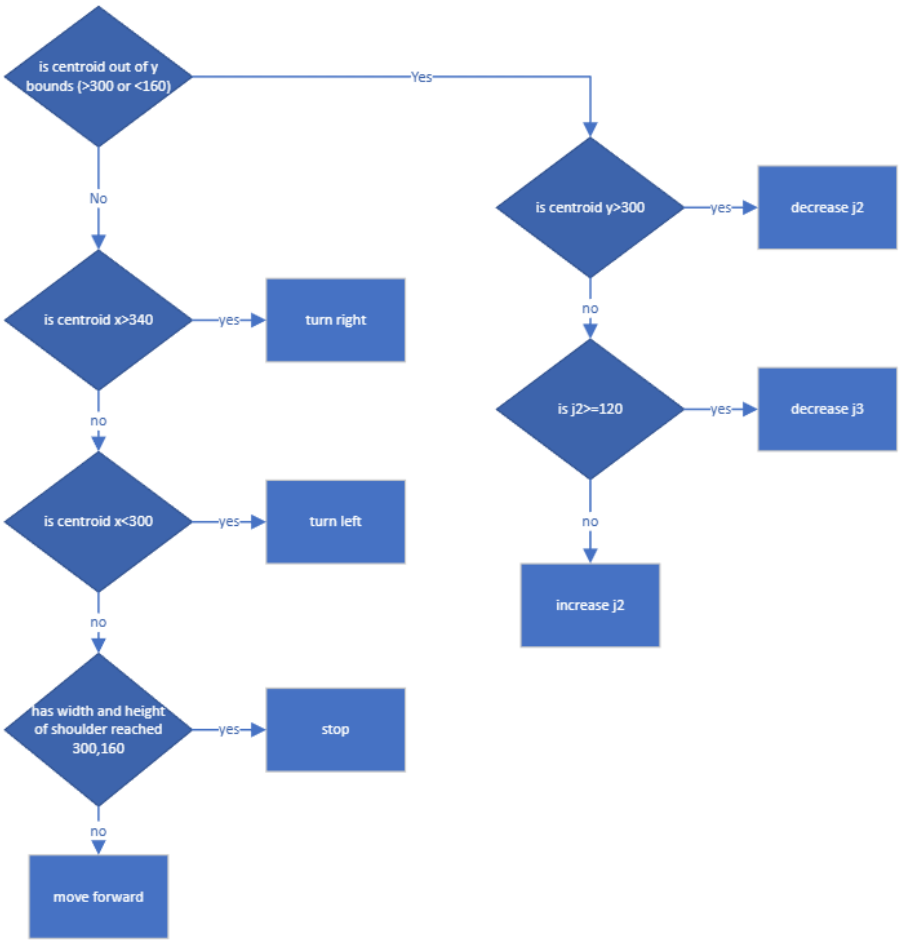


Figure 46: Heavily simplified person tracking Flowchart

4.6.3 Object Tracking

This function is identical to person tracking with two exceptions, firstly, this function tracks the object's centroid instead of the mid-point of the shoulders. Secondly, instead of the width and height of the shoulders, it uses the width and height of the object's bounding box. In Figure 47, the robot wants to track the orange on the left with higher confidence (red dot) and ignore the lower confidence orange (green).

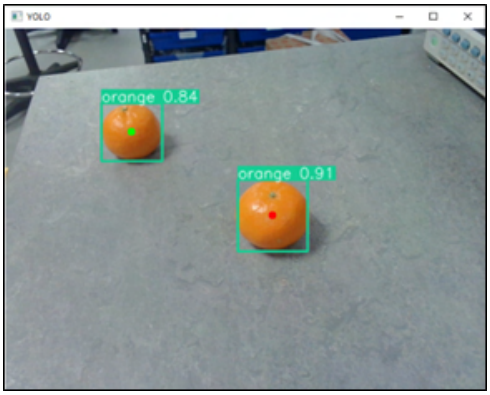


Figure 47: Object tracking

4.7 Python Code Flow

This section outlines where all the previous systems such as hand gesture control, holistic control, voice recognition and object detection modes are implemented within the Python code. Therefore, this section contains the flowcharts and descriptions to label where they are placed and how they are linked within the code.

Within Figure 48, the only blocks that haven't been explained before are the setup blocks, whose purpose is to ensure the robot's starting position gets a good weight balancing and has less stress, therefore draining less battery and increases the life span of the robot.

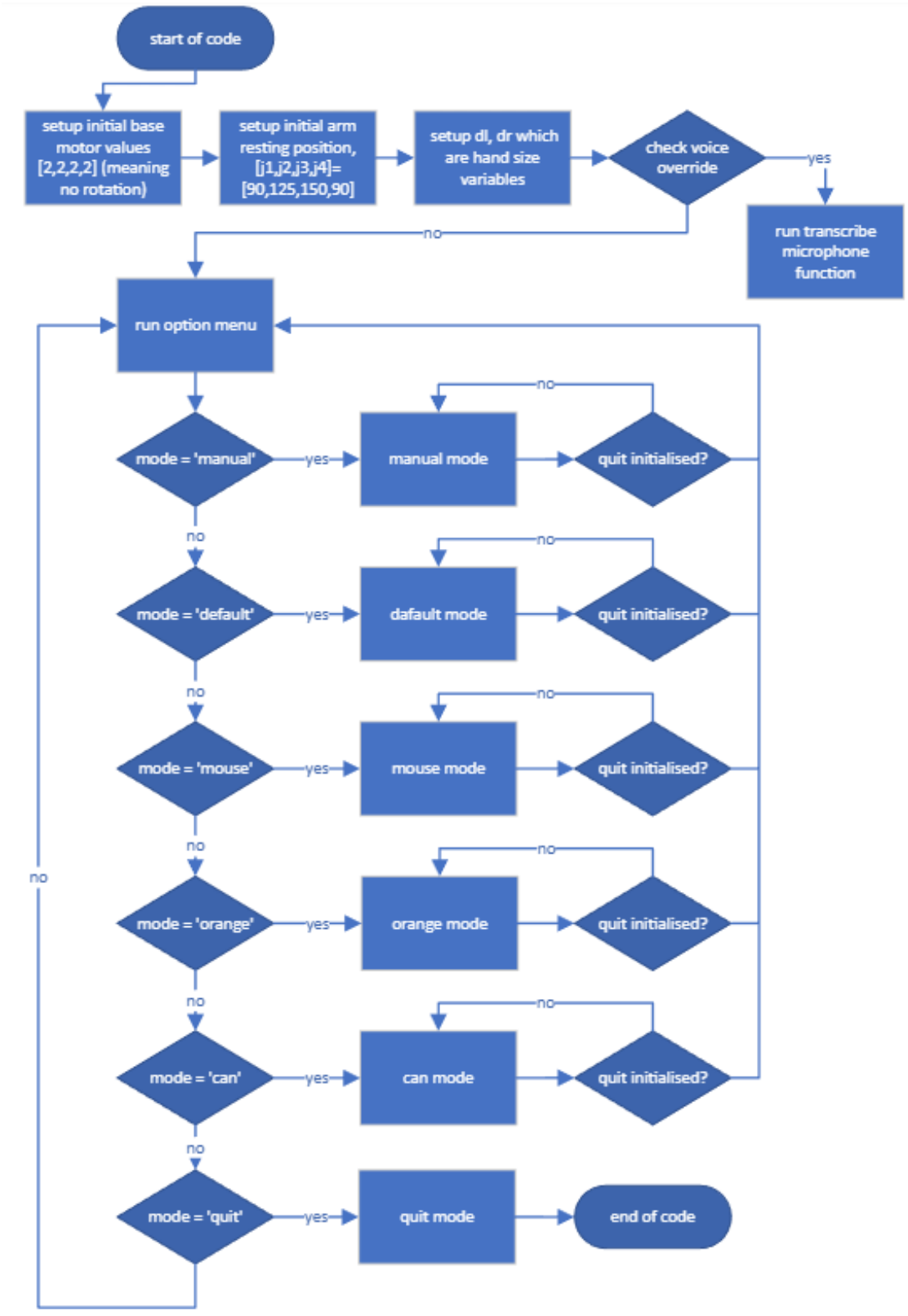


Figure 48: Simplified Python flow

4.7.1 Manual Mode

Below is the flowchart of Manual mode (hand gesture) block in Figure 48:

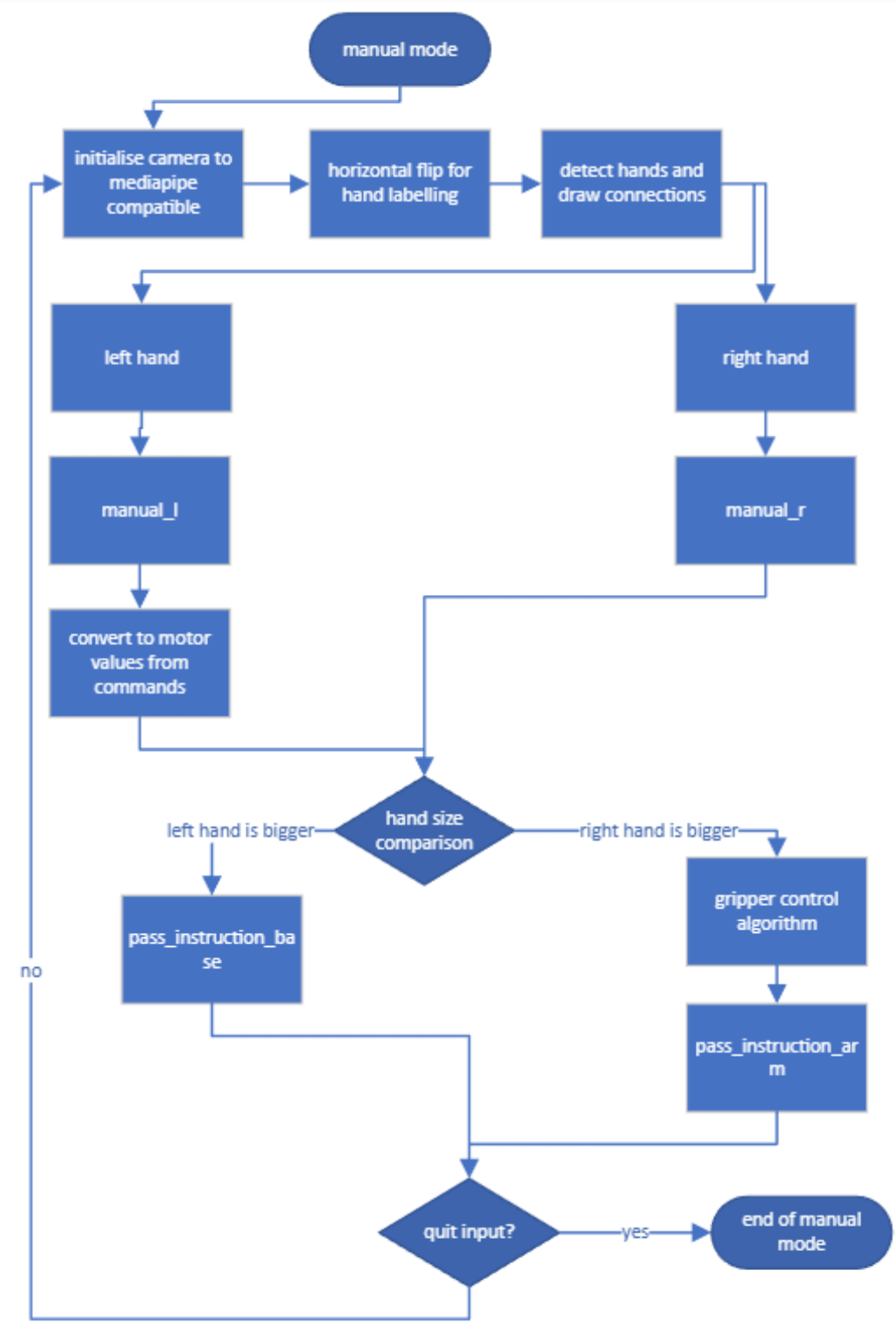
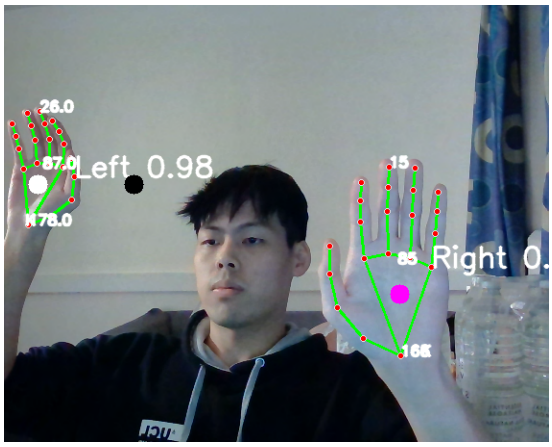
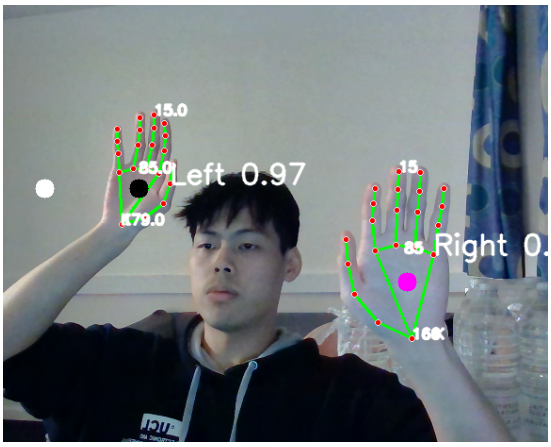


Figure 49: Simplified manual mode flowchart

The gripper control block within Figure 49 is a system that activates when both hands are present in the frame and when the right hand is in front of the left hand. The user can hover the centre of the left hand onto the two pixels on the screen which opens/closes the gripper.



(a) Gripper open



(b) Gripper close

Figure 50: Gripper control

4.7.2 Default Mode

Figure 51 is a flowchart of the 'default mode' block of Figure 48, which is a mode that involves simple and automatic tracking that uses the holistic model.

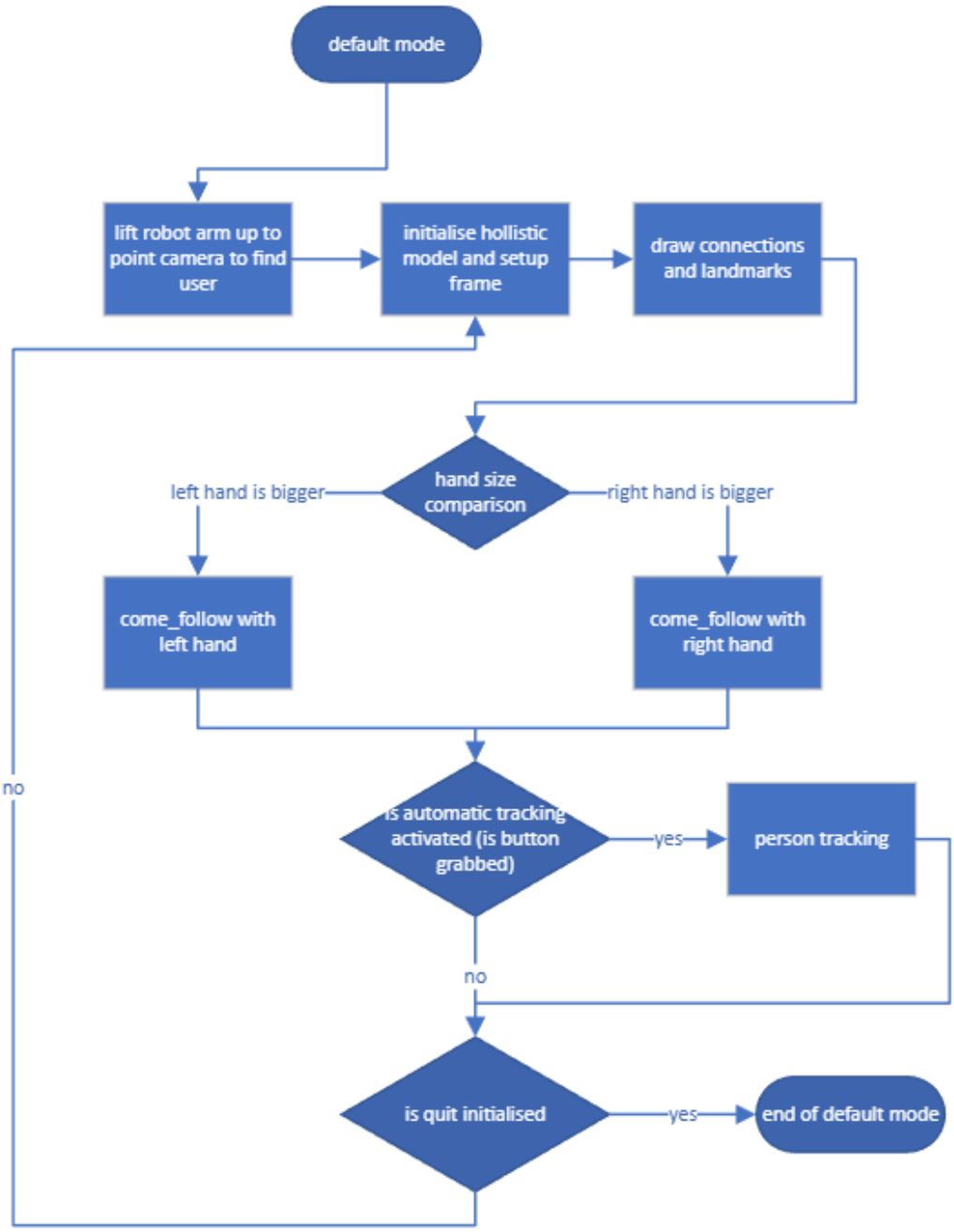


Figure 51: Simplified default mode flowchart

4.7.3 Object Modes

These modes utilise YOLO to carry out object recognition, which then acquires a target object. These modes' objective is to grab and pick up the object class. Each object class is a mode. Since all the modes are the same, except for the bounding box of the objects, only one flowchart will be shown in this section, using the class 'can'.

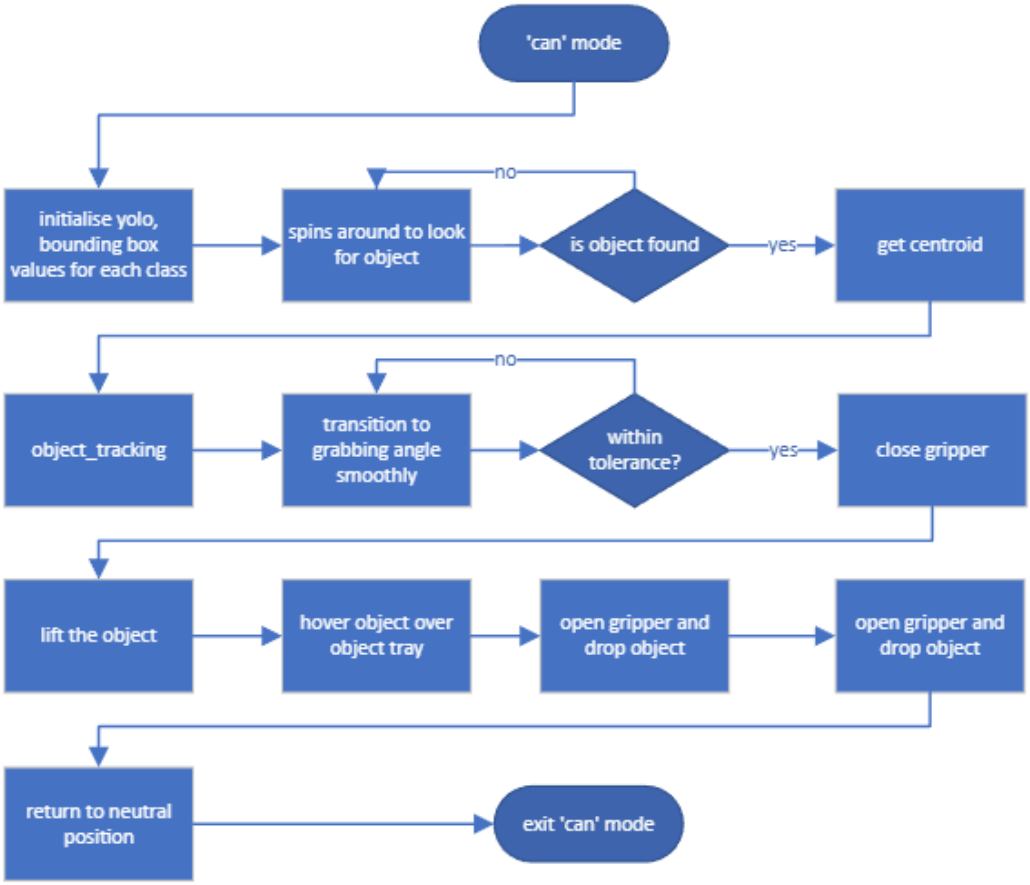


Figure 52: Simplified object modes flowchart

4.7.4 Quit function

This is executed when mode = 'quit', target angles are set and sent smoothly with $[j2, j3] = [120, 150]$, which sits arm down in a position which implies the robot is resting, which is also an angle that allows the arm motors to have less torque which protects its life-span.

4.8 Wireless Connectivity - Wireless Camera and RF

4.8.1 Wireless Camera

To allow for a full wireless robot, we need a wireless camera. This would not be possible without at least a power cable as wireless battery-powered cameras fit for our case would be too expensive. An inexpensive camera with a power cable is an ESP-32 Cam. This would be connected to the same IP address as the device where the program would run, and it opens with 'CV'. The settings could be altered and with a clock frequency of 10KHz, a frame size of SVGA, and some setting alterations, a frame rate of 15fps was achieved.

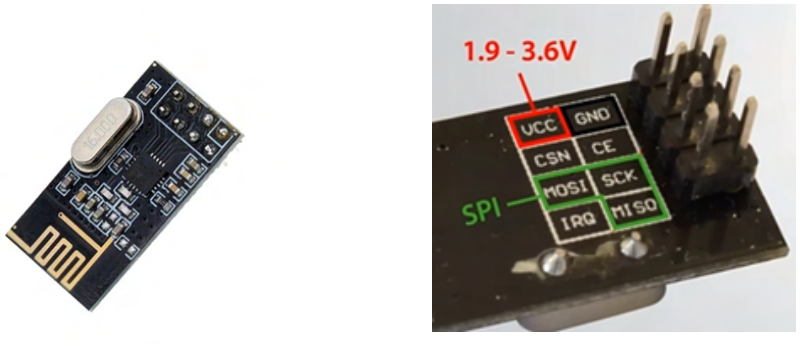
We opted for a wired camera option due to a frame rate of 3fps whilst running in 'grabbing mode'. This would be too slow for the robot's speed and so would end up missing grabs. With the manual and default mode, the frame rate was 15fps and this was enough to get seamless control of the robot with gestures.

Figure 53: ESP-32 Cam

4.8.2 RF

A fit wireless connection is radio frequency, which diminishes the need for good Wi-Fi or Bluetooth and gives an extended usage range. The RF connection utilises transceivers, and the 'nRF24L01' uses the 2.4GHz band. A pair of these each can act as a transmitter and receiver. The way to make them one or the other is using the code 'radio.stopListening();' for transmitting and 'radio.startListening();' for receiving.

The pin connections between the nRF24L01 and the Arduino for the transmitter and receiver ends were with SPI connections. Connections were made by looking at Figure 54. The VCC was connected to the 3.3V of the Arduino and the GND to the ground. The CSN and CE connected digital pins 30 and 31 respectively for the receiver with Arduino Mega and transmitter with Arduino UNO.



(a) nRF24L01 Module **(b)** nRF24L01 Pins

Figure 54: nRF24L01 Module and Pin Configuration

There were some problems in getting this to work making us switch the modules many times. The 5V VCC was used which worked for a while but stopped working. After many tries of getting this to work due to 3.3V inaccessibility due to the motorshield, another Arduino UNO was used to supply the 3.3V and the ground. This again worked for a while until another grounding issue occurred due to pins on the Mega having internal grounding. There was no ground bonding as the ground pin was connected to a separate UNO. Due to this permanent ground bonding by soldering a wire, Figure 55, allowed for full functioning.

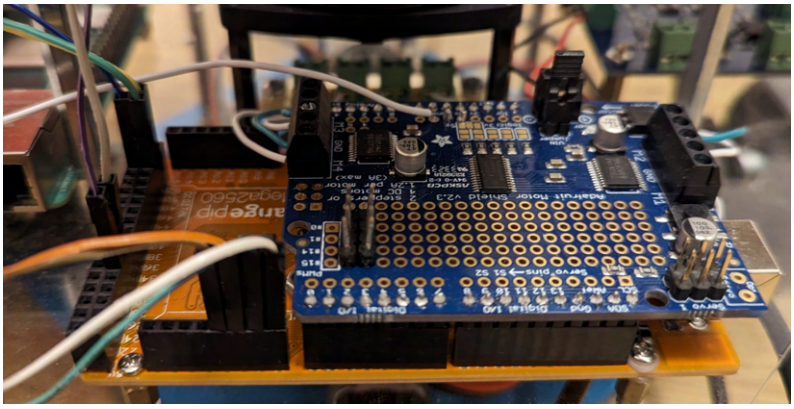


Figure 55: Soldered Wire to Ground

4.9 Joystick SLAM

Below we describe the approach taken to solving the problem of SLAM.

4.9.1 Controller

As mentioned in Section 3, a custom controller was required to be written for our omnidirectional base. The controller written is available online at [43]. We used the ROS2 node, Teleop Twist Keyboard, to provide command velocities to our robot controller. The workings of this node are outlined in Section 3.6.1.2.

The custom controller accepts these command velocities and turns them into angular wheel velocities. The kinematics equations, of 4WD omnidirectional robot are known [44, 45]. These kinematic equations were added to our controller. Upon testing using Teleop Twist Keyboard, it was clear that control of the robot was not robust. Often the robot would only vaguely travel at the desired body velocity. We were not able to pinpoint the exact cause of the differences between theory and real life. However, we believe that it may be due to the theoretical assumptions of a frictionless surface and or the imperfect contract between wheels and a non-flat floor.

As a result, an alternative form of kinematics was developed. Similarities can be seen between ours and those of a differential drive robot's kinematics. The frames of reference of our robot base are outlined in Section 3.5. Figure 56 outlines the implemented kinematics. Through these experimental methods, we achieved the goal of joystick control of the robot.

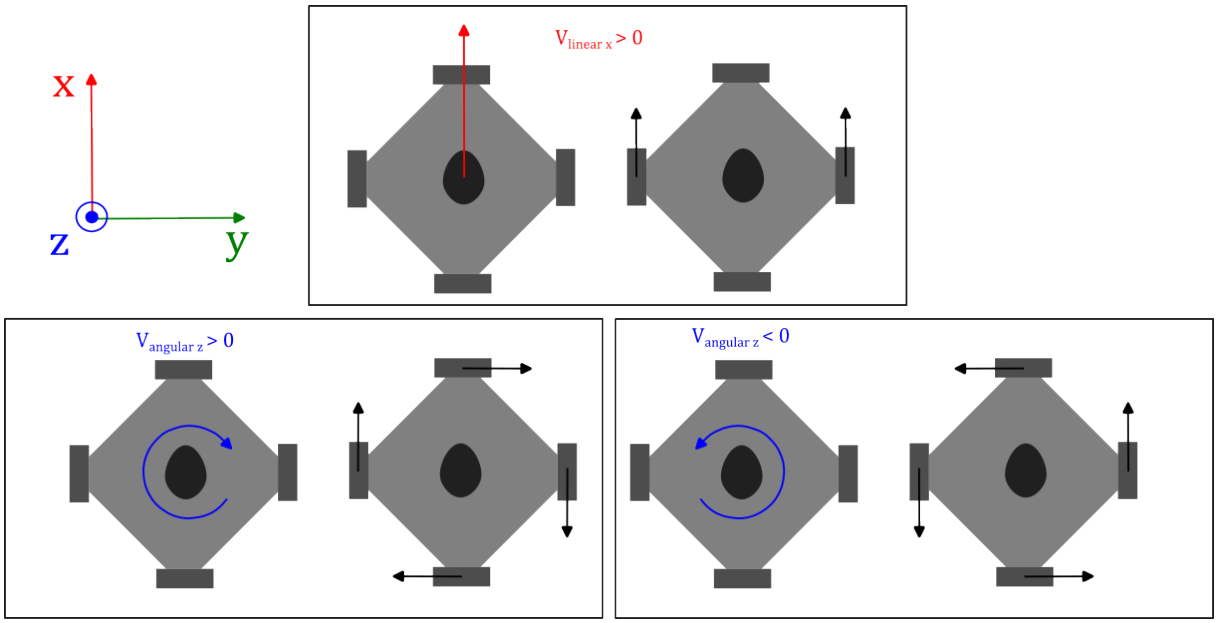


Figure 56: Omnidirectional Controller Kinematics

4.9.2 Hardware Interface

Next, a custom hardware interface was written for our robot. The hardware interface written is available online at [46]. The robot model, as described by the URDF file, included as part of this package, can be seen in Figure 57. This URDF was formed by creating the robot model using Fusion 360 and then converting using 'fusion2urdf-ros2' [47]. The hardware interface is a ROS node that runs on the Raspberry Pi. The calculated PWM values are sent using serial connection from the Raspberry Pi to the Arduino Mega.

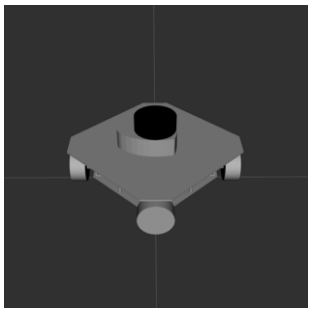


Figure 57: Robot URDF, 1:1 Scale with real robot ensuring interactions with environment are accurate

4.9.3 TF Tree

As mentioned in Section 3.6.3, running SLAM requires the odom → base_link transform to be published. We identified that our wheels were prone to slippage on a lot of surfaces. As a result, we opted to use a canonical scan matcher [48] to do so rather than a wheel encoder-based system. The 2016 paper, by Jaimez *et al*, explains how the scan matcher works [49]. In doing so we increased our robot's robustness to the home environment. As also mentioned in Section 3.6.3 the SLAM algorithm itself publishes the map → odom transform. The SLAM package we are using to do so is Steve Macenski's slam-toolbox [50]. The remaining transforms are static and are defined as part of the hardware interface package [46]. In this way a complete TF tree was developed, see Figure 58.

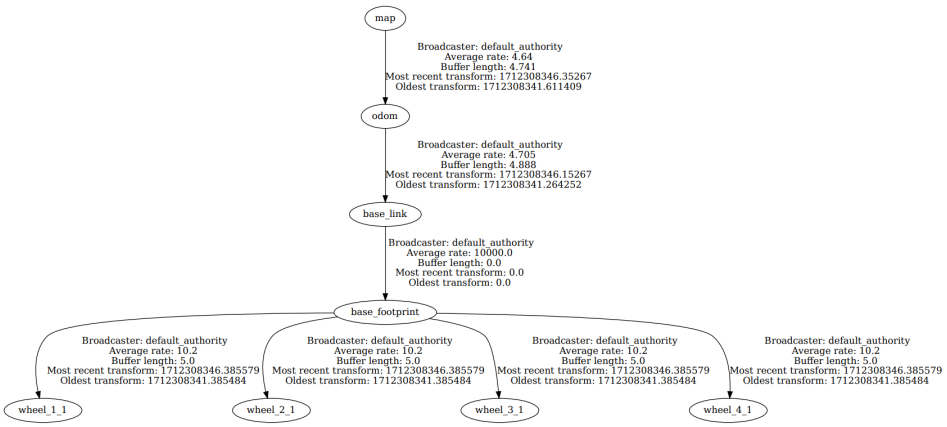


Figure 58: Complete TF Tree for our robot, obtained from ROS2 using 'view_frames ' [51]

4.9.4 Developed Maps

After completing the TF tree, it was time to experiment with slam-toolbox, tuning it for our use. To familiarise ourselves with slam-toolbox several maps were developed. The first map developed can be seen in Figure 59.

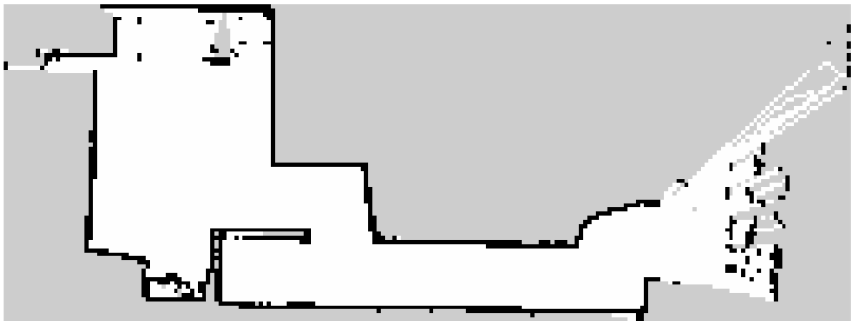
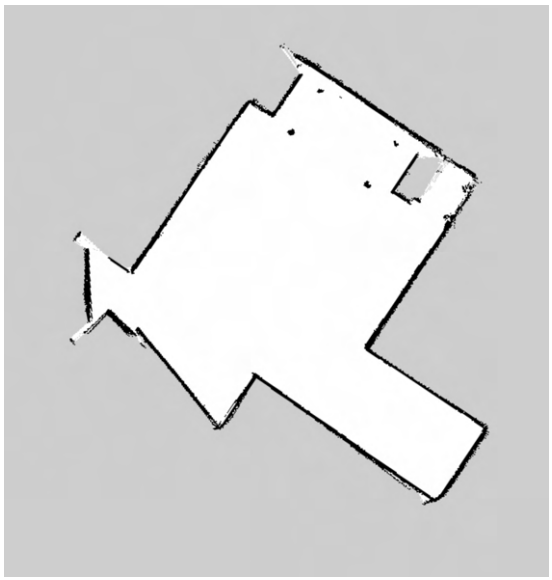
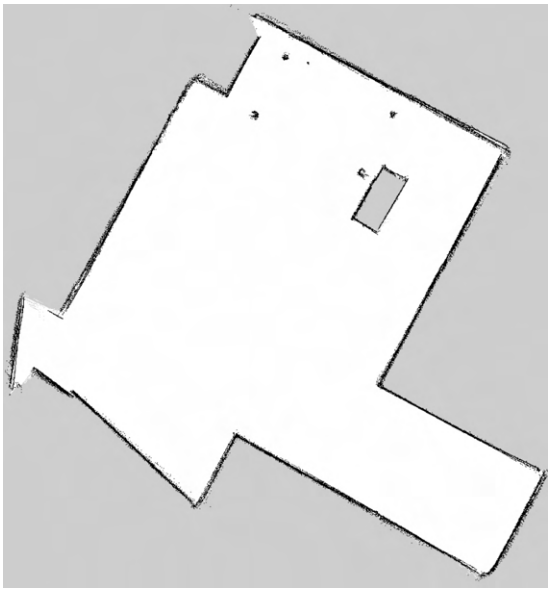


Figure 59: First Captured Map - Resolution 0.05m per pixel

Through testing, we discovered that two main parameters greatly affect the suitability of any developed map for Navigation. 1) It's resolution 2) Rotational of our robot during mapping. Two further iterations of mapping can be seen in Figure 60.



(a) Resolution 0.01m per pixel



(b) Resolution 0.005m per pixel

Figure 60: Subsequent Maps - Both 0.01 and 0.005 provide a good revolution for obstacle detection.

The effect of the robot rotational speed can be seen in Figure 61. We believe one reason for this is the delay between the canonical scan matcher calculating the odom → base_link transform. This could mean the laser scan is registered as part of the slam algorithm at an odometry position of a time step in the past.

Imagine we place the robot in the centre of the room and instruct it to rotate about its centre whilst simultaneously running the slam algorithm. If there is a delay between the laser scan being received and the odom → base_link transform being updated then

features detected as obstacles will be marked in slightly the wrong place. Due to the effect of angular velocity, there will be a greater error in (x, y) positions for those obstacles which are further away from the centre of the robot. This can give way to features appearing as transition zones on produced maps rather than discrete lines. The rotational speed of the robot was set to be 1/3 of that of the linear speed to account for this effect.

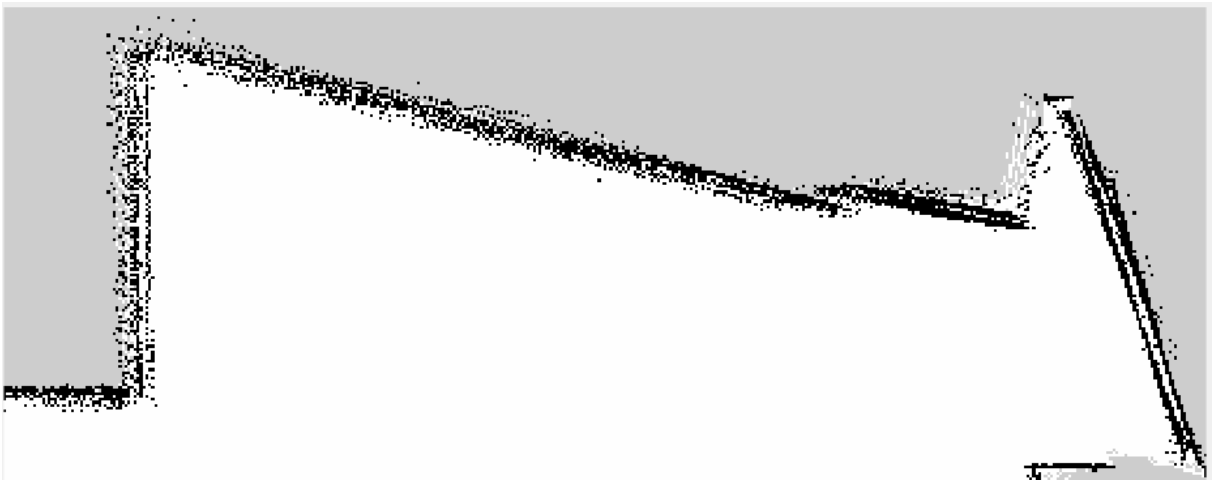


Figure 61: Effect of increased speed on map

The map used for our final pipeline implementation, Section 5.7, can be seen in Figure 62. In this map, a large object has been added in the centre of the room. This is a suitcase. It was added to ensure we ruggedly test the Navigation system's obstacle avoidance and path planning. In addition, this map includes the corridor, again this was mapped to allow for more rugged testing i.e. navigating past sharp corners.

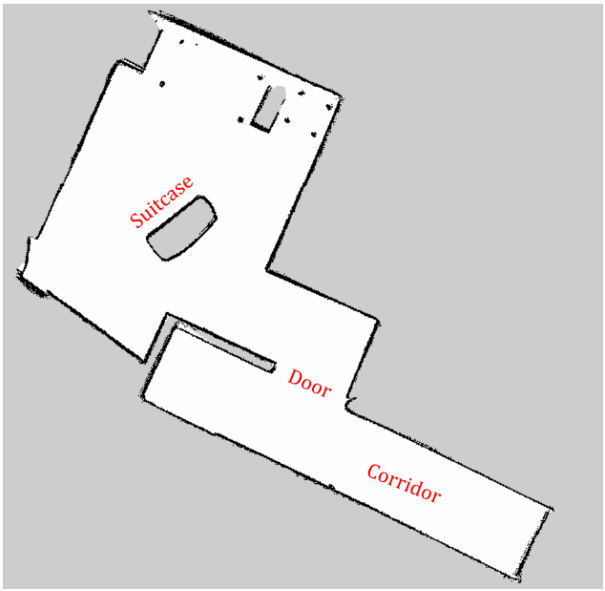


Figure 62: Final Map used for Navigation, Resolution 0.01m per pixel

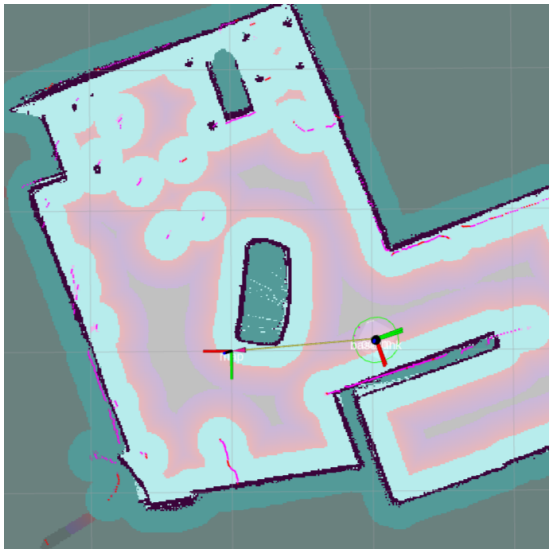
This concludes the experimental methods for implementing joystick SLAM on our robot.

4.10 Robot Navigation

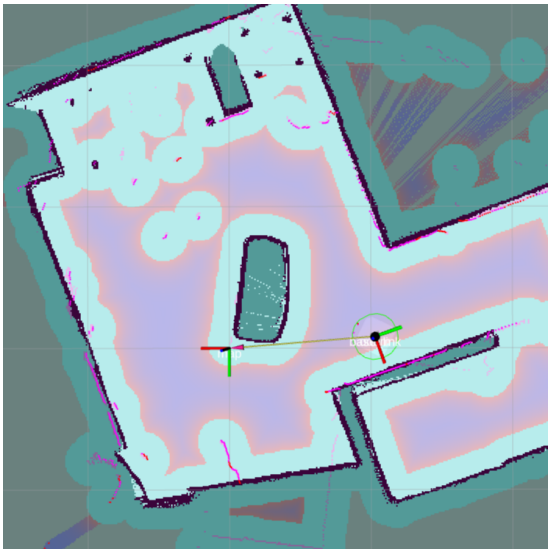
Nav2 is the professionally supported ROS2 navigation stack. Nav2 handles the path planning and outputs the required body velocities to follow the planned path.

To allow Nav2 to control our robot, we first needed to remap the outputted command velocities such that they were published on the topic expected by the custom controller. Once the output topic was remapped the command velocities produced by Nav2 were received by the controller and the robot was able to follow the produced path.

What remained was to tweak the parameters of Nav2 to suit our application. Figure 63 shows two variations of cost maps. Changing the inflation_layer parameters changes how close the planned path will travel near obstacles. We settled on a cost_scaling_factor of 5 and an inflation_radius of 2 after field-testing the navigation. The unshaded part of the map, present in Figure 63a, is considered a safe area by Nav2. Nav2 will always aim to plan a path that is equidistant from all the unsafe zones (light blue). The parameters 'inflation_radius' and 'cost_scaling_factor' affect how far the transition zone (purple) stretches from the unsafe zone. One should aim to create a constant area of purple shading over the parts of the map which are safe as we want the planning algorithm to know that it can travel anywhere in this area. We can see that the area is not constant in Figure 63a. This means that when using these settings the algorithm tends to plan a less smooth path that hugs the boundary between the safe area (unshaded) and the transition area (purple). This can lead to the robot often coming closer than ideal to objects and a more jittery advancement to the goal.



(a) cost_scaling_factor=5.0, inflation_radius=0.35



(b) cost_scaling_factor=10.0, inflation_radius=0.55

Figure 63: Investigating how modifying the 'inflation_layer' parameters of Nav2's global costmap affects the planned path

Figure 64 shows the smooth planned path due to the constant level of purple shading in the safe areas of the map. The final inflation_layer parameters chosen were the same as those used when creating Figure 63 and Figure 64.

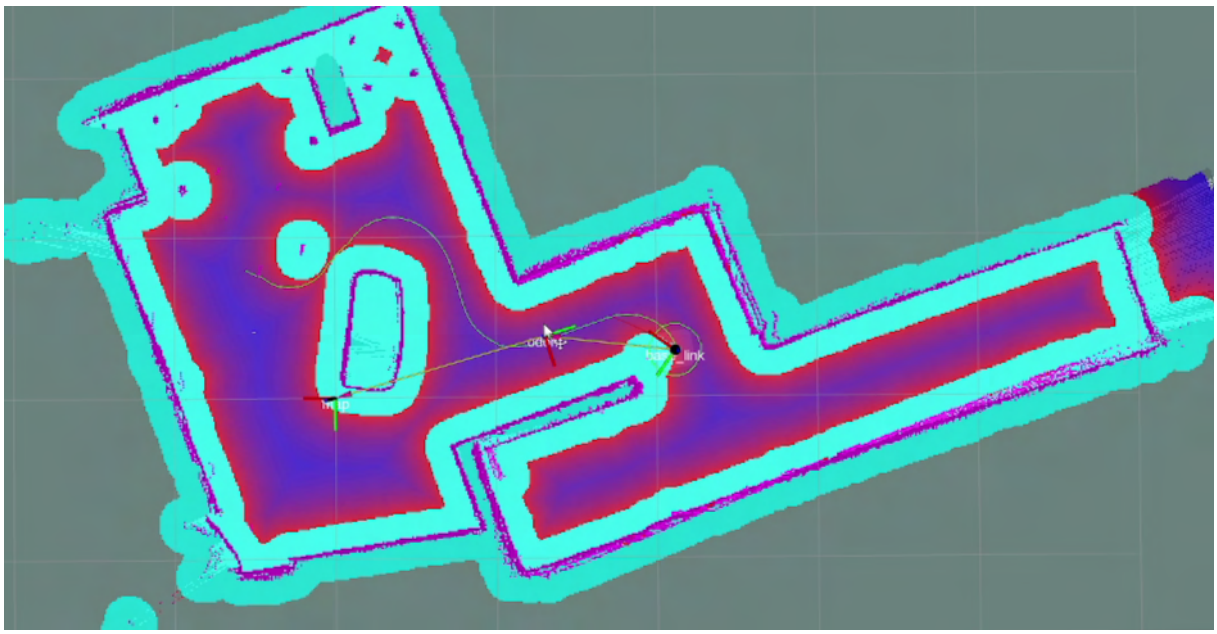


Figure 64: cost_scaling_factor: 5.0, inflation_radius = 0.5, (map contrast has been altered to better see the path)

5 Results

In this section, we critically evaluate how our implementation has solved the problems laid out in the goals and objectives.

5.1 Robot Redesign

The robot was successfully redesigned as outlined in Section 4.1.

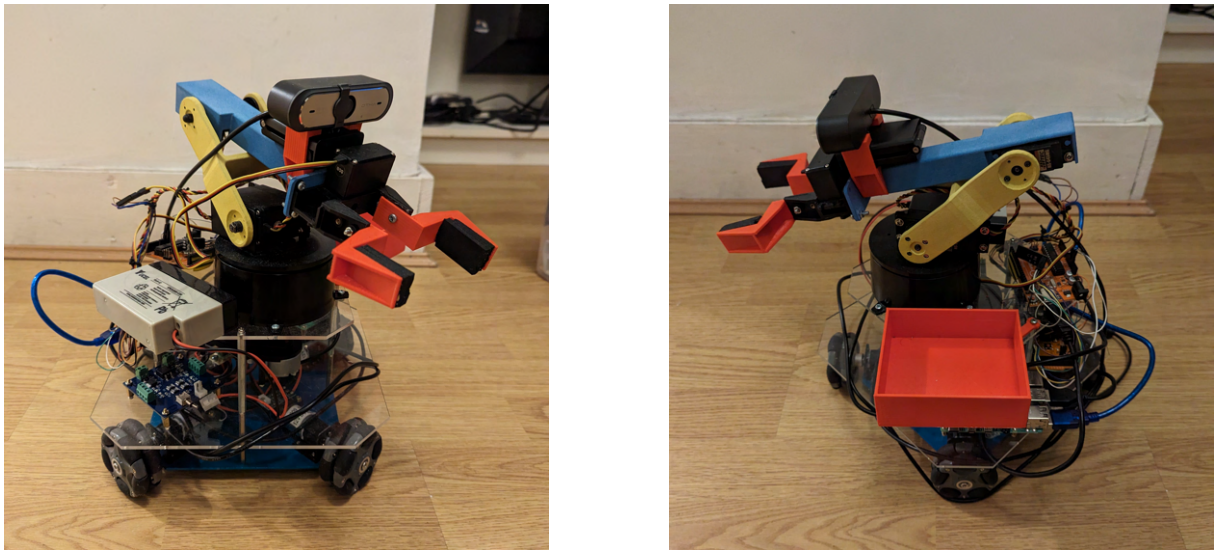


Figure 65: Final Design of the robot

5.2 Hand Gesture Control (Manual mode)

Hand gesture control is successfully implemented with the camera on the working station, the user can control the robot by simply sitting in front of a control camera and using hand gestures.

Firstly, the algorithm can correctly identify the priority hand with 90% accuracy when both hands are present in the frame. Secondly, all arm motors can be simultaneously controlled at the same time with the user's right hand with 85% accuracy. Example right-hand angles are shown in Figure 67. Thirdly, when the left hand is a priority, the robot can stop, move forward, backward, and rotate left and right by the designed left-hand commands with 90% accuracy. Finally, it was agreed that pictures are not sufficient in presenting the results and capabilities of hand gesture control, a supporting video to show the full hand gesture control of the robot is available at [1].

(a) Robot arm mimics the user's right wrist

(b) Wrist upright with fingers up

(c) Wrist leaned back with fingers down

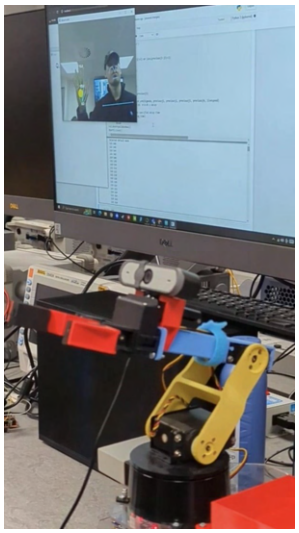
(d) Wrist laid back with fingers up

Figure 66: Hand gesture control results

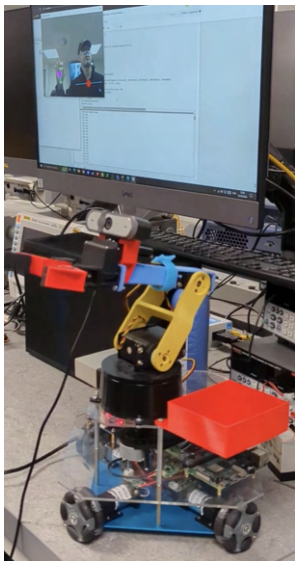
5.3 Holistic and person tracking (Default mode)

Holistic model and tracking are implemented successfully; the robot can be controlled via the robot's camera.

Firstly, simple tracking can track a manoeuvring hand ('come' command) successfully with 95% accuracy. Secondly, when 'fingers_straight' and palm facing the camera, the button shows and can be activated 100% of the time. Thirdly, automatic tracking and stopping are successful with 95% accuracy. Finally, again since pictures cannot represent the full potential and capabilities of the tracking functions, a supporting video showcases is available at [2]. This showcases the automatic tracking's activation (when the button is shown and grabbed), actual tracking (robot following the user) and stopping (cancel the tracking by showing 'stop' command or when the person is too close).

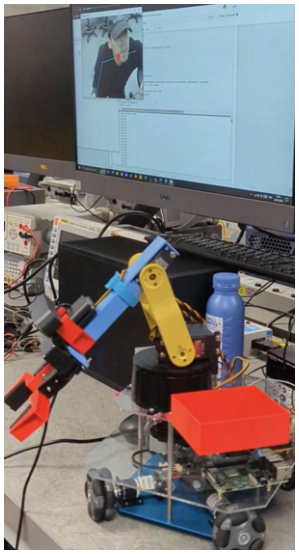


(a) Button rendered on screen



(b) Button grabbed, tracking initiated

(c) Robot base and arm adjusted to point to user



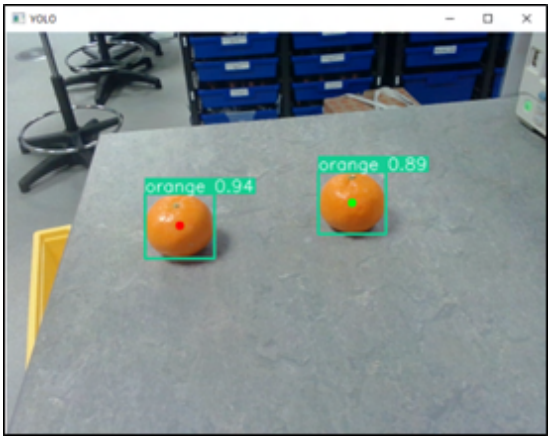
(d) User ducked down and robot tracked down

Figure 67: Automatic tracking results

5.4 Object Detection

After training the YOLOv5 object detection model, it was important to evaluate the efficiency and accuracy of the model. The Figures below demonstrate the object detection model confidence scores, which represent how confident the model is that the identified objects match the label, with values closer to 1 having the highest confidence:

(a) Phone, Fork, Mouse, Can



(b) Orange

Figure 68: Real-Time Object Detection

From Figures 68a and 68b, it can be seen that the YOLOv5 model can detect various household objects with high confidence, overall showing precise identification abilities of the model in real-world scenarios. At first, we had the intention of training more classes so that even more household objects would be able to be detected, however, instead of 100 images per class as we have done for the 5 above objects, we attempted to train 13 classes at 35 images per class. This resulted in incorrect objects being detected with low confidence scores as seen in Figure 69.

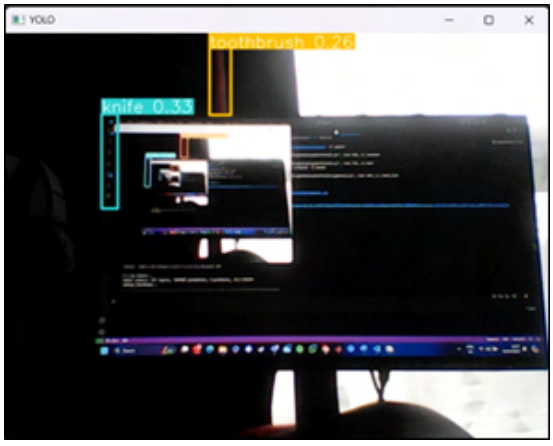


Figure 69: Real-Time Object Detection (Older Model)

From Figure 69, it can be seen that objects are being detected as other objects, with low confidence scores. For example, a crease in the curtains is considered a toothbrush. This inaccurate and poor model is a result of a lack of diversity and quantity in the training data, as the model needs to learn from a large and varied dataset, with variations in

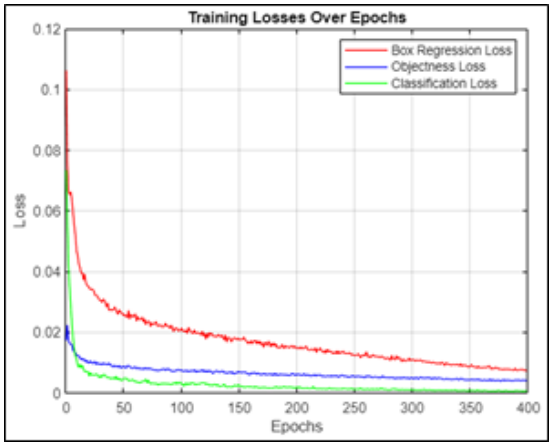
orientation, lighting, and background, to successfully detect objects in real-time. These factors were considered when the latest training run was conducted, resulting in accurate object detections, with constant confidence scores above 0.8.
As mentioned in Section 4.2, once a training run has been conducted, a corresponding 'exp (number)' file is created, containing two folders called 'results' and 'weights'. The 'results' folder will now be discussed. To evaluate the success of an object detection model, it is important to know the different types of losses and parameters used to measure the performance during training. Table 1 shows these values as the number of epochs increases:

Epochs	Train/box_{loss}	Train/obj_{loss}	Train/cls_{loss}	Metrics/Precision	Metrics/Recall	Metrics/$mAP_{0.5}$	Metrics/$mAP_{0.5:0.95}$	Val/box_{loss}	Val/obj_{loss}	Val/cls_{loss}
0	0.10621	0.01715	0.07334	0.006626	0.8	0.032264	0.010694	0.068105	0.00956	0.05025
1	0.076082	0.02234	0.05731	0.15516	0.31541	0.15239	0.051111	0.057046	0.01185	0.03525
2	0.066255	0.01774	0.04241	0.58135	0.46667	0.44612	0.14374	0.054809	0.01223	0.02549
3	0.065296	0.01611	0.03455	0.3785	0.61966	0.46083	0.14739	0.055322	0.00730	0.01554
4	0.066241	0.01612	0.02859	0.24367	0.51111	0.2884	0.1153	0.05854	0.00669	0.01129
5	0.063174	0.01574	0.02223	0.38355	0.75556	0.52577	0.1529	0.055134	0.00632	0.00806
6	0.059199	0.01371	0.01794	0.61209	0.89676	0.76147	0.29764	0.047552	0.00625	0.00695
7	0.055067	0.01381	0.01431	0.68031	0.81339	0.81951	0.31673	0.043652	0.00566	0.00683
8	0.051922	0.01317	0.01120	0.65189	0.8742	0.71231	0.38981	0.038782	0.00501	0.00544
9	0.046831	0.01233	0.01107	0.54817	0.97778	0.84088	0.4493	0.033501	0.00522	0.00441
...	...	...	...	...	...	...	...	...	...	...
397	0.0076679	0.00396	0.00043	0.99317	1	0.995	0.93424	0.006234	0.00623	0.00031
398	0.0073561	0.00416	0.00059	0.99322	1	0.995	0.92984	0.006282	0.00626	0.00031
399	0.007175	0.00390	0.00046	0.99311	1	0.995	0.93236	0.006348	0.00624	0.00031

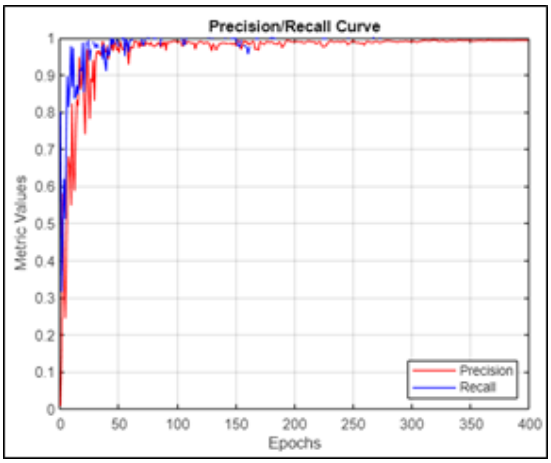
Table 1: Training Progression and Validation Metrics for YOLOv5 Model

Each parameter in Table 1 is important when evaluating the performance of the object detection model. The definition of each parameter can be seen in the Appendix 9.6.

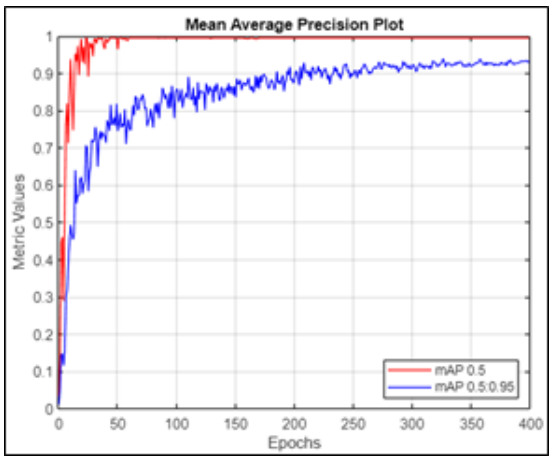
Having obtained the values for several parameters, as seen in Table 1, it is possible to create multiple plots that will allow the performance of the object recognition model to be evaluated. These plots can be seen in Figure 70.



(a) Training Losses against Epochs



(b) Precision/Recall Curve



(c) Mean Average Precision

(d) Validation Losses against Epochs

Figure 70: Performance Evaluation for YOLOv5 Model

From Figure 70a, the values of the box regression, objectness loss, and classification loss can be throughout the training epochs. At first, all three losses rapidly decrease, showing that the model is quickly learning from the training data, and is learning to predict the location and category of the objects accurately. However, as the epochs increase above 50 epochs, the rate of decline of the losses starts to plateau, suggesting that the model is reaching its capacity with the given training dataset. Increasing the number of epochs further could eventually result in very small losses, however, this could cause overfitting, meaning the model has been trained to the dataset too well. This means that the model would give accurate predictions for the training data, but not for new data [20].

From Figure 70b, the values for the precision and recall metrics can be seen to rapidly increase with training epochs, demonstrating that the model is accurately identifying and classifying objects, whilst also detecting all relevant instances. The continuous growth in precision and recall shows the accuracy and effectiveness of the object recognition model.

From Figure 70c, similar to the precision and recall plot, the mean average precision for thresholds above 50% and between 95% rapidly increases throughout the training epochs. This shows that the bounding boxes assigned by the model are successfully overlapping with the actual object. Consistently high mAP values, values close to 1, show that

the model is accurate and robust for objects with different orientations, lighting and backgrounds, thus demonstrating that the model is accurate.

From Figure 70d, similar to Figure 70a, the losses rapidly decrease until they begin to plateau, and as the validation dataset is different to the training dataset, it shows that the model is learning general patterns that could be applied to new data in real-life scenarios. The losses in Figure 6d show greater stabilisation than the losses in Figure 6a, which could suggest overfitting, however, this could easily be improved by training with a lower number of epochs.

Based on Figure 70, the model appears to perform well, with training and validation losses reducing with time, indicating good generalisation for new data. The precision, recall and mAP values are all high after training for 400 epochs, showing that the model is predicting and classifying objects accurately. Plots alone are unable to evaluate the performance of the model in real-life scenarios, however Figures 4a and 4b show that even in real-time object detection, objects are being identified and classified with confidence scores above 0.8, furthermore showing the reliability and robustness of the model.

The results show the model's accuracy and precision, which could set a benchmark for other object detection models in the field to compare against. The model's adaptability to various conditions demonstrates the robustness of the model, hinting at future iterations of YOLO models delivering increased accuracy and training speeds.

5.5 Voice implementation

The voice recognition was set to continuously run from the start and still be listening at times when in a certain mode to see if a 'quit' command is said. If out of a mode, and the 'quit' command is heard, the robot would turn off.

In Figure 71a, the command was recognised as 'default' so that mode was selected. In Figure 71b, the 'grab' command was recognised, therefore the 'grabbing' mode was selected, the class 'orange' was recognised and so that class was selected in the YOLO model and was used to approach and grab the orange. In Figure 71c, the 'manual' command was said so that mode was selected and in Figure 71d, the 'quit' command was heard so that mode was selected which stopped the robot. These are examples of perfect usage of the intended controls.

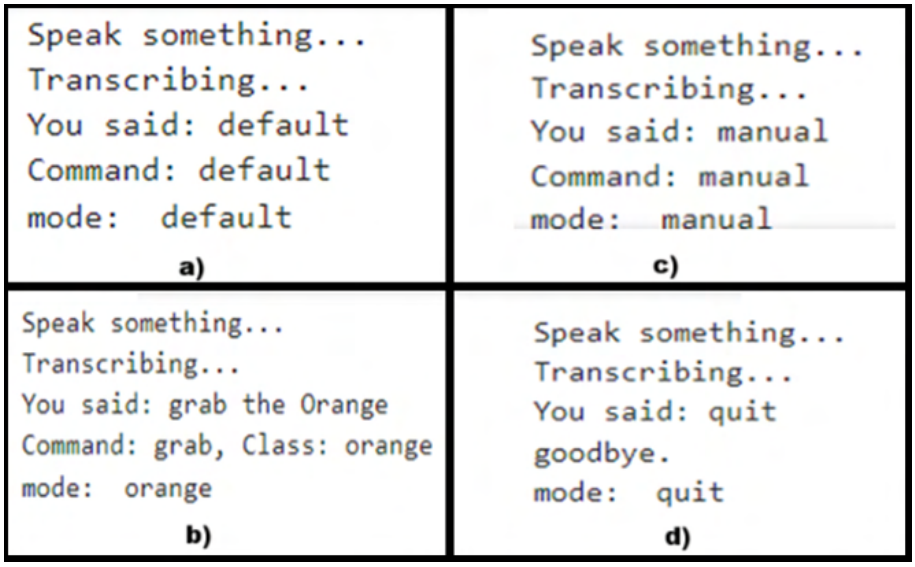


Figure 71: Voice Test (Command Words)

In Figure 72, the text-to-action was tested for the grabbing mode. In Figure 72a, the wrong command, 'hand', was said which prompted no command to be detected, therefore the class was not even sought. The robot then went back to listening. In Figure 72b, the right command was said but an untrained class was heard, here there is understanding about the command but the same outcome occurred.

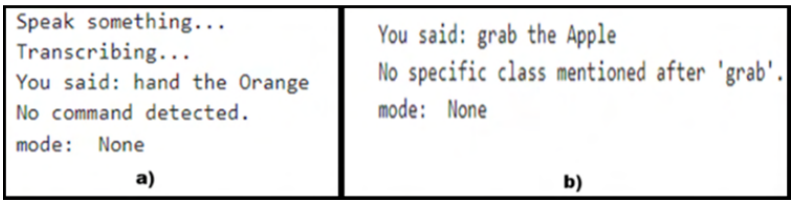


Figure 72: Voice Test (Wrong Command/Object)

In Figure 73, the behaviour is assessed on mistakes in speech and reconsideration of the mode or class. In Figure 73a, the command word 'grab' was said wrong on purpose and rectified later, the program knew to work on the latter command. In Figure 73b, the wrong class that is still in the YOLO model, 'can', was said and then changed to 'orange'. The program disregarded the can and went to grab the orange instead. In Figure 73c, a mode was selected but the change of mind to stop was shown, and the robot stopped.

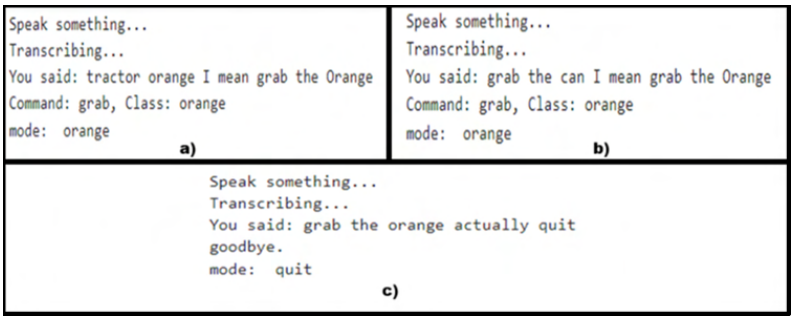


Figure 73: Voice Test (Changed Command)

5.6 Wireless Connectivity - RF

In Figure 74, the messages show weird characters and then nothing, this is due to a loss of communication. As mentioned in the methods section, when the 5V supply was connected, this result would appear, also when the ground bonding was not done appropriately. This prompted us to make many changes as described before until the result showed the instructions.

Figure 74: RF Error Message

Figure 75 shows the joint angles and base movements being sent with a 100ms delay as coded. The joint angles j1, j2, j3, and j4 are shown as the base movements. The instructions were sent at quick speeds and accurately, with no visible delay or error in movement. Movement even looked faster than the wired serial communication meaning there was no compromise and delay in movement when using wireless.

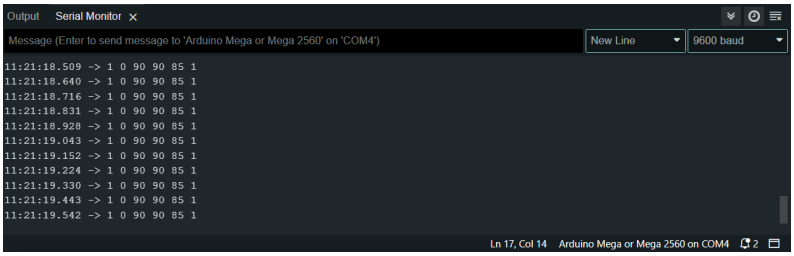


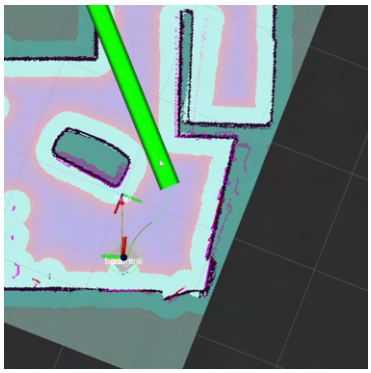
Figure 75: RF Joint Angles

5.7 Object Retrieval Pipeline

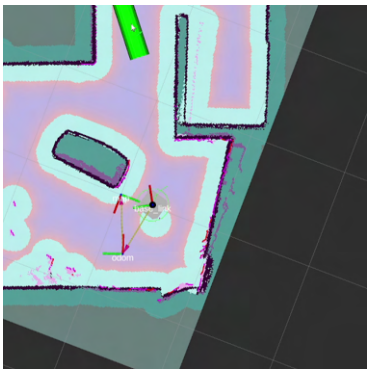
Below we outline the results obtained from our object retrieval pipeline.

5.7.1 Navigation to Object

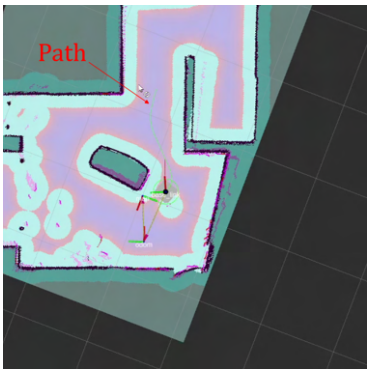
The first step in our pipeline is to get our robot to navigate to the desired object. For this to happen we have to satisfy the following conditions. 1) Know where the object is 2) Have an existing map of the robot's environment. Figure 76 contains screenshots taken from a field test of this pipeline where navigated using the pre-made map to a Coke can, picked up the can with the robotic arm, and navigated back to the initial position.



(a) Goal #1 set

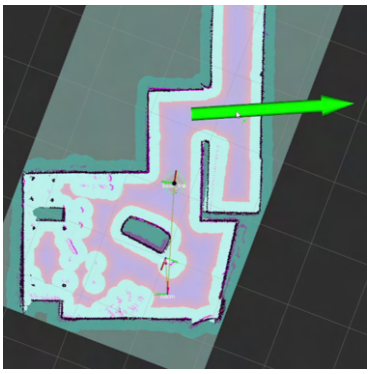


(b) Goal #1 reached and goal #2 set

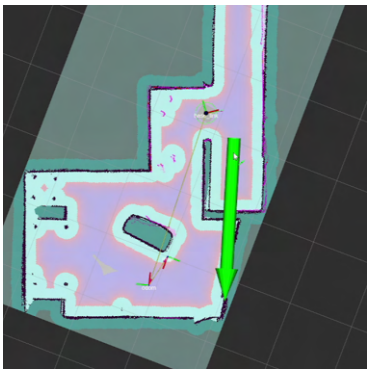


(c) Planned Path for Goal #2 can be seen (Highlighted)

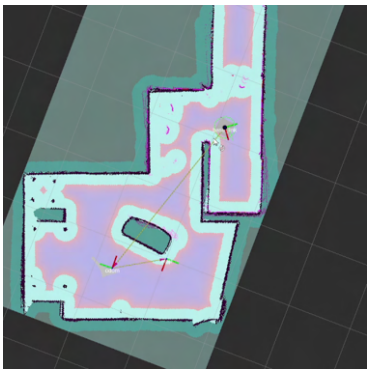
(d) Robot follows planned path and navigates past the suitcase



(e) Goal #2 is reached and goal #3 is set



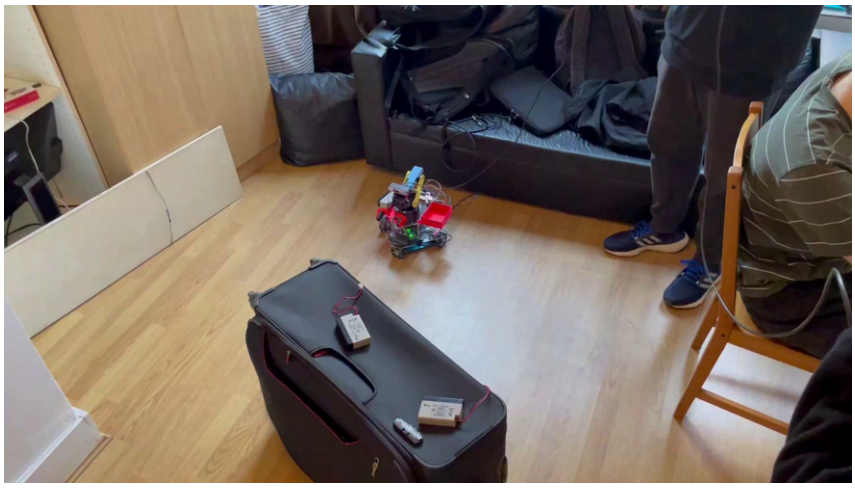
(f) Goal #3 reached and goal #4 set



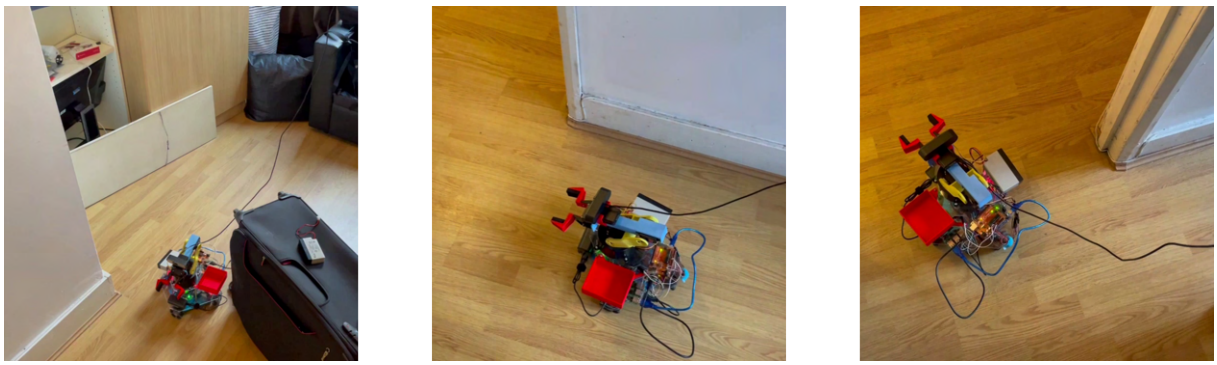
(g) Goal #4 is reached

Figure 76: Navigation to desired object to start position

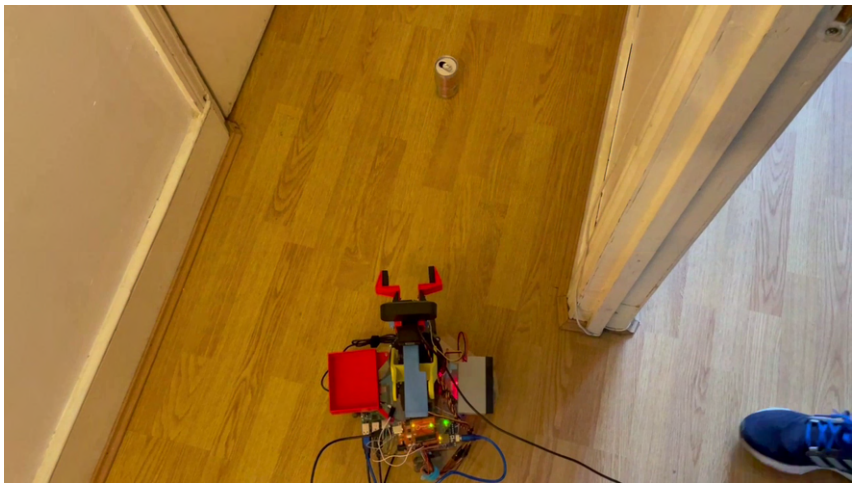
Figure 77 contains photos taken from the same field test shown previously through the lense of RViz2 in Figure 76



(a) Robot in initial position, corresponds to Figure 76a.



(b) Robot travelling the past suitcase and through the door corresponds to Figures 76b to 76f.

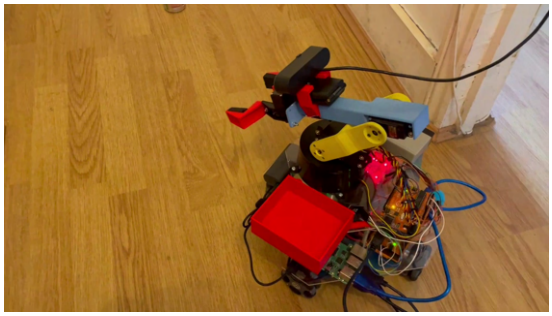


(c) Robot in final position, corresponds to Figure 76g.

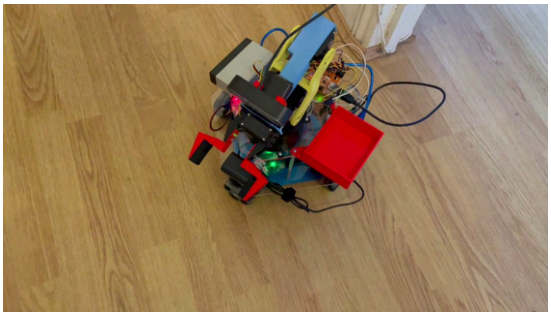
Figure 77: Navigation to desired object from start position

5.7.2 Initial scanning

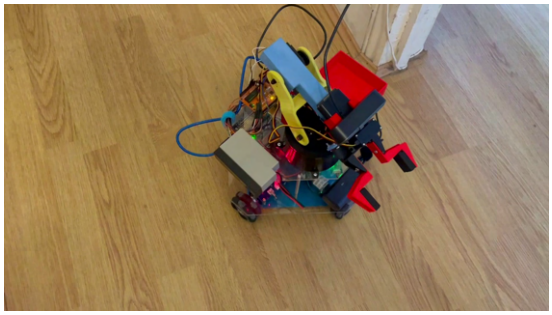
Once the robot reaches the vicinity of the desired object the object recognition subsystem is launched. The first thing the object recognition subsystem does is look for the desired object. If that desired object is not in the field of view, the camera performs an initial scan - the base of the robot rotates until the object is found. Once found, object tracking begins. The initial scan from the field test can be seen in Figure 78.



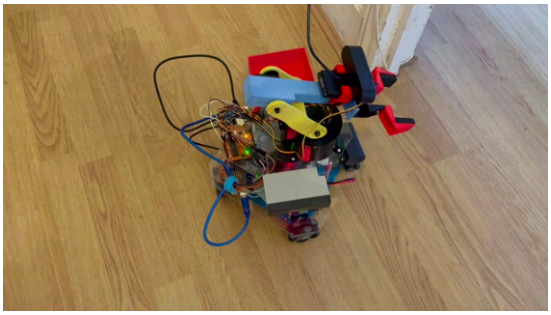
(a) Can not in view, starting initial scan



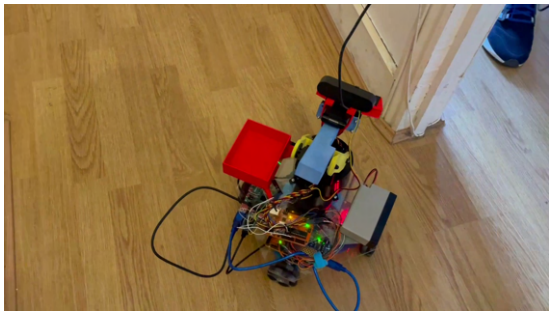
(b) Object not found, rotating



(c) Object not found, rotating



(d) Object not found, rotating



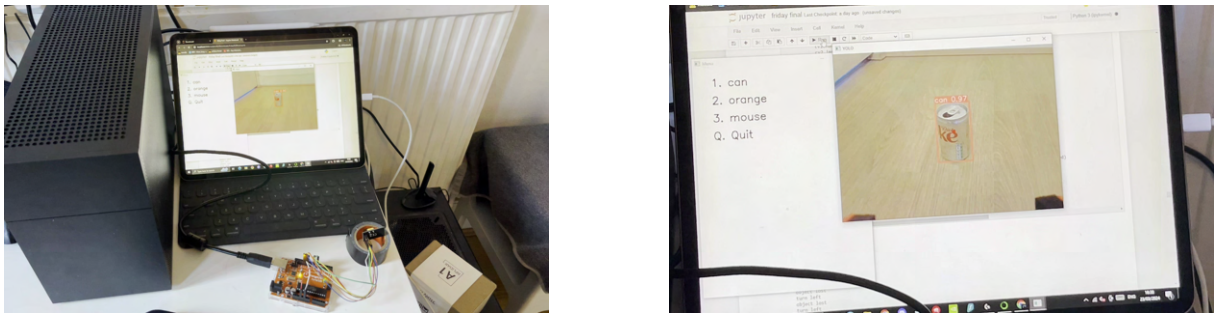
(e) Object not found, rotating

(f) Can found, start object tracking, begin approach

Figure 78: Initial Scan Process

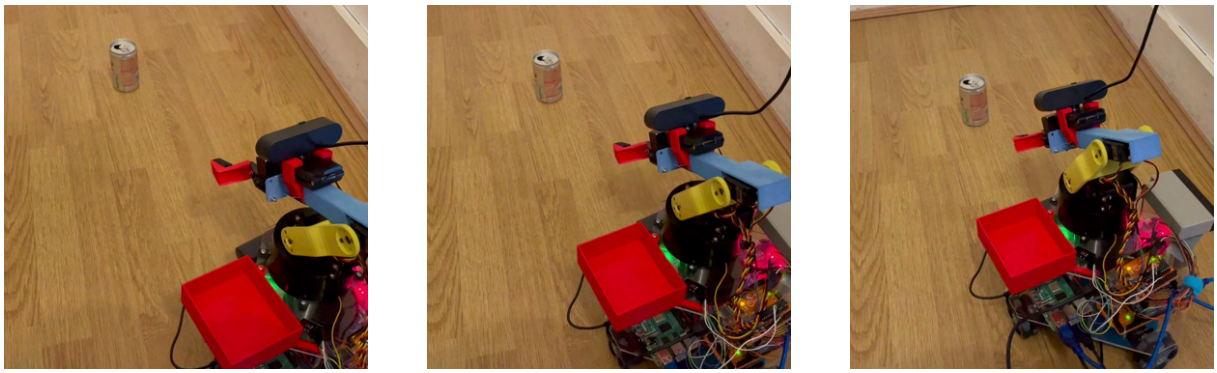
5.7.3 Object Tracking

Once the object has been identified by the initial scan, it needs to be approached. Figure 79 and 80 shows the object approach from the aforementioned field test. Figure 79 shows the object tracking from the user's POV. In its current implementation, the camera feed is being provided by a physical cable, as we were unable to get a wireless feed working, see Section 4.8.1.



(a) **(b)**

Figure 79: Object Tracking User POV



(a) Approaching object, happens directly after Figure 78f

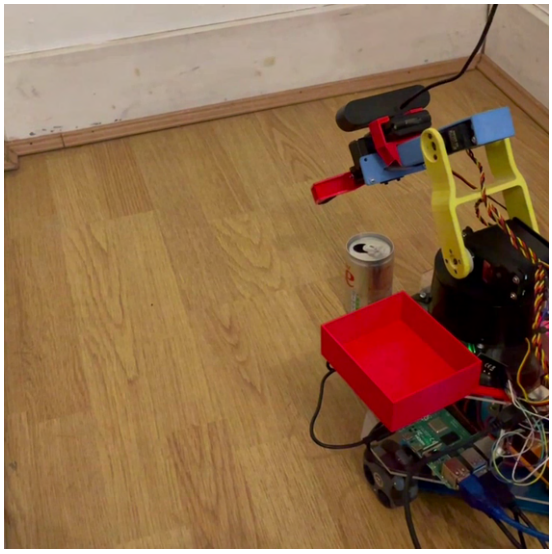
(b) Approaching object

(c) Reached distance to pick up object

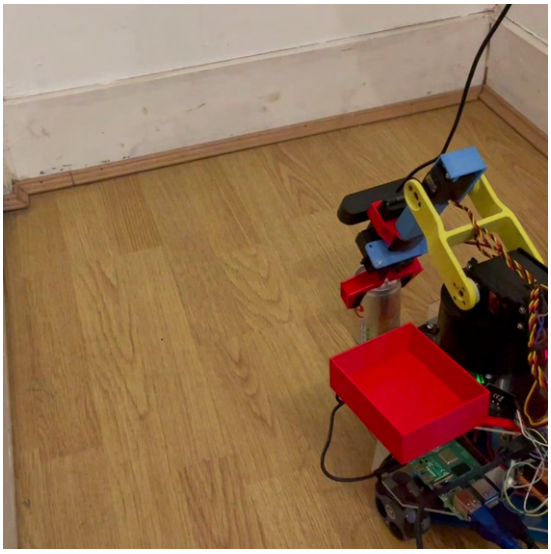
Figure 80: Approaching Object

5.7.4 Object Picking

Once the object has been approached, it needs to be picked up by the robot. Figure 81 outlines the object picking from the aforementioned field test.



(a) Arm moves to pick up object



(b) Gripper closes to grasp the can

(c) Arm retracts and begins to turn to drop the can in the box

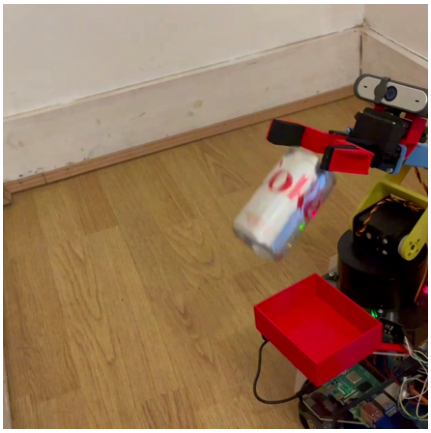
Figure 81: Object Picking

5.7.5 Object Placing

Once the object has been picked up it needs to be placed on the robot so it can be carried back to the user. Figure 82 shows the object placed from the aforementioned field test.

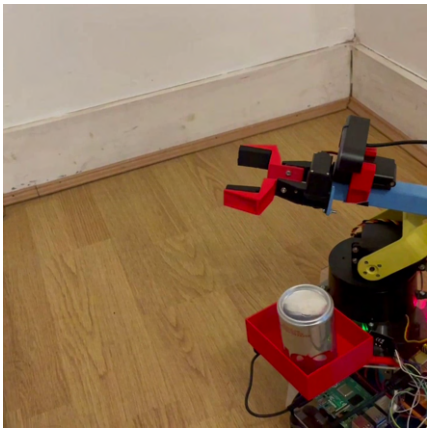
(a) Arm rotates to drop position

(b) Arm at drop position



(c) Dropping can

(d) Can in basket

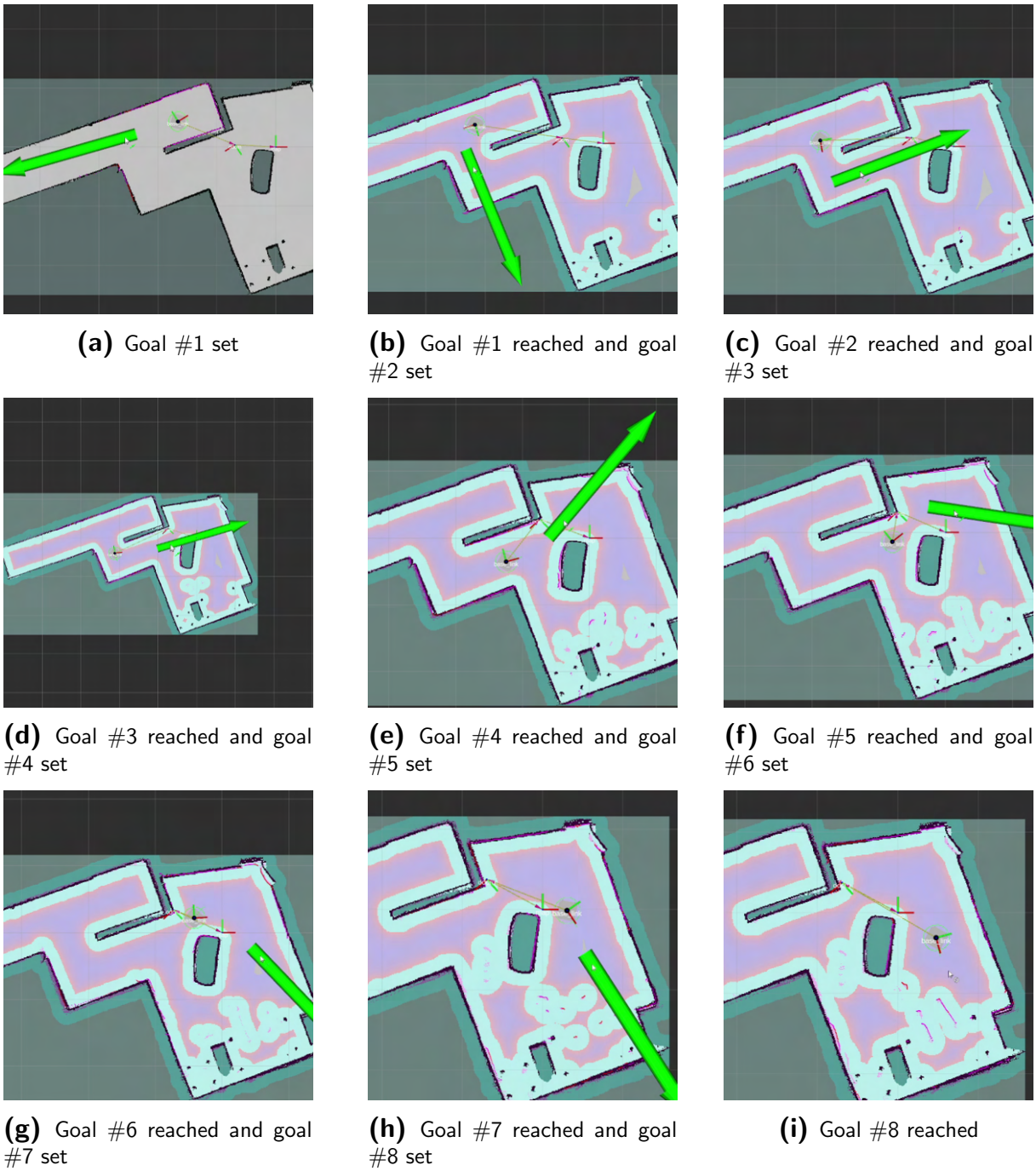


(e) Arm returns to original position

Figure 82: Object Placing

5.7.6 Navigation to Initial Position

Once the object has been placed on the robot we have to bring the robot back to its original position. Again, we do so by setting successive 2D Goal poses via the RViz2 application. Figure 83 shows the navigation to the initial position from the aforementioned field test.



(a) Goal #1 set

(b) Goal #1 reached and goal #2 set

(c) Goal #2 reached and goal #3 set

(d) Goal #3 reached and goal #4 set

(e) Goal #4 reached and goal #5 set

(f) Goal #5 reached and goal #6 set

(g) Goal #6 reached and goal #7 set

(h) Goal #7 reached and goal #8 set

(i) Goal #8 reached

Figure 83: Navigation from desired object to start position

This concludes the results obtained from the object retrieval pipeline.

6 Conclusion & Future Work

6.1 Conclusion

It is important to evaluate our work based on our goals and objectives. The goals and objectives listed in Section 2 represent an updated list, formed from the findings of our interim report.

Goal 1 of the project was to create a robot able to perform pick and place tasks. This goal is largely unchanged from that written in the project proposal. We identified this goal as being achieved in the interim report, but we believe we have built on this from then. We have strengthened our robot's ability to pick and place an object on the object holder onboard the robot all while enabling this to happen wirelessly.

Goal 2 of our project relates to the training and integration of a custom YOLO model for object recognition. This was an area we identified for improvement in the interim report. The initial model showed a maximum confidence score of 0.6 and included 13 classes, however, we opted to streamline the model, reducing the number of classes from 13 to 5. This way we could take more pictures of each class, increasing the average confidence score to above 0.8. This approach resulted in increased performance, allowing for the model to be utilised in our object retrieval pipeline as planned. In addition, Goal 2 relates to implementing hand gesture recognition. We identify this as one of the major successes of this project, achieving a general accuracy of above 80%+ in command and angle recognition, which results in a sophisticated control system of the robot's base and arm. Furthermore, holistic model automatic tracking system has been integrated, with an accuracy of 90%+, which allows for intuitive control of the robot.

Goal 3 of our project relates to our SLAM implementation. To this end, we have successfully implemented joystick SLAM on our robot. In addition, we have integrated our robot with the supported ROS2 Navigation toolbox, Nav2. In this way we have equipped our robot with means to navigate through a pre-mapped area. The notable achievements in this area are highlighted through our object retrieval pipeline results, Section 5.7. One key contribution in this area is the creation of the custom controller implemented in ROS. We believe this represents a potentially useful contribution to the open-source community.

Goal 4 of our project relates to our voice recognition implementation. It revised the goal from the project proposal to represent the group's accomplishments better. Our results show we have achieved both goals 4a and b. Through the utilisation of existing models, we were able to produce more accomplished implementations of other parts of the project leading to them being

Continued development shows potential to assist the disabled and elderly, and also to alleviate burdens on the understaffed social care sector.

6.2 Future Work

We have opted to provide multiple scopes of future work - short-term work, capable of being accomplished within 1 month, and long-term work if another group were to work on the project next year.

6.2.1 Short Term

6.2.1.1 Wireless Camera

As reported in Section 4.8.1 we attempted to integrate a wireless camera. This resulted in a bad frame rate especially when doing the object recognition. There was a need for a wired connection to the robot's camera to run our YOLO model. In doing so this stops our robot from being completely wireless. Successfully integrating a wireless camera would give the project a more rounded feel as we would not be physically tethered to any base station. One potential solution worth investigating could be the usage of USB transceivers between the computer and the robot.

6.2.1.2 Model

As highlighted in Section 5.4, improvements could be made in our custom YOLO model. If we had more time we would have liked to have increased the number of pictures taken from 100 to 1000. This 10-fold increase would allow for many different angles to be included in an item thereby increasing our model's robustness to real-world scenarios. It is reported that 1000 photos would achieve a good level of robustness [52].

6.2.1.3 Raspberry Pi Upgrade

An upgrade could be made to Raspberry Pi. An Nvidia Jetson AGX Xavier could be used instead. This may allow for object recognition to be performed onboard the robot, negating the need for a physical connection or wireless camera. This produced satisfactory results in a robot similar to ours, also using YOLOv5 [6].

6.2.2 Long Term

6.2.2.1 Robot Base & Arm

We were not able to implement true omnidirectional kinematics in this project, this would be a beneficial addition to the robot's mobility, allowing for diagonal movement and path tracking without the constant need for rotating. One scope for the future could be built on the custom controller, [43], by incorporating the true omnidirectional kinematics [44, 45]. Increasing the degrees of freedom of the arm would allow enhanced dexterity, increased flexibility, and potentially optimised path planning. This allows for manipulation strategies to allow for more objects to be picked from various angles. Comprehensive implementations of both would be long-term future work.

6.2.2.2 Application

In our current implementation, we have 2 'applications'. The first is the UI described in Section 6.2.2.2. The second is what we are calling our implementation of SLAM/Navigation RViz2. A more accomplished implementation would merge these two applications. The resultant application would consist of the current menu, showing the mapping of the room(s), allowing for control from the application using directional buttons and allowing the robot to move by clicking on that part of the map. A comprehensive implementation of this merged application would be long-term future work as it requires in-depth knowledge of both ROS/RViz2 and application building.

7 Bibliography

[1] *Hand Gesture Control (Manual)*. en. URL: https://mediacentral.ucl.ac.uk/Player/H10DC1Gg (visited on 04/23/2024).

[2] *Holistic Model (Default Mode)*. en. URL: https://mediacentral.ucl.ac.uk/Player/ffghi9bB (visited on 04/23/2024).

[3] *Ageing and health*. en. URL: https://www.who.int/news-room/fact-sheets/detail/ageing-and-health (visited on 04/20/2024).

[4] Widya A. Ramadhani and Wendy A. Rogers. "Understanding Home Activity Challenges of Older Adults Aging with Long-Term Mobility Disabilities: Recommendations for Home Environment Design". In: *Journal of Aging and Environment* 37.3 (July 2023). Publisher: Routledge _eprint: https://doi.org/10.1080/26892618.2022.2092929, pp. 341–363. ISSN: 2689-2618. DOI: 10.1080/26892618.2022.2092929. URL: https://doi.org/10.1080/26892618.2022.2092929 (visited on 04/20/2024).

[5] Christoph Ohneberg et al. "Assistive robotic systems in nursing care: a scoping review". In: *BMC Nursing* 22.1 (Mar. 2023), p. 72. ISSN: 1472-6955. DOI: 10.1186/s12912-023-01230-y. URL: https://doi.org/10.1186/s12912-023-01230-y (visited on 04/20/2024).

[6] B. M. Sufiyan Ali et al. "ROS Based Autonomous Mobile Manipulator Robot". en. In: Atlantis Press, Nov. 2023, pp. 780–789. ISBN: 978-94-6463-252-1. DOI: 10.2991/978-94-6463-252-1_78. (Visited on 04/11/2024).

[7] Ren C. Luo et al. "Modular ROS Based Autonomous Mobile Industrial Robot System for Automated Intelligent Manufacturing Applications". In: *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. July 2020, pp. 1673–1678. DOI: 10.1109/AIM43001.2020.9158800. URL: https://ieeexplore.ieee.org/document/9158800 (visited on 04/11/2024).

[8] Jakub Czygier et al. "Autonomous Searching Robot with Object Recognition Based on Neural Networks". In: *International Journal of Mechanical Engineering and Robotics Research* (Jan. 2020), pp. 1347–1352. DOI: 10.18178/ijmerr.9.9.1347-1352.

[9] Le Phuong and Vo Cong. "Control the robot arm through vision-based human hand tracking". en. In: *FME Transactions* 52.1 (2024), pp. 37–44. ISSN: 1451-2092, 2406-128X. DOI: 10.5937/fme2401037P. URL: https://scindeks.ceon.rs/Article.aspx?artid=1451-20922401037P (visited on 04/11/2024).

[10] Christian Diego Allena, Ryan Collin De Leon, and Yung-Hao Wong. "Easy Hand Gesture Control of a ROS-Car Using Google MediaPipe for Surveillance Use". en. In: *HCI in Business, Government and Organizations*. Ed. by Fiona Fui-Hoon Nah and Keng Siau. Cham: Springer International Publishing, 2022, pp. 247–260. ISBN: 978-3-031-05544-7. DOI: 10.1007/978-3-031-05544-7_19.

[11] Marthed Wameed, Ahmed M. Alkamachi, and Ergun Erçelebi. "Tracked Robot Control with Hand Gesture Based on MediaPipe". en. In: *Al-Khawarizmi Engineering Journal* 19.3 (Sept. 2023). ISSN: 1818-1171, 2312-0789. DOI: 10.22153/kej.2023.04.004. URL: https://alkej.uobaghdad.edu.iq/index.php/alkej/article/view/843 (visited on 04/11/2024).

[12] Soumyadip Chatterjee. "Design and Implementation of an Automated Hand Gesture based Powerchair using a Robotic ARM and IoT Integration". en. In: 08.08 (2021).

[13] Abdullah Shaif, Suresh Gobee, and Vickneswari Durairajah. "Vision and voice-based human-robot interactive interface for humanoid robot". In: *AIP Conference Proceedings* 2788.1 (July 2023), p. 030002. ISSN: 0094-243X. DOI: 10.1063/5.0148670. URL: https://doi.org/10.1063/5.0148670 (visited on 04/11/2024).

[14] Sudeep Sharan et al. "Implementation and Testing of Voice Control in a Mobile Robot for Navigation". en. In: *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. Hong Kong, China: IEEE, July 2019, pp. 145–150. ISBN: 978-1-72812-493-3. DOI: 10.1109/AIM.2019.8868892. URL: https://ieeexplore.ieee.org/document/8868892/ (visited on 04/11/2024).

[15] Issa AR Djinko and Thabet Kacem. "Video-based Object Detection Using Voice Recognition and YoloV7". en. In: *The Twelfth International Conference on Intelligent Systems and Applications (INTELLI 2023)* (Jan. 2023). URL: https://par.nsf.gov/biblio/10433657-video-based-object-detection-using-voice-recognition-yolov7 (visited on 04/11/2024).

[16] *mediapipe/docs/solutions/holistic.md at master · google/mediapipe*. en. URL: https://github.com/google/mediapipe/blob/master/docs/solutions/holistic.md (visited on 04/23/2024).

[17] Imarticus. *Object detection and its Real-World Applications*. en-US. Running Time: 291. Aug. 2023. URL: https://imarticus.org/blog/object-detection-and-its-real-world-applications/ (visited on 04/13/2024).

[18] Sakshi Khanna. *A Comprehensive Guide to Train-Test-Validation Split in 2024*. en. Nov. 2023. URL: https://www.analyticsvidhya.com/blog/2023/11/train-test-validation-split/ (visited on 04/13/2024).

[19] *Train Test Validation Split: How To & Best Practices [2023]*. en. URL: https://www.v7labs.com/blog/train-validation-test-set,%20https://www.v7labs.com/blog/train-validation-test-set (visited on 04/13/2024).

[20] *What is Overfitting? — IBM*. en-us. URL: https://www.ibm.com/topics/overfitting (visited on 04/13/2024).

[21] *How Compute Accuracy For Object Detection works—ArcGIS Pro — Documentation*. URL: https://pro.arcgis.com/en/pro-app/latest/tool-reference/image-analyst/how-compute-accuracy-for-object-detection-works.htm (visited on 04/13/2024).

[22] *Precision-Recall*. en. URL: https://scikit-learn/stable/auto_examples/model_selection/plot_precision_recall.html (visited on 04/13/2024).

[23] *What is Epoch in Machine Learning?— UNext*. en. Section: Artificial Intelligence & Machine Learning. Nov. 2022. URL: https://u-next.com/blogs/machine-learning/epoch-in-machine-learning/ (visited on 04/13/2024).

[24] Anthony Zhang (Uberi). *SpeechRecognition: Library for performing speech recognition, with support for several engines and APIs, online and offline*. URL: https://github.com/Uberi/speech_recognition#readme (visited on 04/02/2024).

[25] *Understanding topics — ROS 2 Documentation: Foxy documentation*. URL: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html (visited on 04/02/2024).

[26] *ros-controls/roscon2022_workshop*. Mar. 2024. URL: https://github.com/ros-controls/roscon2022_workshop (visited on 04/02/2024).

[27] *Nav2 — Nav2 1.0.0 documentation*. URL: https://navigation.ros.org/index.html (visited on 04/02/2024).

[28] *diff_drive_controller — ROS2_Control: Rolling Apr 2024 documentation*. URL: https://control.ros.org/master/doc/ros2_controllers/diff_drive_controller/doc/userdoc.html (visited on 04/02/2024).

[29] *ackermann_steering_controller — ROS2_Control: Rolling Apr 2024 documentation*. URL: https://control.ros.org/master/doc/ros2_controllers/ackermann_steering_controller/doc/userdoc.html (visited on 04/02/2024).

[30] *bicycle_steering_controller — ROS2_Control: Rolling Apr 2024 documentation*. URL: https://control.ros.org/master/doc/ros2_controllers/bicycle_steering_controller/doc/userdoc.html (visited on 04/02/2024).

[31] *Writing a new controller — ROS2_Control: Rolling Apr 2024 documentation*. URL: https://control.ros.org/master/doc/ros2_controllers/doc/writing_new_controller.html (visited on 04/02/2024).

[32] *Creating a package — ROS 2 Documentation: Foxy documentation*. URL: https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html#what-is-a-ros-2-package (visited on 04/09/2024).

[33] *Controller Manager — ROS2_Control: Rolling Apr 2024 documentation*. URL: https://control.ros.org/master/doc/ros2_control/controller_manager/doc/userdoc.html (visited on 04/02/2024).

[34] *URDF — ROS 2 Documentation: Humble documentation*. URL: https://docs.ros.org/en/humble/Tutorials/Intermediate/URDF/URDF-Main.html (visited on 04/09/2024).

[35] Articulated Robotics. *Using ros2_control to drive our robot (off the edge of the bench...)* Oct. 2022. URL: https://www.youtube.com/watch?v=4VVrTCnxvSw (visited on 04/02/2024).

[36] *ROS Index*. URL: https://index.ros.org/r/teleop_twist_keyboard/ (visited on 04/05/2024).

[37] *REP 105 – Coordinate Frames for Mobile Platforms (ROS.org)*. URL: https://www.ros.org/reps/rep-0105.html#coordinate-frames (visited on 02/12/2024).

[38] *ROS/Tutorials/MultipleMachines - ROS Wiki*. URL: https://wiki.ros.org/ROS/Tutorials/MultipleMachines (visited on 04/09/2024).

[39] *ROS/NetworkSetup - ROS Wiki*. URL: https://wiki.ros.org/ROS/NetworkSetup (visited on 02/17/2024).

[40] *LiDAR Sensors*. en. URL: https://www.iadiy.com/LIDAR-sensors (visited on 04/08/2024).

[41] *RPLIDAR A1 Low Cost 360 Degree Laser Range Scanner Introduction and Datasheet*. URL: https://www.slamtec.ai/wp-content/uploads/2023/11/LD108_SLAMTEC_rplidar_datasheet_A1M8_v3.0_en.pdf (visited on 04/08/2024).

[42] *What is YOLOv5? A Guide for Beginners*. en. June 2020. URL: https://blog.roboflow.com/yolov5-improvements-and-evaluation/ (visited on 04/13/2024).

[43] eddie50488. *eddie50488/omni_drive_controller*. original-date: 2024-03-10T18:40:09Z. Mar. 2024. URL: https://github.com/eddie50488/omni_drive_controller (visited on 04/08/2024).

[44] Masaaki Hijikata, Renato Miyagusuku, and Koichi Ozaki. "Wheel Arrangement of Four Omni Wheel Mobile Robot for Compactness". en. In: *Applied Sciences* 12.12 (Jan. 2022), p. 5798. ISSN: 2076-3417. DOI: 10.3390/app12125798. URL: https://www.mdpi.com/2076-3417/12/12/5798 (visited on 04/05/2024).

[45] Chengcheng Wang et al. "Trajectory Tracking of an Omni-Directional Wheeled Mobile Robot Using a Model Predictive Control Strategy". en. In: *Applied Sciences* 8.2 (Feb. 2018), p. 231. ISSN: 2076-3417. DOI: 10.3390/app8020231. URL: https://www.mdpi.com/2076-3417/8/2/231 (visited on 04/05/2024).

[46] eddie50488. *eddie50488/omni_drive_arduino*. original-date: 2024-03-10T18:37:20Z. Mar. 2024. URL: https://github.com/eddie50488/omni_drive_arduino (visited on 04/08/2024).

[47] Dheenadhayalan R. *dheena2k2/fusion2urdf-ros2*. original-date: 2022-05-17T17:58:26Z. Apr. 2024. URL: https://github.com/dheena2k2/fusion2urdf-ros2 (visited on 04/09/2024).

[48] *MAPIRlab/rf2o_laser_odometry*. Apr. 2024. URL: https://github.com/MAPIRlab/rf2o_laser_odometry (visited on 04/05/2024).

[49] Mariano Jaimez, Javier G. Monroy, and Javier Gonzalez-Jimenez. "Planar odometry from a radial laser scanner. A range flow-based approach". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. May 2016, pp. 4479–4485. DOI: 10.1109/ICRA.2016.7487647. URL: https://ieeexplore.ieee.org/document/7487647 (visited on 04/05/2024).

[50] Steve Macenski and Ivona Jambrecic. "SLAM Toolbox: SLAM for the dynamic world". en. In: *Journal of Open Source Software* 6.61 (May 2021), p. 2783. ISSN: 2475-9066. DOI: 10.21105/joss.02783. URL: https://joss.theoj.org/papers/10.21105/joss.02783 (visited on 02/05/2024).

[51] *Introducing tf2 — ROS 2 Documentation: Rolling documentation*. URL: https://docs.ros.org/en/rolling/Tutorials/Intermediate/Tf2/Introduction-To-Tf2.html#using-view-frames (visited on 04/16/2024).

[52] Lihi Gur Arie PhD. *The practical guide for Object Detection with YOLOv5 algorithm*. en. Feb. 2023. URL: https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843 (visited on 04/05/2024).

[53] *eddie50488/Project: Place for Project code to be uploaded*. URL: https://github.com/eddie50488/Project (visited on 12/12/2023).

[54] Ambreen Hussain et al. “Low Latency and Non-Intrusive Accurate Object Detection in Forests”. In: Dec. 2021, pp. 1–6. DOI: 10.1109/SSCI50451.2021.9660175.

8 Team Contributions

Report Section	Contributing Member(s)
Executive Summary	Zabih, Edward
Introduction and Literature Review	Zabih
Goals and Objectives	All Members
Background Theory	
- MediaPipe	Alex
- Object Recognition	Arlend
- Robot Arm and Base	Tianchen
- ROS	Edward
Experimental Methods	
- Robot Redesign	Tianchen
- Object Recognition	Arlend
- Application and UI	Tianchen
- Voice	Zabih
- Hand Gesture, Holistic	Alex
- Python Code	Alex
- Wireless Connectivity	Zabih
- SLAM	Edward
Results	
- Robot Redesign	Tianchen
- Object Recognition	Arlend
- Voice	Zabih
- Hand Gesture, Holistic tracking	Alex
- RF	Zabih
- Object Retrival Pipeline	Edward
Conclusion and Future Work	Zabih, Edward, Tianchen

9 Appendices

9.1 Appendix A - GitHub Repository

The GitHub repository contains all the codes used within the project including the Python code and all the Arduino codes. It can be viewed at [53].

9.2 Appendix B - Arduino Code Serial Communication

```
#include <Adafruit_MotorShield.h>
#include <Servo.h>


// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
// Or, create it with a different I2C address (say for stacking)
// Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x61);

// Select which 'port' M1, M2, M3 or M4. In this case, M1
Adafruit_DCMotor *myMotor1 = AFMS.getMotor(1);
Adafruit_DCMotor *myMotor2 = AFMS.getMotor(2);
Adafruit_DCMotor *myMotor3 = AFMS.getMotor(3);
Adafruit_DCMotor *myMotor4 = AFMS.getMotor(4);

// Arm Servo pins
#define Joint1Pin 16
#define Joint2Pin 14
#define Joint3Pin 15
#define GripperPin 17
// Servo Objects
Servo Joint1;
Servo Joint2;
Servo Joint3;
Servo Gripper;
// Starting Joint Angles
int Joint1Angle = 90; // Change 5 sets of angles
int Joint2Angle = 125; // Change 5 sets of angles
int Joint3Angle = 145; // Change 5 sets of angles
int Joint4Angle = 90; // Change 5 sets of angles
int GripperOpen = 60; // Open gripper; Need to tune value
int GripperClose = 120; // Close gripper; Need to tune value

// Joint Angle Offsets
int Joint1Offset = 7; // Your value may be different
int Joint2Offset = 0; // Your value may be different
int Joint3Offset = 5; // Your value may be different

//Motor Direction
int Circle_Dir;
int Square_Dir;
int Plus_Dir;
int Triangle_Dir;
```

```
int mode;
int Speed;

void setup()
{
 Serial.begin(9600);
 Serial.setTimeout(10);
 Joint1.attach(Joint1Pin);
 Joint2.attach(Joint2Pin);
 Joint3.attach(Joint3Pin);
 Gripper.attach(GripperPin);
 Joint1.write(Joint1Angle+Joint1Offset);
 Joint2.write(Joint2Angle+Joint2Offset);
 Joint3.write(Joint3Angle+Joint3Offset);
 Gripper.write(GripperOpen); // Open gripper

 if(!AFMS.begin()){
  while(1);
 }


 delay(1000); // 5 seconds before loop function
}

void loop()
{
  if (Serial.available()>0)
  {
   String message = Serial.readStringUntil('\n'); // Read the
       message until a newline character is received
    // Split the message into individual parts
    if (message.length() <= 23)
    {
        int parts = sscanf(message.c_str(), "%d %d %d %d %d %
           d", &mode, &Joint1Angle,  &Joint2Angle,  &
           Joint3Angle, &Joint4Angle, &Speed);
        if (mode==0)
        {
          Joint1.write(Joint1Angle+Joint1Offset);
          Joint2.write(Joint2Angle+Joint2Offset);
          Joint3.write(Joint3Angle+Joint3Offset);
          Gripper.write(Joint4Angle);
        }

      //int parts = sscanf(message.c_str(), " %d %d %d %d", &
         Circle_Dir,  &Square_Dir,  &Plus_Dir, &Triangle_Dir)
```

```
        ;
86      //int parts = sscanf(message.c_str(), "%d %d %d %d %d %
          d %d %d", &Joint1Angle, &Joint2Angle, &Joint3Angle
          , &Joint4Angle, &Circle_Dir, &Square_Dir, &
          Plus_Dir, &Triangle_Dir);
87
88      if (mode==1){
89        Circle_Dir=Joint1Angle;
90        Square_Dir=Joint2Angle;
91        Plus_Dir=Joint3Angle;
92        Triangle_Dir=Joint4Angle;
93
94      if (Circle_Dir == 0)
95      {myMotor3->setSpeed(Speed);  //circle
96       myMotor3->run(BACKWARD);  }
97      if (Circle_Dir == 1)
98      {myMotor3->setSpeed(Speed);  //circle
99        myMotor3->run(FORWARD);}
100     if (Circle_Dir == 2)
101     {myMotor3->setSpeed(Speed);  //circle
102       myMotor3->run(RELEASE); }
103     if (Square_Dir == 0)
104     {myMotor2->setSpeed(Speed);  //square
105       myMotor2->run(BACKWARD); }
106     if (Square_Dir == 1)
107     {myMotor2->setSpeed(Speed);  //square
108       myMotor2->run(FORWARD);}
109     if (Square_Dir == 2)
110     {myMotor2->setSpeed(Speed);  //square
111       myMotor2->run(RELEASE);}
112     if (Plus_Dir == 0)
113     {myMotor4->setSpeed(Speed);  //PLUS
114       myMotor4->run(FORWARD);}
115     if (Plus_Dir == 1)
116     {myMotor4->setSpeed(Speed);  //PLUS
117       myMotor4->run(BACKWARD);}
118     if (Plus_Dir == 2)
119     {myMotor4->setSpeed(Speed);  //PLUS
120       myMotor4->run(RELEASE);}
121     if (Triangle_Dir == 0)
122     {myMotor1->setSpeed(Speed);  //square
123       myMotor1->run(FORWARD);}
124     if (Triangle_Dir == 1)
125     {myMotor1->setSpeed(Speed);  //square
126       myMotor1->run(BACKWARD);}
127     if (Triangle_Dir == 2)
```

```
128         {myMotor1->setSpeed(Speed);   //square
129           myMotor1->run(RELEASE);}
130         }
131       }
132     }
133 delay(10);
134 }
```

Listing 1: Arduino Code for Wired Connection Robot via Serial Communication

9.3 Appendix C - Arduino Code (RF Code)

9.3.1 'Hello World' Code for Transmitter and Receiver

```
#include <SPI.h>
#include <RF24.h>
#include <nRF24L01.h>
#include <Servo.h>

RF24 radio(31,30); // CE,CSN
const byte address[6] = "00001";

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.setTimeout(10);
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_LOW);
  radio.stopListening();
}

void loop()
{
  const char txt[] = "Hello World";
  radio.write(txt, sizeof(txt));
  delay(100);
}
```

Listing 2: Arduino Code for Transmitting 'Hello World' with nRF24L01

```
#include <SPI.h>
#include <RF24.h>
#include <nRF24L01.h>

RF24 radio(31,30); // CE,CSN
const byte address[6] = "00001";

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}

void loop() {
  if (radio.available())
```

```
19   {
20     char txt[32] = "";
21     radio.read(&txt, sizeof(txt));
22     Serial.println(txt);
23   }
24 }
```

Listing 3: Arduino Code for Receiving 'Hello World' with nRF24L01

9.3.2 Joint Angle Code for Transmitter and Receiver

```
1  #include <SPI.h>
2  #include <RF24.h>
3  #include <nRF24L01.h>
4  #include <Servo.h>
5
6  int mode;
7  int Joint1Angle;
8  int Joint2Angle;
9  int Joint3Angle;
10 int Joint4Angle;
11 int Speed;
12
13 RF24 radio(31,30); // CE,CSN
14 const byte address[6] = "00001";
15
16 void setup() {
17   // put your setup code here, to run once:
18   Serial.begin(9600);
19   Serial.setTimeout(10);
20   radio.begin();
21   radio.openWritingPipe(address);
22   radio.setPALevel(RF24_PA_LOW);
23   radio.stopListening();
24 }
25
26 void loop()
27 {
28   if (Serial.available()>0)
29   {
30     String message = Serial.readStringUntil('\n');
31     int parts = sscanf(message.c_str(), "%d %d %d %d %d %d",
         &mode, &Joint1Angle, &Joint2Angle, &Joint3Angle, &
         Joint4Angle, &Speed);
32
33     // Convert the values into a formatted string
34     char txt[30]; // Adjust the size as per your requirements
```

```
35      sprintf(txt, "%d %d %d %d %d %d", mode, Joint1Angle,
           Joint2Angle, Joint3Angle, Joint4Angle, Speed);
36
37      // Send the string over radio
38      radio.write(txt, sizeof(txt));
39    }
40    //const char txt[] = "Hello World";
41    //radio.write(txt, sizeof(txt));
42    delay(100);
43  }
```

Listing 4: Arduino Code for Transmitting Joint Angles, Speed, and Mode with nRF24L01

```
1  #include <SPI.h>
2  #include <RF24.h>
3  #include <nRF24L01.h>
4
5  RF24 radio(31,30); // CE,CSN
6  const byte address[6] = "00001";
7
8  void setup() {
9    // put your setup code here, to run once:
10   Serial.begin(9600);
11   radio.begin();
12   radio.openReadingPipe(0, address);
13   radio.setPALevel(RF24_PA_MIN);
14   radio.startListening();
15 }
16
17 void loop() {
18   if (radio.available())
19   {
20     char txt[32] = "";
21     radio.read(&txt, sizeof(txt));
22     Serial.println(txt);
23   }
24 }
```

Listing 5: Arduino Code for Receiving Joint Angles, Speed, and Mode with nRF24L01

9.4 Appendix D - Final Arduino Code

9.4.1 Transmitter Code

```
#include <SPI.h>
#include <RF24.h>
#include <nRF24L01.h>
#include <Servo.h>

int mode;
int Joint1Angle;
int Joint2Angle;
int Joint3Angle;
int Joint4Angle;
int Speed;


RF24 radio(7,8); // CE,CSN
const byte address[6] = "00002";

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.setTimeout(10);
  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MAX);
  radio.stopListening();
}

void loop()
{

  if (Serial.available()>0)
  {
    String message = Serial.readStringUntil('\n');
    int parts = sscanf(message.c_str(), "%d %d %d %d %d %d",
       &mode, &Joint1Angle, &Joint2Angle, &Joint3Angle, &
       Joint4Angle, &Speed);

    // Convert the values into a formatted string
    char txt[30]; // Adjust the size as per your requirements
    sprintf(txt, "%d %d %d %d %d %d", mode, Joint1Angle,
       Joint2Angle, Joint3Angle, Joint4Angle, Speed);

    // Send the string over radio
    radio.write(txt, sizeof(txt));
  }
```

```
  //const char txt[] = "Hello World";
  //radio.write(txt, sizeof(txt));
  delay(100);
}
```

Listing 6: Arduino Code for Final Tranmitter Code

9.4.2 Receiver Code

```
#include <SPI.h>

#include <nRF24L01.h>
#include <RF24.h>

#include <Servo.h>
#include <Adafruit_MotorShield.h>

// Create the motor shield object with the default I2C
   address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
// Or, create it with a different I2C address (say for
   stacking)
// Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x61);

// Select which 'port' M1, M2, M3 or M4. In this case, M1
Adafruit_DCMotor *myMotor1 = AFMS.getMotor(1);
Adafruit_DCMotor *myMotor2 = AFMS.getMotor(2);
Adafruit_DCMotor *myMotor3 = AFMS.getMotor(3);
Adafruit_DCMotor *myMotor4 = AFMS.getMotor(4);

// Arm Servo pins
#define Joint1Pin 16
#define Joint2Pin 14
#define Joint3Pin 15
#define GripperPin 17
// Servo Objects
Servo Joint1;
Servo Joint2;
Servo Joint3;
Servo Gripper;
// Starting Joint Angles
int Joint1Angle = 90; // Change 5 sets of angles
int Joint2Angle = 125; // Change 5 sets of angles
int Joint3Angle = 145; // Change 5 sets of angles
int Joint4Angle = 90; // Change 5 sets of angles
int GripperOpen = 60; // Open gripper; Need to tune value
int GripperClose = 120; // Close gripper; Need to tune value

```

```
38 // Joint Angle Offsets
39 int Joint1Offset = 7; // Your value may be different
40 int Joint2Offset = 0; // Your value may be different
41 int Joint3Offset = 5; // Your value may be different
42
43 //Motor Direction
44 int Circle_Dir;
45 int Square_Dir;
46 int Plus_Dir;
47 int Triangle_Dir;
48 int mode;
49 int Speed;
50
51 //radio setup
52 RF24 radio(30,32); // CE,CSN
53 const byte address[6] = "00002";
54
55
56
57 void setup() {
58   // put your setup code here, to run once:
59   Serial.begin(9600);
60   radio.begin();
61   radio.openReadingPipe(0, address);
62   radio.setPALevel(RF24_PA_MAX);
63   radio.startListening();
64
65   Joint1.attach(Joint1Pin);
66   Joint2.attach(Joint2Pin);
67   Joint3.attach(Joint3Pin);
68   Gripper.attach(GripperPin);
69   Joint1.write(Joint1Angle+Joint1Offset);
70   Joint2.write(Joint2Angle+Joint2Offset);
71   Joint3.write(Joint3Angle+Joint3Offset);
72   Gripper.write(GripperOpen); // Open gripper
73
74   if(!AFMS.begin()){
75     while(1);
76   }
77
78 }
79
80 void loop() {
81   if (radio.available())
82   {
83     char txt[30]=""; // Adjust the size as per your
```

```
        requirements
84      radio.read(txt, sizeof(txt));
85
86      int parts = sscanf(txt, "%d %d %d %d %d %d", &mode, &
           Joint1Angle,  &Joint2Angle,  &Joint3Angle, &
           Joint4Angle, &Speed);
87      if (mode==0)
88          {
89            Joint1.write(Joint1Angle+Joint1Offset);
90            Joint2.write(Joint2Angle+Joint2Offset);
91            Joint3.write(Joint3Angle+Joint3Offset);
92            Gripper.write(Joint4Angle);
93          }
94      if (mode==1)
95      {
96          Circle_Dir=Joint1Angle;
97          Square_Dir=Joint2Angle;
98          Plus_Dir=Joint3Angle;
99          Triangle_Dir=Joint4Angle;
100
101       if (Circle_Dir == 0)
102       {myMotor3->setSpeed(Speed);  //circle
103        myMotor3->run(BACKWARD);  }
104       if (Circle_Dir == 1)
105       {myMotor3->setSpeed(Speed);  //circle
106         myMotor3->run(FORWARD);}
107       if (Circle_Dir == 2)
108       {myMotor3->setSpeed(Speed);  //circle
109         myMotor3->run(RELEASE); }
110       if (Square_Dir == 0)
111       {myMotor2->setSpeed(Speed);  //square
112         myMotor2->run(BACKWARD); }
113       if (Square_Dir == 1)
114       {myMotor2->setSpeed(Speed);  //square
115         myMotor2->run(FORWARD);}
116       if (Square_Dir == 2)
117       {myMotor2->setSpeed(Speed);  //square
118         myMotor2->run(RELEASE);}
119       if (Plus_Dir == 0)
120       {myMotor4->setSpeed(Speed);  //PLUS
121         myMotor4->run(FORWARD);}
122       if (Plus_Dir == 1)
123       {myMotor4->setSpeed(Speed);  //PLUS
124         myMotor4->run(BACKWARD);}
125       if (Plus_Dir == 2)
126       {myMotor4->setSpeed(Speed);  //PLUS
```

```
127           myMotor4->run(RELEASE);}
128         if (Triangle_Dir == 0)
129         {myMotor1->setSpeed(Speed);  //square
130           myMotor1->run(FORWARD);}
131         if (Triangle_Dir == 1)
132         {myMotor1->setSpeed(Speed);  //square
133           myMotor1->run(BACKWARD);}
134         if (Triangle_Dir == 2)
135         {myMotor1->setSpeed(Speed);  //square
136           myMotor1->run(RELEASE);}
137       }
138       // Print the received string
139
140
141       Serial.println(txt);
142       //Serial.println("hello");
143       delay(100);
144     }
145 }
```

Listing 7: Arduino Code for Final Receiver Code

9.5 Appendix E - ESP-32 CAM Arduino Code

```
#include "esp_camera.h"
#include <WiFi.h>
#define CAMERA_MODEL_AI_THINKER // Has PSRAM
#include "camera_pins.h"

// ===========================
// Enter your WiFi credentials
// ===========================
const char* ssid = "";
const char* password = "";

void startCameraServer();
void setupLedFlash(int pin);

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_UXGA;
  config.pixel_format = PIXFORMAT_JPEG; // for streaming
  //config.pixel_format = PIXFORMAT_RGB565; // for face
     detection/recognition
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
```

```
45    config.jpeg_quality = 12;
46    config.fb_count = 1;
47
48    // if PSRAM IC present, init with UXGA resolution and
         higher JPEG quality
49    //                      for larger pre-allocated frame
         buffer.
50    if(config.pixel_format == PIXFORMAT_JPEG){
51      if(psramFound()){
52        config.jpeg_quality = 10;
53        config.fb_count = 2;
54        config.grab_mode = CAMERA_GRAB_LATEST;
55      } else {
56        // Limit the frame size when PSRAM is not available
57        config.frame_size = FRAMESIZE_SVGA;
58        config.fb_location = CAMERA_FB_IN_DRAM;
59      }
60    } else {
61      // Best option for face detection/recognition
62      config.frame_size = FRAMESIZE_240X240;
63  #if CONFIG_IDF_TARGET_ESP32S3
64      config.fb_count = 2;
65  #endif
66    }
67
68  #if defined(CAMERA_MODEL_ESP_EYE)
69    pinMode(13, INPUT_PULLUP);
70    pinMode(14, INPUT_PULLUP);
71  #endif
72
73    // camera init
74    esp_err_t err = esp_camera_init(&config);
75    if (err != ESP_OK) {
76      Serial.printf("Camera init failed with error 0x%x", err);
77      return;
78    }
79
80    sensor_t * s = esp_camera_sensor_get();
81    // initial sensors are flipped vertically and colors are a
         bit saturated
82    if (s->id.PID == OV3660_PID) {
83      s->set_vflip(s, 1); // flip it back
84      s->set_brightness(s, 1); // up the brightness just a bit
85      s->set_saturation(s, -2); // lower the saturation
86    }
87    // drop down frame size for higher initial frame rate
```

```
88    if(config.pixel_format == PIXFORMAT_JPEG){
89      s->set_framesize(s, FRAMESIZE_QVGA);
90    }
91
92  #if defined(CAMERA_MODEL_M5STACK_WIDE) || defined(
        CAMERA_MODEL_M5STACK_ESP32CAM)
93    s->set_vflip(s, 1);
94    s->set_hmirror(s, 1);
95  #endif
96
97  #if defined(CAMERA_MODEL_ESP32S3_EYE)
98    s->set_vflip(s, 1);
99  #endif
100
101 // Setup LED FLash if LED pin is defined in camera_pins.h
102 #if defined(LED_GPIO_NUM)
103   setupLedFlash(LED_GPIO_NUM);
104 #endif
105
106   WiFi.begin(ssid, password);
107   WiFi.setSleep(false);
108
109   while (WiFi.status() != WL_CONNECTED) {
110     delay(500);
111     Serial.print(".");
112   }
113   Serial.println("");
114   Serial.println("WiFi connected");
115
116   startCameraServer();
117
118   Serial.print("Camera Ready! Use 'http://");
119   Serial.print(WiFi.localIP());
120   Serial.println("' to connect");
121 }
122
123 void loop() {
124   // Do nothing. Everything is done in another task by the
        web server
125   delay(10000);
126 }
```

Listing 8: Arduino Code for Wireless ESP-32 Camera

9.6 Appendix F - Object Recognition Table Column Definitions

'Epochs': Each Epoch is a complete pass through the dataset.

'Train/box_{loss}': This is the box regression loss, and it measures how well the model predicts the bounding box for each detected object. (Training Dataset) [54].

'Train/obj_{loss}': This is the objectness loss, and it measures how well the model can differentiate between the background and object. (Training Dataset) [54].

'Train/cl_{loss}': This is the classification loss, and it evaluates the accuracy of the model when classifying objects. (Training Dataset) [54].

'Metrics/Precision': Shows the accuracy of the positive predictions made by the model. It is the ratio of true positive predictions to the total number of positive predictions. [21].

'Metrics/Recall': Shows the models ability to detect all relevant instances/objects in the dataset. It is the ratio of true positives to the sum of the true positives and false negatives [21].

'Metrics/$mAP_{0.5}$': Called the Mean Average Precision. It measures the overlap between the actual object and the bounding box given by the model, and in this case, if the overlap is above a threshold of 50%, the object detected is considered a true positive, thus the model is correct [21].

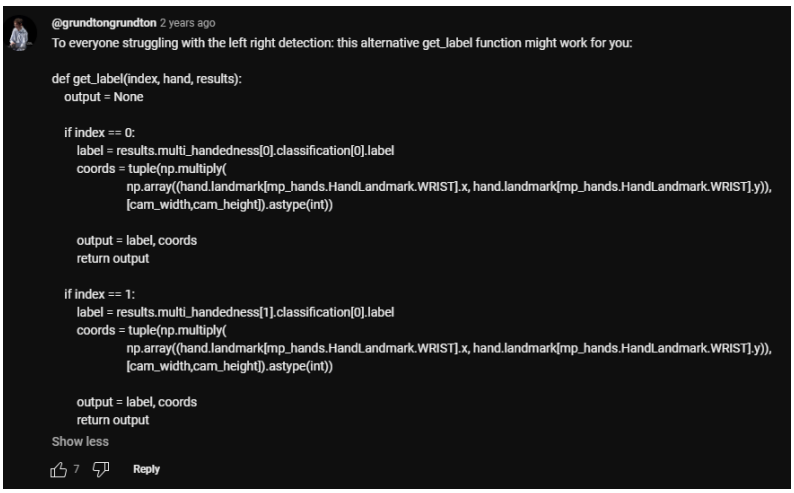
'Metrics/$mAP_{0.5:0.95}$': Instead of a threshold of 50%, the threshold is between 50% and 95%. Once again, if the overlap is between these values, the detected object is considered a true positive, and the model is therefore accurate [21].

'Val/box_{loss}': This is the box regression loss, and it measures how well the model predicts the bounding box for each detected object. (Validation Dataset) [54].

'Val/obj_{loss}': This is the objectness loss, and it measures how well the model can differentiate between the background and object. (Validation Dataset) [54].

'Val/cl_{loss}': This is the classification loss, and it evaluates the accuracy of the model when classifying objects. (Validation Dataset) [54].

9.7 Appendix G - Extra Information



@grundtongrundton 2 years ago

To everyone struggling with the left right detection: this alternative get_label function might work for you:

```
def get_label(index, hand, results):
    output = None

    if index == 0:
        label = results.multi_handedness[0].classification[0].label
        coords = tuple(np.multiply(
                    np.array((hand.landmark[mp_hands.HandLandmark.WRIST].x, hand.landmark[mp_hands.HandLandmark.WRIST].y)),
                    [cam_width,cam_height]).astype(int))

        output = label, coords
        return output

    if index == 1:
        label = results.multi_handedness[1].classification[0].label
        coords = tuple(np.multiply(
                    np.array((hand.landmark[mp_hands.HandLandmark.WRIST].x, hand.landmark[mp_hands.HandLandmark.WRIST].y)),
                    [cam_width,cam_height]).astype(int))

        output = label, coords
        return output
```

Show less

7 Reply

Figure 84: get label function