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*— On a video call with multiple people speaking on one smart device, holding the smart device and moving it towards the person speaking is a hassle. The solution we envisioned was a rotating platform that holds your phone and adaptively rotates towards the person who is speaking. This document details the design and development of this solution. It outlines the use case and problem space addressed, our vision for the product, system design choices and trade-offs. It also covers the outcome of the design process, metrics for success, testing and validation, and project management details.**

I. Introduction

As the 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 during this COVID-19 pandemic, connecting with family and friends through video calls has become of paramount importance.

Video calls with multiple people on either end are often incredibly inconvenient as the phone has to be moved around constantly to face the person speaking. Human tendency is to be lazy and this can often lead to the phone being set down somewhere and causing confusion among participants on the call as to who is speaking. In order to alleviate this problem, we came up with COMOVO. The name COMOVO stands for Control, Motion, Voice.

Our initial thought process was to have a user be able to simply download an application onto their phone which would register and configure their COMOVO device to communicate with any other user's COMOVO around the world. User A would place their phone in a little notch in their COMOVO, choose to video call any group of people (also with a registered COMOVO on their end) over any video calling application such as FaceTime, WhatsApp, Messenger, etc and have a hands-free enjoyable video chat experience. Inter-COMOVO communication would be enabled by the application sending commands over the internet.

After considering budget and time constraints, our goal evolved into a rotating platform with a notch to hold your phone that would sit on a table on both ends of a video call. It would operate in two modes - automatic and manual. In manual mode, participants on either end of the video call would be able to control the rotation of the platform (and consequently, the phone) on the other end through hand gestures. In this mode, the COMOVO on one end of the call communicates the direction and duration of rotation of the COMOVO on the other end. In automatic mode, the platform would rotate the phone to face the loudest speaker in the room on the same end of the call. In this mode, there is no inter-COMOVO communication. These platforms work independently of the phones that sit on top of them. As a result, they are also independent of the video calling application used.

Due to the COVID-19 pandemic, its implications on our access to resources, and the location of our team members, we had to further modify our goal. We decided to simulate the rotation of the platform using an animation that would behave almost exactly as a physical platform with a motor would have behaved in both automatic and manual mode.

In this report, we have outlined the hardware and software implementation designs for our pre and post-pandemic goals for COMOVO as well as our metrics for testing and verification. We also discuss the final outcome of the design process, tradeoffs we made throughout, roadblocks that we overcame, and details of our project management.

II. Design Requirements

The metrics discussed below were chosen assuming certain testing conditions. For manual mode, we assumed that there would be good lighting conditions. For automatic mode, we assumed a reasonable conversation (one person speaking at a time), minimal background noise, and that 4 participants were seated equally spaced around the table.

Tables 1a and 1b shown below detail the factors and metrics that we were concerned with for testing our device in manual and automatic mode respectively. For the accuracy of gesture detection in manual mode, we looked at the ability of the COMOVO to correctly identify the presence of a hand gesture or absence (just a person’s head). If there was a hand gesture,

we looked at the ability of the COMOVO to correctly identify the direction denoted by the hand gesture (left or right). We required that the classifier produce greater than 85% accuracy on the validation dataset that we created. This metric was based on our research of existing gesture classifiers used in the industry [1], [2]. We assumed that when people gesture in manual mode, they would be approximately a foot away from the COMOVO. Thus, we expected the COMOVO to detect a hand located a foot away from the platform. This metric came from the average distance of a person from their phone when they are video calling, assuming the phone is sitting on a surface.

For the accuracy of 'loudest speaker' detection, we required a minimum 95% hit rate of the COMOVO rotating to within 45° of the person speaking. Assuming 4 people are seated equally spaced around a table, this rotates directly towards the person speaking. This metric was based on our estimate of users' tolerance towards errors. We used this justification as there was no substantial existing research or documentation regarding the accuracy of simple loudest speaker detection. Our estimate for the distance of the person speaking from the device was 3 feet. This number came from our research on the average dinner table radius [3].

Finally, we tested latency by building log creation into our scripts and recording time differences through these logs. In automatic mode, we measured the difference in time between when the user began speaking and when the COMOVO had finished rotating towards them. In manual mode, we measured the time difference between when the gesture was made by the user and when the COMOVO had finished rotating in the direction specified. Based on our research, video calling (which is over a UDP connection) latency is approximately 300 ms [4]. However, the more relevant metric that we used to measure success, was if the total latency was less than 2.3 seconds. This metric came from our own data collection on the average time taken for a person to move from their seat and rotate their phone to the current speaker.

TABLE 1a. Metrics and Testing: Manual Mode

Feature	Metric	Success Values
(M) Gesture Detection	% accuracy of classifier on testing data	> 85%
(M) Distance of person motioning from COMOVO	Distance in feet	< 1ft
(M) Latency	Time taken to receive, process, and execute command	Begin rotating in < 2.3s

TABLE 1b. Metrics and Testing: Automatic Mode

Feature	Metric	Success Values
(A) Accuracy of 'loudest speaker' detection	% times COMOVO rotates to correct quarter	> 95%
(A) Distance of people speaking from COMOVO	Distance in feet	~ 3ft
(A) Latency	Time taken to receive, process, and execute command	Begin rotating in < 2.3s

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 shown below in Fig. 1a describes our hardware specifications before we had to make modifications due to the pandemic. The hardware block diagram shown in Fig. 1b depicts our updated hardware interfaces.

We used a Raspberry Pi 4 Model B [5] for the COMOVO's main processor. We decided to use this model over the Raspberry Pi Zero Model W [6] 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 was also plenty of documentation, user guides, and support readily available for this specific model of the RPi since it was one of the newest models.

To capture gestures, we used a Raspberry Pi Camera Module V2-8 [7]. We decided on this camera as it provided the image quality that we required, was a custom RPi add-on, and had been used in previous projects with similar use cases. We required relatively high quality camera output as we had to feed the frames produced by the camera into our gesture classifier.

To capture audio input, we initially planned on using Adafruit Mini USB Microphones [8] as they were inexpensive, small, and compatible with the RPi. However, due to the pandemic, the manufacturer of these microphones stopped shipping. We also realized after the design review that we required directional microphones for our use case. Unfortunately, due to the pandemic the only microphones we could find that were still available for shipping, were omnidirectional. Thus, we decided to use omnidirectional Zaffiro USB Microphones [9] augmented with baffles to provide directionality.

The 4 microphones are connected to the RPi through the 4 USB ports. The camera is connected to the RPi through the MIPI CSI port. The RPi is powered by a USB-c cable connected to a PC which provides the 5V and 3A required.

Our initial vision for the hardware included an Adafruit stepper motor in the NEMA-17 size [10], to control the rotation of the COMOVO. We also planned to use an Adafruit DC and Stepper Motor HAT [11] connected through the ground and motor ports to the motor. The motor HAT would have been connected to the RPi through two specific GPIO pins, SDA and SLC. The stepper motor required 12V and drew 350 mA and the motor HAT would have been powered by an external 12V NiMH 8xAA battery pack [12]. With regards to the structure of the physical platform, we planned for the microphones to be mounted on the platform and remain stationary. The camera and phone would have been attached to the rotating part of the COMOVO as they both needed to be in sync.

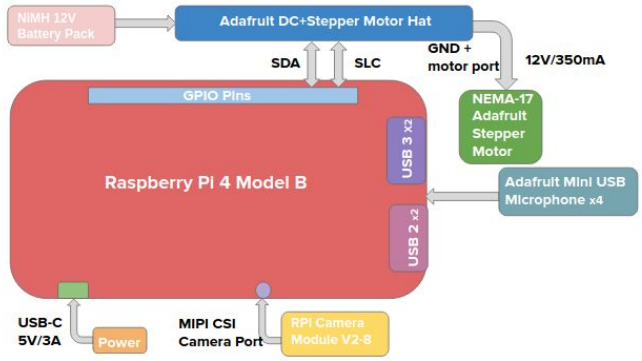


Fig. 1a. Hardware Block Diagram (before modifications)

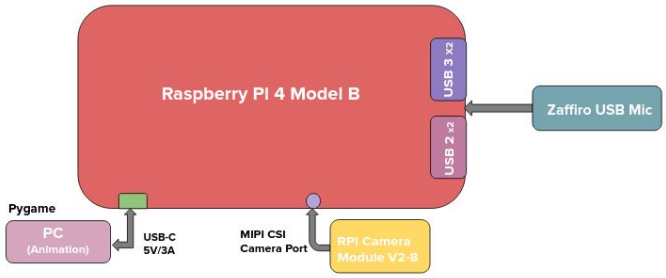


Fig. 1b. Hardware Block Diagram (after modifications)

The software block diagram shown below in Fig. 2a describes our software specifications before we had to make modifications due to the pandemic. The software block diagram shown in Fig. 2b depicts our updated software interfaces.

The RPi runs Raspbian OS [13] which simplified a lot of our interfaces as we did not have to implement our own device drivers to communicate with external hardware and had Python already installed. To capture frames at the chosen frame-rate from the COMOVO camera's output video stream we used PiCamera module [14]. These frames were then pre-processed using OpenCV [15], NumPy [16] and imutils [17] and fed into the convolutional neural network (CNN) that we built using Keras [18], TensorFlow [19] and sklearn [20]. The CNN predicts whether the input frame is a left hand gesture ('L' symbol), right hand gesture ('ok' symbol) or not a gesture (just a person's head). The architecture of the CNN was modeled on the VGG-16 architecture [21]. The predictions made by the classifier were passed to the Pygame [22] animation running on the PC through a TCP connection.

To read and process the audio signals from each of the COMOVO's microphones we used PyAudio [23]. Once the microphone with the loudest signal is detected, the microphone ID is passed to the Pygame animation running on the PC through a TCP connection.

Our initial plan involved the use of the Python Adafruit MotorHAT Library [24] to control the speed and direction of rotation of the stepper motor. The RPi would have communicated with the motor HAT and consequently the motor through the I2C bus. Instead, we created an animation in Pygame to simulate the rotation of the COMOVO.

We used socket programming to send and receive messages between the RPis over TCP and between the RPIs and PC over TCP.

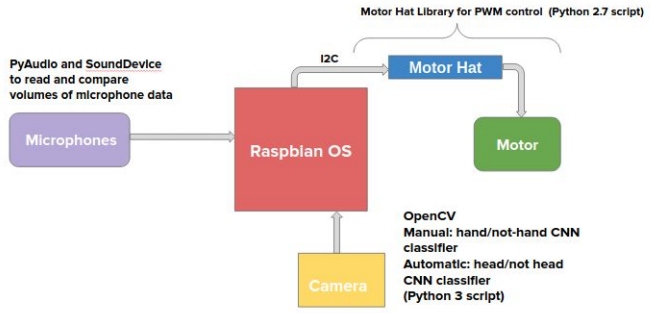


Fig. 2a. Software Block Diagram (before modifications)

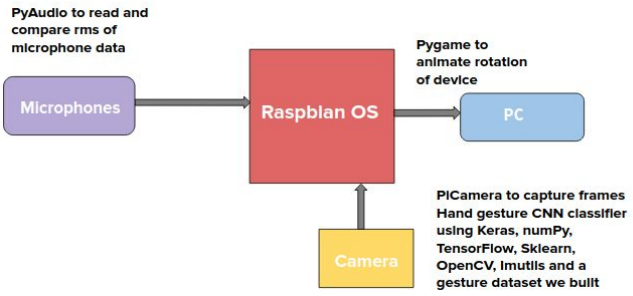


Fig. 2b. Software Block Diagram (after modifications)

IV. Design Trade Studies

In our initial design, we opted to use the Adafruit DC and Stepper Motor HAT for our project as it would have conveniently sat on top of the RPi enabling us to connect the motor to the RPi with less external wiring. It also would have contained motor controllers necessary to power the motor as the GPIO pins would not have been able to provide enough power. Additionally, the motor HAT would have allowed us to control the stepper motor through the motor HAT library which provided a high level of abstraction for pulse-width modulation (PWM) control and for setting the direction, speed and degrees of rotation of the motor.

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

Also in our initial design, we chose a NiMH 8xAA battery pack as the additional power supply that would have been connected to the motor HAT to power the stepper motor. We selected this power supply as it was inexpensive, easily available, simple, and provided the 12 V supply we needed.

While building our gesture classifier, we had to make many trade-offs. First, we arbitrarily chose 'thumbs up' and 'thumbs down' as our left and right hand gestures respectively. However, we realized after experimentation, that we needed hand gestures that were more distinguishable from one another and also from the wall or background. Thus, we chose the 'L' symbol as our left gesture and the 'ok' symbol as our right gesture as they each have a distinct number of fingers raised versus the number of fingers folded.

Second, we started by following an existing Medium tutorial on how to build a CNN hand gesture classifier [25]. This tutorial utilized a 4000-image Kaggle dataset created using a Leap Motion sensor [26]. The classifier we built by following this tutorial, resulted in a 99% accuracy when validated on images from the Kaggle dataset. However, when we tested the classifier with real images that we captured of ourselves making the gestures, we found that the accuracy dropped to approximately 50%. We realized that this was due to the lack of diversity of the images in the dataset, as well as the fact that the images were captured with a motion sensor and looked dramatically different from those captured with a regular camera. We then decided to train the same classifier on real images of hand gestures we collected. At this point, we began the process of crowdsourcing images of four types: left hand gesture against a blank background, right hand gesture against a blank background, right hand gesture against a person's face and left hand gesture against a person's face. Initially, we collected around 600 images. We experimented with training the existing classifier on different subsets of our collected dataset. After experimentation, we were able to achieve a maximum of 80% accuracy by training on only images of the left and right hand gestures against blank backgrounds.

Then, we tried to tweak the hyperparameters of the same classifier to increase its validation accuracy. We experimented with changing the loss function, optimization model, number of epochs, learning rate and test-to-train ratio of the CNN. We also added a feature to reduce the learning rate of the CNN when it reached a plateau of accuracy for a fixed number of consecutive epochs. This raised the validation accuracy of the classifier to between 82% and 85%. However, even after increasing the count of images in the dataset to 800 images, the accuracy of this classifier seemed to plateau at 83%. Thus, we decided to revamp the architecture of the CNN itself.

After researching industry standard gesture classifiers, we discovered an architecture called VGG-16 created by K. Simonyan and A. Zisserman at the University of Oxford. This CNN was extremely large, contained 41 layers and learned 138 million parameters or weights. Due to the limited processing power of the RPi, we decided to use a subset of the layers of VGG-16. After experimenting with different subsets of layers, playing with the number of nodes in the fully connected layers, and reading more tutorials, we reduced the total number of layers to 13 and the number of learnable parameters or weights to 3.7 million. This resulted in a 88% validation accuracy when trained on our dataset which at this point contained 1000 images. Once again we hit a plateau at 88% and realized that we needed to collect more images to cross the 90% mark. With 1200 images in the dataset, we were able to achieve 90%.

We attempted to further increase the accuracy by implementing the skin detection algorithm [27] in the pre-processing phase, to separate the foreground hand gesture image from the background by clustering pixels between certain color values. This boosted the accuracy to 92% and after collecting a total of 1966 images we hit 95%. To normalize images before feeding them into the classifier, we attempted to also add Otsu binarization [28] which would convert the image into only black and white pixels. However, this did not help as predicted and we decided against using it.

During the integration phase, we realized we were missing an entire class that should have been predicted by our classifier. The CNN also needed to detect the difference between a person's head and a hand gesture. Collecting more images of people's heads would have been very time consuming.

Instead, we found an online dataset of 909 headshots of celebrities' faces [29] and added these images to our training dataset. With a new total of 2875 images, our CNN still averaged an accuracy of 93-95%.

The last important trade-off we made with regards to our gesture classifier was the value we chose for the frame-rate of the COMOVO's camera. Selecting a value too high gave us almost perfect predictions when we took the arithmetic mode of the predictions of multiple consecutive frames. However, as the CNN is quite large, it took some time to make inferences on images passed in when it was running on the RPi. Thus, we could not choose a frame rate too high, as that drastically increased the prediction latency. We settled on using 4 frames per second and decided to get the predictions of these 4 frames simultaneously on the 4 cores (one frame per core) of the RPi to save even more time.

Apart from those trade-offs related to the classifier, we had to make several others during the design process. First, we had to purchase microphones that were omnidirectional and larger than we had initially planned, as there was a lack of manufacturers shipping during the pandemic.

Second, we decided to use the Python module Pygame instead of Simulink [30] (a MATLAB-based programming environment) for our animation. Pygame was simpler to integrate with the rest of the code, which was already in Python and we had some experience using Python animation libraries in prior projects.

Third, the omnidirectional Zaffiro microphones innately had a certain level of directionality that became apparent when tested in our use case of a four-person, reasonable conversation. However, to further augment this directionality and maximize the margin of difference between the microphone detecting the loudest speaker and the other microphones, we constructed baffles, or plastic cones layered with bubble wrap, that encircled the microphones.

Fourth, we had initially intended to create a head detector which would recognize the presence of a person's head within a frame. This detector would have augmented the loudest speaker detector, by allowing us to more accurately zone in on the location of the person speaking. However, we realized that the number of people in our use case and the number of microphones we used for the loudest speaker detection were the same. We also recognized that a phone camera at a distance of approximately 3 feet (radius of a dinner table) can display more than one person's head in a singular frame. Based on the time we invested into building and tuning the gesture classifier to our required accuracy, and considering that the head detector would also have to be a machine learning model like a CNN, we decided that the cost of building this classifier outweighed the benefits and that it would not be a very useful addition to COMOVO.

Fifth, we had to make a trade-off with regards to the sampling rate of the microphones. We had to experiment to find a sampling rate that would give us a fast response and meet our accuracy requirements for loudest speaker detection. On the one hand, sampling at a higher rate used a lot of processing power and memory, and lengthened the response time. On the other hand, we believed that using a higher sampling rate would improve the accuracy of the loudest speaker detection. However, we learned that the loudest speaker detection accuracy fared better at a lower sampling rate as the microphones picked up less noise from other directions.

Sixth, we had previously considered more complicated algorithms such as array processing and triangulation [31] [32] to detect the loudest person speaking or sound source. However, we realized that these algorithms were far too complicated to be a subsection of our project given the time and budget constraints. Thus, we decided to use our current, simple approach of calculating and comparing the root mean squares (which estimates the energy or loudness) of the audio signals from each microphone, to detect the microphone near the loudest person speaking.

Seventh, we made the design choice to increase the audio-input-to-rotation latency as we required that a person remain speaking for a period of time before the COMOVO rotates towards them. We made this choice to avoid jitter caused by the COMOVO rotating towards a speaker who only utters a word, before rotating immediately back to the primary speaker.

The final tradeoff we had to make was due to limited resources during the pandemic. We were not able to acquire eight microphones (four for each of the RPis), and thus, decided to designate one RPi to exhibit automatic mode (referred to in the following sections as RPi A) and the other RPi to exhibit manual mode (referred to in the following sections as RPi M). This also significantly changed our control loop as we no longer had to switch between modes for each of the RPis. As a result, for manual mode we had RPi M receive the video stream from the camera, predict the direction of rotation using the classifier, communicate this direction to RPi A, which would then communicate this direction of rotation to the animation running on the PC. For automatic mode, we had the RPi A receive audio signals from the microphones, detect the microphone receiving the loudest signals, and communicate this microphone ID to the animation running on the PC.

V. System Description

Our system was developed following the two modes of operation of the COMOVO - automatic mode and manual mode. The overarching control loop that drives the animation

runs on the PC and both the RPis run a generic script that acts according to the current mode of operation as well as according to the specific RPi (A or M), as described briefly in the previous section. By making the script generic, we ensured that if we continued to work on the COMOVO in the future, we would easily be able to extend the control loop to allow both RPis to switch between automatic and manual mode.

In the initial setup step, the control loop script is started up on the PC, configuring it to act as the server. This initializes a socket which listens for incoming TCP connections from both RPis which act as clients to the PC. Then, RPi A and RPi M are both switched on, the microphones which are connected to RPi A are also switched on. The camera is connected to RPi M. The generic script is started on both of the RPis and on each RPi, establishes a connection from the RPi to the PC. The PC displays a splash screen and awaits user keystroke input to either quit ('q') or select a mode of operation ('a' for automatic or 'm' for manual). The user keystroke input is communicated to both RPis. The descriptions of what occurs on the PC and each RPi in each mode of operation are in the subsections below.

A. Automatic Mode

In automatic mode, RPi M does nothing and simply waits for a message from the PC in the event of the user quitting or requesting a mode switch. RPi A runs all the code for loudest speaker detection. First, 4 audio streams are opened from the 4 microphones connected, with the sampling rate specified. The 4 microphones have fixed positions such that each microphone ID (1, 2, 3 or 4) maps to a specific microphone (and thus, person) on the animation screen and the corresponding microphone (and thus, person) in real life.

The script running on RPi A then reads in the audio samples coming from the 4 microphones at the specific sampling rate (8 kHz) for a specific amount of time (2 seconds). This is done simultaneously using multiprocessing on the 4 cores to provide audio snapshots from all 4 microphones during the same 2 second time interval. The root mean squares of the audio samples are then also simultaneously calculated. The RMSes are compared to determine the ID of the microphone receiving the highest energy audio signals (largest RMS), which consequently indicates the loudest speaker during that 2 second interval. This process is repeated twice to get the loudest speaker for 4 seconds. Then, the microphone ID is communicated to the animation on the PC over the TCP connection established earlier. The above sequence of steps occurs in an infinite loop on RPi A, unless interrupted by a message received from the PC, in the event of the user quitting or requesting a mode switch.

On the PC, the control loop awaits messages from RPi A communicating the microphone ID that the animation must rotate to. When a microphone ID is received, the animated COMOVO rotates to face the person at the position mapping to this ID, an arrow moves to point to that person and the screen is updated. The animation continuously listens for any user keypress event to quit or switch modes. If an event occurs, the PC communicates this information to both RPis over the TCP connections established previously. If the event is to quit, the connections between the PC and the RPis are closed and RPi A closes the connection between itself and RPi M (if this connection exists).

B. Manual Mode

In manual mode, RPi A listens on a socket for an incoming TCP connection from RPi M. Once the connection is established, RPi A listens for prediction messages from RPi M and also continues to listen for messages from the PC in the event of the user quitting or requesting a mode switch.

RPi M runs all the code to capture and pass frames through the gesture detector. First, the software interface to the camera is set up using PiCamera module, and the camera resolution and frame rate are initialized to specific values. The trained CNN gesture classifier and its weights are loaded from files they were saved in and the CNN is compiled.

The script captures frames continuously at 4 frames per second and uses multiprocessing to simultaneously make predictions for one frame per core. Predictions are made by forward propagating the frames through the compiled CNN. In order to make nearly perfect predictions, the arithmetic mode of the 4 individual frame predictions is calculated as the final prediction for that iteration. In other words, if there are 3 or more of the same gesture predicted, the final prediction is that gesture (left or right). In all other cases, the final prediction is no gesture (head). The final prediction is communicated to RPi A which in turn forwards the prediction to the animation on the PC. The above sequence of steps occurs in an infinite loop on RPi M, unless interrupted by a message received from the PC, in the event of the user quitting or requesting a mode switch.

On the PC, the control loop awaits messages from RPi A communicating the gesture prediction. If the prediction received is 'left', the animated COMOVO rotates 10° in the counterclockwise direction, if the prediction received is 'right', it rotates 10° in the clockwise direction, and if the prediction received is 'head', no rotation occurs. The screen is then updated. As in automatic mode, the animation continuously listens for any user keypress event to quit or switch modes. If an event occurs, the PC communicates this information to both RPis over the TCP connections established previously. If the event is to quit, the connections between the PC and the RPis are closed and RPi A closes the

connection between itself and RPi M.

VI. Results

We were able to surpass our benchmarks for success for both automatic and manual mode.

In automatic mode, the measured latency from the time a person began speaking to the time the COMOVO began rotating was approximately 2 seconds. This latency outperformed the benchmark of 2.3 seconds that we had set. This benchmark came from averaging measurements of time taken to physically rotate a phone placed on a dinner table. The 2 second latency includes the delay we added as a design choice and without this delay, the sound processing pipeline returned results almost instantaneously.

We also tested the accuracy of our loudest speaker detector over multiple runs with at least 3 people and with variation in terms of sampling frequencies, room size, echo, voice pitch, and placement of microphones. The accuracy was consistently over 95% for every test and surpassed our benchmark. Additionally, we noticed that at lower sampling frequencies the accuracy remained the same, but the margin of difference between the microphone signals from the loudest person speaking, and the signals from other microphones increased. This was attributed to the fact that at lower sampling frequencies, the microphones picked up less noise and there was less information to process.

In manual mode, the measured latency from the time a person began making a hand gesture to the time the device began rotating, was 7 seconds at the time of our demo. This was several seconds past our benchmark, and in part was due to the processing occurring on the RPi. However, after our demo, we noticed that we were opening the file containing our saved CNN, reading the file, closing the file, and loading and compiling the CNN every time we made a prediction on a frame. After changing this to only load the saved classifier once on start-up, our latency went down to 1-2 seconds and outperformed our benchmark of 2.3 seconds.

Each gesture registered rotates the device 10°, allowing the person making the gesture to have adequate control over the rotation while taking latency into consideration. Thus, it takes roughly 5 seconds to rotate from one person another, assuming they are seated approximately 90° apart. Given more time, we would experiment with using a neural stick [33] (which is a Visual Processing Unit with much higher processing power), connected to the RPi to process images and run the classifier to further improve latency. We would also use the low-level binary version of TensorFlow instead of the Python version as the high level abstraction increases latency.

Finally, over many runs and epochs, our hand gesture classifier consistently produced a validation accuracy of 93-95% and outperformed our 85% validation accuracy benchmark.

VII. Project Management

A. Schedule

Our original schedule in the proposal was changed to account for delays in ordering and receiving parts, changes to our proposed sound localization algorithm and the unprecedented time it took to set up the RPis. Our schedule then drastically changed again in the second week of March due to the pandemic and the resulting changes we had to make to our project plan. Our schedule over the whole semester reflecting these changes is shown on the last page.

B. Team Member Responsibilities

Neeti used her experience with machine learning to build the initial gesture classifier as well as create the pipeline to train and test the classifier. She also used her experience with Python animation to create the final simulation using Pygame.

Gauri used her experience with machine learning to improve the accuracy of the initial gesture classifier and restructure its architecture. She worked with Shrutika on the camera-RPi and microphone-RPi interfaces. Initially, she also worked on setting up the RPis.

Shrutika worked on setting up and testing all the hardware components. She handled PC-RPi communication and also worked with Gauri on the camera-RPi and microphone-RPi interfaces.

All team members worked together to collect and construct the training dataset for the gesture classifier. We also worked together to integrate the animation with the camera-RPi and microphone-RPi interfaces and create the overall control loop.

C. Budget

The project Bill of Materials is shown in Table 2 and Table 3 shows the parts we acquired from the inventory for free. The rows highlighted in green are the items we used to achieve the modified goals.

D. Risk Management

The first risk factor we faced was related to the gesture detection input. The risk here was that the accuracy rate of identifying the gestures depended on the size and quality of our dataset. To mitigate this risk, we initially narrowed the scope of classification to recognize a gesture with a blank background. We also created our own dataset. We were able to collect more images as needed to completely mitigate this risk and no longer require a blank background.

The second risk factor we considered was the accuracy of the loudest speaker detector. We required a very high accuracy rate, and we identified several risks regarding ambient sound, sound from the phone speaker and the

accuracy of the microphones. We were able to mitigate this risk by creating baffles for the omnidirectional microphones that we used. We also narrowed the scope to detecting the loudest person speaking among 4 people having a reasonable conversation.

The third risk factor was a higher latency than we expected due to the gesture classifier being a CNN as well as the low processing power of the RPi. We mitigated this risk by saving the trained model in a file and loading and compiling the model from this file only once on start-up. To further speed up the classification and produce real-time predictions, we would connect a neural stick to the RPi to harness its processing power. We would also use low-level binaries of TensorFlow instead of the version created for Python.

Finally, the performance of our device was heavily dependent on parameter tuning. We mitigated this risk by experimenting heavily with different parameters and architectures for the classifier, different sampling frequencies for the microphones and different frame-rates for the camera.

TABLE 2: Bill Of Materials

Item	Count	Supplier	Cost of Item	Total Cost	Purchased
RPi 4 Model B	2	Amazon	$45.78	$91.50	Y
Raspberry Pi Camera Module V2-8 Megapixel 1080p	2	Amazon	$28.20	$56.40	Y
ZAFFIRO USB Microphone	4	Amazon	$17.99	$71.96	Y
Adafruit DC Stepper and Motor HAT for Raspberry Pi	2	Amazon	$29.95	$59.90	Y
Adafruit Accessories 8xAA Battery Holder	2	Amazon	$9.53	$19.06	Y

TABLE 3: Other Parts (acquired for free)

Item	Count	Contributor
SD Card	2	ECE Inventory (Came from RPi Zeroes)
Ethernet Cable	2	CMU IT
Baffles (Amazon Prime wrapping + Construction Paper)	4	-
Box to hold RPi	1	-

VIII. Related Work

We found that the closest product to COMOVO is Facebook Portal [34]. 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 [35] and Zoom [36] have such loudest speaker detection capabilities but are used primarily in office conferencing settings and do not involve a portable platform that can physically rotate your phone. Additionally, 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. Cisco Webex and Zoom business licenses are often too expensive for individuals to purchase.

IX. Summary

Over the course of the semester, our goals for this project evolved multiple times due to unforeseen circumstances. Our initial goal was to create a physical rotating platform to make multi-person video calling a better experience. However, due to the pandemic (a tired phrase) we had to drastically modify our goals to create a simulation of the physical platform instead. Although the circumstances were frustrating, we delivered a product that met our modified goals and surpassed all of our benchmarks for success. Section VI details our results and the performance of the COMOVO measured against our benchmarks.

We would like to thank Professor Tom Sullivan and Jens Ertman for their guidance and support throughout this project, all of the other ECE Capstone faculty for sharing their knowledge, our ECE professors and peers for their valuable feedback, our friends, family and everyone else who provided us with images to help build our dataset. Finally, we would also like to thank Carnegie Mellon University for teaching us the building blocks that were fundamental in creating this project and giving us this opportunity to use and build on what we have learned over the last four years.

References

[1] Pisharady, Pramod & Saerbeck, Martin. (2015). Gesture Recognition Performance Score: A New Metric to Evaluate Gesture Recognition Systems. 157-173. 10.1007/978-3-319-16628-5_12.

[2] Badi, H. Recent methods in vision-based hand gesture recognition. *Int J Data Sci Anal* 1, 77–87 (2016). https://doi.org/10.1007/s41060-016-0008-z

[3] How To Calculate The Best Dining Table Size For Your Room. (2019, May 2). Retrieved from http://www.parotas.com/en/calculate-best-dining-table-size/

[4] Grigorik, I. (n.d.). High Performance Browser Networking. Retrieved from https://www.oreilly.com/library/view/high-performance-browser/9781449344757/ch01.html

[5] Raspberry Pi 4 Model B specifications – Raspberry Pi. (n.d.). Retrieved from https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

[6] Buy a Raspberry Pi Zero W – Raspberry Pi. (n.d.). Retrieved from https://www.raspberrypi.org/products/raspberry-pi-zero-/

[7] Camera Module. (n.d.). Retrieved from https://www.raspberrypi.org/documentation/hardware/camera/README.md

[8] Mini USB Microphone. (n.d.). Retrieved from https://www.adafruit.com/product/3367

[9] USB Microphone, ZAFFIRO. (n.d.). Retrieved from http://www.zaffiro.cc/P_view.asp?pid=227

[10] Stepper motor - NEMA-17 size - 200 steps/rev, 12V 350mA. (n.d.). Retrieved from https://www.adafruit.com/product/324

[11] Adafruit DC & Stepper Motor HAT for Raspberry Pi - Mini Kit. (n.d.). Retrieved from https://www.adafruit.com/product/2348

[12] Industries. (n.d.). 8 x AA battery holder. Retrieved from https://www.adafruit.com/

[13] Raspbian. https://www.raspberrypi.org/documentation/raspbian/

[14] picamera. (n.d.). Retrieved from https://picamera.readthedocs.io/en/release-1.13/

[15] OpenCV. (2020, April 9). Retrieved from https://opencv.org/

[16] NumPy. (n.d.). Retrieved from https://numpy.org/

[17] imutils. (n.d.). Retrieved from https://pypi.org/project/imutils/

[18] Keras: The Python Deep Learning library. (n.d.). Retrieved from https://keras.io/

[19] TensorFlow. (n.d.). Retrieved from https://www.tensorflow.org/

[20] scikit-learn. (n.d.). Retrieved from https://pypi.org/project/scikit-learn/

[21] Simonyan, K., & Zisserman, A. (2015, April 10). VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. Retrieved from https://arxiv.org/pdf/1409.1556.pdf

[22] Pygame Front Page. (n.d.). Retrieved from https://www.pygame.org/docs/

[23] PyAudio Documentation. (n.d.). Retrieved from https://people.csail.mit.edu/hubert/pyaudio/docs/#pyaudio-documentation

[24] Adafruit-Motor-HAT-Python-Library. (2017, February 28). Retrieved from https://github.com/adafruit/Adafruit-Motor-HAT-Python-Library

[25] https://towardsdatascience.com/tutorial-using-deep-learning-and-cnns-to-make-a-hand-gesture-recognition-model-371770b63a51

[26] Hand Gesture Recognition Database. (2018, July 30). Retrieved from https://www.kaggle.com/gti-upm/leapgestrecog/version/1

[27] Baig, W. U., Rosebrock, A., SaadEddin, Tsipenyuk, B., Landau, J., Nowicki, P., … Orion Gump. (2020, April 18). Tutorial: Skin Detection Example using Python and OpenCV. Retrieved from https://www.pyimagesearch.com/2014/08/18/skin-detection-step-step-example-using-python-opencv/

[28] Image Thresholding. (n.d.). Retrieved from https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html#otsus-binarization

[29] Liu, Z., Luo, P., Wang, X., & Tang, X. (2016, July 29). Large-scale CelebFaces Attributes (CelebA) Dataset. Retrieved from http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[30] Simulink - Simulation and Model-Based Design. (n.d.). Retrieved from https://www.mathworks.com/products/simulink.html

[31] S. Ganguly, J. Kerketta, P. K. Kumar and M. Mukhopadhyay, "A study on DOA estimation algorithms for array processing applications," *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*, Gurgaon, 2017, pp. 62-65, doi: 10.1109/IC3TSN.2017.8284451

[32] Lam, A. (2017). *3D sound-source localization using triangulation-based methods* (T). University of British Columbia. Retrieved from https://open.library.ubc.ca/collections/ubctheses/24/items/1.0357459

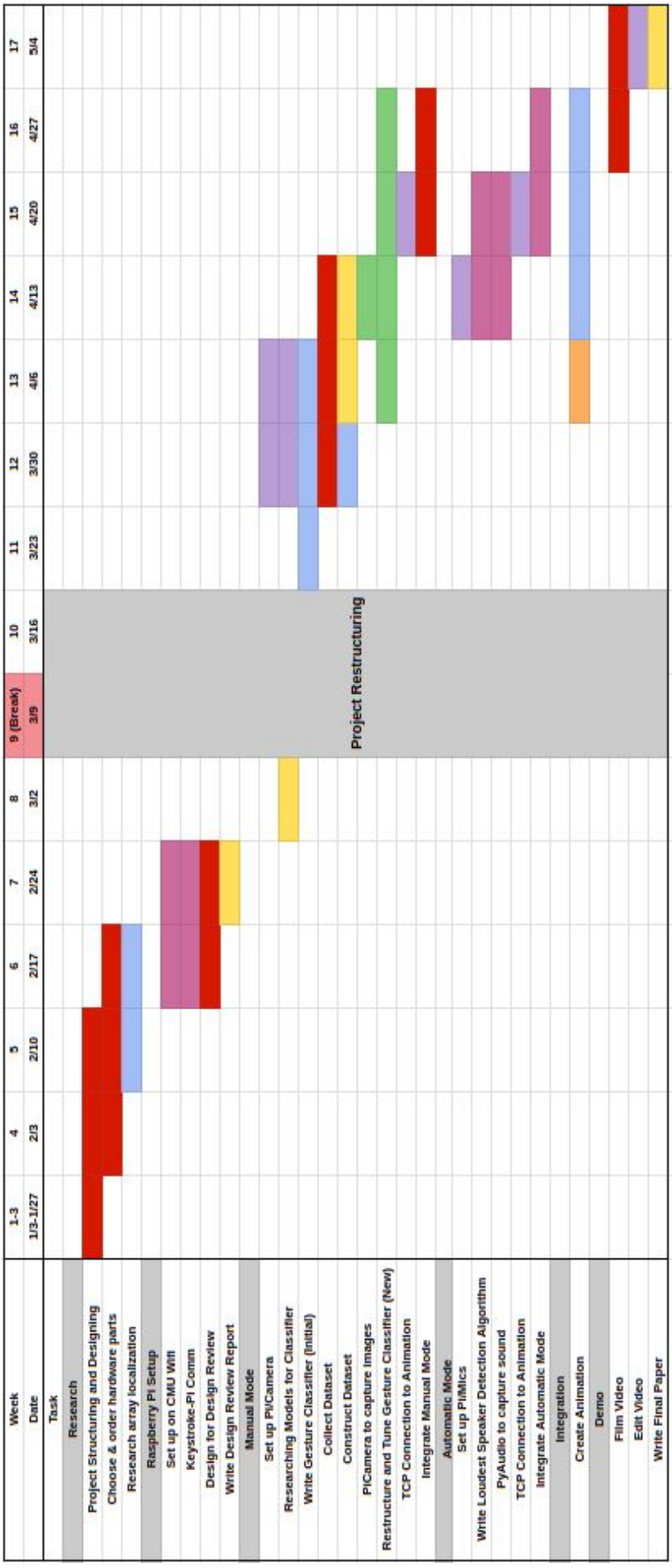
[33] Intel® Neural Compute Stick 2. (2019, October 11). Retrieved from https://software.intel.com/en-us/neural-compute-stick

[34] Portal from Facebook. (n.d.). Retrieved from https://portal.facebook.com/

[35] Video Conferencing, Online Meetings, Screen Share: Cisco Webex. (2020, May 6). Retrieved from https://www.webex.com/

[36] Video Conferencing, Web Conferencing, Webinars, Screen Sharing. (n.d.). Retrieved from https://zoom.us/

[37] Girma, A. (2019, March 1). Keras - Convolutional Neural Network (CNN) Implementation for Hand Gesture Recognition. Retrieved from https://medium.com/@aggirma/keras-convolutional-neural-network-cnn-implementation-for-hand-gesture-recognition-d7dd11958af6


Week
Date
Task
1-3
1/3-1/27
4
2/3
5
2/10
6
2/17
7
2/24
8
3/2
9 (Break)
3/9
10
3/16
11
3/23
12
3/30
13
4/6
14
4/13
15
4/20
16
4/27
17
5/4
Research
Project Structuring and Designing
Choose & order hardware parts
Research array localization
Raspberry Pi Setup
Set up on CMU Wifi
Keystroke-Pi Comm
Design for Design Review
Write Design Review Report
Manual Mode
Set up PiCamera
Researching Models for Classifier
Write Gesture Classifier (Initial)
Collect Dataset
Construct Dataset
PiCamera to capture images
Restructure and Tune Gesture Classifier (New)
TCP Connection to Animation
Integrate Manual Mode
Automatic Mode
Set up PiMics
Write Loudest Speaker Detection Algorithm
PyAudio to capture sound
TCP Connection to Animation
Integrate Automatic Mode
Integration
Create Animation
Demo
Film Video
Edit Video
Write Final Paper
Project Restructuring


Neeti
Shrutika
Gauri
Neeti and Gauri
Neeti and Shrutika
Gauri and Shrutika
All