iContact

Authors: Heather Baker, Anna Li, Edward Lucero: Electrical and Computer Engineering
Carnegie Mellon University

***Abstract*—**iContact is a mobile camera system that points directly at whomever is currently speaking, whether it is a single user moving around in a room or a conference room full of people. Using both audio detection and facial recognition via computer vision, iContact can identify the speaker and maneuver the camera to face that person. Its motors can rotate the cameras about the yaw axis, and vertically raise and lower the cameras. With iContact, video calls will be made to feel more personal and immersive.

***Index Terms*—Acoustic location, Haar Cascades, I2C, Jetson Nano, MIPI CSI-2, OpenCV, PWM, ReSpeaker**

I. Introduction

With the onset of COVID-19, video calls have become an absolutely indispensable part of everyone's daily lives, whether it is for attending lectures via Zoom, calling friends and family, or even remote internships, most people cannot go a day without a video call anymore. Even before COVID happened, people have needed video calls for keeping in touch with distant friends and family, and many companies have relied on conference calls for linking their various branches and workers around the world.

The world has seen how video calls have become increasingly crucial over recent years, but video call mechanics have not really evolved much – conference calls are all still primarily done through a laptop camera or a desktop webcam. The question we asked was: How can this project better immerse the remote viewer into a video call? The answer is iContact, an agile camera that keeps the focus on the speaker in any conversation by physically adjusting to center on the speaker's face.

There are four areas of functionality that the design requirements categorize into: conference viewing, working range, algorithm accuracy, and speed. For the viewing requirement, iContact should be compatible with any conferencing software and able to operate at 1080 pixels at 30 frames per second. For the working range, iContact will aim to have a 360-degree field of view, one foot of vertical panning, and ten feet of microphone audio pickup and person detection radius. The requirements for algorithm accuracy will be set for 90% with respect to centering, speaker identification, and cerebral command comprehension, as well as 95% motor positioning accuracy. Lastly, within the speed category, iContact should complete motor positioning adjustments, audio processing, and video processing within one second. The speeds for audio and video processing are important to have minimal lag between the conferencing video feed and iContact.

II. Design Requirements

There are different tests for the various areas of functionality. To meet the high compatibility requirement, iContact will be tested on Zoom, WebEx, and Google Hangouts. The frame rate requirement can be determined by counting the number of frames that get sent between iContact and the host computer within a certain amount of time. Regarding the working range and algorithm accuracy requirements, there are two tests to be performed: a stationary speaker test and moving speaker test. The stationary speaker test will be conducted at varying distances and heights from iContact to test the acoustic location range, the vertical panning range, and the centering accuracy of an out-of-frame speaker. The distances will be between three to fifteen feet, in and out of frame. The heights will be set such that the speaker's head is above and below the camera frame as well as above and below the center of the frame. The moving speaker test will also be conducted at varying distances and varying heights, determined by the working range of iContact found from the stationary test. The moving speaker test will be used to verify centering accuracy, field of view, and vertical panning range. In addition to the previous two tests, there is an additional multiple speaker test, which will gauge how accurately iContact is able to identify speakers. All previous tests mentioned will also record the time of each processing component to determine how well iContact meets the speed requirements. One additional test to specifically test speed is to have multiple speakers conversing back and forth for varying speaking time durations. These requirements and tests are listed in Table I.

Functionality	***Requirements***	***Testing***
Viewing	High compatibility, 1080p @30fps	Run with Zoom, Webex, and Google Hangouts
Working range	360-degree field of view, 1ft vertical panning range, 10ft acoustic location range, 10ft person detection radius	Stationary or moving speaker around the room at various distances and angles from iContact, speaking (50-65dB)
Algorithm accuracy	90% centering accuracy, 90% speaker identification accuracy	Stationary speakers converse back and forth, Subject moving while continuing to talk
Speed	<1s motor control for camera adjustment, <1s audio input processing latency, <1s video input processing latency	Stationary speakers conversing back and forth, taking turns speaking one sentence at a time

TABLE I. Design Requirements

III. Architecture and/or Principle of Operation

There are two degrees of freedom for the physical design: vertical movement and yaw axis rotation. A stepper motor is used to control the elevation of the cameras. Each camera is attached to a micro servo, which controls yaw rotation (see Figure 1).

The main computing is done through the Jetson Nano (see Figure 3) and is powered by a 5-volt, 4-amp power supply. Connected to the Nano are two cameras, a microphone array, a servo shield, and a motor HAT. The motor HAT connects a stepper motor, powered by a separate 12-volt, 5-amp power supply, to the Jetson. The two servos are connected to the servo shield and powered through the Jetson.

For the overall software design of the project there are three separate components that interact with the main program. These three components are software programs that control the audio processing, the motor controller, and the computer vision software. The main program is responsible for the communications between other components by controlling when they start up and making use of the return values from each program.

The system first starts out in an idle state where the camera positioning is the most recently set angle from either initial startup or from the most recent speaker detected (see Figure 2). The first step is to retrieve the most recently detected speaker from the audio processing component. The Jetson utilizes the microphone array's firmware to determine the direction of arrival of any noises detected, although the Jetson only waits for a human voice to be detected. After filtering out noisy values and outliers, it then updates the speaker angle to which the camera must rotate (about the yaw axis) to point in the general direction of the speaker. It sends this angle to the main program upon request.

Once the main program receives an angle from the audio component, it can then send instructions to the motor controller. The motor controller is responsible for turning the specified motors to the desired angle. By using the Adafruit Motor Library, motor movement can be done by specifying the angle to turn to for the micro servos and the angle amount to turn by for the stepper motors. For the stepper motor, a counter tracks the current angle relative to the initial angle, to calculate the degrees the stepper motor needs to turn. Once the motors have set the angle to the correct angle, it responds back to the main program that it is finished. The angle calculated by audio detection points the camera in an approximate direction of the speaker. Computer vision face detection is then utilized to perfectly center the speaker in the camera's view, through precise adjustments. Thus, the next step is for the main program to tell the computer vision component to start looking for a speaker, as well as which of the cameras' views the speaker is in.

After the computer vision portion has started up, it looks in the specified video feed for the speaker. If a speaker is detected in the updated frame, the final recentering for the speaker can then be computed by telling the main program the final set of motor instructions that needs to be set to complete the centering. This is a single complete loop of the software state machine.

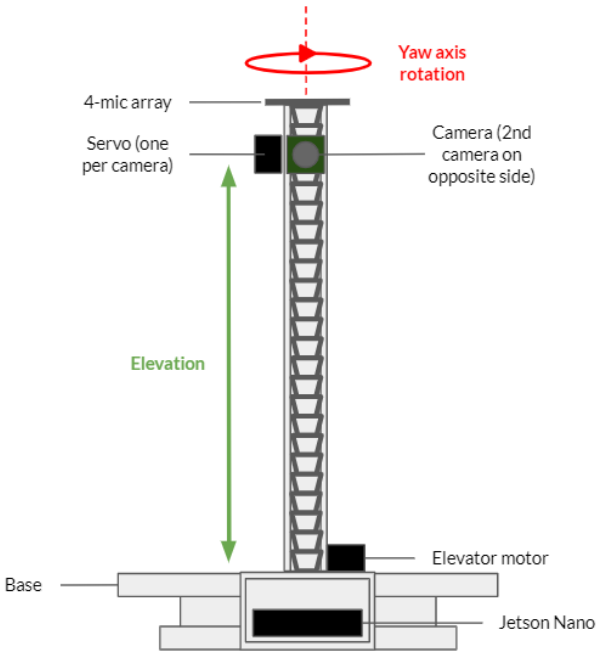


Fig. 1. iContact mechanical design

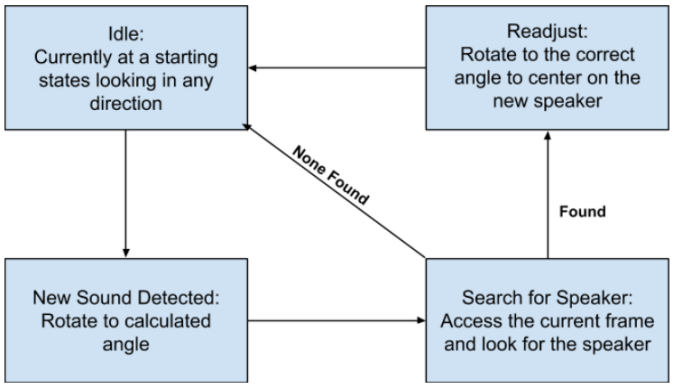


Fig. 2. Software state machine

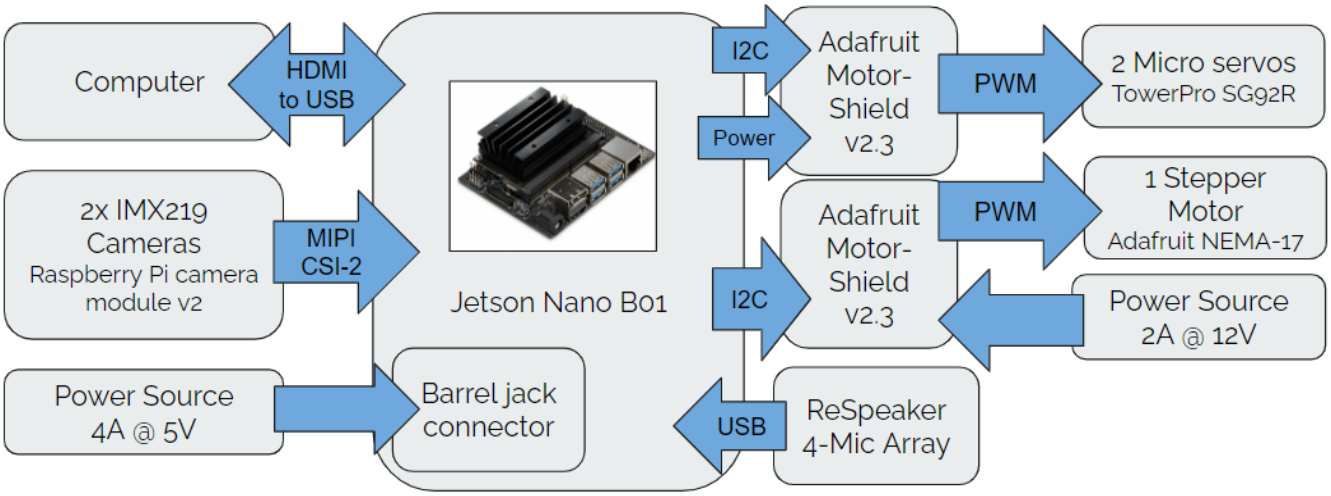


Fig. 3. Hardware system specification

IV. Design Trade Studies

The following are justifications for the component selections and any changes that were made from the project proposal.

A. *Jetson Nano vs. Raspberry Pi*

We opted to use the Jetson Nano for iContact. As shown in Table 1, the Jetson Nano provides more power for video decoding and better support for peripherals than the Raspberry Pi does. A big advantage of the Jetson Nano is the increase in MIPI CSI-2 lanes as this increases the number of cameras we could use, which reduces the latency for speaker detection (see Section G). The Jetson Nano also has I2S for audio, which is a better choice since it is digital, resulting in less noise compared to analog and thus better audio quality. While this was our original plan, it was later changed as detailed in Section C.

	Specs and Cost	
	Jetson Nano B01	***RaspberryPi Model 4***
Price	$99	$55
RAM	4 GB	4 GB
Video	2 MIPI CSI-2 DPHY lanes	1 MIPI CSI-2 DPHY lanes
USB	4 USB 3.0	2 USB 3.0, 2 USB 2.0
GPIO	40 pin	40 pin
Video Decoder	H.264 up to 1080p240	H.264 up to 1080p60
Audio	2 I2S	No I2S
CPU	ARM A57	ARM A72
Motor	4 I2C, 1 PWM	6 I2C, 2 PWM

TABLE II. Jetson Nano Vs. Raspberry Pi Spec Comparison

B. *Audio Hardware Comparisons*

Our initial plan for the audio was to use four microphones, two connected in stereo to each of the Jetson's I2S pins. However, we soon realized that we would need each individual microphone's audio feed to determine the difference in arrival time of the speaker's sound at each of the microphones. By connecting two microphones on one I2S input, we would not be able to process their audio feeds separately. We briefly decided to switch from using four microphones to just two – one on each I2S pin. We moved away from this idea when we realized we would never know from which side of the microphone array the source would be. As illustrated in Figure 4, the difference in arrival time would be the same for the two speakers, since they are equidistant from each microphone; the two-microphone array would not be able to distinguish between them.

From here, we abandoned the I2S microphones and made use of the three remaining I2C pins on the Jetson (the Jetson has four in total, and one is used for the motors). This meant we had to switch to analog microphones and an I2C ADC (analog to digital converter), as opposed to our I2S microphones with digital output. In the search for new analog microphones, we had to make some more tradeoff analyses. We narrowed our options down to Adafruit's MAX4466, which has adjustable gain, and MAX9814, which has automatic gain control. The MAX4466 was cheaper, but customer reviews revealed the production quality varied widely, and it picked up a great deal of noise. We ended up choosing the 9814, which was pricier but also seemed much more reputable and higher quality.

However, the ADS1115 ADC, that we obtained to work with these new MAX9814 microphones was not sampling quickly enough. Even after optimizing our code and setting the Jetson's I2C bus speed to the maximum frequency, we could only achieve about 300 samples per second, which, split among the three microphones, meant 100 SPS per microphone. This was far from the precision we needed, since our goal was to be able to detect the difference in arrival time between the microphones. Plus, we learned that we would need to sample at least 8000 Hz, least twice the maximum frequency of the typical voice range to avoid Nyquist aliasing.

After realizing the MAX9814 microphones were a lost cause, we moved to the next idea: mini-USB microphones. These had a sampling rate of 44100 Hz, which was a massive improvement and well above the Nyquist frequency. Using the Jetson's three USB ports for three microphones, each 1.5 ft from each other, we were able to get a rudimentary acoustic location algorithm working -- it simply detected sudden noises or rises in amplitude among the microphones (i.e. interpreted as someone beginning to speak), compared the arrival times of the noise to each microphone, and returned which microphone had the earliest arrival time. With this implementation seeming to work thus far, we attempted to integrate more microphones using a USB splitter. However, this was soon abandoned after encountering issues with the PyAudio module we were using in our code -- it could not consistently assign the same device indices to each microphone, even if, physically, the setup and connections were precisely the same. Unfortunately, things continued going downhill from here with regards to consistency. From the beginning stages of implementing the USB microphone solution, we found that each microphone had some delay in its audio detection; we easily managed this by determining that delay and hardcoding it into the acoustic location program. After more and more trials, however, we discovered that this delay fluctuated unpredictably depending on the number of processes running on the Jetson. Even worse, the microphones were sometimes "hard of hearing" and could not dependably detect voices at normal speaking levels.

Yet again, we changed to a new implementation. This time, figuring that the sporadic delay with the USB microphones was in part due to the built-in audio interface in each microphone and the USB protocol, we opted once again for analog microphones and an ADC. The analog microphones that we used in the I2C implementation worked fine, so we only sought out a new ADC. We managed to find an ADC with a 200 kHz sampling rate, the MCP3008, which communicated over SPI. This was far beyond the frequency we needed, but we anticipated that the sampling rate listed in the product's description would inevitably drop in nonideal conditions. Upon testing out our new hardware, we were shocked to discover that the sampling rate fell to a mere 600 SPS, i.e. 200 SPS for each of the three microphones, again much too low for our purposes.

At this point, after much trial and error and research, we realized that we simply would not be able to achieve the sampling rate and real-time precision required for accurate acoustic location without a dedicated microcontroller running a real-time OS, as opposed to our general-purpose Jetson Nano running Linux, a time-sharing OS. Luckily, we at last managed to find an easily integratable solution: the ReSpeaker v2.0 4-microphone array. Instead of having to connect four discrete microphones to the Jetson, the ReSpeaker had all four microphones already integrated and synchronized on this singular chipset; all we had to do was plug it into one of the Jetson's USB ports. The ReSpeaker was designed for functionality like acoustic location -- in fact, acoustic location was one of its built-in algorithms. This solution, not being our own implementation from scratch, was not ideal, but it perfectly suited our purposes and allowed us to advance our project.

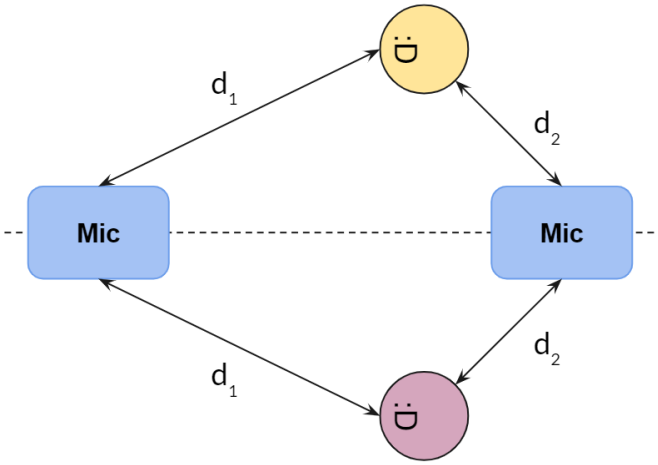


Fig. 4. Two microphone setup

C. *Audio Detection Software Optimizations*

The ReSpeaker's built-in algorithms for voice detection and direction of arrival came in very handy and were easy to use, but there were still optimizations to be made. In our original code, we updated the angle of the speaker every time a voice was detected. This resulted in the speaker angle being updated too often, typically only by a few degrees, which made the motors "jitter" as they continuously made these insignificant adjustments. There was also the occasional outlier, sometimes in the complete opposite direction of the speaker. To manage these issues, we changed the algorithm such that, upon detecting a voice, it would record the next 20 speaker angles (with 50 ms between readings) as returned from the built-in direction of arrival functionality. From this sequence of angles, it would pull the median and use this as the new speaker angle, which helped eliminate outliers. Next, it would compare this new angle to the current angle; if they differed by at least 10 degrees, the current angle would be updated to the new angle. This prevented the speaker angle from being updated too frequently. We also found that the noise from the motors moving was interfering with the audio processing, so we adjusted the algorithm to pause listening until the motors finished moving.

After making the optimizations, our speaker identification accuracy improved considerably. We tested and measured this metric by holding conversations between two to three people located at various positions 4ft from the iContact. If the iContact could detect someone beginning to speak and turn to an angle such that that person was somewhere in the camera's view (not necessarily centered), we would count this as a successful speaker identification. We ran tests in which there were 30 times that a new person would begin speaking. The iContact was able to achieve 100% speaker identification accuracy between two speakers (on opposite sides of the iContact) and 90% among three speakers (equidistant from each other and therefore closer together than the two speakers were), thus fulfilling our design requirement for this metric.

For one lone person moving around the room and speaking

at different distances from the iContact (beginning to speak 30 separate times with at least five seconds between sentences), the acoustic location was able to angle the camera such that the speaker was in frame 100% of the time up to 4ft, and 90% up to 6ft. This fell short of our 10ft acoustic location range design requirement. We believe this is, in part, due to the lower volume of the speaker's voice in conjunction with the increased amount of sound waves bouncing off of surfaces as the speaker moves farther away, which makes it more difficult for the iContact to assess voice detection and time delays.

Throughout testing, we also kept track of the audio input processing latency, which we discovered was well below the upper bound we had set in our design requirements, averaging at 0.6451 seconds with negligible variance.

D. Panning Range Comparison

Early on in our project design, the vertical panning range was changed from the original plan of three feet to one foot. There were multiple justifications for this design change. The taller the vertical panning range, the heavier the elevator would be. This added weight presented two issues: the base would need to be heavier to keep the center of balance low, and the base motor would have to be more powerful to provide a higher torque to the elevator. In addition, the heavier and larger the entire project was, the less practical the iContact became for the user. Another metric driven decision was the latency time given by needing to move the entire three feet. Using the average data from Table #, we can determine that the latency for stepper movement is 0.313 on average and thus for three feet the movement would likely be 0.939 which heavily cuts into our goal of moving in under one second. In the end, we were still able to capture a reasonable vertical range for our use cases, the elevator was able to detect a range of 20 inches when users were one foot away from the camera (see Table III).

Distance from Camera (feet)	*Range of Camera View (inches)*
1	20
3	43
6	87

TABLE III. RANGE BETWEEN BOTTOM AND TOP OF ELEVATOR AND CAMERA FRAME

E. Motor Selection

For servo selection, we needed a small servo to fit on the elevator and minimize the weight of the device. In addition, we wanted the servo to move fast to reduce the motor movement latency. At the time, we wanted the servos to have a range of 180 degrees because the initial design was to move them in the pitch axis, so there was no need for movement past 180 degrees as it would hit the elevator shaft. We selected the TowerPro SG92R micro servos because on paper it met our needs. In practice, the servo only moves between 10 and 170 degrees; any farther and the servo begins to emit noise, which is undesirable as it can affect the speaker detection. Later when we repurposed the servo motors to rotate in the yaw axis, the degree limitation prevents full 360-degree motor movement; however, the cameras have over a 60-degree field of view, so all 360 degrees are accessible by camera, and with some adjustments of the video feed, the face is centered. For the stepper motor selection, we chose to have stepper motors for their angular precision. We selected the Adafruit Nema-17 stepper motor as it provided a torque of 28 oz-in, as we anticipated our device to be under one pound, we believed this would be enough.

F. Motor Design

Originally, the goal for the project was to have three degrees of freedom; however, the final design choice was to only have two degrees: yaw and vertical. The design change was made by repurposing the servos from rotating around the pitch axis to the yaw axis, and the need for the change arose from the base stepper motor not having enough torque to be able to rotate the entire system around the yaw axis. There were some benefits that came from using servos instead of steppers. Firstly, the movement for the servo is much quicker than the movement for the stepper because the servo can take one fluid motion to go from one location to the other, while the stepper motor requires to move step by step. This fluid motion also reduces the amount of jitter as seen in the camera frame. Secondly, the micro servos are two separate systems. As a result, if there are two speakers such that one speaker is in the range of one camera and the other speaker is in range of the other camera, the servos will keep the cameras pointed at the speakers, while the software swaps between the camera inputs.

Test description	*Motor Time Average (seconds)*	*Motor Time Variance*
Only servo movement	0.115	0.092
Stepper and servo movement	0.449	0.136

TABLE IV. MOTOR TIMES WHEN RUNNING IN THE FINAL PRODUCT

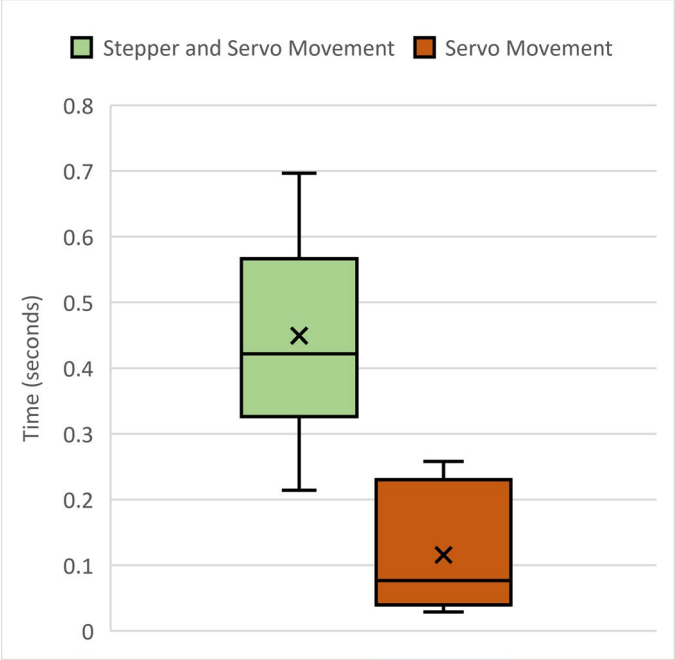


Fig. 5. Motor movement speed plot

G. Video Transmission Selection

Originally, we planned to send video from one camera to the Jetson Nano for processing and then to the user's computer. The design was to edit the Linux kernel of the Jetson Nano before flashing it. Linux has Linux Gadget Drivers, such that when a USB probe from the user's computer arrives, the Linux computer, AKA the Jetson Nano, would present itself as a USB Video Class gadget. For the scope of this class, this seems like it could be a risk point because of the time it would take to try and get this component working, and how it would cut into the already tight time frame we had scheduled for everything else.

One of the alternatives that we tried was to make use of Gstreamer to send the video feed from the Jetson to a port on our local machines through local internet. This led to a couple of problems. Only Windows 10 can detect the video stream as a camera source without additional setup, and the video stream needed to be ONVIF (a standardized interface for internet protocol-based video streams).[5] We had to reduce the framerate down to a maximum of 10 frames per second and the pixel width down to 64 pixels and even then, there was blurring from missing pixels being sent. There were too many sacrifices being made for this design and we needed to pivot to something else.

In the end, our design choice was a video capture card which takes in an HDMI input and converts it into a USB video output. This solution worked well and was smooth to integrate into our system as the Jetson could send video over HDMI and no install was needed to have the video capture card be detected as an input camera. Table II and III detail the metrics for this design choice. In Table II, maximum FPS (frames per second) is the maximum FPS possible as specified by our code. In Table 3, it was measured by determining the total number of frames sent for the whole test run, excluding the time for initializations on startup. Latency was measured with a stopwatch by comparing the real-time value to the value seen on the video feed.

	Video Capture Card	***IP-Camera***
Maximum FPS	21	10
Pixel width	1080	64
Latency (seconds)	Less than 2	Greater than 2
OS compatibility	Windows, Linux, MacOS	Windows 10, Linux
Setup required	No	Yes

TABLE V. VIDEO CAPTURE CARD VS. IP-CAMERA

Test description	***Latency Average (seconds)***	***Latency Variance***
No face in frame	1.04	0.09
Face in frame	1.79	0.12

TABLE VI. VIDEO CAPTURE CARD LATENCY

H. Camera Comparison

For our cameras, we had the choice between USB cameras and MIPI camera modules. USB cameras are easy to use and are readily available but the biggest concern for them was the greater latency, since the USB protocol includes routing through the CPU before it can be used, whereas a MIPI connection is sent straight to the memory. In addition to this, MIPI cameras were cheaper than a webcam. The final product that we landed on was the Raspberry Pi Camera Module V2 since it was highly rated and known to being compatible with the Jetson Nano and other Jetson environments. Our second design choice for the cameras was the number of cameras that we wanted to use. Part of our design requirements is to minimize the amount of time for speaker transitions and part of that will come from minimizing the time it takes for the motors to adjust the camera. Given that each of the chosen cameras has a 62-degree horizontal field of view, we could always have full coverage, but we felt that this was unnecessary, as we would still need to include rotation for the proper recentering. We wanted to choose the number of cameras that would reduce the maximum amount of yaw-axis rotation to a reasonable degree (see Figure 1). For example, in a single camera setup, the max we would need to rotate is 180 degrees to get to the furthest point away from our current field of view. We initially wanted to work with 3 cameras since that would reduce it to 60 degrees as our maximum rotation, but then we ran into the limitations of the Jetson Nano. The Nano could potentially accommodate additional cameras, but that would require us to purchase additional hardware to mux on the MIPI lanes. For this reason, we felt that we would still have a reasonable amount of rotation using only two cameras, and it would not require additional hardware, thus minimizing the total project cost. While testing with the servo and stepper motors we eventually came down to using only 2 cameras where each was responsible for half of the total 360 field of view.

V. System Description

The hardware system is centered around the Jetson Nano B01, which does most of the computation. It is powered by a 5-volt, 4-amp barrel jack power adapter. Directly connected to the Nano is a ReSpeaker v2.0 4-microphone array on one of the USB 3.0 ports, two Raspberry Pi V2-8 camera modules connected via MIPI CSI-2, and a Adafruit v2.3 motor HAT and Adafruit 16-channel Servo Shield on one of the I2C ports. Connected to the servo shield are two TowerPro SG92R micro servos, and to the motor HAT is a Adafruit Nema-17 stepper motor. All motors communicate to their respective motor controllers via PWM.

For the overall structure of the software, there are three distinct parts that all need to communicate with each other. These three components are the motor controller, the audio processor, and the speaker tracker. These components are all connected to a main program that will keep track of the current state of the software cycle. It will request new input from the audio processor, give motor instructions to the motor controller and signal to the speaker tracker when to start and expect a result from it as well. Threads for each of these components are all going to be concurrently running. [7, 11]

A. Motor Software

To control the motors, we utilized Adafruit CircuitPython, specifically the MotorKit and ServoKit libraries. [3] The micro servos are easily controlled by specifying angle within a range of 180 degrees to turn to. The stepper motors can only move by taking a 1.6-degree step at a time. To keep track of the location of the cameras, the main program keeps track of the latest angle for each servo. This is tracked in the main program because the information is needed to determine new angles based on the audio software. For the stepper motor, the motor controller software keeps track of how many steps the stepper has taken. The bottom of the elevator is at a 0 step count, and the top is at a 480 step count. To ensure our software can accurately place the vertical location of the cameras, iContact has a predetermined starting position for the stepper motors at the bottom of the elevator, and at shutdown will return to this position.

The thread for motor control also has some control over the audio and video processing threads. This changed from our original plan of the main thread controlling all communications between threads. During integration we discovered two issues. Firstly, and most concerning, was the stepper motor started to move very slowly. Secondly, the servo movement was loud and sudden enough for the audio processing to recognize as a voice. We reasoned the first issue arose from the addition of the audio processing; with the increase of workload for each thread, the motor thread was being interrupted more. Since the stepper motor could only move one step at a time, the audio processing would interrupt this critical section and thereby increase the latency between steps. To resolve this issue, we utilized the event class from Python's threading library. One thread signals an event and other threads wait for it. The motor controller thread signals an event and then waits for the other threads to finish their current iteration of processing. After the other

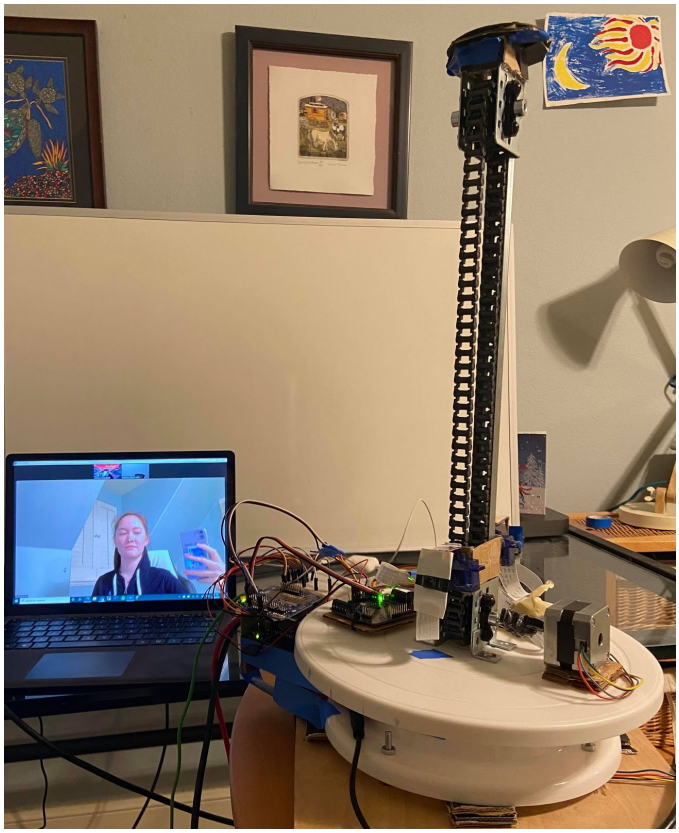


Fig. 6. Our project

threads are done processing; they wait for the motor event to be done. Once motor movement is complete, the motor controller thread clears the event, and all threads resume (See Figure 7).[11] This solution solved the slow stepper movement by only letting the motor controller thread run for that code section. This solution also fixed the second issue because the audio processor no longer collected data during the time the motors were running.

B. Audio Subsystem

The audio program makes use of the Python library [8] provided by the manufacturer of the ReSpeaker microphone array. Three of the built-in algorithms that we utilized were voice detection, which returns a Boolean value indicating whether the sound being picked up is that of a person's voice, and direction of arrival, which returns the angle of the sound being picked up relative to a designated zero-position on the microphone array. Both functions retrieve their values from the firmware on the ReSpeaker.

The voice detection algorithm distinguishes voices from other sounds based on the amplitude, frequency, and duration. With regards to amplitude, human voices, within the pickup range of the ReSpeaker's microphones, tend to be between 30 and 60 dB. Similar to how a voice's amplitude is expected to be within a certain range, human voices are typically between 300 and 3400 Hz (which is also the frequency range that phones

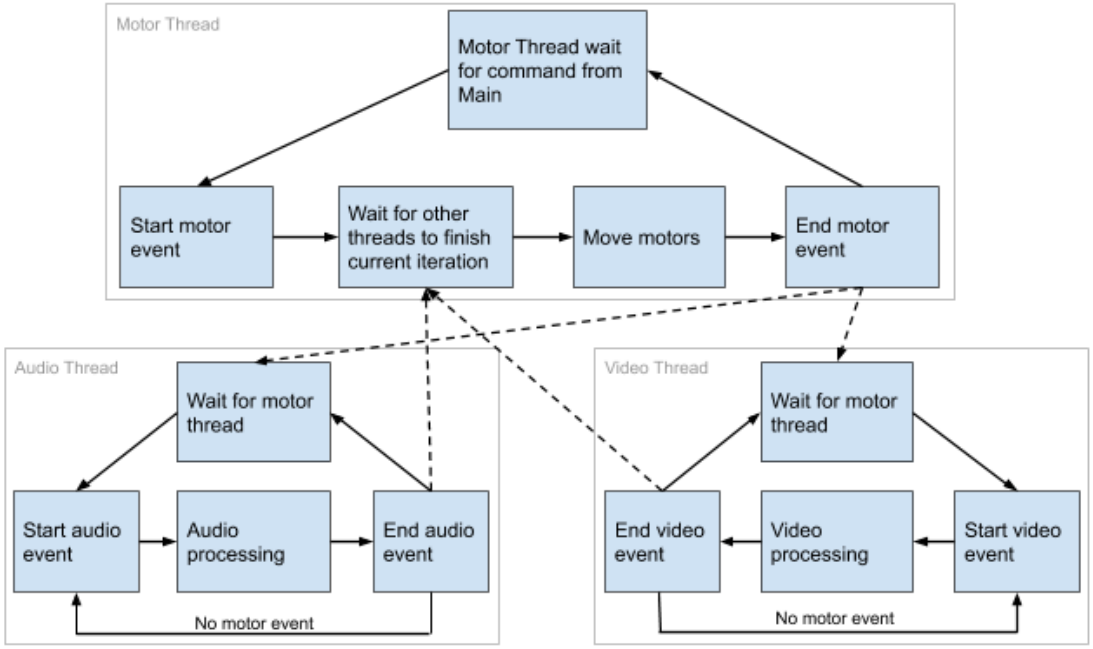


Fig. 7. Event flowchart

accommodate) in frequency. Lastly, a speech segment -- even a single word -- is usually at least 300 microseconds in duration. Any noises beyond these ranges are considered incidental noises and not detected as human voices [12].

The acoustic location algorithm works by utilizing the physics of the traveling of sound waves and the differences in arrival times of the speaker's sound to each microphone. From the measurements noted in Figure 8 and the speed of sound, c, we can get the time difference between one pair of microphones as:

$$\Delta t = (d/c)\ sin(a + \pi/2 - x + \theta) \quad (1)$$

From this [4], we can solve for x, the direction of the source. The ReSpeaker calculates the direction of the source among each pairing of its four microphones and returns the average [8].

Knowing how the built-in capabilities of the ReSpeaker worked, we were able to confidently utilize them for the iContact. While the microphone array can, of course, pick up all kinds of sounds, the iContact is only concerned with human voices. Thus, we took advantage of the ReSpeaker's voice detection functionality, such that the iContact is constantly listening but will ignore any sounds that are not a human voice. In the event that a voice is heard, the program begins collecting a sequence of 20 angles, as returned by the direction of arrival algorithm, waiting 50 milliseconds in between readings. Then, it compares the median of this sequence to the current speaker angle; if the two values differ by at least 10 degrees, the current speaker angle is updated to the median of the sequence. Otherwise, it is ignored, and the loop begins anew by collecting another sequence of angles.

C. *Computer Vision Software*

When called upon, the computer vision software will track for the face of a person [2]. These cameras are connected to the Jetson Nano via two MIPI lanes. The first step is to identify all the people in the current frame. This is done with the use of haar cascades. haar cascades are pre-optimized models for OpenCV and can detect a pattern within the image it is scanning through. After the frame has been processed by the OpenCV, there is a possibility that multiple people are detected. When this happens choose the center most person because we rely on the accuracy of the audio processing to properly give us the most accurate speaker. From here the motor instruction is created by approximating the angle by scaling the position in frame to the camera's field of view. The Raspberry Pi Camera Module v2 has a horizontal field of view of 62.2 degrees the distance of a person's face to the center of the frame was calculated and scaled to the horizontal field of view. After testing we found that the approximations gave us an average accuracy of 87.4% and a standard deviation of 12.6%. This can

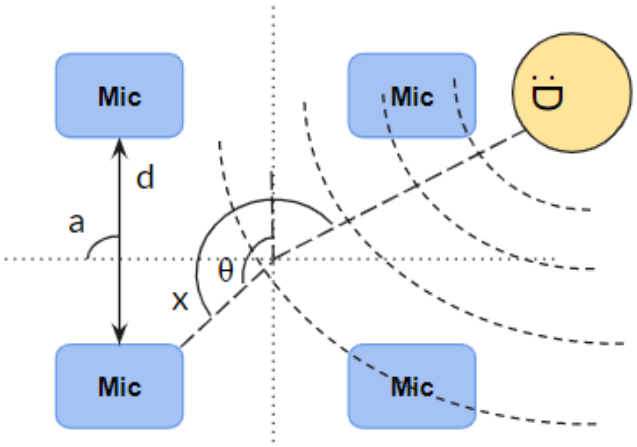


Fig. 8. Four microphone algorithm

be explained by our linear scaling which meant that if the speaker was already close to the center of the frame it would have a more accurate update than if the speaker started off farther away from the center. Due to the relatively accurate acoustic locating, it meant that we were less likely to have the extreme cases that detected faces closer to the edges of the frame. For the vertical update of the cameras, it was harder to calculate an exact degree step because we were using the motor to control a chain that would rotate. We instead gathered data on how an update on the motors would change the image that was being captured. We did our testing by having the program capture a single frame and find a face then update the motor to change its elevation and then look for the same face. The pixel distance was then divided by the number of steps we had the motor run and we found that in a 1080 image, 1 pixel was approximately a double step for our stepper motor. This vertical panning had similar problems in how a linear scale would not always work well but since the vertical field of view of the camera was lower, 42 degrees, the actual accuracy was at 90% with a standard deviation of 5%. Here we were able to barely meet our requirements and the standard deviation is still quite high, but the vertical centering was significantly better than the horizontal centering. With these two combined we were able to accurately center a face in the frame after a motor update. One of the key issues with using servo motors to adjust these cameras was that we could not actually have either camera set to watch at the degree ranges of 350 to 10 degrees and 170 to 190 degrees. Setting the servos to those degrees would lead to the servos to function improperly so we set a max range that the cameras could be set to. In order to still get a centered image, we crop the image around the face instead and aim to use as many pixels as possible. Instead of altering the image to fit the screen size we opted to maintain the quality of the image and just filled the empty pixels with black. This allowed us to maintain centering for the edges of our cameras' range.

D. Motors

The Jetson controls the stepper motor through an Adafruit motor HAT v2.3 and the servos through an Adafruit 16-channel servo shield. Located on each shield is a PWM chip driver, specifically the PCA9685 from NXP, which is a I²C-bus controlled 16-channel LED controller, but the PWM signal for controlling LEDs works for controlling motors. The Jetson communicates to each PWM chip driver using one of its I²C ports at a frequency of 100Hz. I²C is a communication protocol that allows one master, in this case the Jetson, to communicate to multiple devices of different addresses. There are two wires that are used as the bus: SDA for data and SCL for the clock. To communicate to the devices, the Jetson sends out the address of the device to establish communication with. Then the Jetson sends data regarding which motor it wants to control and at what speed or angle the motor should turn to. In our project, the two different PWM chip drivers are addressed such that the motor HAT's address is 0x60 and the servo shield's address is 0x40. After the PWM chip driver gets an instruction from the Jetson using I²C then it controls the motors by signaling through PWM.[6]

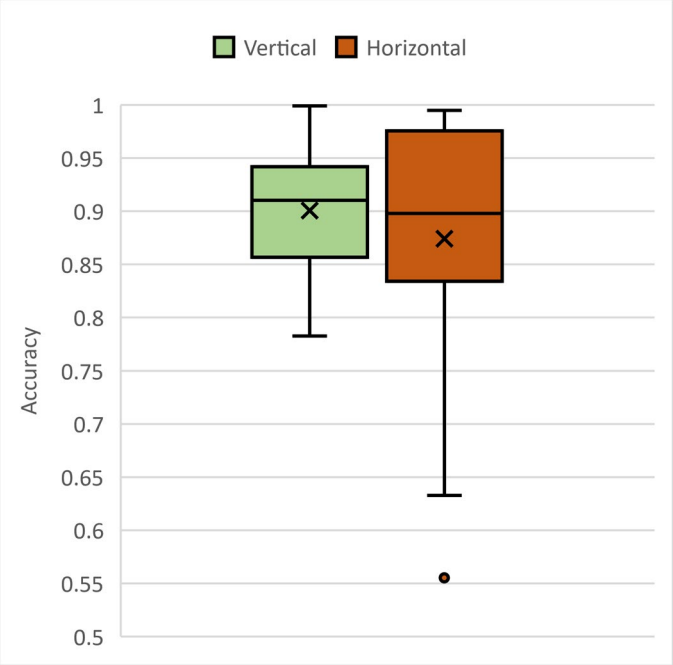


Fig. 9. Centering accuracy plot

PWM stands for pulse width modulation. There is one wire for signaling and it conveys information using duty cycle. One period of the signal would be the length of time the signal is high plus the length of time the signal is low before it goes high again. Duty cycle is the percentage of the period that the signal was turned on for. The signal has a set frequency. For the stepper motor, the frequency needs to be above 1600Hz. The servos require 50Hz for operating. This difference in frequency necessitated the use of two shields because the PCA9685 cannot be set to operate at two different frequencies.

For the two servos, the width of the PWM signal pulse indicates the angle at which the servo should turn to. 0.75 milliseconds are the minimum pulse width and indicated an angle of 0 degrees, while 2.25 milliseconds is for an angle of 180 degrees. For the stepper motor, six PWM signals go to a Driver IC for Dual DC motors (TB6612FNG). The stepper motor has two coils and the two separate signals for each DC motor instead go to each coil. Each coil is controlled by three PWM signals that are used to determine which of the five states of operation the coil should be in: clockwise, counterclockwise, standby, short brake, and stop.[9, 10]

VI. Project Management

A. Schedule

Our project schedule had changed little up to Week 7, as we had managed, for the most part, to stay on track. The only major change was the removal of the section of our plan where we intended to implement verbal command functionality, which we realized early on would be too great of a challenge on top of our existing design.

Beyond Week 7, there were some relatively substantial changes to the schedule for some of the individual components of the project as we scrambled to get them working for integration. Most notable of these changes were the extended time allocated for video passthrough, which proved to be a greater challenge than anticipated because of a lack of well-documented solutions, and the multiple changes in audio implementation plans. The latter resulted in the most schedule deviation, but these changes only minorly impacted integration and testing, which we were able to complete on time.

B. Team Member Responsibilities

We have divided the work for our project such that Heather's primary task is motor control, Edward's CV, and Anna's audio. As for our secondary tasks, Heather and Edward are working jointly on hardware communication for the video feed (i.e. camera to Jetson to computer), while Anna and Heather are working together on hardware communication for the audio feed (i.e. microphone to Jetson to computer). Given that we are all in different locations for this remote semester, Heather, alone, took on the additional task of physically assembling the device. When it was time to integrate and test, we worked together remotely over Zoom while Heather ran the iContact.

C. Budget

Our bill of materials may be referred to in Figure 10. We had to purchase multiple (sets of) items, such as the Jetson Nano, cameras, and microphones, so that each of us could have our own partial implementation of the project, since we are all working remotely. The blue rows are the items that we planned to purchase from the start; the yellow rows are the items that we did not initially plan to purchase but ended up needing; and the red rows are the items that we purchased but did not use. The "Cost to Budget" column shows what we spent our given $600 budget on (shipping and tax included), with the green rows at the bottom showing the total spent and how much we had left over. The neighboring "Cost of iContact" column shows the cost (not including shipping and tax) of the components required to construct one iContact, with the total in the green row at the bottom.

Item	Quantity	For Whom	Cost to Budget (includes shipping and tax)	Cost of iContact (does NOT include shipping and tax)
Jetson Nano	1	Heather	$107.91	$99.00
Jetson Nano	1	Anna	$104.94	
Jetson TX2	1	Edward	Rented	
MicroSD Card (128GB)	1	Heather	Already owned	$19.49
MicroSD Card (128GB)	1	Anna	Already owned	
Adafruit 16-Channel 12-bit PWM/Ser	1	Heather	Already owned	$17.50
Adafruit MotorHat v2.3	1	Heather	Already owned	$18.90
Adafruit TowerPro SG92R	2	Heather	Already owned	$11.90
Adafruit Nema-17 Stepper Motor	1	Heather	Already owned	$14.00
12V, 5A Power Adapter	1	Heather	Already owned	$10.99
5V, 4A Power Adapter	1	Heather	Already owned	13.99
Video Capture Cards, HDMI to USB 2	1	Heather	Already owned	$14.99
Raspberry Pi Camera Module V2-8	2	Heather	$45.40	$39.98
Raspberry Pi Camera Module V2-8	2	Edward	$45.40	
Adafruit I2S MEMS Microphone	2	Heather	$11.00	
Adafruit I2S MEMS Microphone	4	Anna	$41.12	
USB TTL	1	Heather	$16.62	
Adafruit ADS1115 16-bit ADC	1	Anna	$15.59	
Adafruit MAX9814 Microphone	3	Anna	$27.30	
Capacitor Assortment	1	Anna	$4.72	
Adafruit Mini USB Microphone	3	Anna	Already owned	
Adafruit MCP3008 ADC	1	Anna	Already owned	
ReSpeaker v2.0 4-Microphone Array	2	Anna & Heather	$167.48	$79.00
Total Spent			**$587.48**	**$339.74**
Budget			**$600.00**	
Amount Leftover			**$12.52**	

Fig. 10. Centering Accuracy Plot

D. Risk Management

From the beginning of our project, we did a great deal of planning to minimize the many risks -- particularly with respect to integration -- that would inevitably come with working on a mechanical design while the three of us were all in different locations. For example, knowing integration would be perhaps the most challenging part of our project, we made sure that none of the integration tasks were solo tasks.

Throughout our project, there were multiple times where we realized we had misjudged just how critical and arduous an individual's task was. To ensure that these tasks were executed well and that all our bases were covered, we would designate one or two other team members to join in the work. For example, the video transmission communications turned out to be much more involved than any of us had anticipated. As a result, we changed this problem into a two-person endeavor for the next couple of weeks. During this semester, there were many instances where we had to reprioritize.

Moreover, we spent a great deal of time coming up with what components to use in our design and the potential risks that came with each decision. One of the biggest decisions we made was whether to use a Jetson Nano or a Raspberry Pi, which would be the brains of our entire device. The primary features we were examining while deciding between the two was the peripheral support. Our overarching ideology was "the more peripherals, the better" -- it is, of course, always easier to add components or change protocols with an excess of ports than a lack thereof. For instance, we knew our project would require the use of either one or two cameras, but at this early stage, we were still unsure of which quantity we would ultimately agree upon. In that respect, the Jetson, having two video lanes, was the safer bet compared to the Raspberry Pi, with its singular video lane. When it came to choosing microphones, we initially opted for I2S, in part because it is designed to be digital (and would thus have less noise than analog), and because it would not use up any of the USB ports, which we wanted to keep open in case we needed to add any other components to our project down the road. Lastly, in general, we did our best to minimize our spending to leave us enough room in the budget for unexpected purchases to be made in the future. We went through numerous rounds of trial and error with different audio hardware, so our frugality from earlier in the semester paid off and allowed us the flexibility to try new implementations.

VII. Related Work

Aside from iContact, there are several other similar products that seek to make video calls more personal using smart cameras that dynamically focus on whomever is commanding attention. One such existing solution is the Meeting Owl, a smart video conferencing camera that captures 360-degree video and audio. Its single omnidirectional camera takes in a static, panoramic view of the room from the center of the table, then displays the section of that view where the current speaker is located. One of last semester's CMU ECE Capstone projects, COMOVO, was also a smart video conferencing camera to be placed at the center of the table. Instead of having a panoramic view of the room like the Meeting Owl, this unidirectional camera was motorized and capable of turning to whoever is speaking either automatically or by interpreting physical gestures. Other devices, like Google Meet's video conferencing hardware, Polycom's Poly Studio, and Facebook's portal feature a stationary, unidirectional camera to be placed at the front of a conference room such that it has a full view of everyone present and can zoom in on the current speaker. What sets iContact apart from all these products, which seem limited to one plane of view, are its abilities to raise and lower, giving it a vertical range of view that existing solutions cannot achieve.

VIII. Summary

Overall, we were able to meet the majority of the requirements that we set out. The requirements that we did not reach were in the fields of acoustic location range and the centering of the cameras. For the acoustic location detection, we were primarily limited by the complexities of sound wave reflections as the speaker moved farther away from the iContact. The horizontal centering accuracy varied a lot because of our linear approximation for calculating the motor angle update. Given additional time, we could improve our system performance with more sensitive microphones and enhancing the centering algorithm to be more accurate. Another improvement we could try is converting our software to be based in C rather than Python, which would increase execution speed on the overall system so that every update is fastern the overall system so that every update is faster.

A. Lessons Learned

One crucial lesson we learned was that, for a project of this scale, something will go wrong, and the team needs to be prepared for that from the beginning. That means, in the early stages of the project, leaving lots of room in the schedule for rerouting, as well as minimizing costs to ensure that the budget can sustain changes in implementations throughout the semester. It is also important to document as much as possible, including failed implementations and even ideas that were never implemented. Looking back on these notes will help with defining the trajectory of the project and making decisions in the future.

Functionality	***Requirements***	***Results***	
Viewing	High compatibility	✓	Worked with Zoom, WebEx, Google Hangouts
	1080p @30fps	✕	1080p @21fps
Working range	360-degree field of view	✓	Can turn to any direction
	1ft vertical panning range	✓	Full range of motion up/down 1ft shaft
	10ft acoustic location range	×	100% accuracy up to 4ft; 90% up to 6ft
	10ft person detection radius	✓	100% accuracy up to 10ft
Algorithm accuracy	90% centering accuracy	×	Horizontal: 70-90%; vertical: 90%
	90% speaker identification accuracy	✓	100% with 2 speakers; 90% with 3 speakers
Speed	<1s motor control for camera adjustment	✓	0.3957s
	<1s audio input processing latency	✓	0.6451s
	<1s video input processing latency	✓	0.2749s

TABLE VII. Design Requirements Results

REFERENCES

[1] Industries, Adafruit. “Micro Servo.” Adafruit Industries Blog RSS, www.adafruit.com/product/169.

[2] Kumar, Akshay. “Real-Time Face Detection on Jetson Nano Using OpenCV: Nvidia Jetson.” Maker Pro, Maker Pro, 7 Feb. 2020, maker.pro/nvidia-jetson/tutorial/real-time-face-detection-on-jetson-nano-using-opencv.

[3] LeBlanc-Williams, M. “CircuitPython Libraries on Linux and the NVIDIA Jetson Nano.” Adafruit Learning System, Adafruit, learn.adafruit.com/circuitpython-libraries-on-linux-and-the-nvidia-jetson-nano/circuitpython-dragonboard.

[4] Nehorai, A. “Interaural Time Difference.” Interaural Time Difference - an Overview | ScienceDirect Topics, Elsevier B.V., 2015, www.sciencedirect.com/topics/engineering/interaural-time-difference.

[5] ONVIF, www.onvif.org/.

[6] PCA9685, NXP Semiconductors, 16 Apr. 2015, www.nxp.com/docs/en/data-sheet/PCA9685.pdf.

[7] “Queue - A Synchronized Queue Class.” Queue - A Synchronized Queue Class - Python 3.9.1 Documentation, docs.python.org/3/library/queue.html.

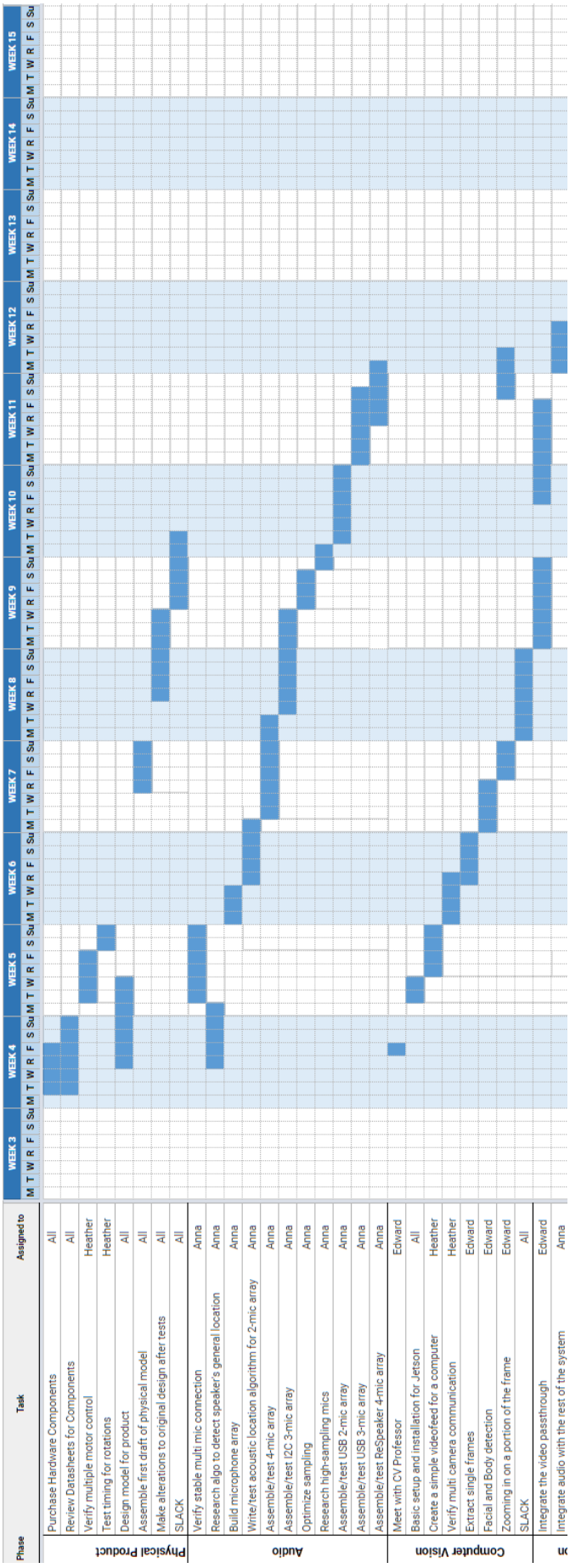
[8] ReSpeaker, jerryyip. “Respeaker/usb_4_mic_array.” GitHub, Seeed, 11 June 2019, github.com/respeaker/usb_4_mic_array.

[9] “STEMMA.” Adafruit Learning System, learn.adafruit.com/assets/9536.

[10] “TB6612FNG.” TB6612FNG | Brushed DC Motor Driver ICs | Toshiba Electronic Devices & Storage Corporation | Americas – United States, toshiba.semicon-storage.com/us/semiconductor/product/motor-driver-ics/brushed-dc-motor-driver-ics/detail.TB6612FNG.html.

[11] “Threading - Thread-Based Parallelism.” Threading - Thread-Based Parallelism - Python 3.9.1 Documentation, docs.python.org/3/library/threading.html.

[12] Zuo, Baozhu. “ReSpeaker Mic Array v2.0.” Seeedstudio, Seeed Technology Co., wiki.seeedstudio.com/ReSpeaker_Mic_Array_v2.0/.


Phase
Task
Assigned to
WEEK 3
WEEK 4
WEEK 5
WEEK 6
WEEK 7
WEEK 8
WEEK 9
WEEK 10
WEEK 11
WEEK 12
WEEK 13
WEEK 14
WEEK 15
Physical Product
Purchase Hardware Components All
Review Datasheets for Components All
Verify multiple motor control Heather
Test timing for rotations Heather
Design model for product All
Assemble first draft of physical model All
Make alterations to original design after tests All
SLACK All
Audio
Verify stable multi mic connection Anna
Research algo to detect speaker's general location Anna
Build microphone array Anna
Write/test acoustic location algorithm for 2-mic array Anna
Assemble/test 4-mic array Anna
Assemble/test I2C 3-mic array Anna
Optimize sampling Anna
Research high-sampling mics Anna
Assemble/test USB 2-mic array Anna
Assemble/test USB 3-mic array Anna
Assemble/test ReSpeaker 4-mic array Anna
Computer Vision
Meet with CV Professor Edward
Basic setup and installation for Jetson All
Create a simple videofeed for a computer Heather
Verify multi camera communication Heather
Extract single frames Edward
Facial and Body detection Edward
Zooming in on a portion of the frame Edward
SLACK All
Integrate the video passthrough Edward
Integrate audio with the rest of the system Anna