COMOVO - Control, Motion, Voice

Neeti Ganjur: Electrical and Computer Engineering, Carnegie Mellon University
Gauri Laxman: Electrical and Computer Engineering, Carnegie Mellon University
Shrutika Ruhela: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—**This document details the design and development of our solution to multi-person video calling difficulties. In a multiple person video call, especially for personal use, holding a smart-device and moving it around towards whoever is speaking is a hassle. This device is a rotating platform that holds your phone and has the ability to rotate towards the person who is speaking. This device will have two operation modes, manual and automatic, which allow different levels of control to the users. Over the last few weeks, we conducted further research on the hardware components necessary for the project and created a schematic for the way in which the hardware components will fit together and connect to one another. We also delved into details of our software implementation and designed the interfaces between the various sensors, the motor and the Raspberry Pi. Finally, we elaborated on the overall control loop for the project, our plan for testing the device, the potential risk factors of this project, and our projected schedule for project development.

I. Introduction

AS a majority of our team has family located internationally, we are familiar with the pains of video calling into family dinners and events from across the world. Especially with multi-person video calls it becomes increasingly hard for the phone to be passed around between every new speaker. In order to alleviate this problem, we decided to come up with a solution in the form of a rotating device that sits on your table and holds your phone. We decided to name our device COMOVO which stands for Control, Voice, Motion.

Existing solutions include Facebook Portal [7] integrated with Amazon Alexa, Cisco Webex [8] and Zoom [9]. All of these are quite expensive, video-calling platform dependent and are mostly for office space conferencing. Our goal is to make an easily affordable, video-calling platform independent device that is meant for personal use.

The device will be able to function in two modes - automatic and manual. In automatic mode, the device collects video frame data through a camera and sound data through multiple microphones, processes both these types of data to detect the location of the speaker and then rotates the motor (and thus, the phone that sits atop the device) to face the speaker. In this mode, there is no inter-COMOVO communication. In manual mode, the user on one end of the call will be able to use hand gestures to control the direction and duration of rotation of the motor on the other end of the video call. In this mode, the two COMOVO devices will communicate with each other through a protocol we define, independent of the phones that sit on top of them. The device will boot up by default in automatic mode and we will allow mode switching through our inter-device communication protocol.

In this report, we have outlined the hardware and software implementation designs for COMOVO as well as our plans and metrics for testing and verification. We also discuss potential risk factors and details of our project management.

II. Design Requirements

Table 1 shown below details the factors and metrics that we are concerned with for testing our device. For the accuracy of gesture detection we will be looking at the ability of the CNN classifier to correctly identify a hand when it is being used to gesture in manual mode and identify the direction of rotation intended. We aim to have the classifier produce greater than 85% accuracy on the testing dataset that we will create and feed it. This metric is based on our research of existing models that classify similar data. We expect that when people gesture in manual mode, they will be approximately a foot away from the device. Thus, our model should be able to detect a hand that is a foot away from the camera. This metric comes from the average distance of a person from their phone when they are video calling.

For the accuracy of 'loudest speaker' detection, since we will be working with four directional microphones, we expect that the accuracy of rotating to the correct quarter will be quite high and so we are aiming for a minimum 95% hit rate. Our estimate for the distance of the person speaking from the device is 3 feet. This number came from our research on the average dinner table radius.

Finally, we will be testing latency by building log creation into our scripts and recording time difference through these logs. In automatic mode we will be measuring the difference in time between when the speaker starts talking and when the device has finished rotating towards the speaker. In manual mode we will be measuring the time difference between when the gesture or keystroke input is fed in to the device and when the device has finished rotating towards the speaker. Based on our research, video calling latency which is over a UDP connection is approximately 300ms. Our latency for manual mode will be slightly higher as TCP connections are slower than UDP connections. However, the more relevant metric that we will be using to measure success for latency is if the total latency is less than 2.3 seconds. This metric comes from data collection on the average time taken for a person to move from their seat and rotate their phone to the current speaker.

TABLE 1. Metrics and Testing

Feature	**Metric**	**Success Values**
(M) Gesture Detection	% accuracy of gesture classifier on testing dataset	> 85%
(M) Distance of person motioning from COMOVO	Distance in feet	< 1ft
(A) Accuracy of 'loudest speaker' detection	% times COMOVO rotates to correct quarter	> 95%
(A) Distance of people speaking from COMOVO	Distance in feet	~ 3ft
(Both) Latency	Time taken to receive, process and execute command	Begin rotating in < 2.3s
(Both) RPi - RPi communication latency	Time taken to transmit data over TCP	~ 300 ms

III. Architecture and/or Principle of Operation

Our system architecture has two main parts. The hardware interfaces and connections and the software interfaces and specifications. The hardware block diagram is shown below in Fig. 1. We will be using a Raspberry Pi 4 Model B [4] for the COMOVO's main processor. We decided to use a stepper motor and a motor hat to easily drive and power the motor. Since we need visual input for our computer vision processing we have a camera and since we need audio signal input, we decided to use four directional microphones to get the range and accuracy we want. These microphones will be mounted on the platform and will be stationary. The camera and phone will be on the rotating part of our device since they both need to be in sync.

The stepper motor requires 12V and draws 350 mA and will be driven by the motor hat which is powered by an external 12V battery pack. It will be connected to the motor hat through the ground and motor ports. The motor hat is connected to the RPi through two specific GPIO pins, SDA and SLC.

Our current plan is to use 4 USB microphones connected to the RPi's USB ports through adapters so that we have extra wire length to mount the microphones correctly on the platform. It is possible that we might switch to different microphones in the near future if we find out that the microphones we have chosen do not have the sensitivity we require or do not have the directional capabilities we need. In this case, the new microphones we choose might be connected to the GPIO pins instead of the USB ports.

Our camera will be connected to the MIPI CSI port of the RPi. We also have a temporarily connected USB keyboard that we will be using as an incremental testing tool for our gestures in manual mode. This will allow us to separately develop motor control and gesture detection using computer vision. The RPi itself will be powered by a USB-c cable which provides the 5V and 3A required.

The software block diagram is shown below in Fig. 2. The RPi runs Raspbian OS which will simplify a lot of our interfaces. The RPi will communicate with the motor hat and consequently the motor through the I2C bus. We will be able to control the speed, direction, and degrees of rotation of the motor through specific methods of the stepper motor object that we create as an instance of the PWM controller class through the motor hat library. We have already used PyGame to process keystroke input from the USB keyboard. We will use OpenCV [3] to process frames from the camera's output video stream and we plan on using PyAudio [2] and SoundDevice [1] to read and compare the volumes of microphone data received by the RPi.

Depending on which mode the devices are in, we will apply either a hand/not-hand classifier or a head/not-head classifier on the video frames. In automatic mode, we will use the head/not-head convolutional neural network in conjunction with our microphone amplitude data to locate the speaker. In manual mode, we will use the hand/not-hand convolutional neural network and information about the location of the hand in the frame to recognize "right" and "left" gestures and classify the implied direction of rotation.

In terms of the overarching control loop, both devices (holding phones on either end of the video call) will begin by running a headless script that establishes a client-server model TCP connection (which we have already been able to implement). We will use socket programming to send and receive messages between the RPis. They will then run the appropriate python scripts to process sensor data, depending on the mode. Subsequently, if the sensor data reaches a specific threshold, the loop will trigger the script for motor rotation (also dependent on the mode).

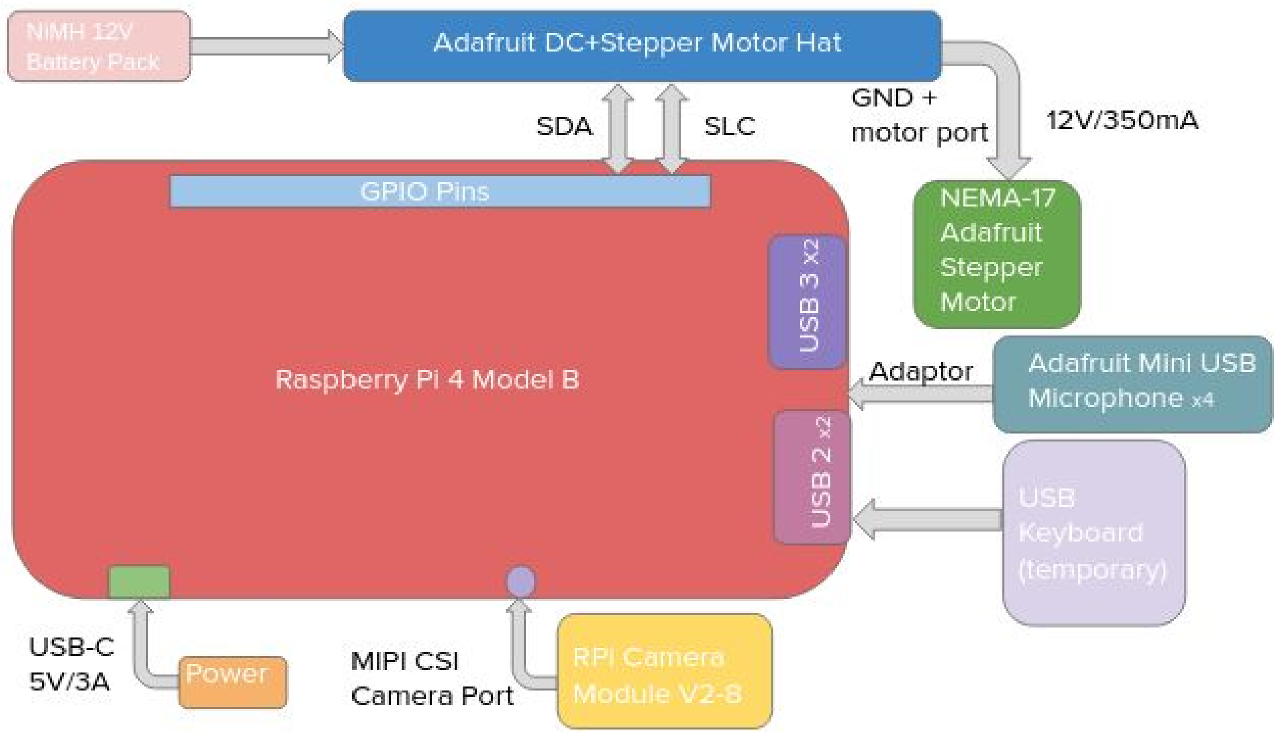
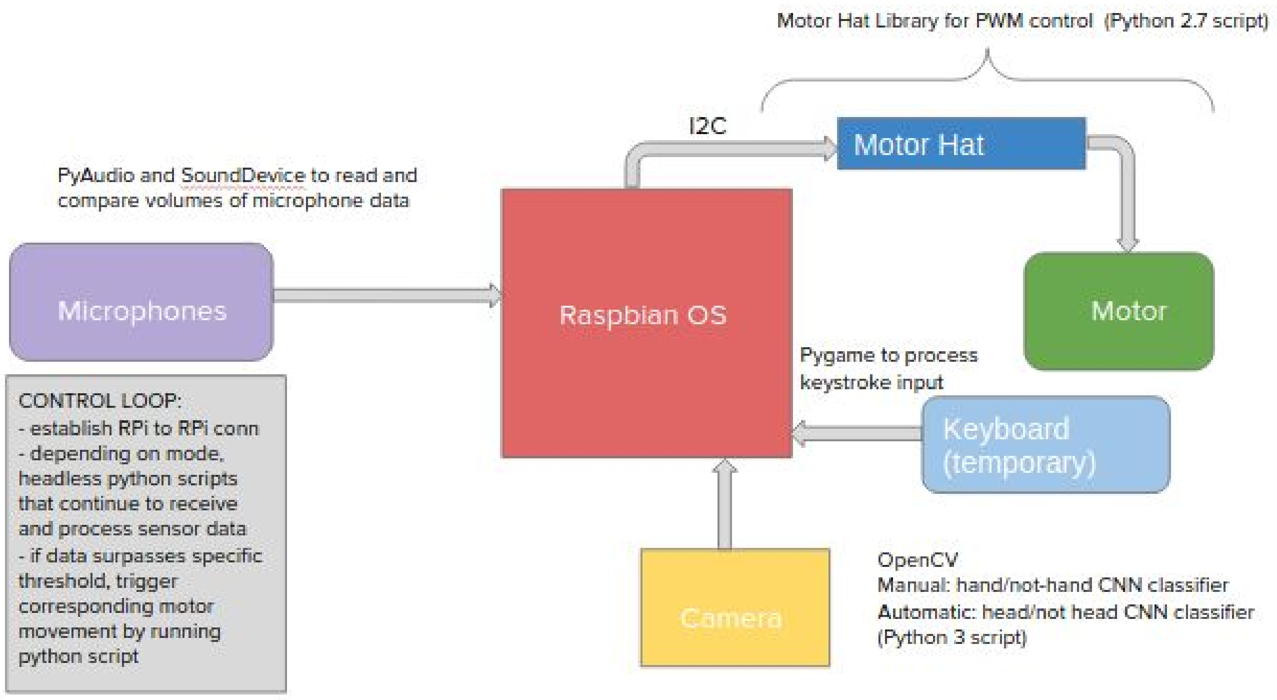


Fig. 1. Hardware Block Diagram


Motor Hat Library for PWM control (Python 2.7 script)
I2C
Motor Hat
PyAudio and SoundDevice to read and compare volumes of microphone data
Raspbian OS
Motor
Microphones
Pygame to process keystroke input
Keyboard (temporary)
Camera
OpenCV
Manual: hand/not-hand CNN classifier
Automatic: head/not head CNN classifier
(Python 3 script)

CONTROL LOOP:
- establish RPi to RPi conn
- depending on mode, headless python scripts that continue to receive and process sensor data
- if data surpasses specific threshold, trigger corresponding motor movement by running python script

Fig. 2. Software Block Diagram

IV. Design Trade Studies

We decided to use the Raspberry Pi 4 model B over the Raspberry Pi Zero Model W which we were initially considering, because it provides more sensor input ports, is still small enough for our use case and has WiFi capability. There is also plenty of documentation, user guides and support readily available for this specific model of the RPi since it is one of the newest models.

We opted to use the Adafruit DC and Stepper Motor Hat for our project as it conveniently sits on top of the RPi and contains motor controllers which are necessary to power to the motor since the GPIO pins cannot provide enough. The motor hat also allows us to control the stepper motor through the motor hat library which provides a high level of abstraction for PWM control and for setting the direction, speed and degrees of rotation of the motor.

For the motor we decided to use the Adafruit Stepper Motor in the NEMA-17 size as it is small enough to uphold our size requirement for the device and provides high torque rotation at low speeds. This particular motor also has a precision of 200 steps per revolution (or 1.8 degrees per step) which is more than sufficient for our use case. The motor, motor hat and RPi are compatible with one another and there are a lot of existing user guides and previous projects that use the three components together.

The camera we decided on is the Raspberry Pi Camera Module V2-8mp as it provides the image quality that we require, is a custom RPi add-on, and has been used in previous projects with similar use cases. We wanted relatively high quality camera output since we will be performing computer vision analysis on the frames received from the camera.

Finally, for the additional power supply that will be connected to the motor hat to power the stepper motor, we will be using a simple NiMH 8xAA battery pack as it inexpensive, easily available, simple, and provides the 12V supply we need.

V. System Description

Our main interfaces are described in further detail in the subsystems below. As described in the block diagrams section III, each interface has a hardware and a software component. Additionally, the code for these interfaces will be modular in order to compartmentalize the pieces and enable unit testing. The integration will be primarily in the control loop.

A. Motor-RPi Communication

The first thing we have to do to allow the RPi to communicate with the motor through the motor hat is to enable I2C on the RPi. This is a simple step that can be accomplished using raspi-config. We will be using PWM (Pulse Width Modulation) to control the speed, direction and degrees of rotation of the motor. Instead of writing the drivers ourselves, using the motor HAT allows us to use the Python Adafruit DC + Stepper Motor HAT library [6]. We will instantiate an object of the PWM controller class to represent the stepper motor and feed in arguments to specify the connection port and steps per revolution that we want the motor to rotate at. This object has methods that we will use to direct the motor's rotation.

For manual mode, we will take in the direction specified by the gesture and translate that to a specific amount of degrees to rotate. This information will then be sent to the other COMOVO and the motor on that side will be rotated by the specified amount.

For automatic mode, we will calculate the degrees to rotate locally on each Pi based on the sound and camera signals data and rotate the motor on the same side by that amount.

B. Camera-RPi Communication

We will connect the camera to the RPi through the MIPI CSI port which allows Python OpenCV [3] to directly recognize the camera. We can read video frames at a fixed rate (based on how frequently we want to process the data) using the OpenCV video capture method and feed these frames to the appropriate classifier based on the mode.

For manual mode, we want to use the camera feed to detect the gestures on the controller's side. For automatic mode, we want to use the camera feed to detect the presence and location of a head in the quarter specified by the microphone input. This will enable us to rotate the platform within the chosen quarter until we find a head.

C. Microphone-RPi Communication

This is the interface that will require the most testing and experimentation in order for us to figure out the best way to analyze audio data for our project. Similar to OpenCV for camera output, PyAudio [2] is capable of reading audio signals from the microphones connected to the RPi if we specify the ports. However, it is difficult to get volume information easily using PyAudio and so we found another open source library called SoundDevice [1] that enables users to calculate volume given a sound signal.

Since the microphones are stationary on the platform, each one corresponds to a specific quarter in the full circle. Our plan is to intermittently pick the quarter whose microphone is receiving the maximum volume sound input. Using cardioid directional microphones with parabolic baffles, we hope to get an accurate estimate of which quarter the speaker is located in. This will also take care of ignoring sound coming out of the phone's speaker as that will be located behind each microphone. The microphones are only relevant for automatic mode.

We did initially consider using a common method for sound localization called Time Difference of Arrival (TDOA) [5] but this did not seem like a good choice of algorithm for our use case because the microphones on our platform will be placed less than 6 inches apart and thus there will likely be no significant time difference between the arrival of sound at each microphone. This problem of the distance between the microphones may also prove to be an issue with our chosen method of volume comparison, but we think that using directional microphones should reduce the severity of the problem should it arise.

D. RPi-RPi Communication

The two COMOVOs will communicate over a TCP connection using standard socket programming functions like send and recv. We will establish a protocol of commands and sequences of events to perform the two key functionalities needed: sending the calculated direction to other RPi in manual mode and mode switching.

E. Control Loop

The main control loop will run on both COMOVOs and begin as headless scripts that startup on boot. This loop will continually process sensor inputs based on the current mode and will also check for mode switch commands. This means that in manual mode, the camera video feed will be processed by this loop on the controller's side and in automatic mode the camera and microphone data will be processed on both sides. The loop will also keep polling the listening socket for any new command information received from the other end. We will write this in Python for ease of integration with the other interfaces' code which will all be in Python.

VI. Project Management

A. Schedule

Our schedule has been affected since the original proposal planned schedule because of delays in ordering and receiving parts, changes to our proposed sound localization algorithm and certain setup procedures taking slightly longer than anticipated. We had to parallelize some items in our schedule in order to account for these changes. The schedule is shown in Fig. 3 on page 7.

B. Team Member Responsibilities

Neeti has experience in embedded software, networks and machine learning. She will lead the effort to build the ML models and create datasets to train and test them. She will also work with Gauri to control the motors and convert camera output to motor rotation.

Gauri has experience with operating systems, embedded systems and networks. She will be working on interfacing with the motors and processing sound data from the microphones in order to perform sound localization. She will also be working with Neeti on building the ML models since both of them have taken Pattern Recognition Theory.

As a software engineer with experience in networks, Shrutika is responsible for designing the inter-RPi communication protocol and ensuring that they communicate reliably and effectively. A lot of this will be in the integration phase once the devices are set up and receiving sensor input. She is also in charge of building the physical platform and fitting all of the different parts together.

C. Budget

The project Bill of Materials is shown in Table 2 on page 6 and Table 3 on page 6 shows the parts we acquired from the inventory for free.

D. Risk Management

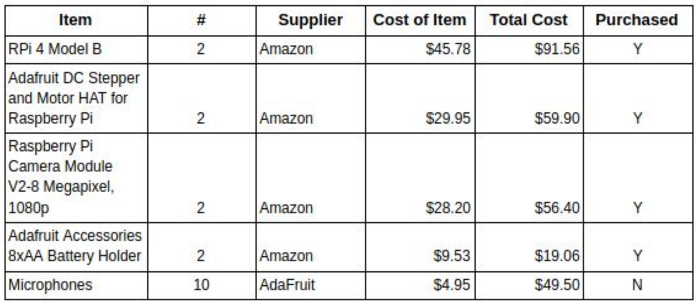
We have identified several risk factors for our project. The first risk factor is related to the gesture detection input, taken by the Raspberry Pi Camera Module to classify left and right gestures, and forward them to the paired COMOVO. The risk here is that the accuracy rate of identifying the gestures depends on the size and quality of our dataset. To mitigate this risk we plan to narrow the scope for the purpose of the demo to recognizing a gesture with a specific background (against a plain black/white background). This way our dataset can be high quality and specific, and our gesture detection and rotation will work with a higher rate of accuracy.

The second risk factor is the accuracy of the sound localization. We are planning for a very high accuracy rate of the COMOVO rotating towards the loudest speaker, and we have identified several risks regarding ambient sound, sound from the phone speaker and the accuracy of directional microphones. We plan to use cardioid directional microphones to ensure that too much ambient noise is not picked up, but to further mitigate this risk, for demo purposes, we may use a quieter room, and narrow the scope to detecting between a specific number of people rather than an undefined number of people surrounding the device.

The third risk factor is the potential for higher latency than is anticipated. Since our use case is for long distance video calls, our short distance testing might not accurately represent the latency for our original use case.

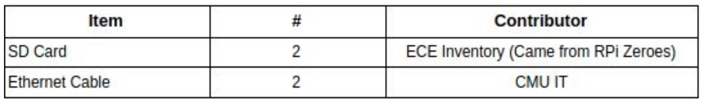
Finally, the performance of our device is heavily dependent on parameter tuning and the physical design of the device - the classifiers will need to be tuned to accurately recognize heads and hands, the directional microphone sensitivities needs to be sufficient to offset the short distance between them and we will need to experiment with different speeds for the motor rotation to pick a stable one.

TABLE 2: Bill Of Materials



Item	#	Supplier	Cost of Item	Total Cost	Purchased
RPi 4 Model B	2	Amazon	$45.78	$91.56	Y
Adafruit DC Stepper and Motor HAT for Raspberry Pi	2	Amazon	$29.95	$59.90	Y
Raspberry Pi Camera Module V2-8 Megapixel, 1080p	2	Amazon	$28.20	$56.40	Y
Adafruit Accessories 8xAA Battery Holder	2	Amazon	$9.53	$19.06	Y
Microphones	10	AdaFruit	$4.95	$49.50	N

TABLE 3: Other Parts (acquired for free)



Item	#	Contributor
SD Card	2	ECE Inventory (Came from RPi Zeroes)
Ethernet Cable	2	CMU IT

VII. Related Work

As talked about in the introduction, we found that the closest product to COMOVO is Facebook Portal [7]. This is quite expensive, is integrated with Amazon Alexa for intelligence and only works with a certain set of video calling platforms. It is an independent tablet-like device that rotates on a platform and cannot be used in conjunction with any phone.

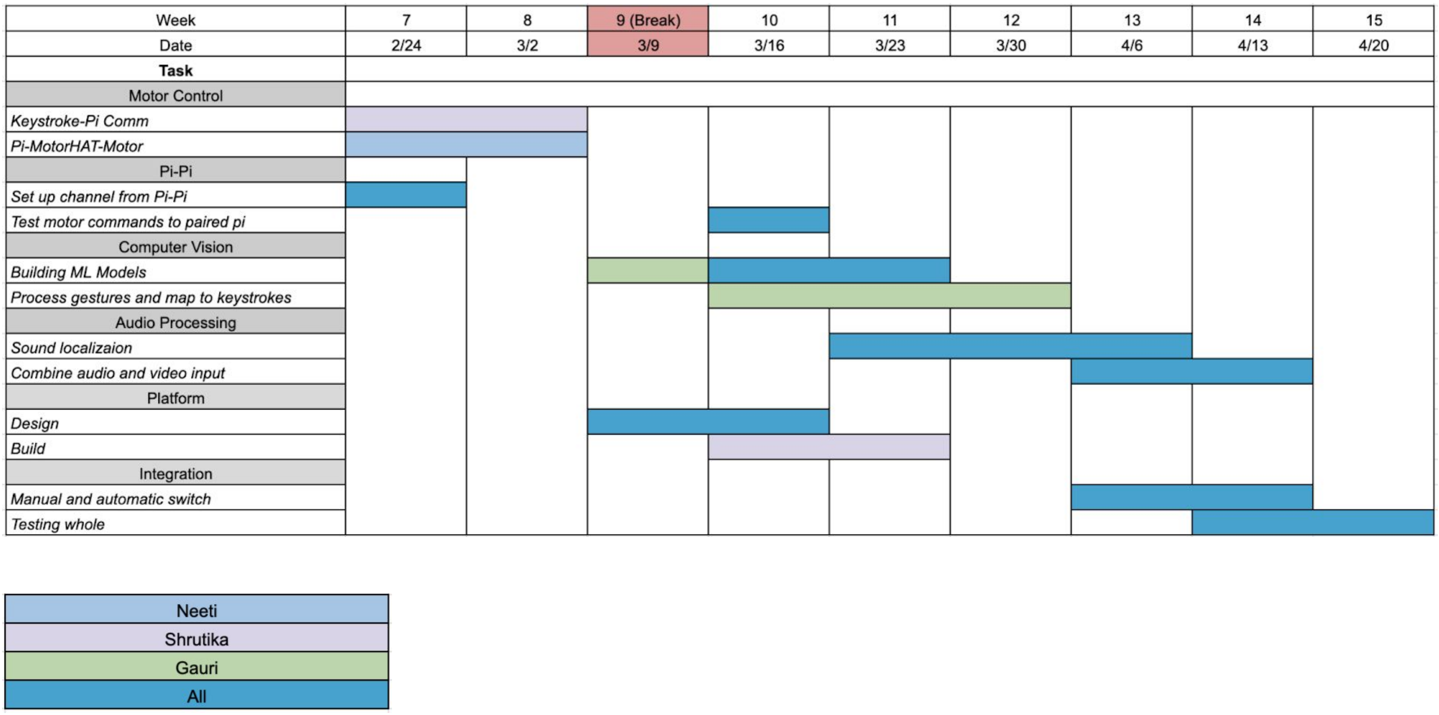
Cisco Webex [8] and Zoom [9] have such audio localization capabilities but are used primarily in office conferencing settings where typically only one person is talking at a time. Also since these products are meant for office conferencing, they are meant to be bought by companies in bulk to be integrated with conference rooms with strategically placed cameras and microphones and are not for personal use.

VIII. Summary

We hope that over the course of the rest of the semester, we are able to build COMOVO successfully and satisfy all our basic requirements. We anticipate hurdles during this process but should be able to handle these with the contingency plans outlined in the risk management section.

References

[1] https://python-sounddevice.readthedocs.io/en/0.3.14/
[2] https://people.csail.mit.edu/hubert/pyaudio/docs/
[3] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html
[4] https://projects.raspberrypi.org/en/projects/raspberry-pi-getting-started/4
[5] https://en.wikipedia.org/wiki/3D_sound_localization
[6] https://github.com/adafruit/Adafruit-Motor-HAT-Python-Library
[7] https://portal.facebook.com/
[8] https://www.webex.com/
[9] https://zoom.us/



Week	7	8	9 (Break)	10	11	12	13	14	15
Date	2/24	3/2	3/9	3/16	3/23	3/30	4/6	4/13	4/20
Task									
Motor Control									
Keystroke-Pi Comm	🟪	🟪							
Pi-MotorHAT-Motor	🟦	🟦							
Pi-Pi									
Set up channel from Pi-Pi	⬛								
Test motor commands to paired pi				⬛					
Computer Vision									
Building ML Models			🟩	⬛	⬛				
Process gestures and map to keystrokes				🟩	🟩	🟩			
Audio Processing									
Sound localizaion					⬛	⬛	⬛		
Combine audio and video input							⬛	⬛	
Platform									
Design			⬛	⬛					
Build				🟪	🟪				
Integration									
Manual and automatic switch							⬛	⬛	
Testing whole								⬛	⬛

🟦	Neeti
🟪	Shrutika
🟩	Gauri
⬛	All

Fig. 3. Schedule - Gantt Chart