Droid Project

Jackson Williamson

Introduction

Background Information

In the rapidly evolving field of technology, hands-on projects are invaluable for understanding the practical applications of theoretical knowledge. This project aimed to develop an Astromech droid, inspired by the beloved Star Wars franchise, which could serve as a versatile and helpful companion. The goal was to integrate advanced features such as computer vision, projection capabilities, and robotic arm functionality.

Objective

The primary objective of this project was to design, build, and test an Astromech droid that could follow the user, perform helpful tasks, and interact in a meaningful way. The aim was to push the boundaries of current robotics and create a device that was both functional and reminiscent of the iconic droids from the Star Wars universe.

Inspiration

The inspiration for this project stemmed from my lifelong love of Star Wars and a passion for hardware development. Initially, I envisioned creating a droid that could follow me around, project images and videos, and perform various tasks with the help of a robotic arm. Leveraging the rapidly advancing field of computer vision, I aimed to create a companion that could interact with its environment and assist in everyday tasks, surpassing the capabilities of existing robotic systems. Although the project's challenges were significant and the outcome was not as successful as hoped, the journey provided valuable insights and learning experiences in the realm of hardware development.

Research

I have always been fascinated by robots and wanted to make my own. Most complex robots are developed on a large scale by big companies, and while there are hobbyists all over the world, I wanted to be able to say that I was able to create something truly remarkable. To achieve this, I looked into various tools and techniques available in the field of robotics and computer vision.

Computer Vision

The field I was most excited to study and look into was the field of computer vision this was something we looked into at the tail end of robotics and saw this as the logical continuation of last year where I would be able to build upon the basic robots we had used to complete rudimentary tasks and give them life specifically sight with an eye a camera which was a founding idea to me.

- **Facial Recognition**: Facial recognition is a technology is something that has come along way and know its use cases are far and wide and a lot of people leverage it everyday by using it to lock and unlock there phone.

- **Object Detection**: Object detection is an interesting technology because it is very useful and probably one of the easier ones to work with especially when you have a pretrained model but also one of the most restrictive if you need it for collision detection it needs to recognise what its looking at which means its trained on that so you would need a large amount of data for a reliable model which will also negatively impact processing time.

- **Depth Mapping**: Depth mapping was one that surprised me as I had not even known about it prior to starting this project but the idea of it is amazing for navigational purposes it allows for checking what is close and far away in an environment.

OpenCv

Too bring these computer vision concepts together I will utilize the power of opencv supports a broad range of built in functions that can be used in tandem with different techniques it is allows for easy use of external tools such as using a webcam.

These technologies offered endless potential to a robot that is using them and could theoretically be bolted into any machine to make it better although having the weakness of processing time.

Robot Operating System (ROS)

In addition to computer vision, I considered using Robot Operating System (ROS), a middleware designed to act as a robot's brain. ROS facilitates the integration of various components and allows developers to utilize nodes created by others, enhancing the robot's capabilities. However, ROS has its drawbacks

- User Friendly: ROS has a steep learning curve for new users and this is referenced in its own documentation on installing it I have installed ROS twice to play around with it when installing it on my Raspberry Pi it took hours by my est about 3.

- System: While Ros as a tool could of shot my productivity up on working robot after learning its system and getting it all installed it would of meant going down the rabbit hole and using all these cool and well built nodes that have been designed for my purpose which is great but it would of taken away from learning about these different methodology in programming which at the heart of this is most important.

Ultimately, ROS isn't all bad either. It allowed me to learn about different techniques which would be interesting to experiment with later. For example, while I settled on the idea of having a robot move around with a camera using computer vision, ROS taught me about an alternative technique using SLAM (Simultaneous Localization and Mapping). SLAM is a process used in ROS for navigation, which involves building a digital map of the environment (mapping) and determining the robot's position on this map (localization). Additionally, ROS uses path planning to decide where to move, ensuring consistent and purposeful navigation. This method would have been desirable for my project, as it would avoid aimless and inconsistent navigation, ensuring the robot knows where it's going at all times.

And while these are some great tools and idea I wanted to make my own mistakes and my own control software so I could learn the pros and cons of using computer vision.

Other Concepts:

Serial: When looking at how to communicate between my controllers I settled on usb Serial communication this is because it is widely available format and it also allows me to work interchangeably between Computers and Raspberry Pi while working on the project and especially for prototyping and testing the code out.

Audio: I wanted to implement some kind of speakers to simulate the worbling of a droid.

Project Design and Planning

Initial Idea

The initial concept of this project was to create an astromech droid that would be able to do this things we see in star was much like an R2 unit I wanted him to be able to roam around freely be able to grab things for us as well as being able to be sent on missions for Example a mission might be to find me and show me a video which he could use a projector for or too follow me or someone in particular around if they were a target for example.

Design Process

Concept

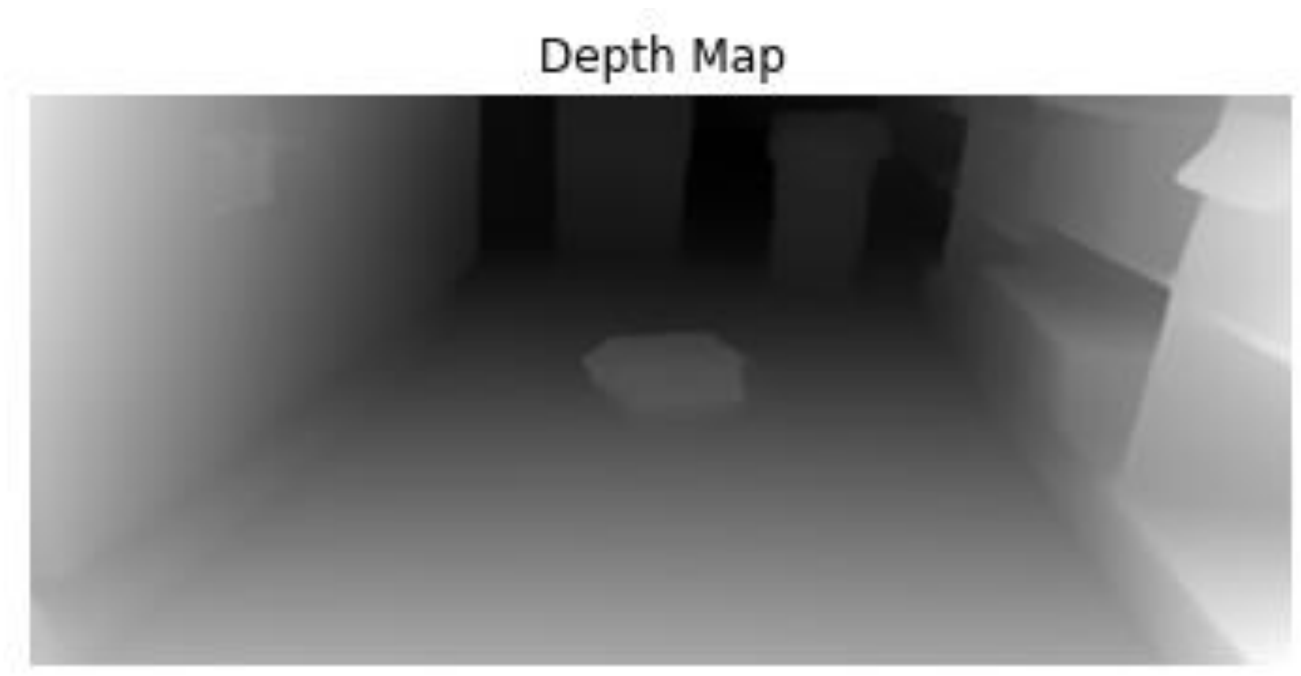
- In the beginning I brainstormed ideas I want to be in the droid this included Recognising people, roaming, following, being able to project an image, hold and carry things and go on missions.
- Next I started thinking about how I wanted to make it work which I did end up settling on CV where I would write my own scripts for it.

Components

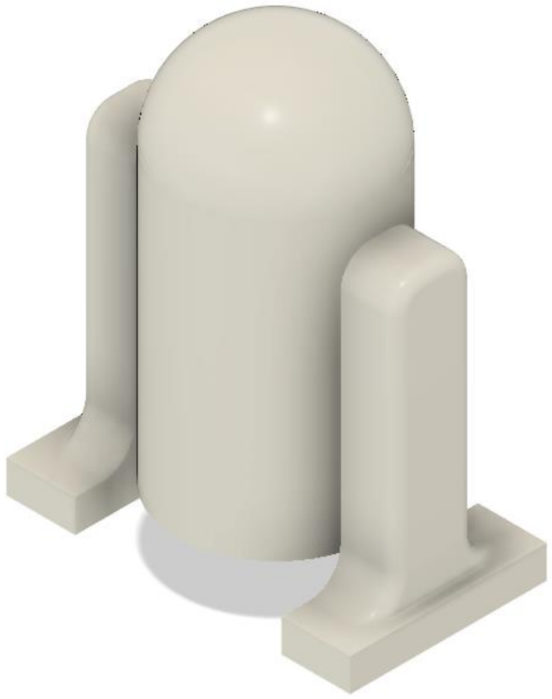
- I identified a need for DC motors for movement now I went with standard Motors over a stepper because I wanted it to move quickly and smoothly.
- I wanted it to be able to see so it needed a camera I was originally planning on going with a raspberry pi camera but eventually decided it was too restrictive and would cause issues if I wanted the head to move. So I settled on a webcam as it has a usb cable and would be able to move more freely the advantage with this is its cheaper and more widely available as well.
- For the head movement I have decided on using a servo motor as from my research I had been informed that they were more efficient in comparison to a stepper motor.
- For the Hand/Grabber we will want to use a stepper motor due to its ability to move in a highly precise manner it also has a high stationary torque which will allow it to better hold on to things in comparison to a stepper motor which maybe more likely to open and loosen when under pressure. All though I am taking a tradeoff with a stepper motor which can be set to specific positions and remember them which is good for resetting.
- I have decided to use Raspberry Pi as my main controller this is due to it being compact device that’s widely available and has good processing power. The downside to this is that using computer vision is highly processing intensive that may not work very fast. So if were working with a larger Robot it might be more useful to use a motherboard ideally with a graphics card this wouldn’t need to be a large board Mini or Micro Atx would likely be perfect.

Languages

- I have started off by picking my primary programming languages being Arduino C and Python the advantage to python is it's a versatile language as well as being the go to language for computer vision and it also works very well with a raspberry pi.
- I would also utilize a service that will be connected to a button on startup so I can start the script with the press of a button.
- I also have taken a look at daemons for running a script on startup aswell as task scheduler on windows which yes is more of a tool but they achieve the same purpose.

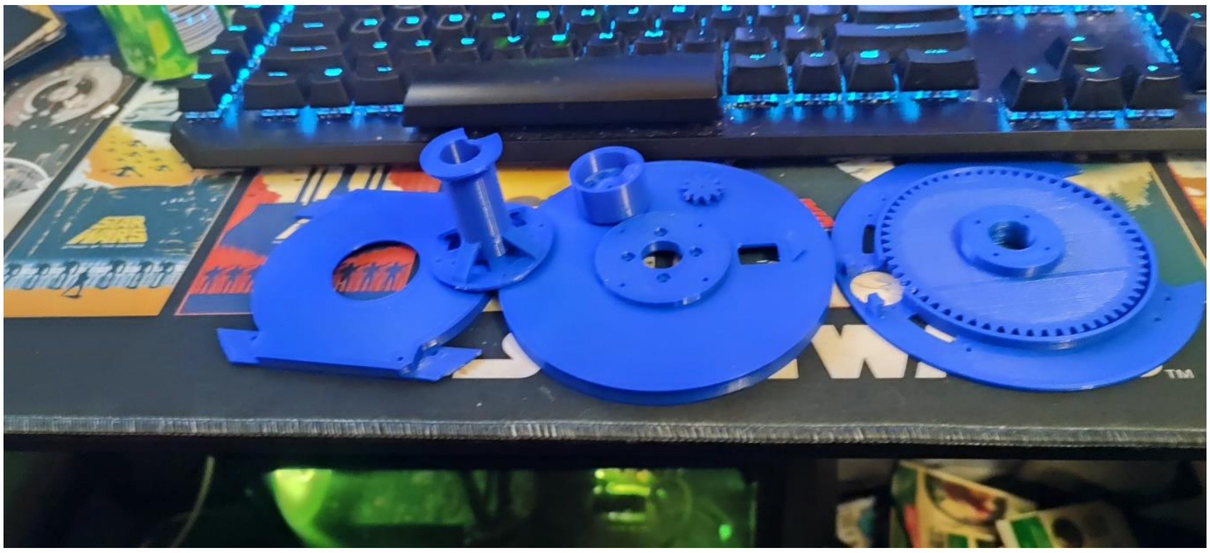


Implementation



I initially begun with my own design of an astromech droid I figured this would be the best approach but shortly after beginning on it I realised this would be a tall task and it might take up more time than I have to come up with working comprehensive design so I ended up abandoning it for a version I found on thing verse.

I settled on this new model because it had seemingly been designed to be motorized although plenty of people had done seemingly successful makes of this model no one seemed to have motorised it including the maker himself but I figured and issues there I would be able to tackle myself and I first focused on the rotation of the head as that seemed like an easy place to start.



It's a fairly simple configuration I had to resize the driving gear for the stepper motor I ended up using as well find alternatives to the screws for the model or redo all the screw holes for the model I ended up using tape blue tack and glue as well as I redesigned and removed some pieces as this has been designed to limit the head movement to about a 90deg movement mine would be able to turn 360 degrees if I wanted it too.

During the initial setup of the servo motor, I encountered several challenges that impacted its performance. Initially, I selected a large servo motor, which, in hindsight, was oversized for this project. This motor exhibited mechanical issues, such as starting and stopping intermittently and generally failing to operate consistently. Given that it was powered through a modular power box designed to provide the necessary amount of power, I ruled out power supply issues as the cause.

After swapping to a smaller servo motor, these issues were resolved, and the motor functioned reliably.

The major focus of this project was not just the hardware but the programming involved. Several scripts were developed around computer vision technologies to enhance the droid's functionalities.

Object Detection Algorithm for Navigation

One of the initial scripts utilized an object detection algorithm to create a more advanced navigation system. The goal was to enable the droid to detect and avoid obstacles dynamically. However, after extensive use, it became clear that this approach was not the most effective for collision avoidance. The processing requirements and the need for extensive training data made it less practical for real-time navigation.

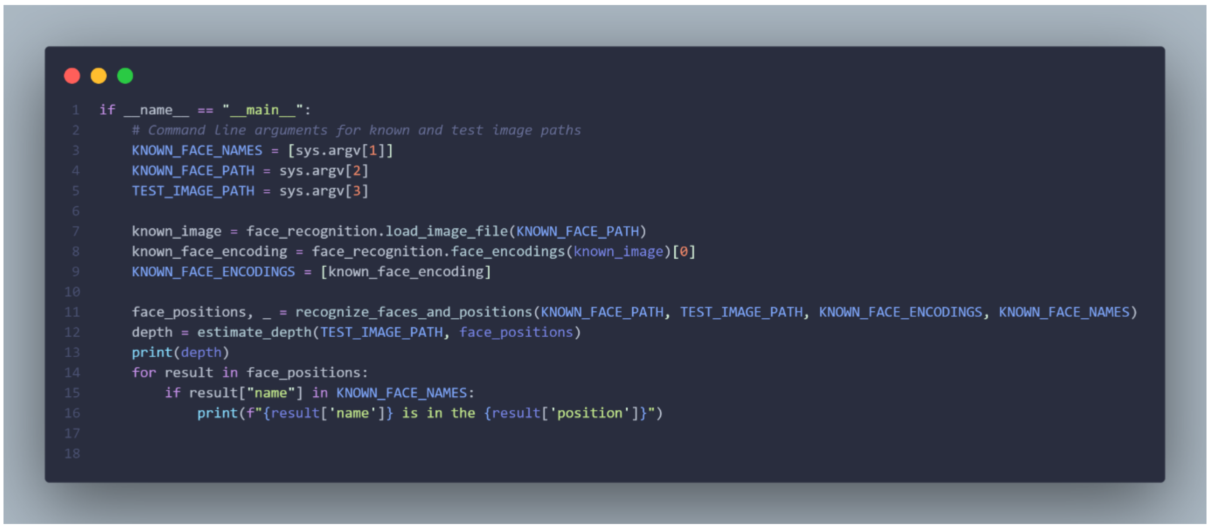
Shift in Approach

Upon discovering new computer vision techniques, I decided to repurpose the object detection algorithm. Instead of using it solely for collision avoidance, I integrated it with the robotic hand functionality. This shift allowed the droid to perform more complex tasks, such as recognizing and picking up specific objects. For instance, if the droid was commanded to pick up a bottle, the object detection algorithm would first verify the presence of a bottle before the robotic hand attempted to grasp it.

Facial Detection Algorithm

The originally I planned on doing some kind of pet mode with the facial recognition algorithm now while the code for that exists inside the script it has been migrated with a more mission orientated focus. My initial plan was so to have a self learning algorithm and while I figured this wouldn’t be impossible it would be extremely intensive programmatically and not the most reliable. I had also planned on self training the model myself specifically on me but this quite quickly became an unfeasible task due to the large number of images I would need to take of myself and preprocess after further research I changed to a premade model in the form of the Facial Recognition algorithm which is built of the dlib module.

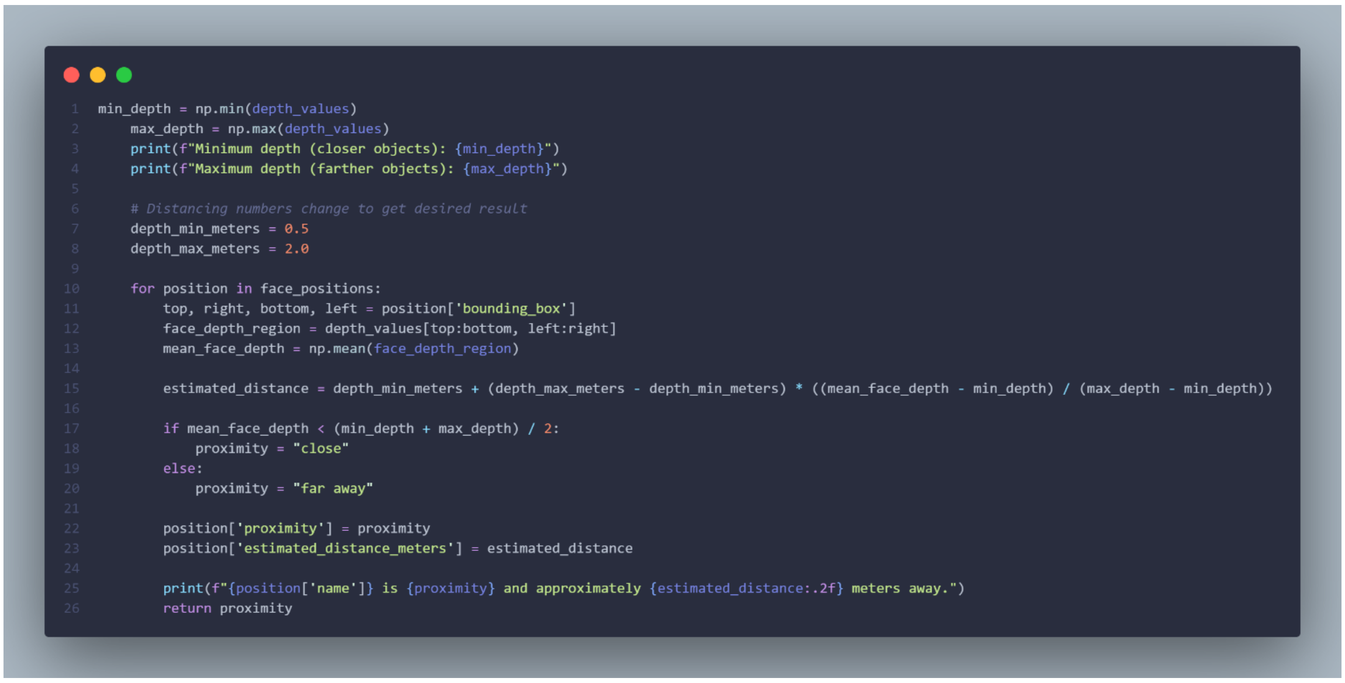
Figure 1Headshots from when i was trying out a trained by me model



```python
if __name__ == "__main__":
    # Command line arguments for known and test image paths
    KNOWN_FACE_NAMES = [sys.argv[1]]
    KNOWN_FACE_PATH = sys.argv[2]
    TEST_IMAGE_PATH = sys.argv[3]

    known_image = face_recognition.load_image_file(KNOWN_FACE_PATH)
    known_face_encoding = face_recognition.face_encodings(known_image)[0]
    KNOWN_FACE_ENCODINGS = [known_face_encoding]

    face_positions, _ = recognize_faces_and_positions(KNOWN_FACE_PATH, TEST_IMAGE_PATH, KNOWN_FACE_ENCODINGS, KNOWN_FACE_NAMES)
    depth = estimate_depth(TEST_IMAGE_PATH, face_positions)
    print(depth)
    for result in face_positions:
        if result["name"] in KNOWN_FACE_NAMES:
            print(f"{result['name']} is in the {result['position']}")
```

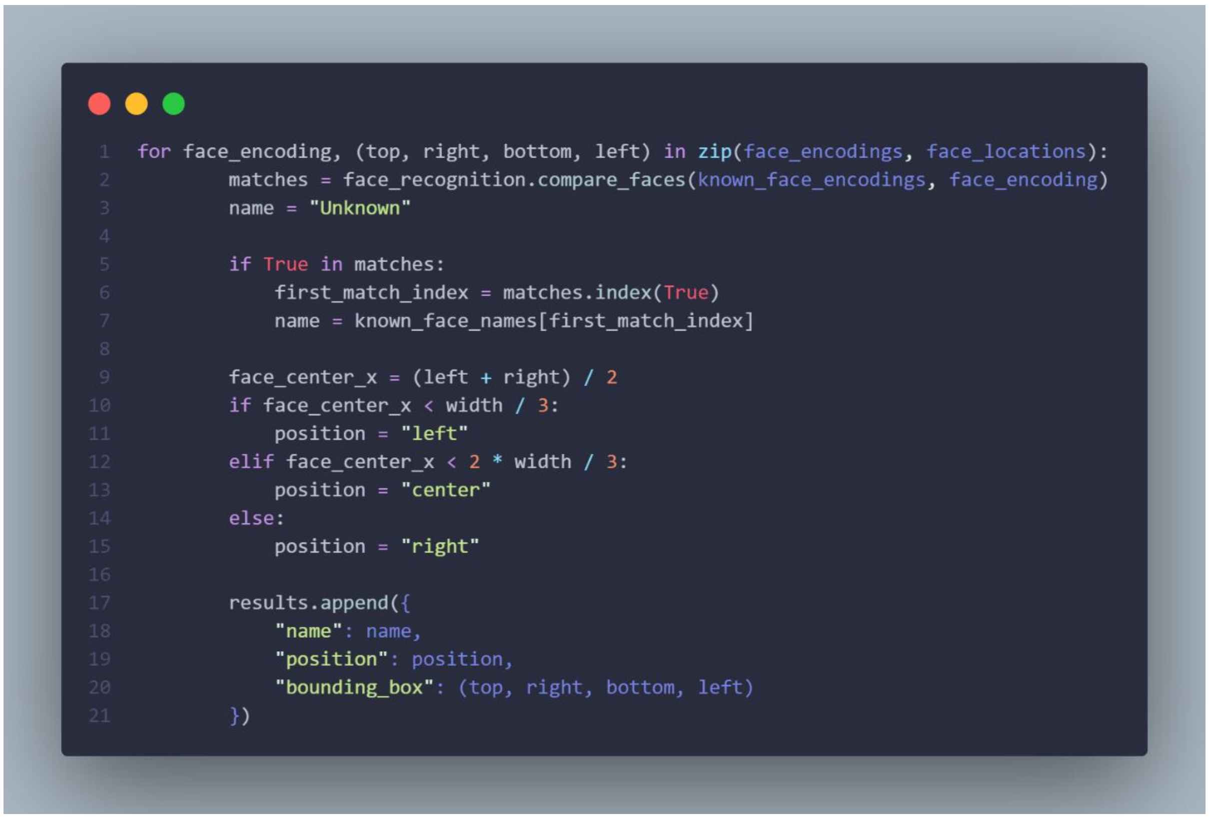
- We will start by ingesting the name of the person we are looking for there image and the image path we want to check against this will be the photo taken by the camera.

- If the person is found it will be printed out which will be used to decide on the next course of action in the control script I.e move forward, strafe left and right.



```python
min_depth = np.min(depth_values)
    max_depth = np.max(depth_values)
    print(f"Minimum depth (closer objects): {min_depth}")
    print(f"Maximum depth (farther objects): {max_depth}")

    # Distancing numbers change to get desired result
    depth_min_meters = 0.5
    depth_max_meters = 2.0

    for position in face_positions:
        top, right, bottom, left = position['bounding_box']
        face_depth_region = depth_values[top:bottom, left:right]
        mean_face_depth = np.mean(face_depth_region)

        estimated_distance = depth_min_meters + (depth_max_meters - depth_min_meters) * ((mean_face_depth - min_depth) / (max_depth - min_depth))

        if mean_face_depth < (min_depth + max_depth) / 2:
            proximity = "close"
        else:
            proximity = "far away"

        position['proximity'] = proximity
        position['estimated_distance_meters'] = estimated_distance

        print(f"{position['name']} is {proximity} and approximately {estimated_distance:.2f} meters away.")
        return proximity
```

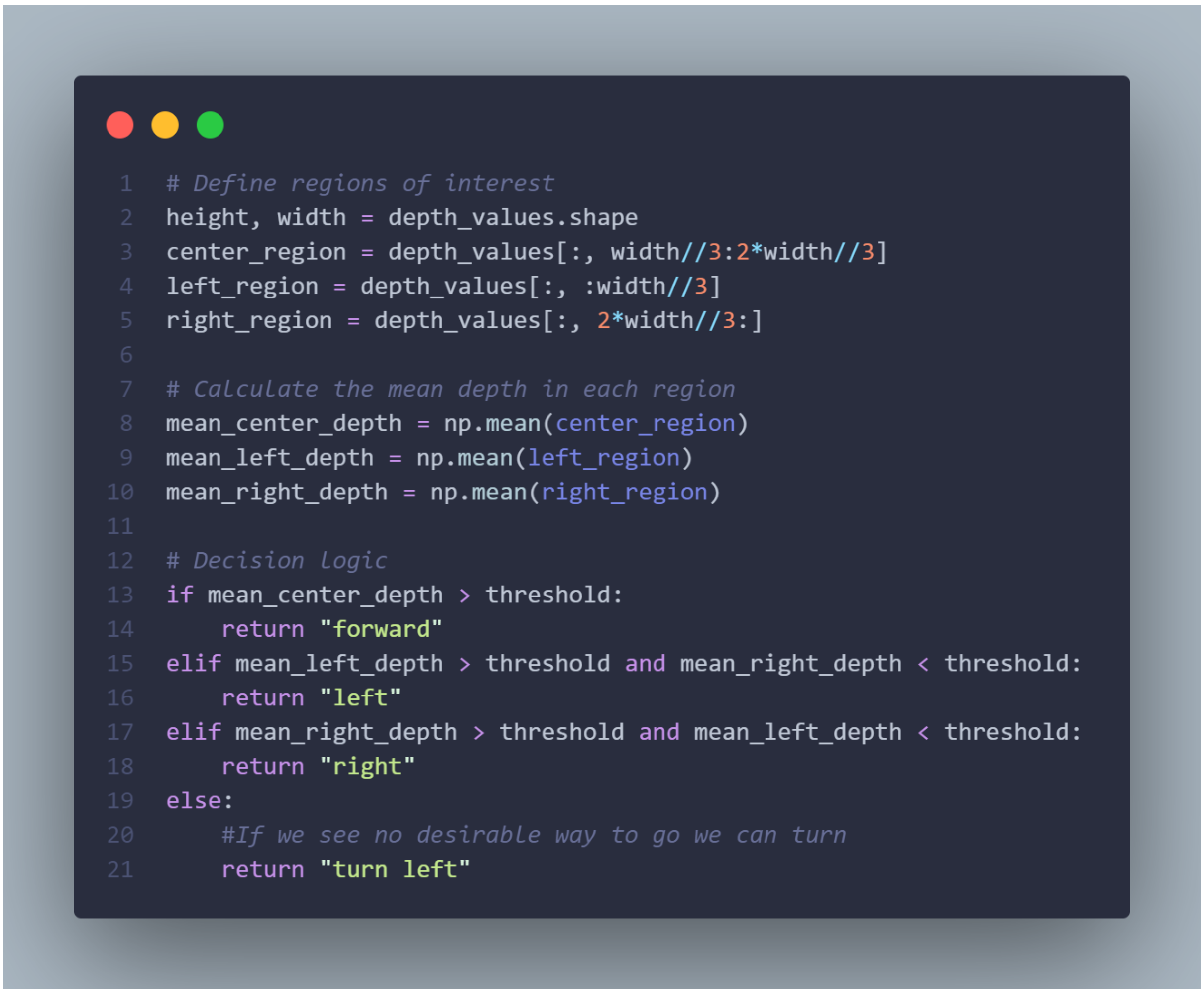
- We take the minimum and maximum from the depth_values and we will also set our own aswell to act as what will be close and far.
- Due to the nature of the model it wont be perfect at detecting the exact distance and we will want to adjust our values to help with this aswell.
- It will then return a basic value of close or far away. Keeping it simple we could alternatively return the distance value and move to the distance but due to the inconsistency.



```python
for face_encoding, (top, right, bottom, left) in zip(face_encodings, face_locations):
        matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
        name = "Unknown"

        if True in matches:
            first_match_index = matches.index(True)
            name = known_face_names[first_match_index]

        face_center_x = (left + right) / 2
        if face_center_x < width / 3:
            position = "left"
        elif face_center_x < 2 * width / 3:
            position = "center"
        else:
            position = "right"

        results.append({
            "name": name,
            "position": position,
            "bounding_box": (top, right, bottom, left)
        })
```

- We will check through face encodings to see if the face we inputted to look for is found.
- Next it will check the location of the face on the screen if it was found and return left, right or centre based on where it was found

Depth Mapping



```python
# Define regions of interest
height, width = depth_values.shape
center_region = depth_values[:, width//3:2*width//3]
left_region = depth_values[:, :width//3]
right_region = depth_values[:, 2*width//3:]

# Calculate the mean depth in each region
mean_center_depth = np.mean(center_region)
mean_left_depth = np.mean(left_region)
mean_right_depth = np.mean(right_region)

# Decision logic
if mean_center_depth > threshold:
    return "forward"
elif mean_left_depth > threshold and mean_right_depth < threshold:
    return "left"
elif mean_right_depth > threshold and mean_left_depth < threshold:
    return "right"
else:
    #If we see no desirable way to go we can turn
    return "turn left"
```

- We start off by defining the regions by take width and height from the depth values.
- We are then able to take the width and divide it into 3s to give us access to the left, right and center of the screen.
- After this we will then calculate the mean of each region and we will compare this against our threshold value.
- After comparing we will then return a direction to move or to turn too.

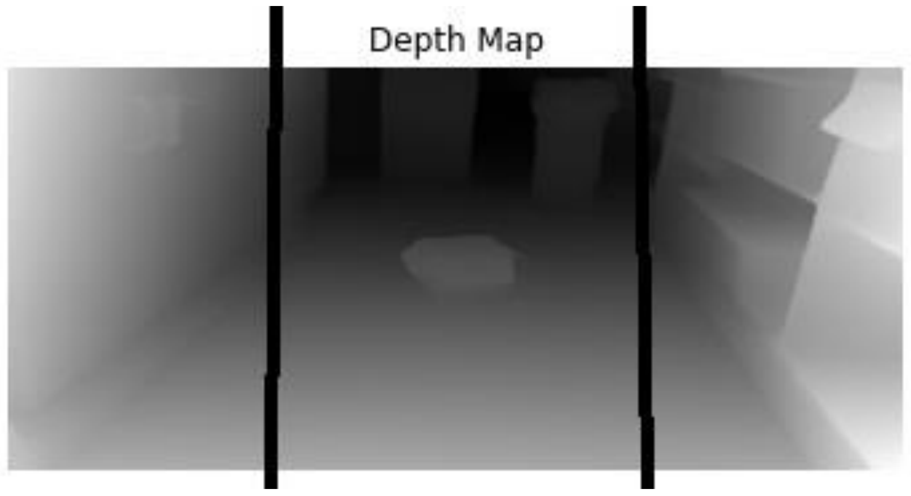
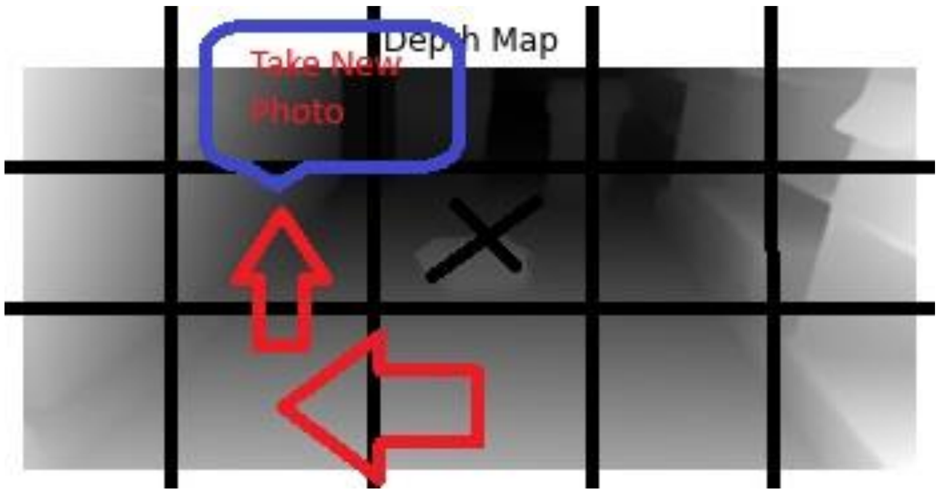


Figure 2 Depth Estimated Img Default Segmentation

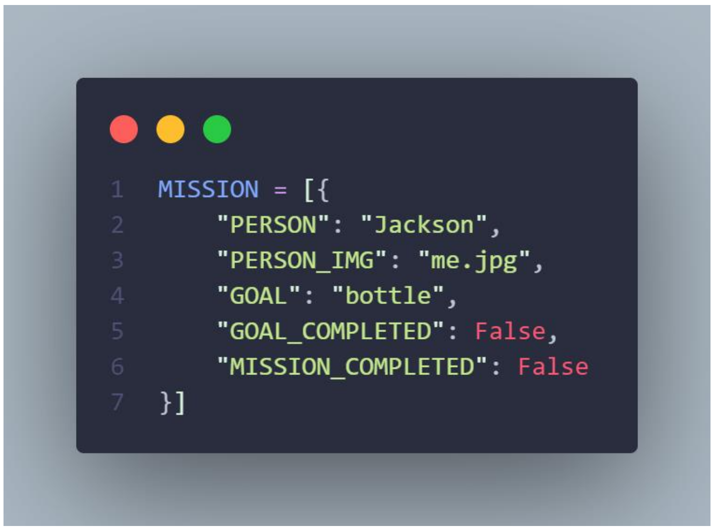
So this is a visualization of how the image looks as well as the segmentation I have applied.

A future implementation for this script would be to use it for creating pathfinding algorithm and the way I would achieve this would be by segmenting the image into a grid and build up a path in the screen this would once more require additional processing power.



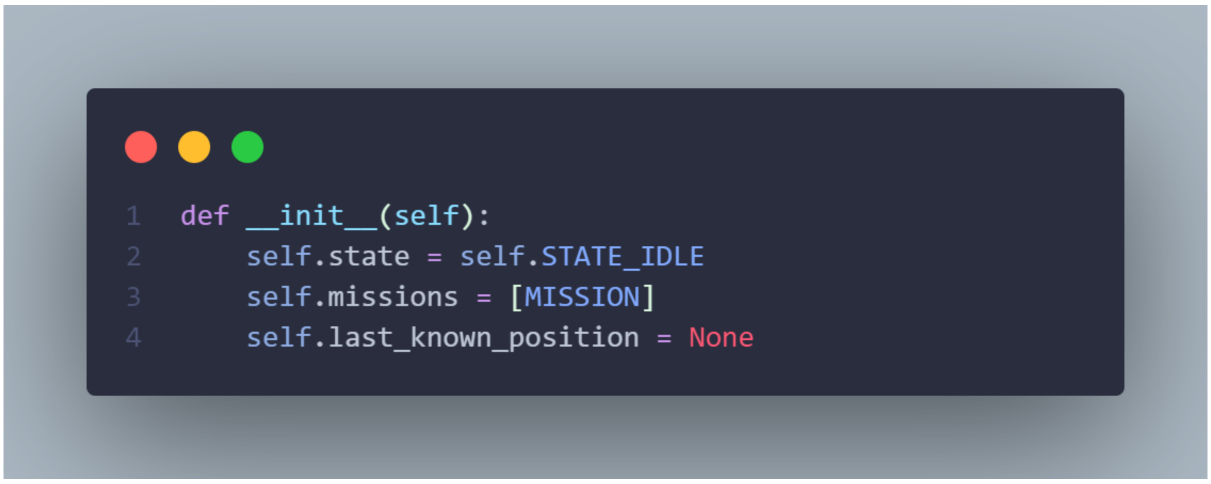
The process would look something like this where we would take the middle close row as the starting point and then tell it to look at the middle squares around it and decide which one is clearer and repeat until the next square meets a sufficiently distant square.

Controller



```
MISSION = [{
    "PERSON": "Jackson",
    "PERSON_IMG": "me.jpg",
    "GOAL": "bottle",
    "GOAL_COMPLETED": False,
    "MISSION_COMPLETED": False
}]
```

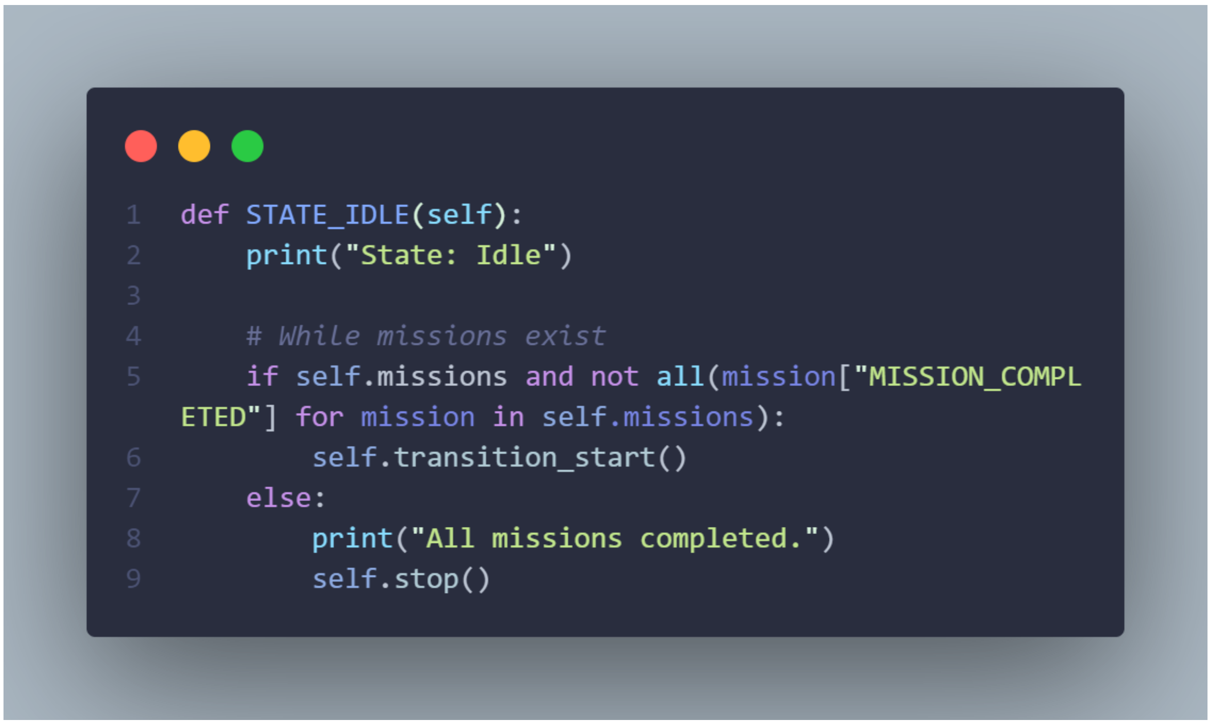
- In the controller we have missions which will control how the robot will want to move.
- For example with this one it will take the name of the Person its looking for as well as a picture of them to be used as the training image.
- The goal is the thing they are looking for and if the goal.
- While the goal isn't completed the robot will only be in a searching mode and it will not worry about looking for people this saves on processing time as it takes awhile when just doing one of these checks let alone 2.
- When all missions are completed the robot will then end the script and shutdown.



```
def __init__(self):
    self.state = self.STATE_IDLE
    self.missions = [MISSION]
    self.last_known_position = None
```

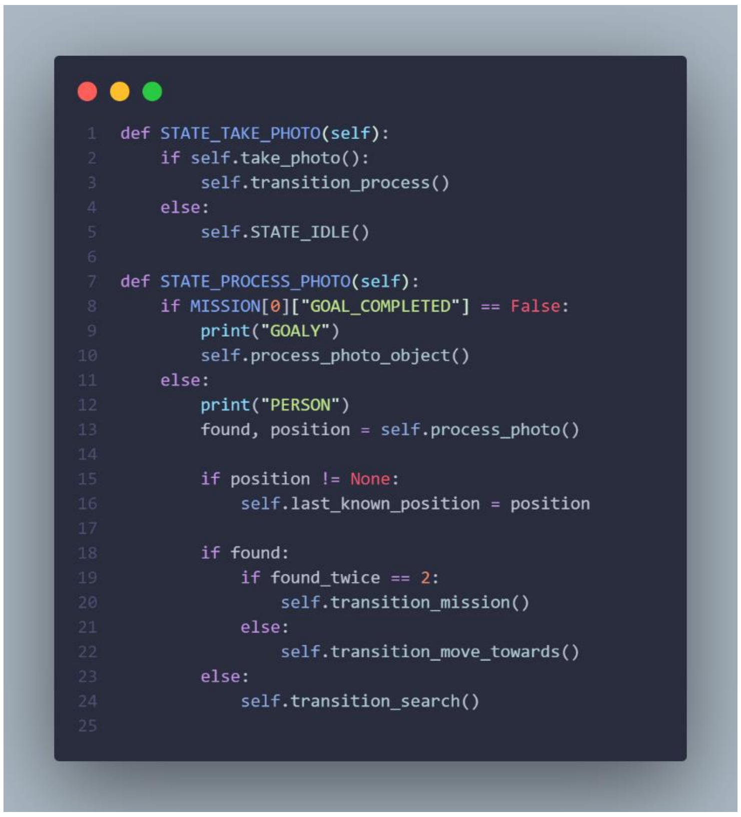
- The initialization state will set the default state to STATE_IDLE
- Then Ingest the MISSION/MISSIONS if there is more than one in the list.

- It will set last known position to none. This will be used later on if when a second photo is taken the target is lost it will give it an idea of where to look next.



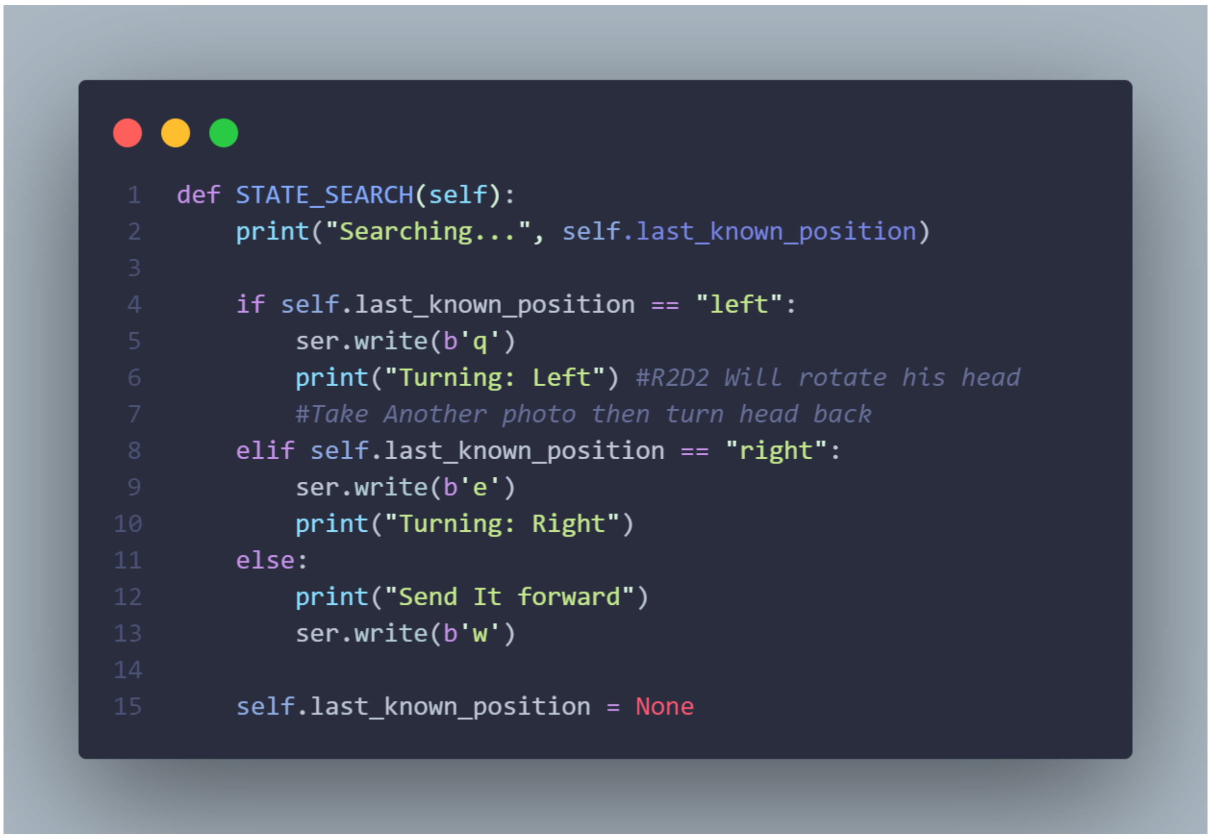
```python
def STATE_IDLE(self):
    print("State: Idle")

    # While missions exist
    if self.missions and not all(mission["MISSION_COMPLETED"] for mission in self.missions):
        self.transition_start()
    else:
        print("All missions completed.")
        self.stop()
```

- In this state we will do our check for missions not being completed.
- If the missions aren’t completed we will enter the transition_start state.
- Otherwise we will stop which will be its state until it receives more missions alternatively we can also set it to a roaming state. Where it will just be able to use the Depth Mapping Pathfinding I mentioned earlier.



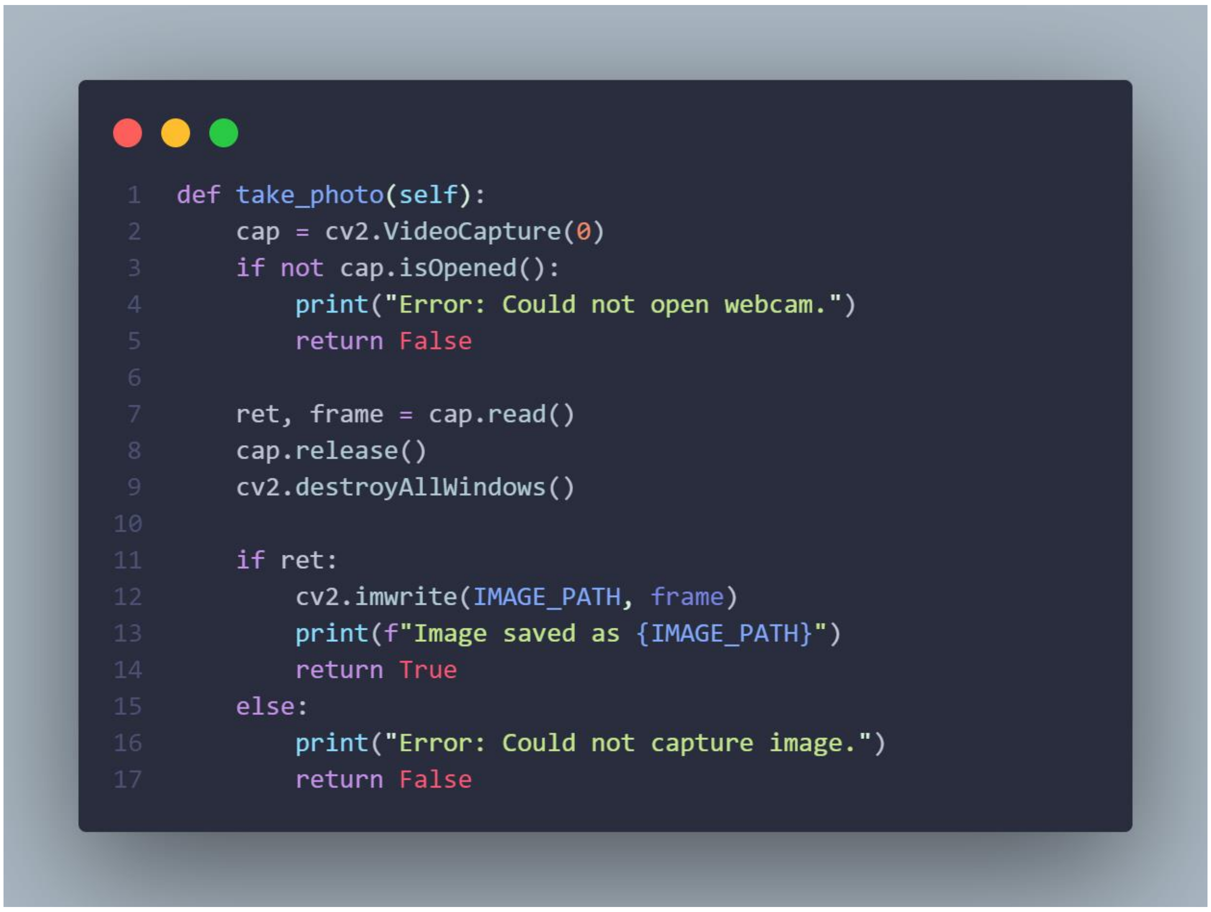
```python
def STATE_TAKE_PHOTO(self):
    if self.take_photo():
        self.transition_process()
    else:
        self.STATE_IDLE()

def STATE_PROCESS_PHOTO(self):
    if MISSION[0]["GOAL_COMPLETED"] == False:
        print("GOALY")
        self.process_photo_object()
    else:
        print("PERSON")
        found, position = self.process_photo()

        if position != None:
            self.last_known_position = position

        if found:
            if found_twice == 2:
                self.transition_mission()
            else:
                self.transition_move_towards()
        else:
            self.transition_search()

```

- Starting with the Take photo function we will take a photo that will be used for this loop in the run.
- After the photo is taken we will enter the transition state.
- If the photo failed we will enter the idle state
- After entering the process photo state we will check if the goal is completed if its not we will process with the object detection script.
- If the goal is completed we will look into the facial recognising script instead.
- We will update last known position if a value was returned.
- If the person being searched for is found we will then check if they were found just the once or twice.
- The reason for checking twice is because while the depth analysis is being refined we want to be able to still move closer to the person.
- If the person isn’t found we will enter the searching state.



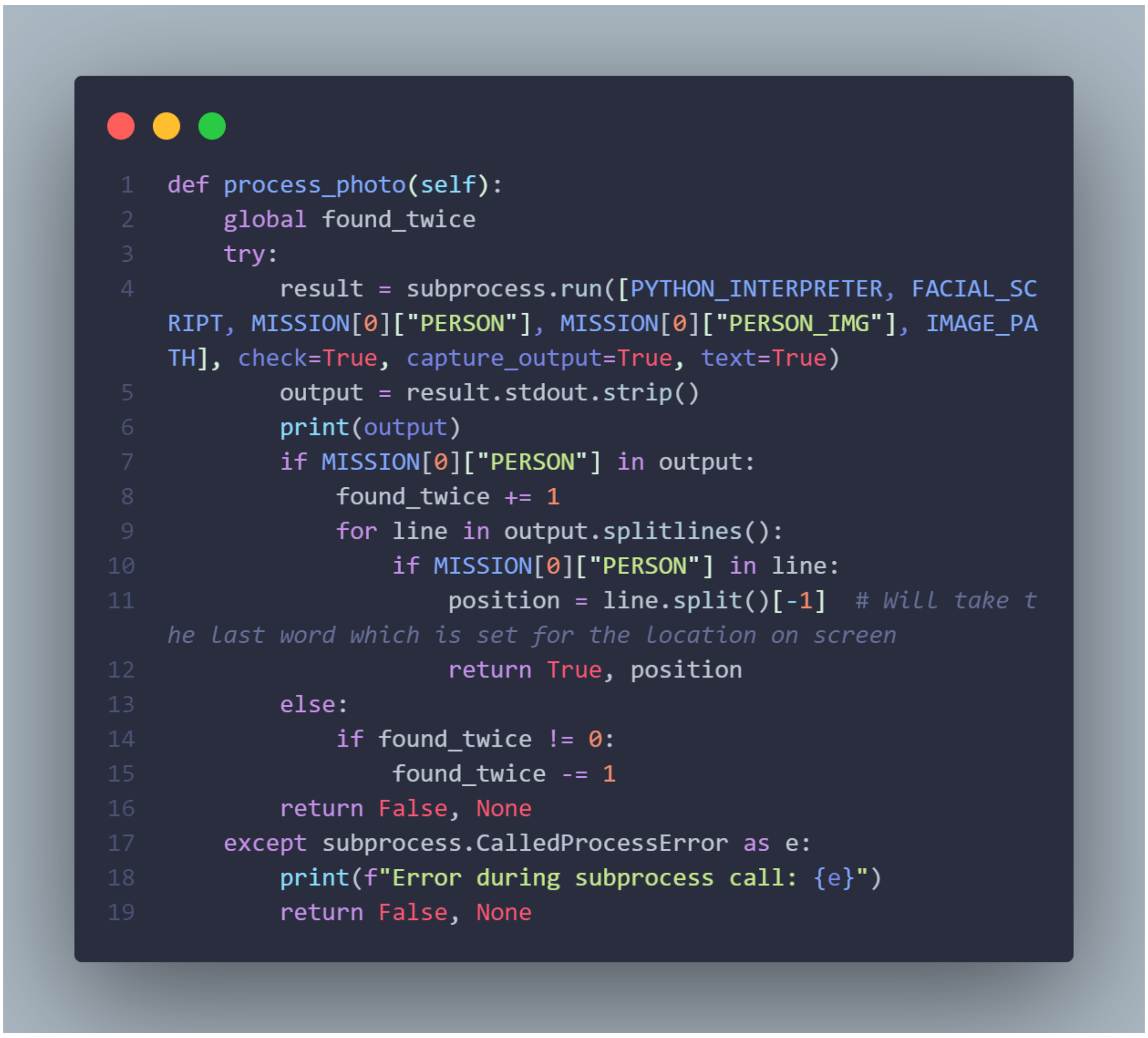
```python
def STATE_SEARCH(self):
    print("Searching...", self.last_known_position)

    if self.last_known_position == "left":
        ser.write(b'q')
        print("Turning: Left") #R2D2 Will rotate his head
        #Take Another photo then turn head back
    elif self.last_known_position == "right":
        ser.write(b'e')
        print("Turning: Right")
    else:
        print("Send It forward")
        ser.write(b'w')

    self.last_known_position = None
```

- When entering the search we will be checking the last known position if the last known position
- If it was left we will turn the head and in doing so the camera left in the case that we are working with a stationary camera we will adjust Arduino script to compensate for turning the chassis instead this all goes the same for turning right.
- Otherwise we will send it forward and reset the last known position back to none.



```python
def take_photo(self):
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Could not open webcam.")
        return False

    ret, frame = cap.read()
    cap.release()
    cv2.destroyAllWindows()

    if ret:
        cv2.imwrite(IMAGE_PATH, frame)
        print(f"Image saved as {IMAGE_PATH}")
        return True
    else:
        print("Error: Could not capture image.")
        return False
```

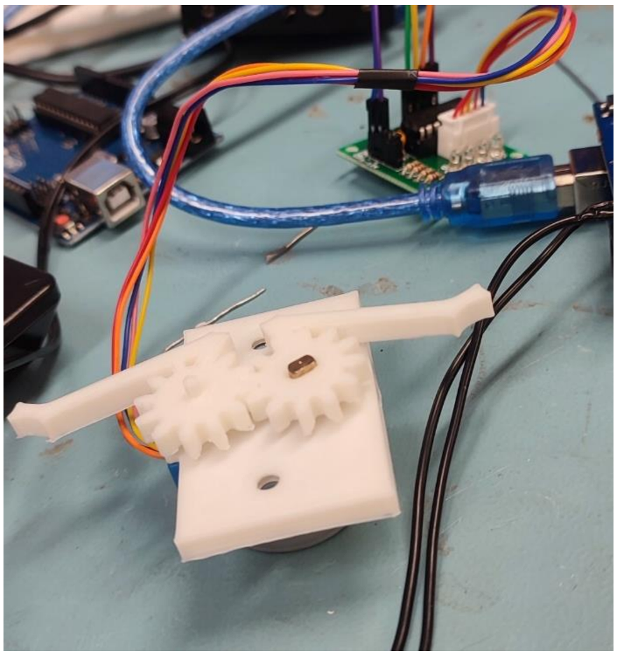
- The take_photo function will be used when ever we need to take a photo it utilizes open cv2
- If the camera cant be opened we will thrown an error
- If the photo is successfully taken successfully it will then be saved under the IMAGE_PATH variables name which will make it accessible to the other scripts



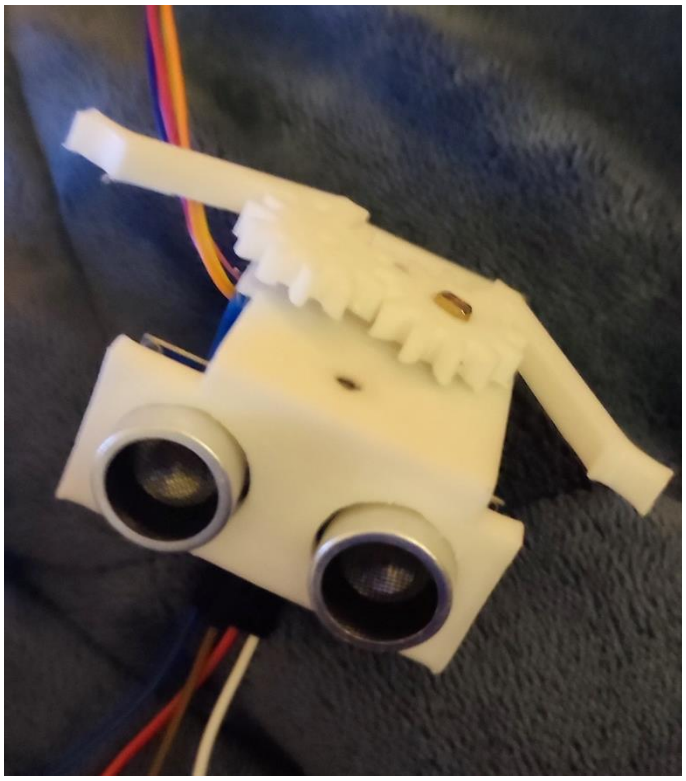
```python
def process_photo(self):
    global found_twice
    try:
        result = subprocess.run([PYTHON_INTERPRETER, FACIAL_SCRIPT, MISSION[0]["PERSON"], MISSION[0]["PERSON_IMG"], IMAGE_PATH], check=True, capture_output=True, text=True)
        output = result.stdout.strip()
        print(output)
        if MISSION[0]["PERSON"] in output:
            found_twice += 1
            for line in output.splitlines():
                if MISSION[0]["PERSON"] in line:
                    position = line.split()[-1]  # Will take the last word which is set for the location on screen
                    return True, position
        else:
            if found_twice != 0:
                found_twice -= 1
        return False, None
    except subprocess.CalledProcessError as e:
        print(f"Error during subprocess call: {e}")
        return False, None
```

- When processing the photo we will call the facialdistancing script which will return our results.
- The results will then be processed and we will extract the relevant data and return it
- The first thing that will be returned will be True or False which represents if the person was found.
- The second thing that will be returned will be the position so we know where there located on the screen and whether to drive straight forward, strafe left and strafe right.
- If they are not found we will decrease the found tracker by one till it hits 0.

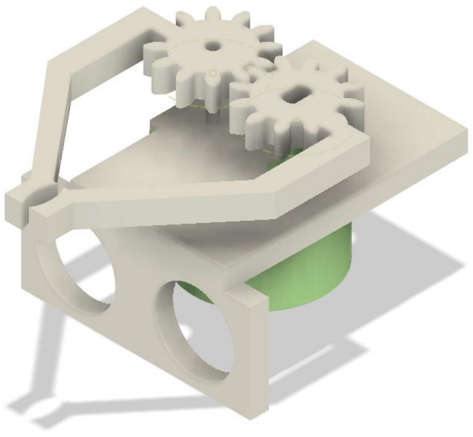
Models



I designed this compact gripper to implement the ability to hold something that will be optioned by the mission the robot is on in tandem with the object detection script.



The gripper will close when the echo sensor detects the object at a specify distanced which in my code base is currently set too 4 CM The echo sensor will lock be locked until the object detection script spots the item.

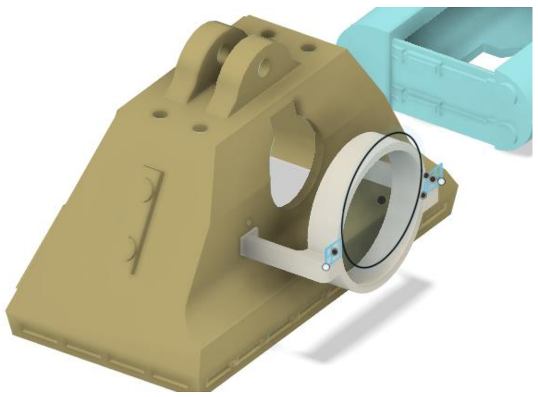


The use of a stepper motor allows for greater grip strength.

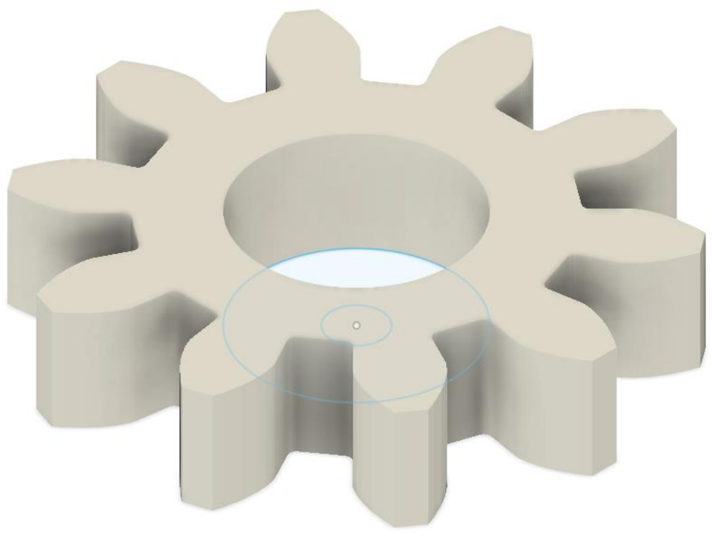
The disadvantage is that I don't have access to specific positions which makes setting an open and close a less precise movement.

I altered the original droid model to use this basic cylinder as its connection between gear that rotates the head and the head. Mainly I made this change because the original design was made for use with parts I wasn't able to get my hands on it also gives me more space for wires in it specifically the webcam.

This motor mount was designed to be used on the foot. After having issues with the wheels and stopping the slipping I had to move away from the movement implementation. My fix to this issue would be to redesign the feet myself.

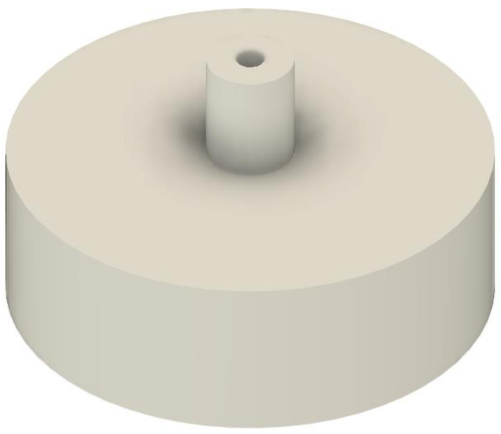


Finding a motor like this would aid in the issue as the dc motors I had access to and was able to find had round shafts Ideally I would want something like this using a D shaft in combination with a tight fit wheel and D shaped hole this would remove the slipping on the wheel itself.

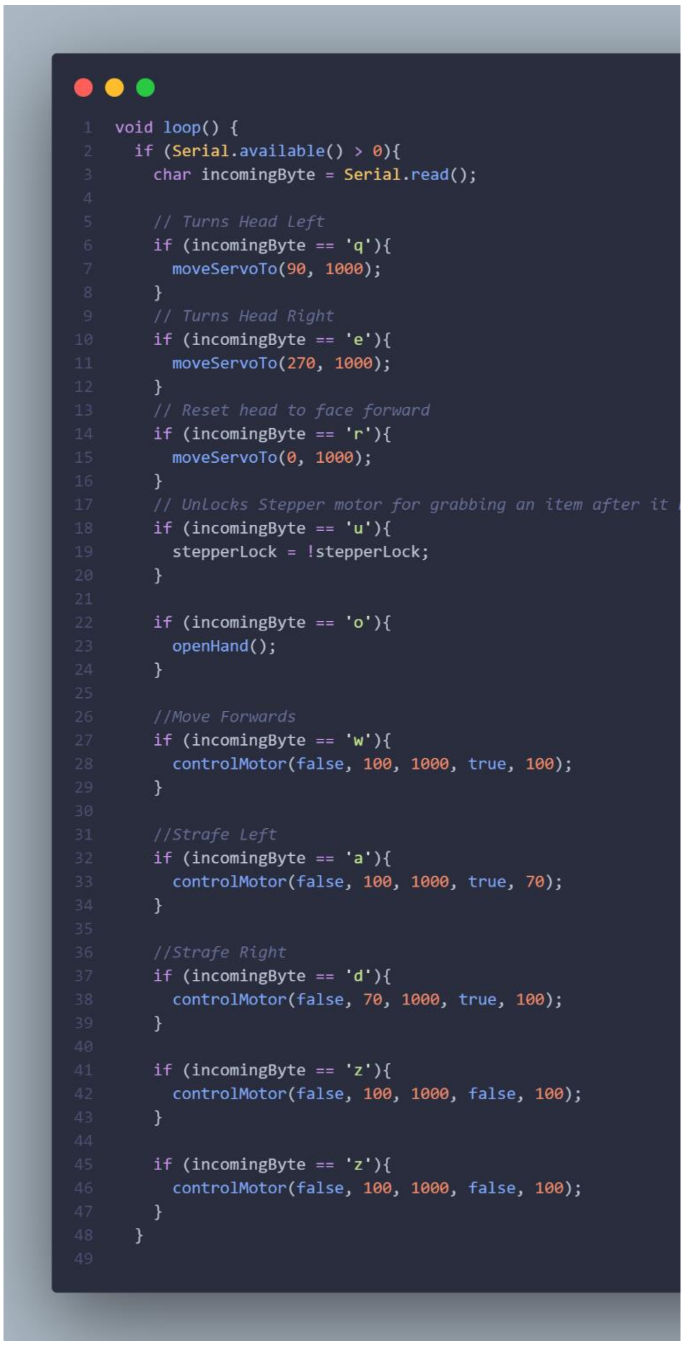


I redesigned this gear to work with my servo motor to turn the head.

The wheel I initially designed to work this the original models feet. I added the shaft to allow for extended reach to compensate for the strange dimensions of the foot and limited reach off the motor shaft.



Arduino



```
void loop() {
  if (Serial.available() > 0){
    char incomingByte = Serial.read();

    // Turns Head Left
    if (incomingByte == 'q'){
      moveServoTo(90, 1000);
    }
    // Turns Head Right
    if (incomingByte == 'e'){
      moveServoTo(270, 1000);
    }
    // Reset head to face forward
    if (incomingByte == 'r'){
      moveServoTo(0, 1000);
    }
    // UnLocks Stepper motor for grabbing an item after it
    if (incomingByte == 'u'){
      stepperLock = !stepperLock;
    }

    if (incomingByte == 'o'){
      openHand();
    }

    //Move Forwards
    if (incomingByte == 'w'){
      controlMotor(false, 100, 1000, true, 100);
    }

    //Strafe Left
    if (incomingByte == 'a'){
      controlMotor(false, 100, 1000, true, 70);
    }

    //Strafe Right
    if (incomingByte == 'd'){
      controlMotor(false, 70, 1000, true, 100);
    }

    if (incomingByte == 'z'){
      controlMotor(false, 100, 1000, false, 100);
    }

    if (incomingByte == 'z'){
      controlMotor(false, 100, 1000, false, 100);
    }
  }

```

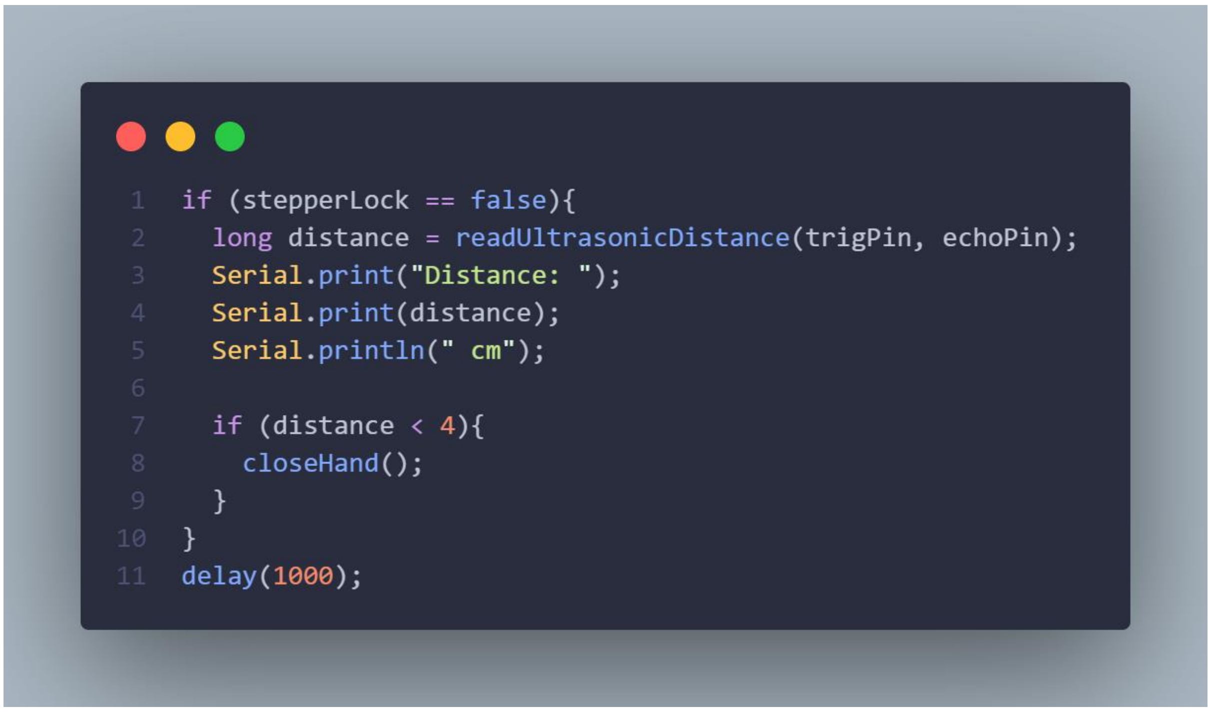
In our loop we setup Serial input this will control most of the moves the Arduino makes.

Our first 3 Commands will be used to turn the head of the droid used for searching for a person.

Our 4th Command will control the locking and unlocking of the Gripper.

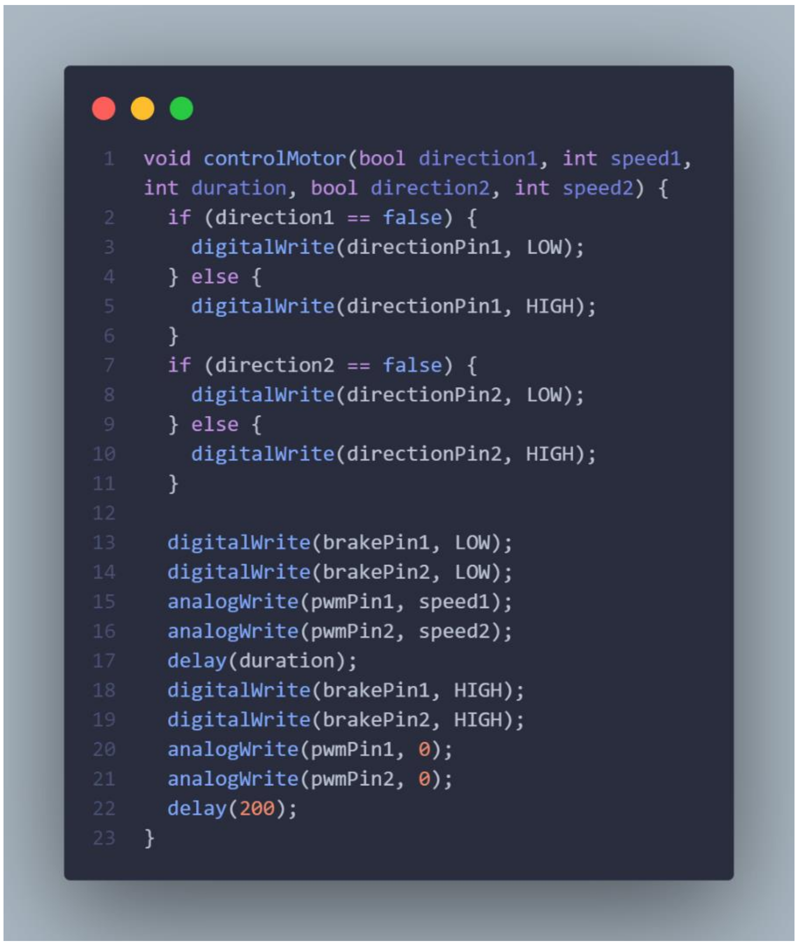
Our 5th Command will allow us to open the hand this will be un affected by the locking mechanism and will be done as the final command when ending a mission.

Our 6th through 10th commands will be used in controlling the movement motors.



```
if (stepperLock == false){
  long distance = readUltrasonicDistance(trigPin, echoPin);
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  if (distance < 4){
    closeHand();
  }
}
delay(1000);
```

When the stepper is unlocked we are able to use the echo sensor to tell us if the item is in the range of the gripper.



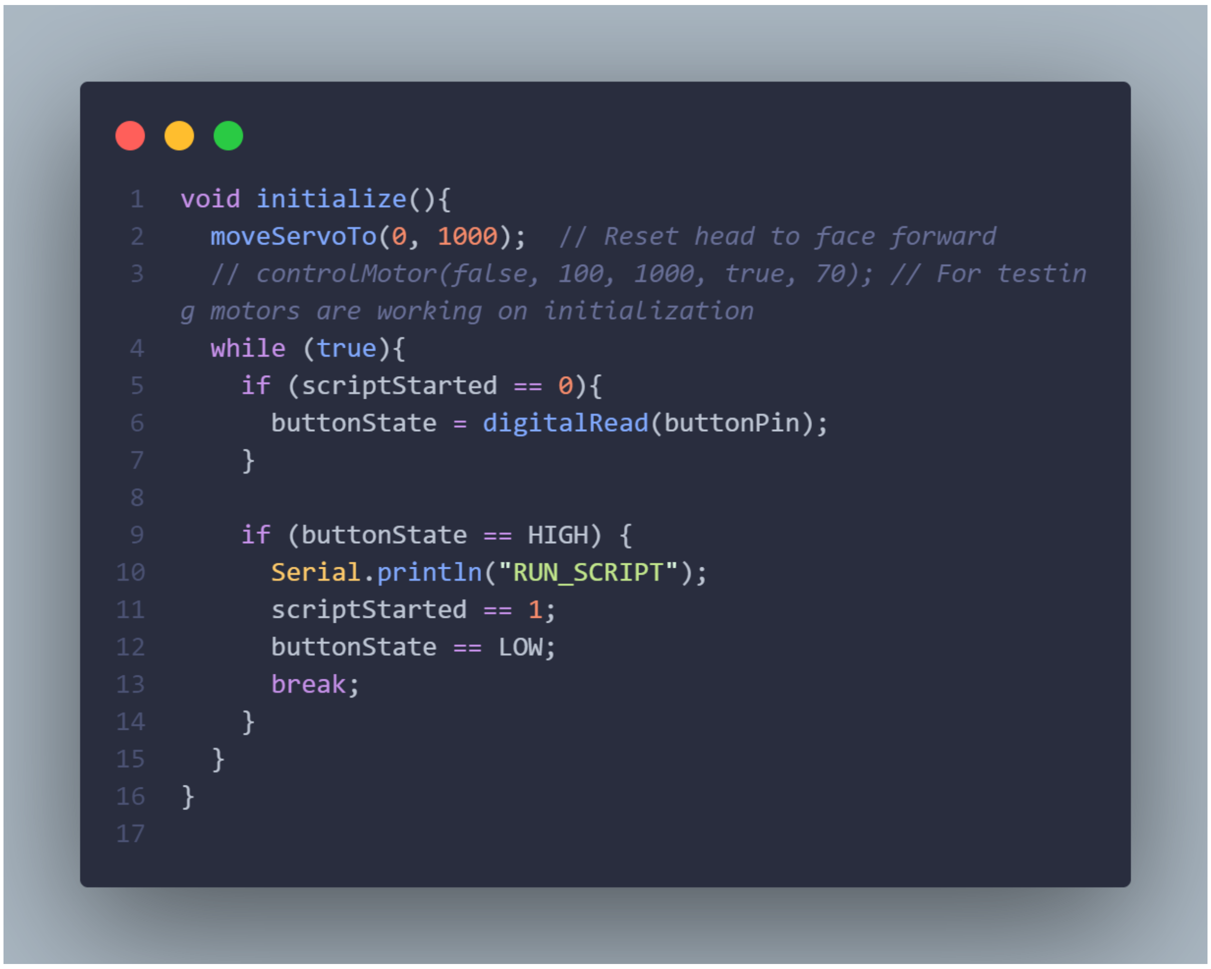
```
void controlMotor(bool direction1, int speed1,
int duration, bool direction2, int speed2) {
  if (direction1 == false) {
    digitalWrite(directionPin1, LOW);
  } else {
    digitalWrite(directionPin1, HIGH);
  }
  if (direction2 == false) {
    digitalWrite(directionPin2, LOW);
  } else {
    digitalWrite(directionPin2, HIGH);
  }

  digitalWrite(brakePin1, LOW);
  digitalWrite(brakePin2, LOW);
  analogWrite(pwmPin1, speed1);
  analogWrite(pwmPin2, speed2);
  delay(duration);
  digitalWrite(brakePin1, HIGH);
  digitalWrite(brakePin2, HIGH);
  analogWrite(pwmPin1, 0);
  analogWrite(pwmPin2, 0);
  delay(200);
}
```

This function will take in the direction and speed for Motor 1 and the same for motor 2 and a duration they will go for that they will both share.

Based on the given direction we will write the direction to the motors and then write the brakes to low turning them off and then setting speed.

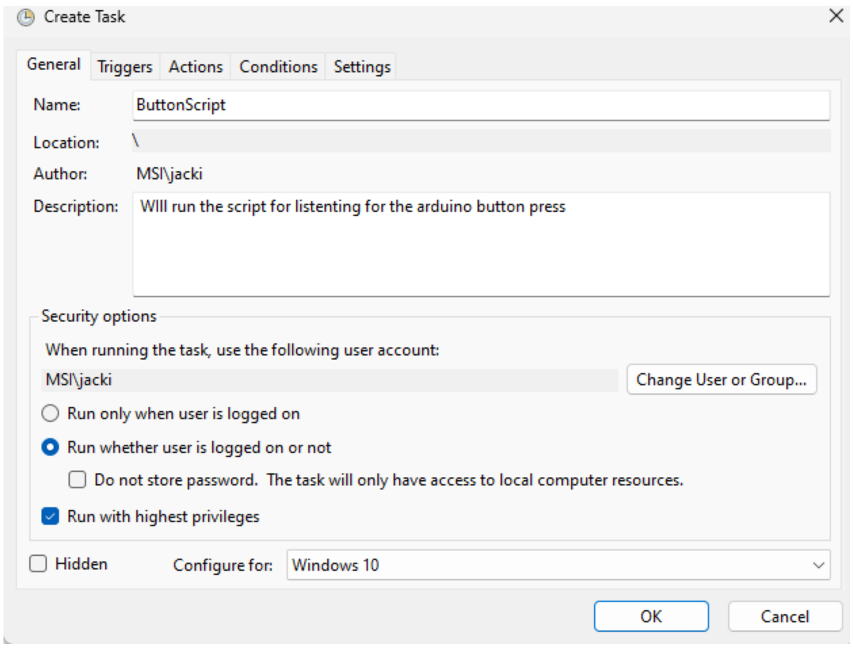
We will then end it by

setting the speed to 0 and re enabling the brakes.



```
void initialize(){
  moveServoTo(0, 1000);  // Reset head to face forward
  // controlMotor(false, 100, 1000, true, 70); // For testin
g motors are working on initialization
  while (true){
    if (scriptStarted == 0){
      buttonState = digitalRead(buttonPin);
    }

    if (buttonState == HIGH) {
      Serial.println("RUN_SCRIPT");
      scriptStarted == 1;
      buttonState == LOW;
      break;
    }
  }
}
```

I also have an initialize function which is for any commands that need to be done during startup its good for component testing aswell but at the moment is currently in use for resetting the servo motor to be at position 0 meaning the head always will be facing forward. It is also being used to wait for initalising the script this is done by having a separate script setup to run on launch that is waiting to receive RUN_SCRIPT over serial



Create Task

General | Triggers | Actions | Conditions | Settings

Name: ButtonScript

Location: \

Author: MSI\jacki

Description: WIll run the script for listenting for the arduino button press

Security options

When running the task, use the following user account:

MSI\jacki — Change User or Group...

( ) Run only when user is logged on

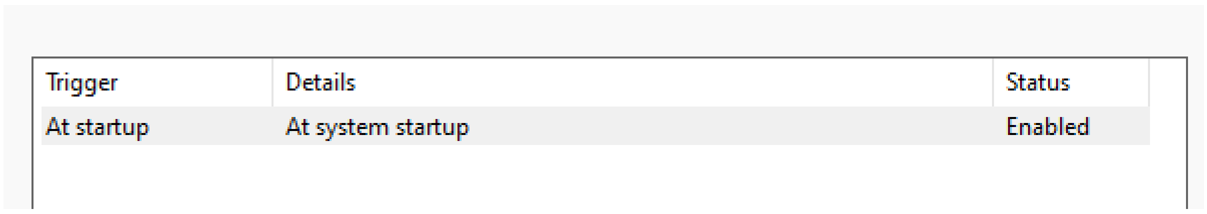
(●) Run whether user is logged on or not

[ ] Do not store password. The task will only have access to local computer resources.

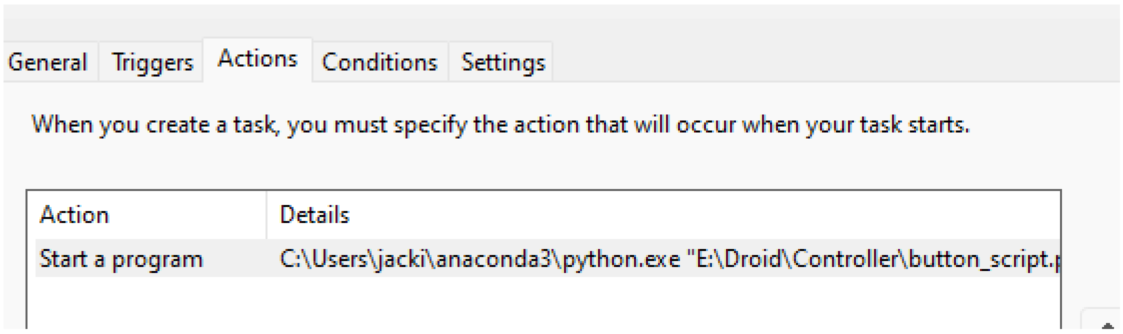
[✓] Run with highest privileges

[ ] Hidden — Configure for: Windows 10

OK — Cancel



Trigger	Details	Status
At startup	At system startup	Enabled

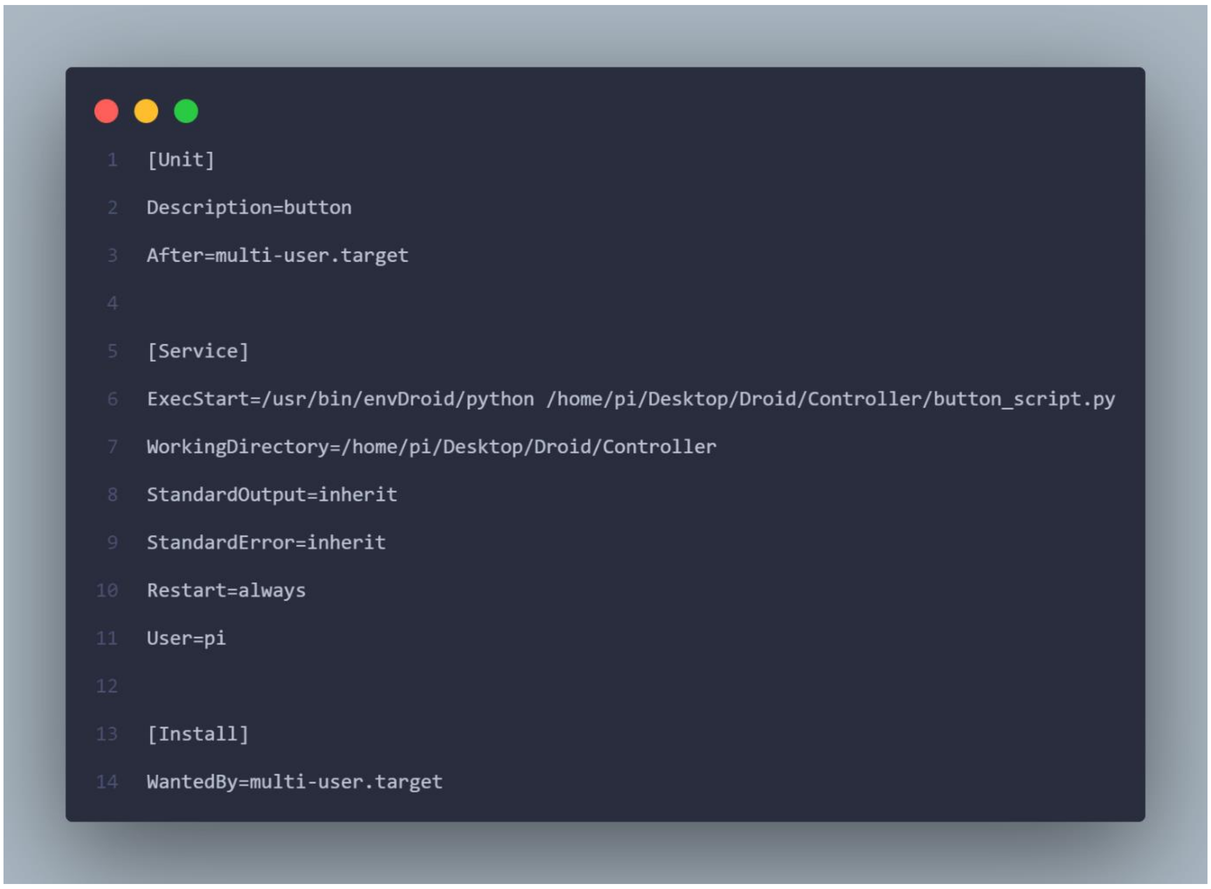


General | Triggers | Actions | Conditions | Settings

When you create a task, you must specify the action that will occur when your task starts.

Action	Details
Start a program	C:\Users\jacki\anaconda3\python.exe "E:\Droid\Controller\button_script.

To always have the script running for windows as I have been using throughout my test process you would setup a task to always startup on launch as seen above alternatively we could use a service if we were Linux based such as on a Raspberry Pi



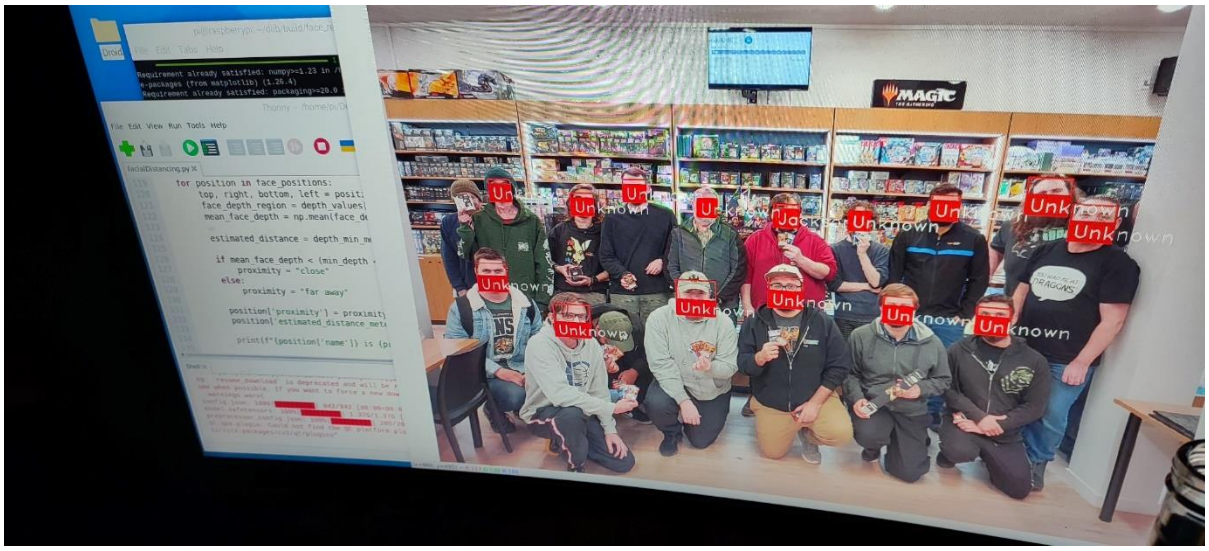
```
[Unit]
Description=button
After=multi-user.target

[Service]
ExecStart=/usr/bin/envDroid/python /home/pi/Desktop/Droid/Controller/button_script.py
WorkingDirectory=/home/pi/Desktop/Droid/Controller
StandardOutput=inherit
StandardError=inherit
Restart=always
User=pi

[Install]
WantedBy=multi-user.target
```

If we are using a Linux based system such as a raspberry pi which was the original intention you would use Daemon like this to achieve the goal.

The biggest issue I encountered when working on this project was hardware based it came down to parts not working or parts not working as expected or just not having access to the parts that would be optimal.

I also faced issues when working with serial causing intermittent signals to be sent I believe this might be due to using usb to serial as when I want doing my research I read some content about it not always being available or designed in mind with modern computers meaning even though it was working from time to time on my laptop it may have been having issues sending which may also be related to some of the port conflicts that were happening from time to time.

An issue/complication was also the speed in which the raspberry pi was able to process the opencv data the below image took around 3 minutes to process and while there were a fare few faces in it these are types of situations that can and will be encountered. When processing this on my laptop it took about 20-30 seconds in comparison to the raspberry pi this is significant. I would choose in future to change a motherboard ideally with a graphics card to allow for fast processing.



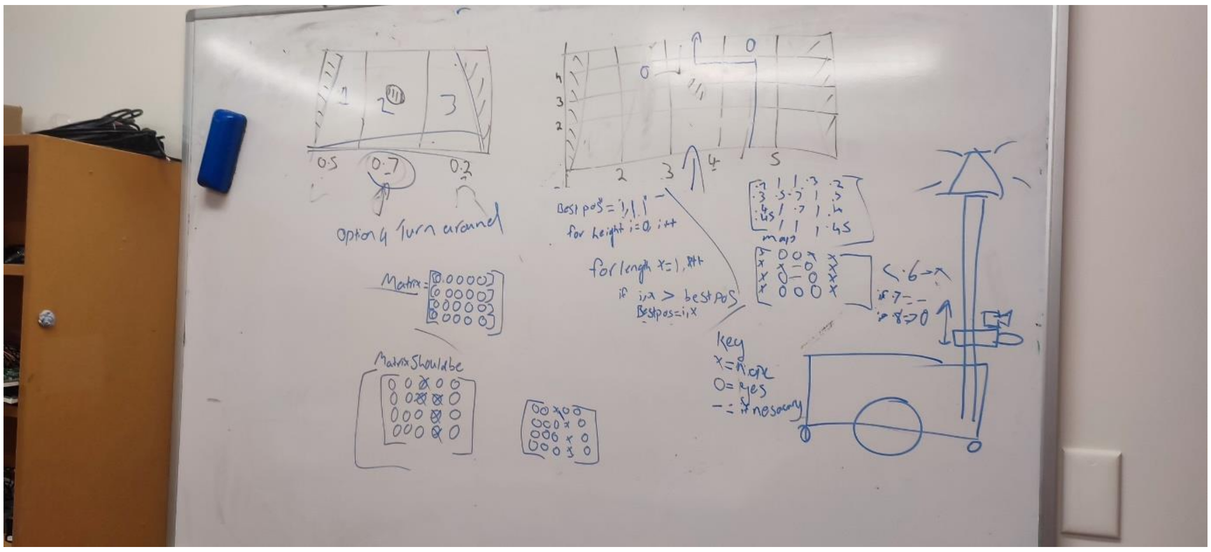
```
Requirement already satisfied: numpy>=1.23 in /
e-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0
```

```
for position in face_positions:
    top, right, bottom, left = posit
    face_depth_region = depth_values[
    mean_face_depth = np.mean(face_de

    estimated_distance = depth_min_me

    if mean_face_depth < (min_depth +
        proximity = "close"
    else:
        proximity = "far away"

    position['proximity'] = proximit
    position['estimated_distance_mete

    print(f"{position['name']} is {p
```

On this front we were able to work out most of the issue I was able to get a Servo Motor that works and a stepper motor that works after some issues and a lot of muddling around I also got my regular old dc motors working but I encountered an issue mentioned earlier I was un able to over come and that was the wheels slipping and I would be unable to solve them in time the way I would be able to fix it would be with motors that don't use a circular driving shaft as for this reason no matter the tolerance put into the wheel hole it would always be at the mercy of slipping with enough force on it and enough ware but if I had D shaft or any kind of shaft that was chamfered with some kind of edge I would have been able to eliminate this issue.

Given the chance to work on such as project again instead of looking to build a platform from scratch like this especially when I am working under a time crunch I think I would use a prefabricated platform like a redbot or more ideally something like FarmBot (The Polytech resident roaming friend) in this situation I would have had most of the hastles of motors sorted and would also give access to use a larger computer to power it than the planned raspberry pi which has proven to not be up to the task. Aswell as would have allowed me to put more focus into my controller which has in my opinion been a standout and is something I wouldn't of imagined being able to make 6 months ago.

I would also have liked to put more time into my pathfinding this was something I couldn't at the moment it work on only a couple of dimensions of information something I concluded near the end of the project was that I could segment up the DepthMeasuring script into a grid of say 3*8 allowing for a matrix to build a path forward like it was moving up a chess board checking which squares are clear enough to move through.



I put this together to workout what some ways of implementing this would be to check through with my own algorithm checking for the optimal position in the first row then checking the same position in the second row if its clear build up otherwise check spaces next to it if ones clear move there check above repeat.

Alternatively I could go through and map positions that cant be used and then take another pathfinding algorithm and provide it with the available positions such algorithm could be A* which was first designed for pathfinding for a robot with the goal of working out the shortest path to a single objective although one factor to consider is processing time as searching for many paths at once will cause slower processing especially with a high amount of grid squares but we will have a fairly low processing time since the amount of squares in the grid are set to limited size.

I would also like to implement a script that would allow for the conversion of an audio file that could be sent over serial to play audio I did attempt to make one but I had issues dealing with tones and frequency once it made it to the Arduino I believe this was caused by the Arduino not reading all of the bytes sent at the right time. I think it would have also been call to have api that would allow for setting new missions and getting them on initialization and

Throughout this project I have learned about different technologies ways of thinking and programming techniques that have helped me build this prototype this ranges from the controller script where I utilised a class based system to build the controller which definitely has some interesting concepts I enjoyed working with. I used a servo motor for the first time in a project which while is nothing impressive it gave me some useful insight into pros and cons with why I would use a stepper motor or a servo. Aswell as this I worked with speakers for the first time which gives me some interesting insight into how speakers work creating audio using varying frequencies and durations is an interesting concept to think about and isn't something I have really ever thought about how it works. It has also allowed me to learn about concepts in 3D design and printing specifically in which it came to the grabber I had never figured out how to properly design gears until now and while I'm definitely no expert I would be able to move forward with basic designs that need gears.

Going forward I have learnt to put less priority on Fabrication and design and to put more time into my programming and work with more prebuilt designs to allow for a more well put together piece of software that will be able to be shown off.

https://github.com/fuzzyers/Droid

Please feel free to go through and use any of the code to build your own machines.