wave Google

Author: Sung Jun Hong, Jeffrey Huang, Claire Ye
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—Current smart home devices require speech and hearing to operate, which makes them inaccessible to the mute and deaf communities. Gesture-based commands can be integrated with the Google Assistant SDK in order to achieve a fully speech- and hearing-free smart home experience. Small gesture recognition through a Nvidia Jetson low-power embedded system allows the user to indicate and form Google Assistant commands, which are then queried online to receive a response from.**

***Index Terms*—Google Assistant, Google Home, Django, AWS, EC2, Elastic Beanstalk, OpenCV, OpenPose, Nvidia Jetson Nano, Support Vector Machine**

I. Introduction

By the end of 2019, over 33% of American residences have at least one smart home device installed in it [1]. Despite the popularity of digital home assistants, these devices lack accessibility. Voice-based commands cannot be used by the deaf or mute community, who may especially benefit from a smart home device due to their condition. Thus, there is an opportunity for a smart home device that takes away the voice-based commands without diminishing the experience. Gesture-based input is a new front being explored by various technology giants, such as Project Soli at Google [2], [3] and Kinect from Microsoft [4]. Kinect is a very primitive gesture recognition hardware that requires clunky hardware and backend customization in order to change into a smart home device. Project Soli is a processor chip that is incorporated into the Pixel 4 [3], and therefore can only be accessible through the purchase of that phone. We wanted to find a middle ground where we could customize the gesture commands to one that is specific to smart home devices (instead of phones) without the need for coding as Kinect would.

We expect to have a product that is on-par with the voice-based Google Home (the smart home device we found to be the "smartest") [2] experience. Therefore, we require our device to respond correctly to queries 80% of the time (the 20% error includes anything from not attempting the question at all and answering it incorrectly). Based on our empirical testing, the average Google Home answer takes around 2 seconds to complete, so the same timing requirement should be applied to this project in order to maintain a Google Home's convenience. Since Google Homes are usually installed in a room with normal lighting, the device should be functional within 500-1000 lux, which is standard indoor lighting conditions [5]. At last, voice-based Google Home can be heard from within the same room, so the project should be able to detect and respond to gesture-based commands within .5 to 3 meters inside the camera's view, which is within a few steps' distance in a small to medium room [6].

II. Design Requirements

Given the target application of creating a gesture-based smart home assistant analogous to a Google Home, our design requirements are largely based on the performance of Google Home.

As stated before the overall command accuracy is 80 percent derived from Google Home's accuracy. This means that for any given command or lack of command, the expected response should be produced 80 percent of the time. Going along with this requirement is gesture recognition accuracy which we set to be 90 percent. This means that for a given gesture, does it successfully identify this as the correct gesture. The accuracy requirement is a bit higher than overall command accuracy, as commands include multiple gestures and although Google Assistant has a means of inferring context and thus correcting some mistakes, too many gesture misclassification would result in overall command errors. Since most commands will include two gestures, having a 90 percent gesture accuracy will allow for an overall accuracy of 80 percent. Accuracy is a core requirement as without reasonable accuracy the smart home device would be unusable, providing wrong responses or none at all. Testing for both will have a similar approach. First, we will have a library of static gestures at 1.75 meters and 750 lux, median distance and lighting. This will include both gestures that are valid as well as false gestures that are not valid. For simple gesture accuracy, we will have an automated stream of the images at 1 per second, recording if it correctly identified the gesture. For each gesture, we will also record the individual accuracy and note other gestures that it is often misclassified with. For overall command accuracy, we will chain a series of gestures, for example activation and what's the weather, and pass in that series of images, and see if the output matches. In addition, false gestures will be added before activation in some cases, to ensure that the device does not accidently activate.

Beyond accuracy, we also have speed requirements. This includes overall speed from testing of Google Home, as well as gesture speed requirements. Like previously stated, our overall speed requirement is derived from the average Google Home Answer after sending in a command. In our case, this would be after doing all our gestures for a given command, our smart home device would relay that information back within 2 seconds. In addition, for gesture recognition speed in particular, this would be included in the 2 second overall

command requirement, but would include just the time our gesture recognition algorithm takes to identify the gesture. We choose a requirement of 50 milliseconds as results from finding from papers running on more expensive hardware took 20 milliseconds, coupled with the fact user will feel anything under 100 milliseconds is instantaneous, 50 milliseconds is an appropriate requirement [7]. Testing for both will be done using the same automated system as described above, but instead of focusing on the output we will capture the time of operation after the gesture or command given, and checking if it is under 50 milliseconds or 2 seconds respectively.

Furthermore, we also have requirements on how fast gestures can be inputted into the system or how many gestures per second. Especially for more complicated commands, it is important for the system to be able to take in gestures at a sufficient rate or else user experience will be negatively impacted. We choose a requirement of 1 gesture per second, given how people talk at roughly 4 syllables per second [8]. Gestures can encapsulate multiple words and syllables, with our most common gesture the activation gesture being analogous to "Hello Google". Testing would use the same framework as above, but the stream would be increased or decreased in speed to test different rates of gesture input.

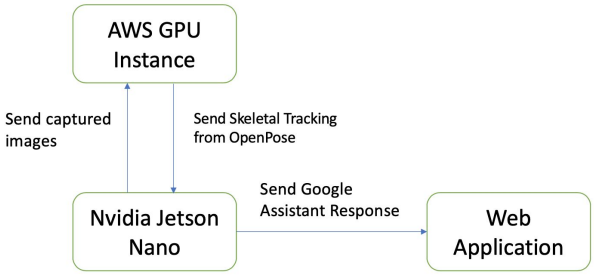
In addition, our smart home device is designed to work in standard lighting conditions as a Google Home would be operated in as well, which is 500-1000 lux. Once again we would implement a similar testing system as before, however this time instead of all images being at 750 lux, we would increment from 500 to 1000 lux. We would include both pictures taken at different lighting conditions, as well as using photo editing software to artificially change images to simulate a certain lighting condition. For our distance requirement, we choose .5 to 3 meters as it falls in range of a standard room size as well as the limitations of having a single mounted line of sight camera. Testing would be done similar to lighting tests, but changing the distance the images are taken. This again will include images taken at different distances as well as edited images to simulate a certain distance.

Finally, we have our last requirement for ease of use. Unlike the previous requirements, this is qualitative rather than quantitative. As a smart home device, it is important for the device to be intuitive and simple to use, else it does not fulfill its purpose of facilitating day to day actions. To test these, we plan on implementing user testing on both gestures themselves, ensuring that none are too difficult to perform, as well as the overall ease and experience of the whole system. Users will be asked to rate each gesture and the experience from 1 to 10. 1 would be considered unusable, 5 would be considered more or less a similar experience as before, and 10 would be considered a substantial improvement. Given this, we choose a rating of 6, which would indicate some improvement over the previous experience.

III. Architecture and/or Principle of Operation

Our architecture has four components so far. These components are the Nvidia Jetson Nano, the Web Application, and the Amazon Web Server running our GPU machine, and our Machine Learning Component.

On a high level, our Nvidia Jetson Nano will take photos at 28 fps, and will send those images to AWS to run OpenPose on those photos. On the AWS, the photos will then be passed through OpenPose to get the skeletal joints of the hand, and will pass the results back to the Nvidia Jetson Nano. The Jetson Nano will then classify the images to some predetermined gesture, and will pass the result to a Google Assistant, and will pipe the result to the Web Application, where the information will be displayed on a screen.



A. Nvidia Jetson Nano

The Jetson's architecture model is described in figure 4. The Jetson will have OpenCV running on its GPU, and on the CPU will handle the classification of gestures as well as communicating with the google assistant and the camera. The Jetson will have to communicate with the camera to take photos at a frame rate of 28 fps. With OpenCV, the images that are taken will be returned with the location of the user's joints on their hands, which will be passed through the classifier to map the user's gestures to a predetermined command. This command will then be passed to the google assistant, which will return the response back to the Nvidia Jetson Nano.

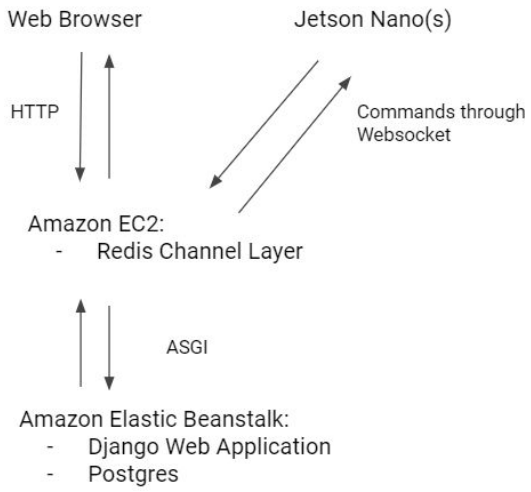
B. AWS GPU Instance

Our architecture model for AWS is a little bit simpler. The only reason we included AWS was that our Nvidia Jetson Nano's GPU capabilities were too weak to support OpenPose, so we decided to use a machine that could withstand the requirements of OpenPose. As such, we will only be running OpenPose on our AWS server, and have the server return the location of joints back to the Nvidia Jetson Nano.

C. Web Application

Our web application also has a simple architecture model. Being a Django channels application, we plan to deploy the web application to AWS Elastic Beanstalk, which simplifies

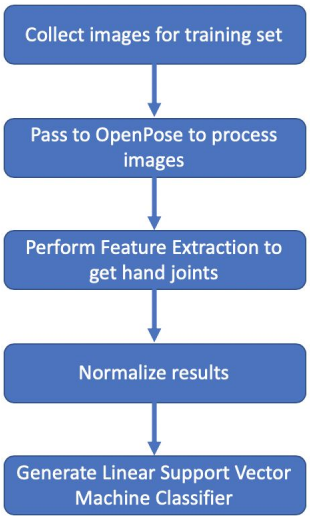
provisioning and managing infrastructure involved such as EC2 instances and Postgres DB. An additional Redis channel layer on EC2 will be used to support the web application as well. The web application will then communicate with the Jetson Nano via simple Web Socket connections.



D. Machine Learning

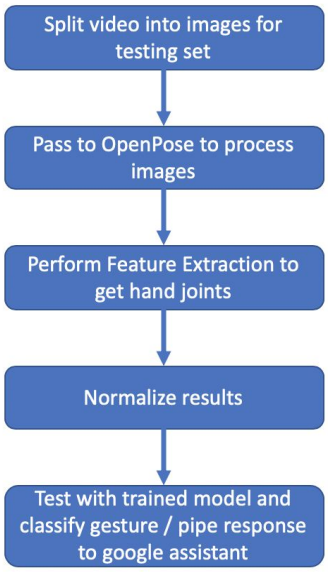
Our Machine Learning Component is as follows. It is loosely split up into two parts, the Training phase, and the Testing Phase.

Data Training



In the training phase, we collected images from the training set. With the 43 different gestures that we have, we collected more than 9000 images to train with. With these images, we feature extracted by getting the location of the joints on the hands of users. We did this by using OpenPose on AWS and using OpenPose's skeletal tracker that is built into the library. The result that we got post-processing from OpenPose and feature extraction was a list of 42 points. These 42 points represented 21 (x,y) joint locations of the hand. Thus, by using OpenPose, we were able to reduce our number of features to 42 features from an image. After getting a list of all the skeletal joints, we performed normalization, to make sure that the data we are using, is scaled properly to not yield any abnormal results. We then trained the linear support vector machine classifier with the normalized data and generated a gesture classifier.

Data Testing



In the testing phase, we will use the camera of the Jetson Nano to split live video into images. From there, we will pass the images to OpenPose on AWS that supports skeletal tracking. We will then feature extract to get the location of the hand joints, and then we will pass the normalized information to our trained classifier, that will classify the gesture we passed in. It will then query the google assistant with the gesture that it classified to.

IV. Design Trade Studies

In coming up with our final design, we made some choices and trade offs to improve ease of implementation at the cost of some ease of use.

A. OpenCV and Glove Tracking

We initially had decided to implement a variant of OpenPose ourselves using OpenCV, which we quickly scaled back on as being out of the scope of the project, pivoting to a glove based approach to run locally on the Jetson Nano. OpenCV would be used to track the glove which would have physical markers on the joints of the hand, producing a similar feature list to OpenPose. However, there were problems that

arose from the glove approach. OpenPose is able to predict the location of joints that are not physically visible. For example, when performing a fist, the markers behind the fingers are covered, but OpenPose is able to extrapolate the location of the markers. OpenCV with a glove with physical markers on the glove lacked the ability to extrapolate covered markers with anywhere near a similar degree of accuracy. 69.7 percent of our gestures had multiple markers that were covered, resulting in features lists that were incomplete and often very similar between gestures. Rather than switching our gestures to support our glove tracking or develop another classifier solely for glove based features with potential poor accuracy, we decided to stick with OpenPose alone for feature extraction.

B. Static Gestures vs Dynamic

Another tradeoff was instead focusing to abandon dynamic gestures and focus on static gestures instead. This tradeoff limits the potential gestures we can choose from. In addition, some fingerspelling gestures are dynamic and by eliminating them, introduces a further learning curve for our gesture set. Furthermore, prior work on dynamic gestures had developed an efficient and lightwork algorithm, Temporal Shift Module (TSM), capable of running at 70 FPS at only 8 watts on the Jetson Nano. However, we decided to make the tradeoff to simply focus on static gestures, as by including dynamic gestures we would have to split our work into static and dynamic, as dynamic gestures would not be able to fully encapsulate the alphabet efficiently. Therefore, to simplify work and limit the number of different algorithms necessary, we choose to stick with static gestures. This eliminates TSM as a solution approach as TSM is time based which is not a factor in static gestures.

C. Gesture Speed

We considered using different algorithms for determining a gesture to be "detected". For example, if within a time frame, we saw the gesture consecutively for ¾ of that time period, the script deemed the gesture as "seen" instead of noise. While there are surely better noise reduction algorithms, in the interest of time and accuracy we opted for 1 gesture per second, despite its lowered usability.

The camera performed, at best, 20 frames per second. If we wanted to do a gesture every half a second, we would need there to be around 7-8 frames of actual gesture. That leaves around 2-3 frames between one gesture and the next. This is *very* little time for a user to switch between hand signs, especially one who is not well-acquainted with the gestures yet. In this case, we would have needed the user to be able to switch between gestures in 0.10 - 0.20 seconds in order for the gestures to be consecutive.

On the other hand, if the metric is kept at 1 gesture per second, then the user has around 0.25-0.50 seconds to switch between gestures, which we found to be more reasonable.

D. Hardware Choices

One choice we had to make was to choose the appropriate hardware for the scope of this project. Because we were focusing on gestures, we needed some reliable way of extracting joint locations from hands. As such, we decided on two different ways we could make this work. One was through using a glove and tracking markers with OpenCV, and the other was using OpenPose and their built in skeletal tracking. OpenCV does not need that much GPU capabilities, but OpenPose needs to run on a GPU with very high capabilities. As such, we contemplated over using the Nvidia Jetson Nano and the Nvidia Jetson Xavier. Upon testing the Nvidia Jetson Xavier, OpenPose showed us a frame rate of 18 fps. Given that the Nvidia Jetson Nano's GPU capabilities are about 8 times less, we made the assumption that the Nvidia Jetson Nano's frame rate would be at the very best 3 fps. The Xavier boards 18 fps would be a reliable frame rate for us to use, but given that our problem was to provide a reliable disability-friendly smart home device, we didn't want to use a board that would be out of the smart home device budget range. As such, we chose to stick with the Nvidia Jetson Nano, and instead of running OpenPose on the Jetson Nano's GPU, we decided to use an AWS Machine with good GPU capabilities to run OpenPose.

E. Feature Extraction

We were considering different ways to do feature extraction. We had the option to train images as a whole, using 2D convolutional neural networks. This would provide to be more accurate, as there would be more total features as a whole since we would be using the full and complete data. However, since each image is initially 5 megapixels with downscaling, this could potentially be a large amount of data to store and process, increasing the overall computational time. By instead only storing the locations of the key joints, we can greatly reduce the amount of information and overhead needed to determine a gesture, and thus reduce training time and improve classification time. However, our main priority is to make sure that we hit our test metrics of accuracy, and if it shows that using the entire image as our feature helps with the accuracy, it might be better to switch over to preserve the accuracy that we need.

F. Machine Learning Models

We thought about using Neural Networks (NN) as our primary means of classification. Particularly, we attempted to use a 2D image based NN classifier. However, when experimenting we found limitations in this approach. First, this required normalizing the images which we attempted to with OpenCV. With a HSV approach to detect and normalize the location of the hand, we were able to successfully normalize the hand less than 60 percent of the time, with conditions like cluttered background and lack of clear

distinction between hand and arm without sleeves. Furthermore, this combined with the fact hand gestures itself when viewed as an outline are often very similar resulted in very poor performance.

By switching to a non-image based approach, with OpenPose which generated points of the markers of the hand we had a much smaller feature list than with images. This minimized the effectiveness of Principal Component Analysis in helping to reduce the dimensionality. Finally, as the features generated by OpenPose were linearly separable, Support Vector Machines proved to be highly effective and fast, becoming our goto gesture classifier.

V. AWS Instance

We opted to use an AWS instance to run OpenPose. This was because of an AWS p2 XLarge instance. This instance had 1 GPU, which gave us a 60x speedup when running OpenPose. With this, we were able to decrease our overall response time. In addition for the instances used to support the web application, we experimented with several different sizes and levels of performance, before settling on the t2.micro, as the difference in performance when upgrading to more powerful models was trivial for the web application as latency between Elastic Beanstalk, Redis channel layer, and the web socket connection itself proved to be the bottleneck rather than performance of the AWS instances.

VI. Size of Image

In an attempt to optimize OpenPose, we also experimented with the size of the image in an attempt to improve performance. A 2MB image even on our AWS instance took 9 seconds. By reducing it to around 200KB, roughly the size of a 1080x720 image, we were able to speed OpenPose by 18 times to .5 seconds. From there we tried reducing the image smaller, but we reached a peak speed of roughly .41 seconds at a resolution of 810x540, having diminishing returns from there. OpenPose was able to work at images as small as 108x72, but with marginal performance gain with a speed of also roughly .41 seconds, and negative performances on maximum sampling, we chose a resolution of 810x540 proved to be the best compromise.

VII. System Description

A. Nvidia Jetson Nano & Peripherals

The Nvidia Jetson Nano is a low-power embedded system equipped with a hefty GPU. It can accept power through either a microUSB connector or a power jack. The microUSB typically supplies 2 Amps at 5 Volts, which is the standard power supply of a phone charger.

The quad-core CPU runs ARM A57, a microarchitecture that implements the ARMv4 64-bit instruction set with out-of-order superscalar pipeline. This particular microarchitecture is suitable for integration with other cores, such as a GPU, into a one-die SoC. [9] The 128-core Maxwell GPU contains proprietary Nvidia microarchitecture that dramatically improves power efficiency through its streaming processor. [10] A GPU is necessary to achieve parallelism in simple computations in image processing and machine learning applications.

The 4-GB 64-bit Low-Power Double Data Rate Synchronous Dynamic Random Access Memory is included in the developer kit. This RAM is targeted at mobile devices, with low power consumption. This allows easier interaction and programming with the Nano through a widely used OS.

The various connections available on the Nvidia Jetson Nano Developer Kit makes it an ideal choice for projects involving multiple peripherals.

The e-con Systems camera SoC for Jetson Nano was chosen as the camera of choice for this application. The camera has 5 MegaPixel resolution, providing clear pictures for small hand gestures. At 5 MP, the camera takes pictures at 28 frames per second, supplying plenty of data for gestures that each lasting roughly 1 second. The SoC is connected to the Nvidia Jetson Nano through a camera connector through the MIPI CSI-2 protocol, which is a widely adopted high-speed camera interface for mobile applications. [11]

The storage for the Nvidia Jetson Nano is provided through 128 GB microSD. This amount of storage allows ~100 GB of free space for the Jetson to work with on top of the Linux Ubuntu 18.0.2 boot image.

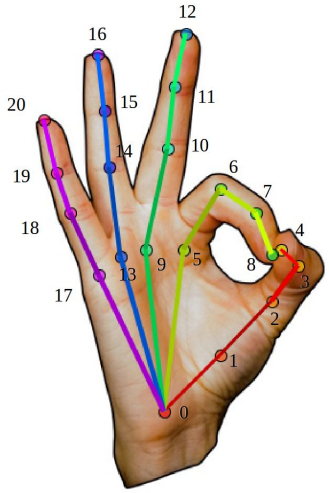
Wireless capabilities are added to the Nano through the Intel 8265.NGWMG.DTX1 Wifi Wireless-Access Point. The WiFi card is integrated through the M.2 connector key E available on the developer kit. This card achieves up to 867 Mbps, but realistically will go up to around ~400 Mbps for both download and upload speeds, which is comparable to standard mobile devices and fast enough for this application. The WiFi functionality needs to be at least industry standard because the device will be uploading images to the AWS server rapidly.

A screen is needed to display the results of the commands. A HDMI connection is used because it provides high bandwidth with a single cable with easy setup. The screen is 7 inches wide, making the overall system portable. The 1024x600 resolution allows clear display of text and images as necessary.

B. Machine Learning

We used OpenPose to help with the skeletal tracking of the hand. OpenPose is an open source software that tracks a targets body, and if hand tracking is enabled, OpenPose will run hand skeletal tracking. OpenPose on AWS will give out a list of joint locations of the user's hand. The process of getting the joint locations of the user's hand is our way of feature extraction. With the image that we get, which is thousands of pixels that could correlate to thousands of features, we opted to use skeletal trackings to track the location of different fingers relative to the hand. We made this decision because of

our strict requirements. Based on our requirements, we need to give a response within 2 seconds after the initial gesture. Because of this, we wanted to reduce as much time classifying our gesture as possible.



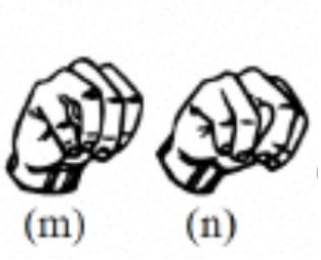
Because of this, we opted out of using an image as our features, and processed our image into a list of all the relevant joint locations. With this, we expected our speed to improve drastically, as there are less features to work with meaning that there is less computation for our learning algorithm to classify the data.

After getting the 21 (x,y) locations of the joints on the hand, normalization was done on the data. A reference gesture was chosen at random, and the distance of the elbow to the wrist was found. For each subsequent image, the same distance was found and a scaling factor between the current image and the reference image was found. With the scaling factor, we scaled the data points that we received, to preserve the overall look of the gesture while scaling. From hand point 0, we measured the distance to all the other points. Thus, we had 20 distances from hand point 0. We scaled these distances using the scaling factor, and we found the angle of all the two points using trigonometry. After we found the angle, we were able to update all the (x,y) locations of the 20 points to accomodate for the scaling factor, while preserving the angle, preserving the initial state of the image. We then placed an offset on all the hand points such that hand point 0 would be located at (0,0). With this, we were able to normalize all our data, regardless of the gesture or hand positioning.

After the normalization, we passed the data into a support vector machine classifier using a linear kernel. After experimenting with a linear kernel, gaussian kernel, and polynomial kernel, a linear kernel yielded the best results. The gestures being linearly separable helped the support vector machine classifier to have a high accuracy rate once trained, which is why we decided to use it.

To train our model, we collected a lot of data with the 3 team members and some students from CMU. One problem that we ran into was that OpenPose was not good enough when it came to near sight detection. This meant that OpenPose would not be able to track hands if they were too close to the camera. After running a few simple tests, we realized that the user had to be about 1 meters away from the camera for OpenPose to recognize the hand. Another thing that we realized while working with OpenPose was that OpenPose requires a face to track in order for skeletal tracking to work well.

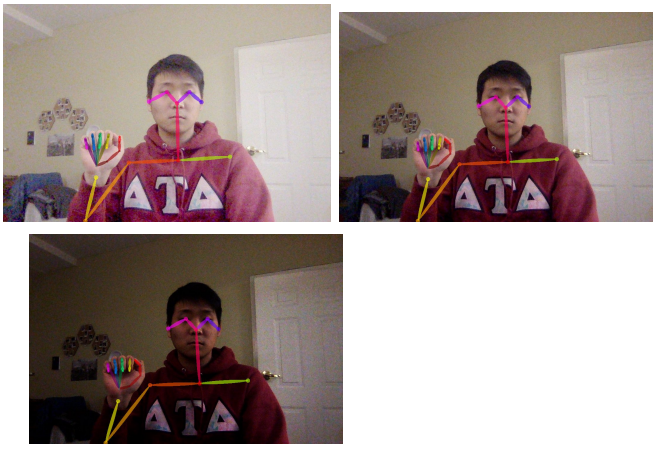
We used specific gestures to tailor our project. We used American Sign Language gestures as well as custom gestures for custom commands for the smart home device. However, there are some ASL gestures that are very similar to one another.



The two gestures above are ASL for the letters “m” and “n” respectively. We suspected that even with skeletal tracking, the feature extraction for these two gestures would be almost identical. As such, we had to use our custom commands that we created. We collected more than 9000 images representing the 43 gestures that we had.

C. Lighting Simulation

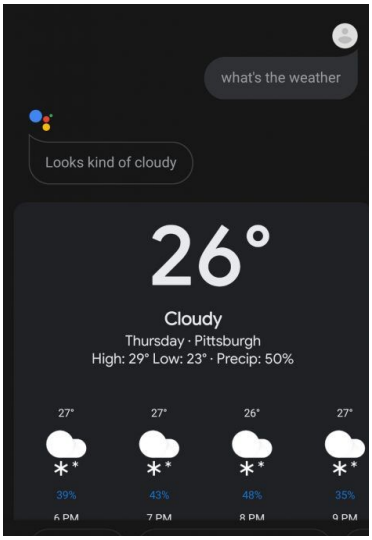
Given the need to test for different lighting conditions, and the difficulty in getting data from different real-world lighting conditions, we opted to simulate different lighting conditions. The method we chose was gamma correction. Humans perceive light non-linearly, as double the light is only a bit brighter. Gamma correction translates the sensitivity of human eyes with the camera. By using OpenCV implementing the following equation: O=I^(1/G), where I is in the input image and g is the gamma value, we can adjust the image to appear lighter or brighter. By using lower values of G we darken the image, and by using higher values we brighten the images.[12]

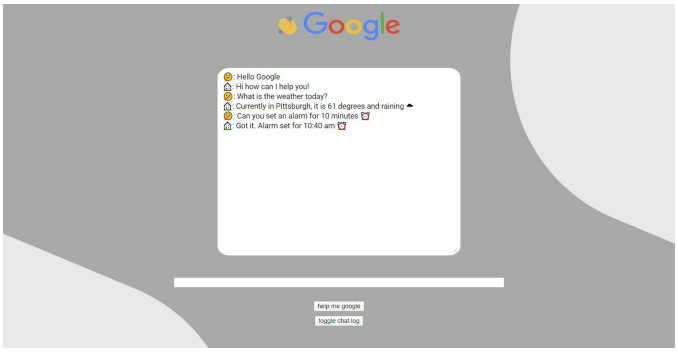


This allowed us to simulate different lighting conditions, well encompassing standard indoor lighting conditions.

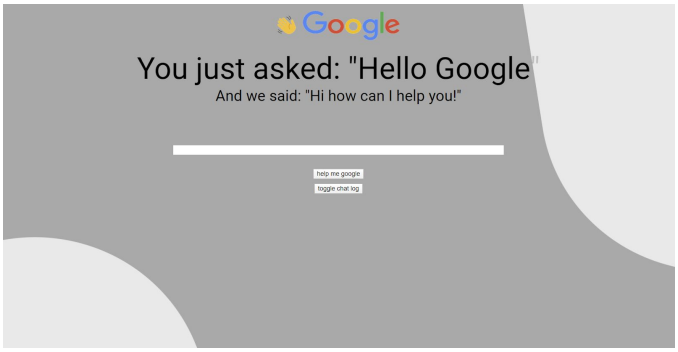
D. Web Application

For the web application, we choose to build on the Django web framework. Using a web framework rather than building from scratch, greatly improves efficiency reducing the amount of repetitive code needed. The web application would be designed to emulate Google Assistant's UI, displaying visual information to the user about its query. For our target use case of the deaf and mute community, this would be essential as without it there would be no means of relaying the information. Following Google Assistant's UI provides a degree of familiarity and ease of use in its chat based approach, where it is easy to see the query followed by the response.





In addition, given that the user could be meters from the screen, we also allowed a mode for the user to quickly and easily see the last query and response from afar.



Given this format it is important for the web application to be asynchronous and update without refreshing the entire page. Django itself runs synchronously, but by also using Django channels, we are able to greatly increase functionality. Django channels work by creating a fully asynchronous layer underneath the core synchronous parts of Django, extending its functionality beyond simple HTTP requests, such as POST and GET. Simply relying on HTTP requests would not provide the real time updates we would need. By adding an asynchronous layer, we can handle connections and sockets asynchronous. This would allow us to use simple Web Sockets to communicate with the web application. Django channels are based on the specification Asynchronous Server Gateway Interface (ASGI), and Django channels provide Consumers an abstraction that allows easy handling of connections and simplifies ASGI applications. The Jetson Nano would then be able to run a simple Web Socket client that would connect with the web applications using the built in websocket library [13]. This would provide a simple and easy connection, as well as allow for easy scaling with multiple smart home devices capable of connecting. The channel layer would use Redis as its backing store, given that Redis is in-memory making it optimal for this use case, being fast and lightweight. This along with the fact that Redis is often used in conjunction with Django along with more documentation makes it an optimal backing store compared to some other solutions like Memcached. Another possible backup would be using the Stream API, which also handles web connections and is often used for real time feeds and chats, similar to our application. However, given the relatively simple nature of applications where there are very few connections, using the default Django channels is sufficient, with Stream API being overkill. Finally, Postgres DB will be used for persistent store of the web application data such as queries and responses.

To deploy our web application, we aim to use AWS Elastic Beanstalk as mentioned earlier. Elastic Beanstalk offers a simpler approach to hosting our web application, as it helps to provision and manage the infrastructure supporting the web application for us such as the EC2 instance and a Postgres instance. This eliminates a lot of unnecessary overhead not essential to running a simple web application. In addition, through the use of Django channels and the resulting channel layer necessary, we decided to deploy the Redis channel layer

in a t2.micro EC2 instance layer. We choose these in particular, as it offers a simple low-cost means of deploying our web application, especially given our AWS credit, and we do not anticipate the EC2 instance would not be very powerful to support our application. Furthermore, by only using AWS for both OpenPose and deploying web applications we help to simplify our approach.

VIII. Results

A. *Gesture Accuracy*

Gesture Accuracy was measured the following way. For each gesture, 20 images of that gesture were chosen randomly to be the testing data. We then tested it with our classifier. Initially, we tested without having any specific gesture classifiers to deal with confusions.

Gesture	Accuracy	Gesture	Accuracy
A	100%	B	80%
C	100%	D	100%
E	100%	F	100%
G	100%	H	100%
I	100%	J	100%
K	100%	L	100%
M	100%	N	100%
O	100%	P	100%
Q	100%	R	95%
S	100%	T	100%
U	100%	V	100%
W	100%	X	100%
Y	100%	Z	100%
0	100%	1	100%
2	100%	3	100%
4	100%	5	100%
6	100%	7	100%
8	100%	9	100%
OK Google	100%	What's the Weather	100%
Set Alarm	90%	Cancel Alarm	100%
Set Timer	100%	Cancel Timer	100%
Stop	100%	Average	99%

With the gestures that had misclassifications, the following confusion matrix was created

True Gesture	Misclassified Gesture	Misclassified Rate
B	F	20%
R	X	5%
Set Alarm	P	10%

After training these individual gestures separately, we made 3 additional classifiers to distinguish between the pairs mentioned above. With this added change, we were able to reduce the misclassified rate slightly, but not remove it completely.

True Gesture	Misclassified Gesture	Misclassified Rate
B	F	0%
R	X	5%
Set Alarm	P	10%

B. *Real Time Gesture Accuracy*

While the former was measuring the accuracy of our classifier itself, Real Time Gesture Accuracy was measuring the accuracy of the total project. An input stream of 1000 images was used.

Real Time Gesture Accuracy	97.5%
OpenPose not processing image	2.5%

Thus, the accuracy of our project was 97.5% which was well above our desired target of 80%.

C. *Real Time Gesture Speed*

Real Time Gesture Speed is the time it takes for a gesture to be classified. This was measured by having a stream of images, and timing how long it took for our classifier to test the images. On average, we achieved 0.47 seconds to classify an image to a gesture.

D. Lighting Requirements

With lighting requirements, we mainly used simulation to test the different lighting requirements. By varying the gamma we used from the gamma correction from .5 to 5, we were able to simulate a variety of lighting conditions, having a higher tolerance for brighter conditions. Outside those ranges, OpenPose was only able to process the image 50% of the time. Unfortunately, we were not able to quantitatively measure the lux levels, due to our lux meter application being highly inconsistent and not matching what we expected for standard indoor lighting. However, qualitative analysis of our pictures taken at various indoor lightings were encompassed by our simulated lighting images.

E. Distance Requirements

With distance requirements, we checked the range we had to be of the camera to have the gestures recognized. We tested 50 images with the same gesture from 1 meter, 2 meter, 3 meter away from the camera. The result we measured was whether or not OpenPose would process the images, as if OpenPose was not able to process the images, we would not be able to feature extract on the result.

Distance	OpenPose Process Rate
1 meter	96%
2 meters	90%
3 meters	22%

Because we compressed our image to decrease the process time of OpenPose, from 3 meter onwards, the image wasn't clear enough for OpenPose to recognize all the hand joints needed. This was something we sacrificed to decrease the total speed.

F. Command Accuracy

Command accuracy describes the overall accuracy from gesture recognition to command parsing. Due to a limited testing sample (i.e. no live testing with people of different physical attributes), most of the testing on this metric was done through simulation. Therefore, we were able to provide tests where the user behaved exactly within the parameters we defined, which may not necessarily be the case when given real users.

Given that the tests are set up based on how we modeled the user's behavior to be, we were able to achieve 100% accuracy *based on what was received from OpenPose.* Thus, the overall command accuracy we achieved is the same as the image classification accuracy.

G. Command Speed

The command speed refers to the time it takes between the first sign being gestured to the camera and the command response, minus the time it took to finish gesturing all the signs for the entire command. Basically, this metric finds the delay between the completion of the command and the response time.

Due to the parallel nature of sending the images while also reading them, getting input from OpenPose is relatively fast and adds negligible time to the command (it varies depending on the length of the command). However, given a spread of gestures ranging from 1 gesture only (built-in commands) to 25 gestures (custom query), we found the average delay between the completion of the query and the delivery of the information to be around 3.2 seconds.

IX. Project Management

A. Schedule

The schedule is included as a figure. Some milestones are getting both machine learning mechanisms to work and incorporated into one process, getting full (not necessarily correct) commands recognized, and finally being able to query and display the results on the screen.

Overall, the stages are split into collection, implementation, and testing. Collection involves setting up the hardware and getting the data necessary. Implementation is where the proprietary software is written. Finally, testing and optimization is where integration and tweaking happens, and where we collect the final data on how well the system performs.

Due to extenuating circumstances this semester, we had to split the project into more distinct chunks and we redid our schedule for the second half of the semester accordingly. While most tasks and delegations remained the same, we had to change some of the integration and testing tasks in order to make the project doable for all team members. See figures 5, 6, 7 for schedules for Jeffrey, Claire, and Sung respectively.

B. Team Member Responsibilities

Sung and Jeffrey are in charge of the machine learning aspect of the project. Sung is working on OpenPose and the AWS setup associated with it, along with the SVM classifier. Jeffrey assisted in helping optimize OpenPose as well as experimenting with OpenCV and NN approaches. Sung also helped to optimize the SVM with confusion matrices.

Aside from machine learning, Claire is working on physical hardware. That includes choosing the right peripherals for the system, determining which system to use, and also coding all the modules that go on the CPU of the Nvidia Jetson. This includes interacting with the Google Assistant and the forming recognized gestures into coherent commands.

Jeffrey is also in charge of the web application, so he will be setting up the AWS server for that and coordinating with Claire on how to display results from the Google Assistant SDK module onto the web application, which will be

displayed on a screen.

Claire is also in charge of writing scripts for validation. She would be in charge of generating tests and verifying them against the output of the overall system.

C. Budget

The bill of materials is included in figure 3. Some materials are included as no cost because the team acquired them for free. Note the Amazon gift card was offered as a gift to all participants who submitted photos of themselves making gestures for training and testing our machine learning algorithm.

D. Risk Management

In terms of personnel, for both hardware and machine learning, there are at least two members who are fairly knowledgeable about these areas. Namely, Sung and Jeffrey are familiar with machine learning, and we all have some experience with embedded systems. This combined with a good deal of reserve budget, allowed us to manage a lot of the risks involved.

The biggest risk was with the gesture recognition algorithm. To minimize risk, we had chosen a proven feature extraction algorithm OpenPose to run as well as chosen an EC2 instance with GPU support known to be able to run OpenPose successfully. We also had the choice to switch to physical hardware such as an Nvidia TK1 or Xavier if possible, which was ultimately unnecessary. However, there are some risks associated with our results as we still had a relatively small sample size of users of only ten people. We attempted to mitigate this by adjusting images by several degrees to generate more data to train and test. However, there still might be some skewed accuracy from a small testing set.

Finally, most of the tasks we scheduled can be done in parallel, meaning if one minor task is unsuccessful another one can be picked up to make up for time later. The way the schedule is designed allows each member room to adjust their own timeline without affecting things progress overall. This proved greatly useful with the switch to fully remote capstone, and being forced to work separately as our schedule allowed for us to work mostly independently from another.

X. Related Work

From previous papers, there has been work done on real time gesture recognition. In particular, projects have been done on the Jetson Nano as well. Like previously mentioned, a paper had used TSM to implement dynamic gestures with high FPS and low power usage. Another project also on the Jetson Nano, specifically focused on recognizing sign language. After training the model, they then cached the model into the Nano achieving live recognition of up to 60 FPS [14]. Many other papers have implemented gesture tracking, but on more expensive hardware like the Jetson Xavier or very specialized camera hardware like the Microsoft Kinect.

In addition, regarding our design choice of using a glove, other papers have also used gloves to aid in their gesture classification. One paper in particular, used a glove with built in sensors to measure the angle of the fingers to recognize finger spelling, illustrating a common trade off of requiring a glove to increase performance [15].

Finally, there has been work linking gestures with smart home devices. As mentioned before, Google with Project Soli is introducing gesture recognition capabilities with their Pixel 4 phones through the use of radar and dedicated chips. Other companies are also introducing gesture based devices that perform simple smart home functionalities. Chicago based FIBARO offers a wireless gesture control pad with 6 programmable gestures [16]. Furthermore, the Clay SDK has been developed that tracks hand's a position using a regular CMOS camera found in most smartphones and a basic 64 bit processor, and they believe that gestures based input will overtake voice control as a better means of interacting with smart home devices [17].

XI. Summary

A. Future work

Future work would include getting more data, and include samples with different skin tones, as well as more images at various distances from the camera and various different real world lighting conditions instead of simulated. In addition, we would also like to do live video testing with more people, being severely limited by the remote aspect of capstone in doing so. Finally, we would like to do more user testing particularly with our target user base of the deaf and mute communities, hoping to gain valuable insight in their thoughts and continue to refine our gesture list to better fit their needs.

B. Lessons Learned

The first issue we ran into was not knowing the hardware specifications because the hardware wasn't available. We looked up the specifications of the Jetson Nano online, and the GPU capabilities of the Nano made it seem like it was powerful enough to handle OpenPose, but after testing with the Xavier, which has roughly 8 times the GPU capabilities of the Jetson Nano, we realized using the Jetson Nano for OpenPose would be near impossible.

We also realized that OpenPose really limits our range of motion for the user. Because the elbow needs to be present in the picture, we were not able to use the majority of the testing samples we gathered online, as people did not include their elbows in the shot. If we were able to do in-person testing this would have proven to be a big flaw in usability. A normal Google Home is able to detect voice both near and far, but our device is only able to do so at a very specific angle and distance.

XII. Amazon Web Services

Amazon Web Services (AWS) proved to be very useful for our capstone project. We used AWS for our Django web application through Elastic Beanstalk, a secondary helper t2.micro EC2 instance to serve as a channel layer to complete with the web application functionality, and finally a p2.xlarge EC2 instance to run OpenPose, the feature extractor, and our support vector machine, the gesture classifier. The GPU on the p2.xlarge EC2 instance greatly sped up our performance compared to running locally without a GPU or on the Jetson Nano with its much limited GPU. We also throughout the process experimented with different EC2 instances sizes, before settling on the instances above. All in all we used 75 dollars in credits, after the initial 50 dollar free credits provided from a new account.

Thank you so much Amazon for your generous contribution, without which our capstone project would not be possible.

References

[1] K. Alt, "16 Smart Home Statistics & Predictions | Safe Smart Living," *Safe Smart Living*. [Online]. Available: https://www.safesmartliving.com/smart-home/statistics-and-predictions/. [Accessed: 27-Feb-2020]

[2] "Rating The Smarts of The Digital Personal Assistants in 2019 | Perficient Digital | Perficient Digital Agency." [Online]. Available: https://www.perficientdigital.com/insights/our-research/digital-personal-assistants-study. [Accessed: 27-Feb-2020]

[3] "Project Soli- Google ATAP." [Online]. Available: https://atap.google.com/soli/. [Accessed: 27-Feb-2020]

[4] "Kinect - Windows app development." [Online]. Available: https://developer.microsoft.com/en-us/windows/kinect/. [Accessed: 27-Feb-2020]

[5] "Illuminance - Recommended Light Level." [Online]. Available: https://www.engineeringtoolbox.com/light-level-rooms-d_708.html. [Accessed: 02-Mar-2020]

[6] "NAHB: Spaces in New Homes." [Online]. Available: https://nahbclassic.org/generic.aspx?genericContentID=216616. [Accessed: 02-Mar-2020]

[7] J. Nielsen, "Response time limits: Article by jakob nielsen," *URL http://www. nngroup. com/articles/response-times-3-important-limits*, 2010 [Online]. Available: https://www.nngroup.com/articles/response-times-3-important-limits/

[8] "How Fast Does the Average Person Speak? - Word Counter Blog," *Word Counter Blog*, 02-Jun-2016. [Online]. Available: https://wordcounter.net/blog/2016/06/02/101702_how-fast-average-person-speaks.html. [Accessed: 02-Mar-2020]

[9] Arm Ltd, "Cortex-A57 – Arm Developer," *ARM Developer*. [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-a/cortex-a57. [Accessed: 01-Mar-2020]

[10] "Maxwell Architecture," *NVIDIA Developer*, 21-Feb-2014. [Online]. Available: https://developer.nvidia.com/maxwell-compute-architecture. [Accessed: 01-Mar-2020]

[11] "1/2.5, 5MP NVIDIA Jetson Nano Camera Board." [Online]. Available: https://www.e-consystems.com/nvidia-cameras/jetson-nano-cameras/5mp-mipi-nano-camera.asp. [Accessed: 02-Mar-2020]

[12] A. Rosebrock, "OpenCV Gamma Correction - PyImageSearch," *PyImageSearch*, 05-Oct-2015. [Online]. Available: https://www.pyimagesearch.com/2015/10/05/opencv-gamma-correction/. [Accessed: 06-May-2020]

[13] "Django Channels — Channels 2.4.0 documentation." [Online]. Available: https://channels.readthedocs.io/en/latest/. [Accessed: 02-Mar-2020]

[14] "Cloud Machine Learning: Train in the Cloud, Deploy at the Edge." [Online]. Available: http://blog.faucher.net/2019/06/cloud-machine-learning-train-in-cloud.html. [Accessed: 02-Mar-2020]

[15] C. Oz and M. C. Leu, "Recognition of Finger Spelling of American Sign Language with Artificial Neural Network Using Position/Orientation Sensors and Data Glove," in *Advances in Neural Networks – ISNN 2005*, 2005, pp. 157–164, doi: 10.1007/11427445_25 [Online]. Available: http://dx.doi.org/10.1007/11427445_25

[16] /author/lauren-shanesy, "Are Gesture-Control Devices the Next Big Thing in Home Tech?," *Builder*, 27-Oct-2016. [Online]. Available: https://www.builderonline.com/products/home-technology/are-gesture-control-devices-the-next-big-thing-in-home-tech_o. [Accessed: 02-Mar-2020]

[17] E. H. Contributor, "Next Big Smart Home Trend: Gesture Control of Lights, Thermostats, and Other Devices? - Electronic House," *Electronic House*, 05-Sep-2017. [Online]. Available: https://www.electronichouse.com/smart-home/next-big-smart-home-trend-gesture-control-of-lights-thermostats-and-other-devices/. [Accessed: 02-Mar-2020]

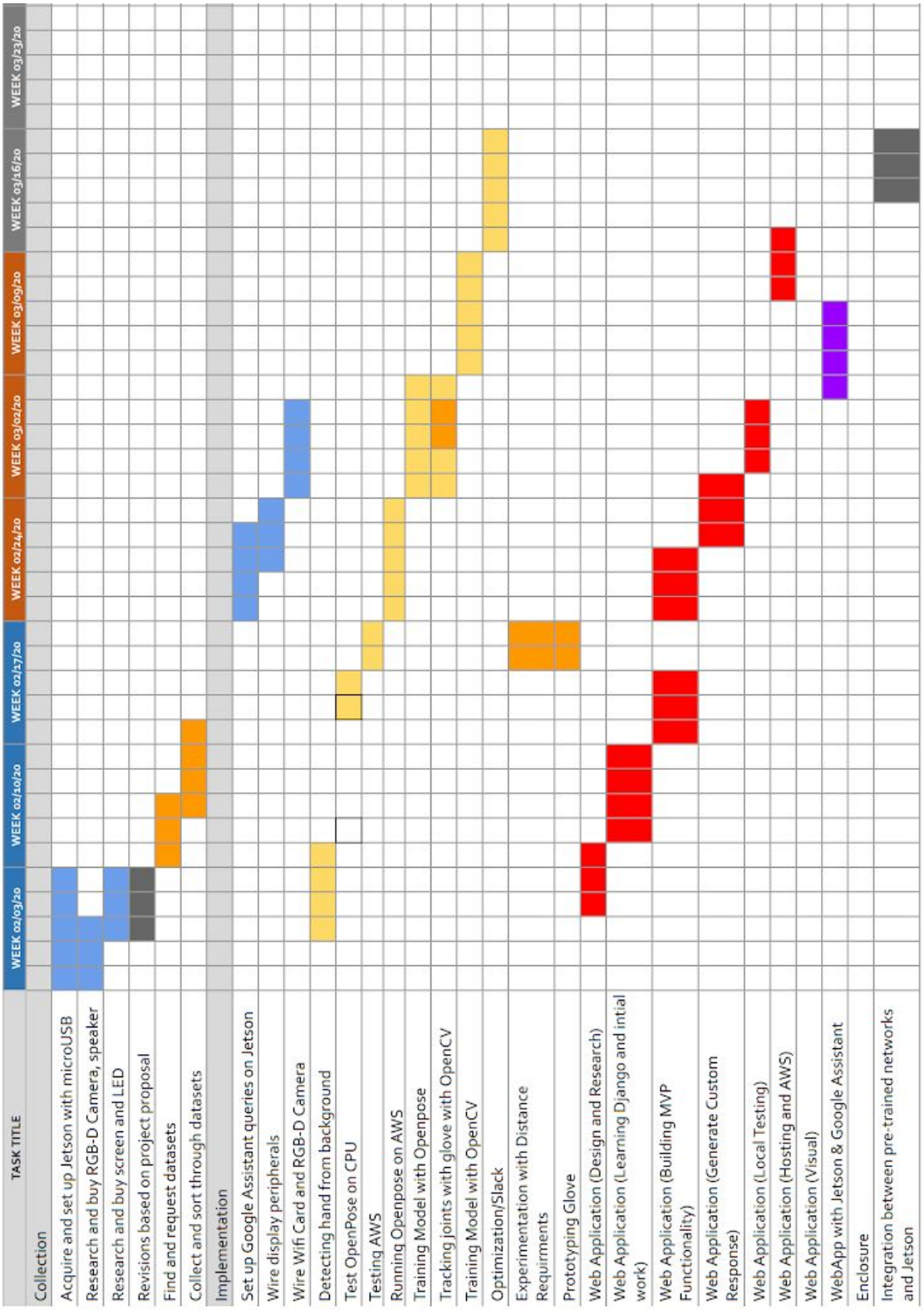


Figure 1. Collection and Implementation stage Gantt charts.

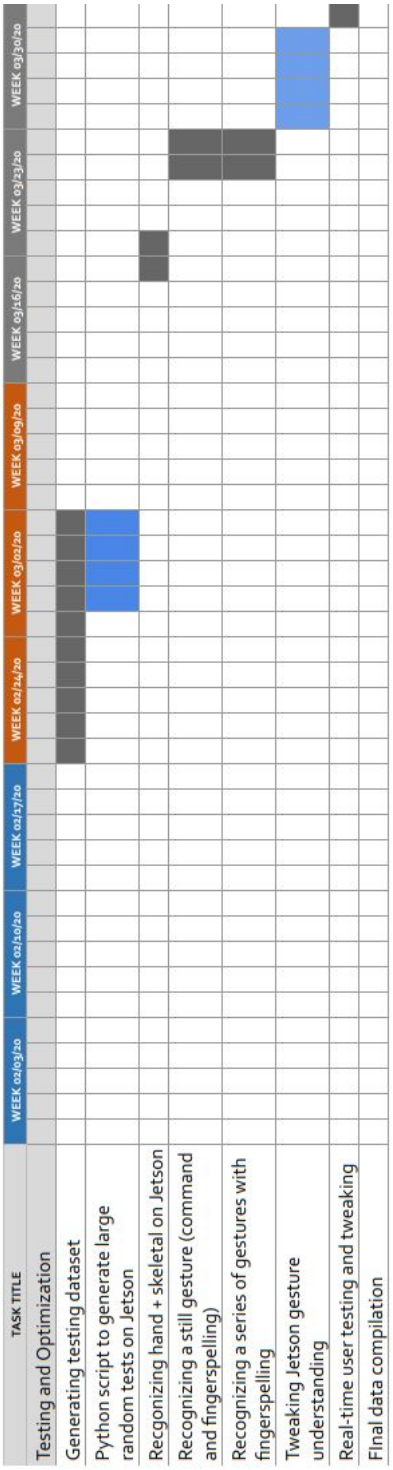


Figure 2. Testing and Optimization stage Gantt chart.

Name	Number	Cost	Used	Provider/Source
Nvidia Jetson Nano	2	99.99	Yes (only 1)	ECE Department + Amazon
Lebula Touchscreen Monitor for Raspberry, 7 Inch 1024X600 with Dual Speaker Small USB Monitor	1	72.99	Yes	Amazon
Samsung 128GB 100MB/s (U3) MicroSDXC EVO Select Memory Card with Full-Size Adapter	2	44.98	Yes	Amazon
e-CAM50_CUNANO - 5.0 MP NVIDIA® Jetson Nano™ Camera	1.00	124.00	Yes	e-con Systems
Intel WiFi Wireless-Access Point 8265 8265.NGWMG.DTX1 Dual Band Desktop Kit	1	30.29	Yes	Amazon
HDMI Monitor	1	0	No	ECE Lab
Mouse	1	0	Yes	ECE Lab
Keyboard	1	12.99	Yes	Amazon
AWS	1	0	Yes	Capstone AWS credit
Amazon Gift Card (raffle prize for data)	1	100	Yes	Amazon
Latex Glove	1	0	No	N/A
Green Tape for Glove Markers	1	4.99	No	Amazon
EDE0001 8 Mega Pixel Digital Camera Terasic P/N: D8M-GPIO	1	0	No	ECE Lending
TP-Link USB Wifi Adapter for PC N150 Wireless Network Adapter for Desktop - Nano Size Wifi Dongle	1	9.99	No	Amazon
		500.22		

Figure 3. BOM

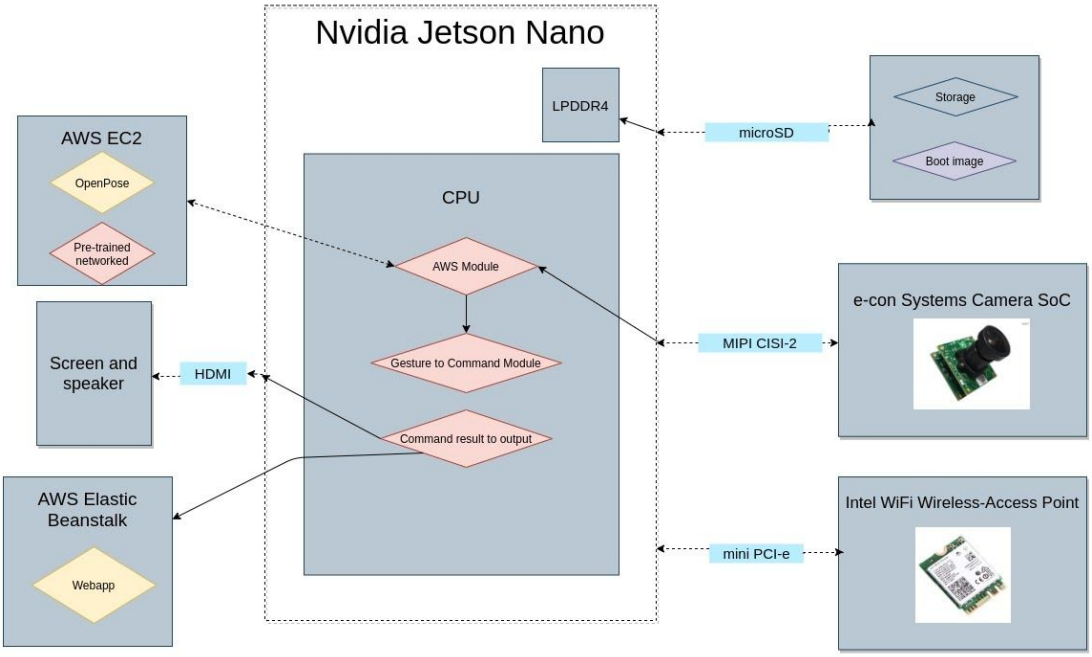


Figure 4. Hardware system chart.

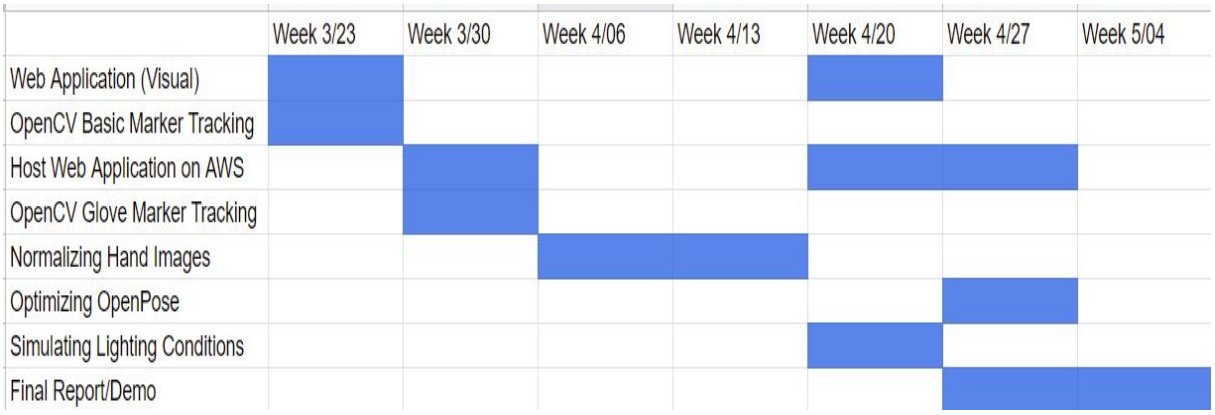


Figure 5. Sung refocus Gantt chart.

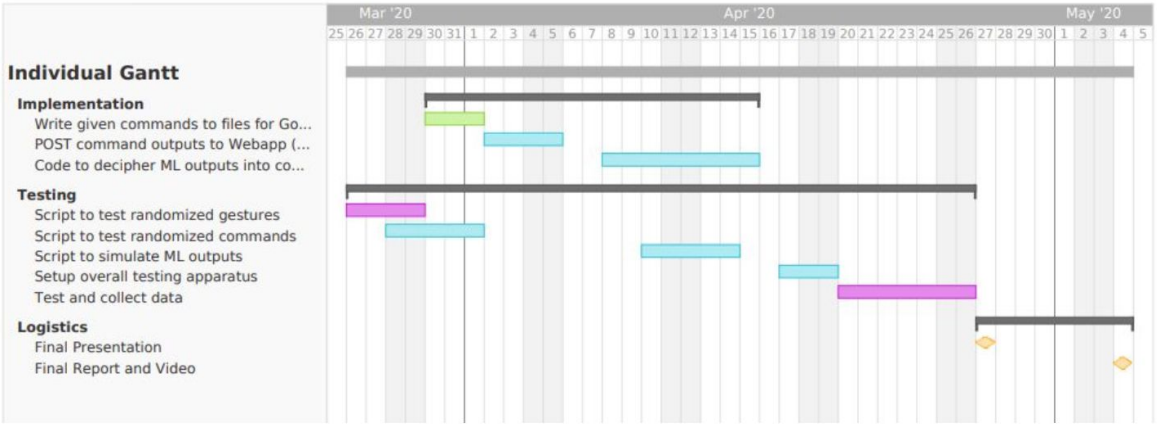


Figure 6. Claire refocus Gantt chart.

TASK TITLE	START DATE	DUE DATE	WEEK 03/23/20	WEEK 03/30/20	WEEK 04/06/20	WEEK 04/13/20
OpenPose						
Collecting Data	3/23/2020	3/30/20				
Finding Examples of Pre-trained Neural Network	3/30/2020	4/1/20				
Working with Pre-trained Neural Network with gestures	4/1/2020	4/6/20				
Making Neural Network from scratch without Pre-trained Neural Network	4/1/2020	4/6/20				
Optimization	4/6/2020	4/9/20				
Writing frame dividing script with backups	4/8/2020	4/11/20				

Figure 7. Jeff refocus Gantt chart.