CV Studio

Authors: Tony Lu, Chris Ng, Mark Prettyman
Electrical and Computer Engineering, Carnegie Mellon University

Abstract—**A system developed to use the body in music production. Users are able to play sounds using motions such as stomps, claps and hits. Furthermore, the user is able to make basic edits on a recorded loop such as quantizing notes to a grid. Latency-critical gestures that are used to play music are classified with simple time-distance-threshold algorithms while editing gestures are classified with image classifiers for greater accuracy. The system is supplemented with a glove with a button on each fingers to give extra functionality to music creation gestures; for example, when the user stomps while holding a button down, the kick drum that plays is different. We use OpenPose to identify joints, which are then be fed into our gesture recognition algorithms.**

Index Terms—**Computer Vision, Gesture Recognition, Action Recognition, Music Production, OpenPose, Pose Estimation**

1 INTRODUCTION

Currently, music production is mostly done through digital audio workstations (DAWs), allowing users to record and edit sounds. Most of this work is done using a mouse and keyboard. Most times, loops are recorded using a MIDI keyboard or entered in using the mouse; however, this input only requires usage of one's hands. CV Studio aims to allow music producers to create input using body movements and allow for editing of already recorded work. Current products that are similar to CV Studio are the MI-MU gloves, which contain sensors that track hand movements [5]. The advantage of CV Studio is that it supports movement of the feet as well as the hands. Furthermore, the Mu.Mi gloves are intended for music performance rather than production.

CV Studio must be able to recognize gestures in less that 80ms for music playing related gestures (stomp, clap, hit). CV Studio must also maintain a 90% accuracy rate of recognizing gestures with a 0% false positive rate. It must function within a reasonable distance from the camera (1.5 - 2.5 m).

2 DESIGN REQUIREMENTS

We have split the design requirements into five components: latency, accuracy, distance, glove functionality, and ease of use.

2.1 Latency

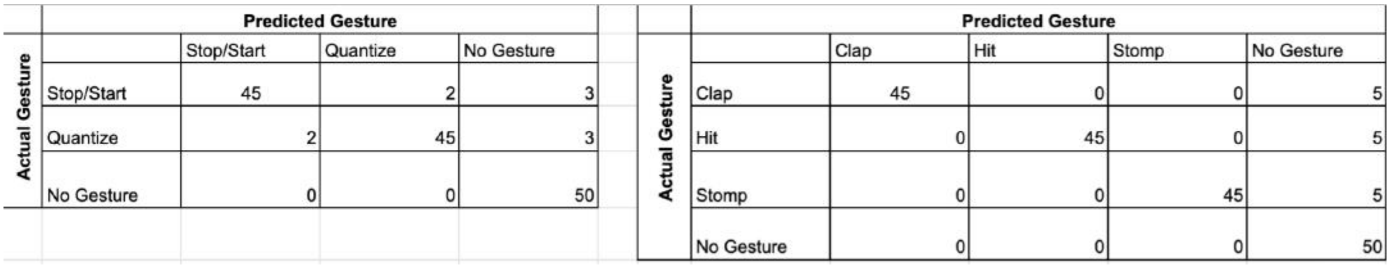
The success of CV Studio relies on its end to end latency of music creation gestures. Users should not feel hindered in any way using CV Studio versus using a MIDI controller to control input. We have split up our requirements of latency into two categories: music creation gestures and editing gestures.

What we call music creation gestures consist of gestures that are mapped to sounds in the studio software. For example, when the user claps, they expect a snare drum to play through the speaker and when the user stomps, they will expect a kick drum to play. We are aiming for a latency of 80 ms or less from the time the gesture is made until the sound is played from the speaker. While there can be noticeable latency for a performer at even 10 ms [10], we were experimenting with a MIDI controller and increasing latency until we could not play on the beat. This time was 80 ms. Since CV Studio is not intended for performances or concerts, we do not need to achieve 10 ms latency. Furthermore, we wanted to make sure that even if the speaker was off and the user wanted to only make gestures without audio feedback that the note would be off by at most a 1/16th note at 150 beats per minute (bpm), which is a fast tempo and should include most types of music users will make [9]. In time signatures with a denominator of 4 (which are very common in music [11]), the quarter note represents a beat, so the 16th note represents a quarter of a beat. This means $\frac{150 \text{ beats}}{1 \text{ minute}} \cdot \frac{4 \text{ 1/16th notes}}{1 \text{ beat}} \cdot \frac{1 \text{ minute}}{60 \text{ seconds}} = \frac{10 \text{ 1/16th notes}}{1 \text{ second}}$. Thus each 1/16th note takes $\frac{1}{10}$ second = 100 ms. Choosing 80 ms made sense because it hits this threshold.

For editing gestures, which consist of stop, start, record and quantize, we relax the requirement to 200 ms. We chose 200 ms because that is when users begin to notice a delay and feel that their commands are being processed [7]. Although this research was done for web applications, we believe our editing gestures have a similar function and should therefore use similar latency constraints.

2.2 Accuracy

For the accuracy of CV Studio, we are aiming for a high rate of true-positive recognition of gestures, a low rate of false-negatives and a very low rate of false-positives. While it may be annoying for the user to perform a gesture that goes unrecognized, CV Studio will be unusable if the user does not perform a gesture and the software executes a command. Furthermore, incorrect classification of a gesture will be quite intolerable since this could put unwanted sounds into a loop or could stop playing a loop unexpectedly. We tolerate at most a 10% false-negative rate, since a user could re-execute the gesture. We tolerate a 5% misclassification rate between quantize and stop/start as there may be some issues with feature extraction and these are not the most critical gestures to make accurate. The con-



		Predicted Gesture		
		Stop/Start	Quantize	No Gesture
Actual Gesture	Stop/Start	45	2	3
	Quantize	2	45	3
	No Gesture	0	0	50

		Predicted Gesture			
		Clap	Hit	Stomp	No Gesture
Actual Gesture	Clap	45	0	0	5
	Hit	0	45	0	5
	Stomp	0	0	45	5
	No Gesture	0	0	0	50

Figure 1: Confusion matrix describing accuracy goals. Note that for each classifier, no gesture also refers to gestures classified in the other classifier.

fusion matrix in Figure 1 describes our accuracy goals.

2.3 Distance

The requirement of distance is that a user can stand from the camera at a reasonable distance while CV Studio retains full functionality. We chose 1.5 m - 2.5 m for usable distance as this will have the body in full frame of the camera, while allowing a few steps of freedom of movement for the user away from the camera. Furthermore, as a user is farther away, the accuracy of pose estimation decreases, making it harder to detect gestures.

2.4 Ease of Use

CV Studio should be easy and intuitive to use. The glove should not be too restrictive and the users should be able to pick up on gestures. We want 75% of users to find CV Studio easy to pick up, rating the intuitiveness as positive or negative and overall satisfaction positive or negative.

3 ARCHITECTURE OVERVIEW

The main components of our design are the glove, a Raspberry Pi Zero, a webcam, an Nvidia Jetson AGX Xavier, and a laptop running Ableton Live.

The buttons on the glove are connected to GPIO pins on the RPi Zero, which are then connected via Bluetooth to the laptop. The webcam is connected to the Xavier via USB and the Xavier connected to the laptop via USB-C. Figure 2 showcases the overall architecture.

4 DESIGN TRADE STUDIES

4.1 OpenPose Latency on Xavier

Running OpenPose without detecting finger keypoints on the Xavier achieves roughly 27 fps and with finger keypoints runs at approximately 10 fps. With our classification algorithms, this will likely be reduced. To achieve the latency requirement of 80 ms, the Xavier with our classification algorithms needs to run at at least $1000/80 \approx 13$ fps.

Initially, we had planned on using finger keypoints and the cosine distances between them as features to classify the stop/start and quantize gesture. Unfortunately, with finger keypoints turned on, the frame rate was too low to reach our latency requirements for the music creation gestures.

Although we planned to implement neural networks to classify all gestures, the latency requirement for music creation gestures and constrained computing resources forced us to resort to simpler approaches for classification. Thus, implemented simpler time-distance thresholding algorithms for the music creation gestures, while retaining neural-networks for the editing gestures.

4.2 Clap, Hit, Stomp Classification Accuracy

4.2.1 LSTM

The music gestures were first classified with an LSTM. The features used were the joint coordinates from OpenPose, centered on the middle of the hip and normalized so that the distance from the middle of the hip to the neck was 1. We appended or truncated frames to a gesture sequence if it was not equal to the median length, which was 23 frames. However, the accuracy of the LSTMs was below 40%, far less than what we required. We also tried removing unnecessary features, such as keypoints in the face, and preprocessing our data so that the joint coordinates were turned into normalized vectors from one joint to the next. Although this decreased training times for the models, this did not improve accuracy.

4.2.2 Spatial-only Algorithm

After trying LSTMs, we resorted to a simpler approach that only used spatial properties of the gestures. We computed the distance between relevant joints for each gesture: for claps, the distance between wrist keypoints; for stomps, the vertical distance between ankle keypoints; and for hits, the cosine distance at the elbow. If it exceeded a tuned threshold, we would detect a gesture. The user would then have to hit a reset threshold that would allow the gesture to be detected again. This approach had very low latency and reasonable accuracy, although it did not meet our goal.

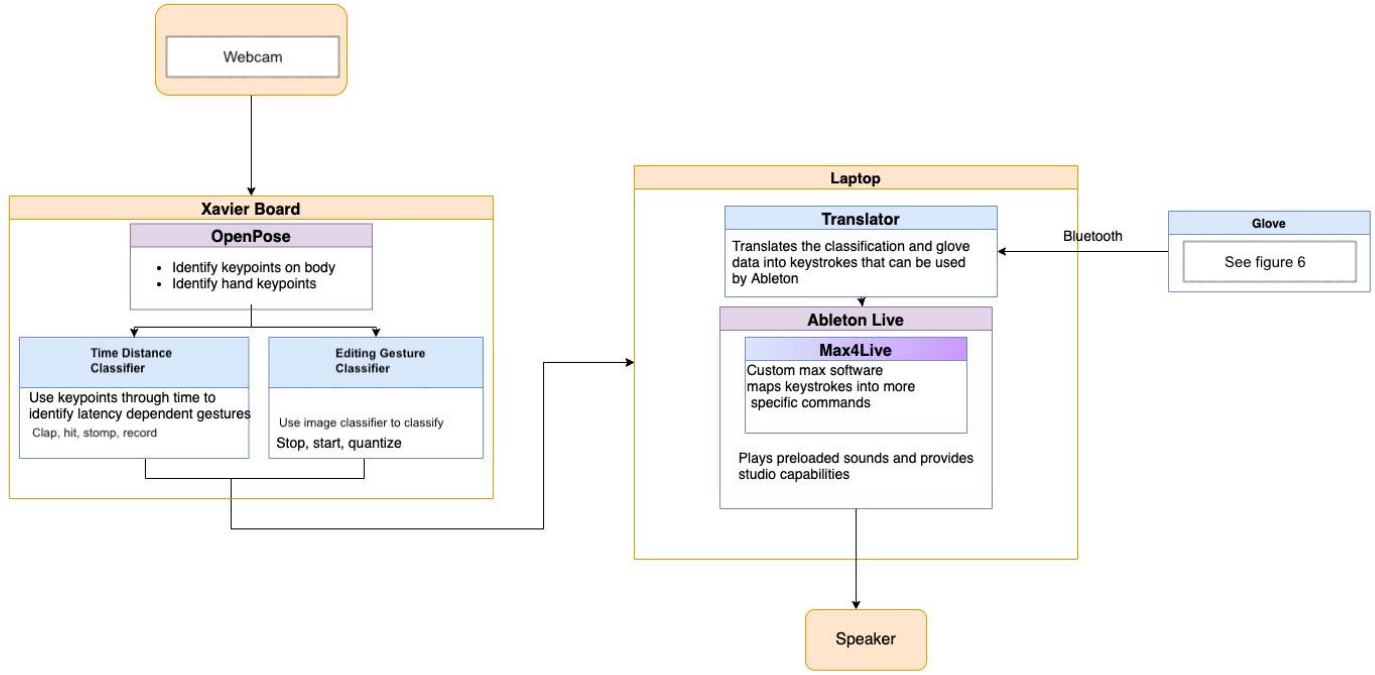

Webcam
Xavier Board
OpenPose
Identify keypoints on body
Identify hand keypoints
Time Distance Classifier
Use keypoints through time to identify latency dependent gestures
Clap, hit, stomp, record
Editing Gesture Classifier
Use image classifier to classify
Stop, start, quantize
Laptop
Translator
Translates the classification and glove data into keystrokes that can be used by Ableton
Bluetooth
Glove
See figure 6
Ableton Live
Max4Live
Custom max software maps keystrokes into more specific commands
Plays preloaded sounds and provides studio capabilities
Speaker

Figure 2: The architecture of the entire system. Orange blocks represent borrowed or purchased parts, blue blocks represent parts we created, and purple blocks represent software that we downloaded. Note that Max4Live is be downloaded but is modified it for our purposes, so it is both blue and purple.

4.3 Stop/Start, Quantize Classification Accuracy

4.3.1 LSTM

The first attempt to classify the stop/start and quantize gestures was to use an LSTM. The features used for this attempt was a convex hull drawn around a normalized mask of the hand of the user, containing far fewer points than trying to use an LSTM to classify a series of images; however, there were some struggles in getting consistent features to pass to the model, resulting in frequent false-negatives. For example, if for a few frames within a stop/start sequence, the mask around the hand was not extracted correctly, the LSTM would classify that as no gesture. The accuracy using LSTM was around 55%. Under highly controlled environments, the recognition accuracy could reach about 70%, both well under acceptable accuracy requirements. These tests were run under non-real time video, and we did not collect latency metrics.

4.3.2 Image Classification

The more successful version of classifying gestures was to do framewise image classification. Instead of passing a convex hull into a classifier, we passed the entire mask as an image into a keras image classifier[2]. For each image, we were able to achieve a very high accuracy of 97%. In order verify whether a gesture was actually performed, a series of 6 frames were classified and if more than half of them were classified as a gesture, that would be the predicted gesture.

5 SYSTEM DESCRIPTION

Figure 2 shows the architecture of CV Studio. Below we describe each component in more detail.

5.1 Xavier Board

The Xavier is the component running OpenPose and the classifiers. The board is connected to the laptop and the webcam through separate USB cables. Information about classifier outputs are passed to the laptop through the USB, acting as a network connection.

5.2 OpenPose

The joint detection algorithm is done by CMU's OpenPose [4]. OpenPose takes in input from the webcam connected to the Xavier and output the main body joint positions. We are not using the finger keypoints or the face keypoints. We ran OpenPose through its Python API.

5.3 Gesture Classification

The gesture classification system is split up between the classification system for the time-distance gestures (clap, stomp, hit) and the gestures for editing (stop/start, quantize). We had initially planned to use a record gesture as well, but this was dropped.

5.3.1 Time-Distance Gesture Classifier

This gesture classifier only relies on major key joint positions, meaning it will not take in any facial or finger keypoints. We use a simple classification method that tracks the important joints for each gesture and determine whether or not they are within the threshold for the gesture to be made.

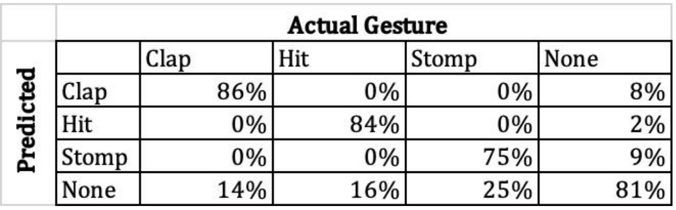
First, the classifier normalizes distances between keypoints based on the distance between the neck and waist keypoint in each frame in order to adjust for different heights and distances from the camera.

For the clap gesture, we take into account the positions of both wrist joints. If they move within a certain threshold, we know that that is the clap gesture. We must take into account that OpenPose will relay that the hands are still moving even if they are not due its slight inaccuracy. We also do not want to output more than one clap as it is being detected. This means that we have to determine when a clap is released. This is done using another threshold, with the distance being a little higher than the original threshold to determine the gesture. These thresholds can be set without having to take into account true distances as the joints positions have been normalized.

For the stomp gesture we determine if the ankles are far enough in their y-coordinates. If this reset threshold has been met, the next time the user brings their ankles within a small enough vertical distance will classified as a stomp.

The hit gesture, which is a gesture that is the user pretending to hit a drum off to the side, will be classified by finding when the arm is fully extended to the side after the forearm is moved up and back down. We calculate the angle between the upper arm and forearm to determine if there is a 180 degree angle (with some margin for error) and then when it retracts we will be able to reset the gesture.

The reasoning for these simple methods of determining these music playing gestures is that they are fast. Since we do not rely on temporal data, we need only one frame to classify when a gesture has broken a threshold. Gestures can also be classified simultaneously, giving the user more freedom in using CV Studio.



		Actual Gesture			
		Clap	Hit	Stomp	None
Predicted	Clap	86%	0%	0%	8%
	Hit	0%	84%	0%	2%
	Stomp	0%	0%	75%	9%
	None	14%	16%	25%	81%

Figure 3: The actual confusion matrix of the time-distance gestures. There were 186 clap samples, 163 stomp samples, 158 hit samples, and 139 no gesture samples.

5.3.2 Editing Gesture Classification

The editing gesture classifier classifies the gestures of stop/start and quantize. The gestures are as follows: stop/start will be a closed fist into palm facing forward and quantize will be a thumbs down.

The image classifier is briefly described in section 4.3.2; We use the Keras image classifier.

Feature extraction of gestures consisted of a binary mask around the right hand of the user, as the left hand would be the one wearing the glove. To extract this mask, first, the OpenPose keypoint of the right hand would be used to track the hand location, then a cropped image would be extracted from the frame. From there, the hand is extracted by extracting the HSV range of skin using OpenCV accounting for noise using a Gaussian blur. We then draw a contour around the hand, normalizing the size and location of the contour. We then fill in the contour with white to get a binary mask around the hand. Please refer to figure 4.

Training is done by manually labeling samples of the hand in the final position of the gesture, for the start/stop it would be the open palm and for the quantize the thumbs down. We trained multiple models on a large set of data and then saved the most accurate one to use in the project to classify. Figure 8 shows a flow chart of the training and classification of the gestures.

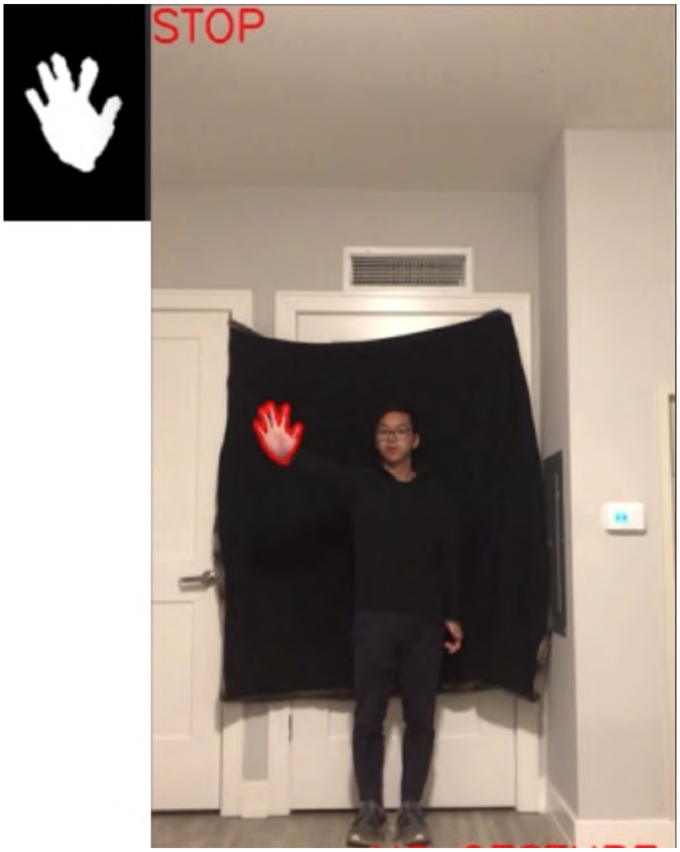
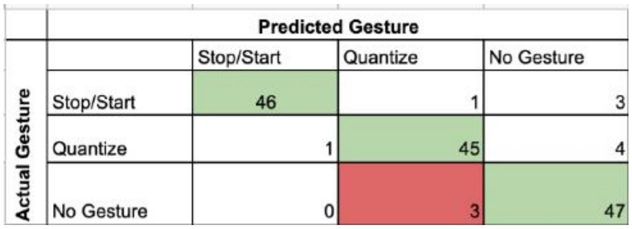


Figure 4: Feature extraction



		Predicted Gesture		
		Stop/Start	Quantize	No Gesture
Actual Gesture	Stop/Start	46	1	3
	Quantize	1	45	4
	No Gesture	0	3	47

Figure 5: Actual confusion matrix of editing gestures run on 50 samples each

5.4 Glove

A system diagram for the glove component is shown in Figure 6. The glove operated as a wireless accompaniment to gestures created by the user, allowing for the user to press buttons on the glove while performing music related gestures, increasing the potential number of instruments and sounds by a factor of four. Four buttons were be wired directly to GPIO pins on a Raspberry PI Zero W. The Pi Zero was chosen due to its portability and ability to process button inputs and easily relay the results to the laptop via a Bluetooth connection. In the figure, GPIO pins 2, 3, 4, and 14, are called out. These pins were chosen because they are in close proximity to each other on the Pi Zero and also are conveniently located next to ground pins. Since the Raspberry Pi Zero requires 5 volts to operate, we used a portable Pi-specific lithium battery module known as the Pi Sugar. This supplied enough power to run the glove component for 6 hours and was easily rechargeable. In the initial design, the glove was intended to have two layers, so the raspberry pi could lay between the layers of gloves and provide a cleaner look. However, the gloves purchased ended up not stretchy enough to accommodate the pi and pi sugar package, so just one layer of glove was used. Buttons were soldered to wire and connected directly to the Pi Zero. The buttons were glued to the area near the fingertips and the Pi Sugar, which was connected to the Pi through four small screws, was glued to the back of the glove. It was considered unnecessary to design a PCB to achieve this purpose, as the complexity of this part of the project does not warrant a custom PCB.

5.4.1 Glove Integration

After completing the assembly of the glove, it was moved into an initial testing phase, where a python script verified each button press. Connections on the first iteration of the glove were generating inconsistent results, as the middle and ring finger buttons were shorting out. Upon reworking the glove and button connections, the hardware was improved so that button presses were recognized and reported almost flawlessly. Due to the nature of the soft tactile buttons purchased, it was difficult to tell whether a press and non-report was a result of not triggering the switch or some other error. As a next step of integration,

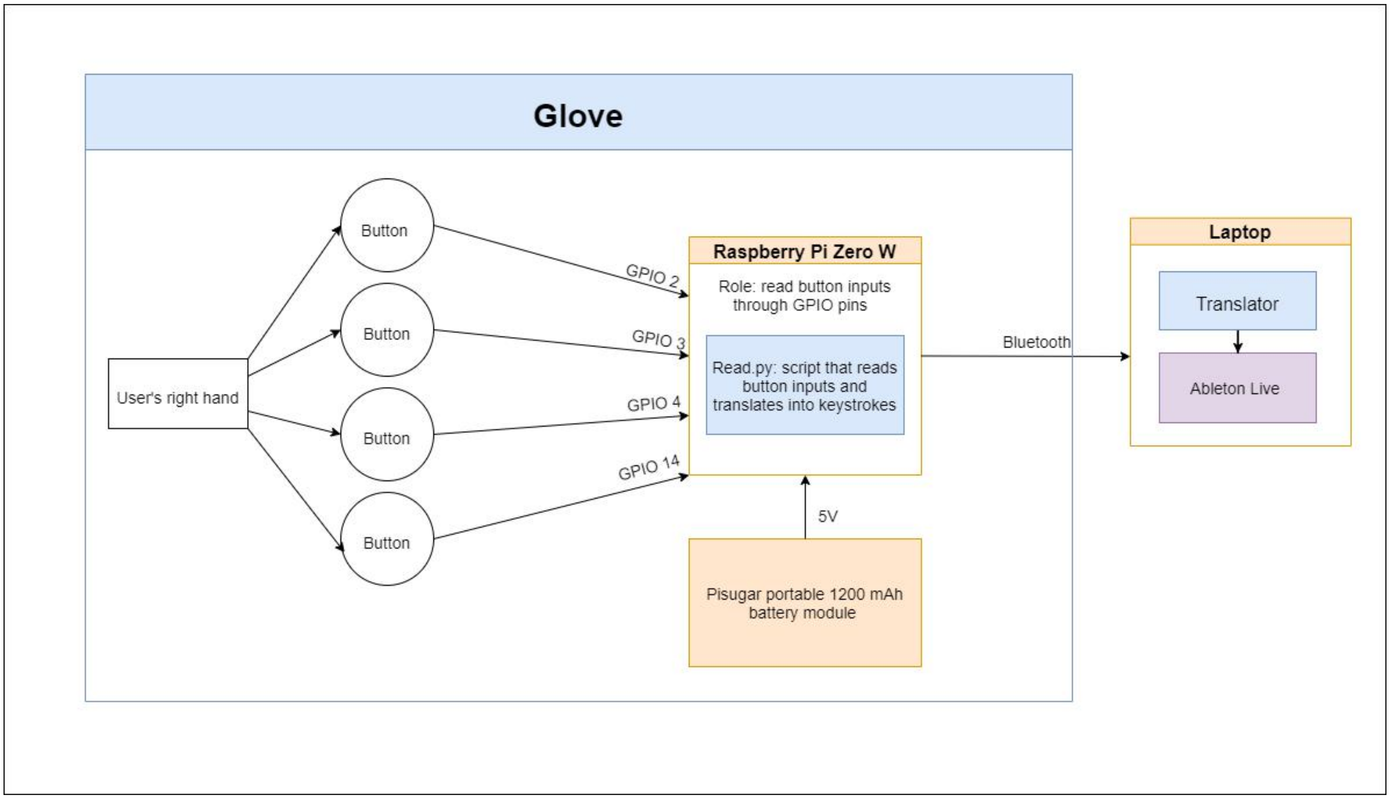
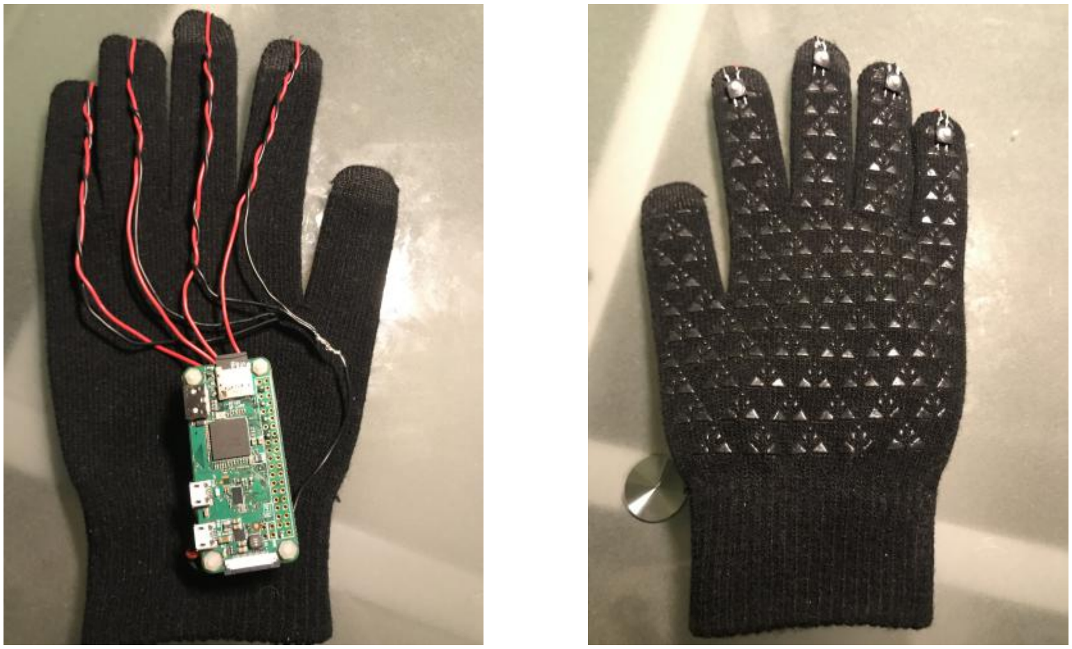

Glove
User's right hand
Button
Button
Button
Button
GPIO 2
GPIO 3
GPIO 4
GPIO 14
Raspberry Pi Zero W
Role: read button inputs through GPIO pins
Read.py: script that reads button inputs and translates into keystrokes
5V
Pisugar portable 1200 mAh battery module
Bluetooth
Laptop
Translator
Ableton Live

Figure 6: The architecture of the glove. Orange blocks represent borrowed or purchased parts, blue blocks represent parts we created, and purple blocks represent software that we downloaded.



(a) Pi side of glove

(b) Buttoned side of glove

Figure 7: Glove component, 2nd iteration assembly.

the buttons were associated with wav file sound samples, which were played through a Bluetooth speaker connected to the Raspberry Pi upon press. This intermediate step was used for testing purposes only prior to the connection of the glove to a laptop. The next phase of glove integration consisted of sending button press info through Bluetooth connection to a laptop, as Ableton ran on one of the team members machines. This was achieved through establishing a socket Bluetooth connection between the pi and the laptop, and sending data to the laptop indicating when any button is pressed. The translator acts as a server and receives incoming data from the pi and our classifier, and outputs the corresponding MIDI notes to Ableton. On the pi side, button presses are only allowed once per second, as the program waits for a final output from Ableton before accepting another press. This constraint was deemed to be reasonable as the time it takes for users to perform a gesture and hear the output is in the range of one second [3].

5.5 Translator

The translator component of our project is one that converts the classification output and the glove data into data readable by Ableton. For the music creation gestures, we use RTMidi, a Python package to output MIDI data directly into Ableton, which reads our translator as a MIDI input. Luckily, there is a standard in how drum kits, the sounds we use in Ableton are mapped to notes on a keyboard, allowing the user to swap different drum kits without having to remap what MIDI notes our translator outputs.

5.6 Ableton Live

Ableton Live is a full digital audio workstation meant for music production and performance. Although we are aiming for production, we utilize the ”Live” component of Ableton live, the performance element to create loops to edit which can then be moved to the production side. Ableton allows us to quantize notes easily with custom settings and to edit clips easily.

5.6.1 Max4Live

Since the classifier outputs keystrokes, we needed a way to convert these keystrokes into Ableton Commands. For example, in order to quantize notes in Ableton, the default method is to double click on a clip, select all the notes, right click, and click on quantize. We didn't want to implement a script that would move the mouse around and do mouse commands so we chose Max4Live, which allows us to create commands within Ableton that does this for us, for all gestures.

6 PROJECT MANAGEMENT

6.1 Schedule

Please refer to the schedules at the end of the document, which contain our original and updated schedule after remote instruction was issued. Although we have left a few weeks of slack and extensive time for testing the system om the original schedule, delays due to the pandemic caused us to use all of that time.

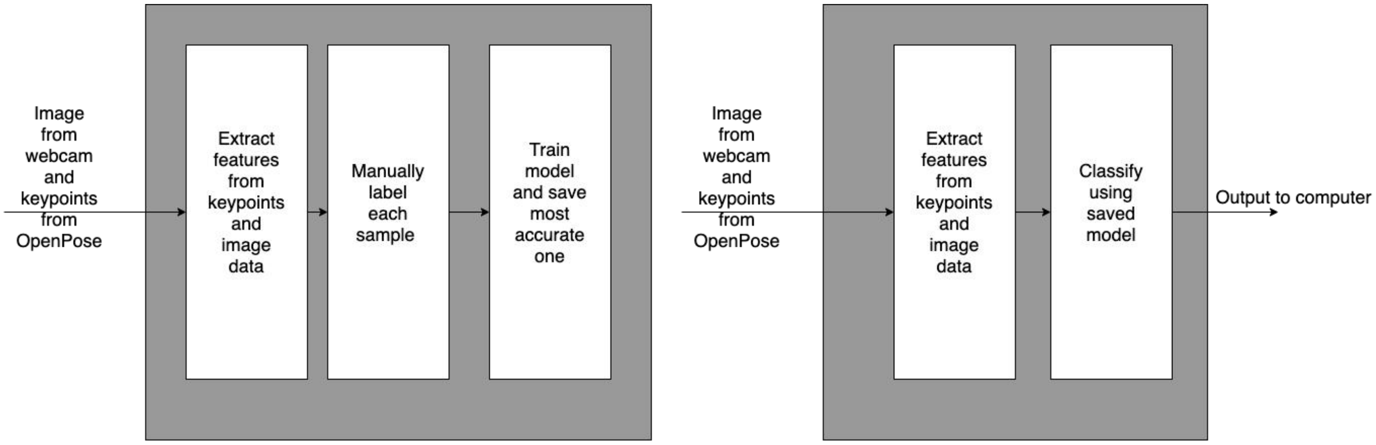


Image from webcam and keypoints from OpenPose
Extract features from keypoints and image data
Manually label each sample
Train model and save most accurate one
Image from webcam and keypoints from OpenPose
Extract features from keypoints and image data
Classify using saved model
Output to computer

Figure 8: Flow diagram describing training of model and classification of gestures.

6.2 Team Member Responsibilities

The classification algorithms were worked on by Chris and Tony, splitting the work up into classifying different gestures, while collaborating on the algorithms themselves to be consistent across the project.

Chris worked on classifying the quantize and stop/start gesture, the ones using image classification and Tony worked on the music creation gestures as well as the record gesture using the Openpose keypoints with time and distance relationships, although the record gesture was dropped.

The glove was designed and built entirely by Mark with decisions about parts and compatibility with the whole system being made by the entire team.

Testing for each component was done by the member who was responsible for that component. For Tony's gestures, the tests were done on videos that were processed on Openpose whereas for Chris' gestures, the tests were done on real time video. This was solely due to the restriction that the Xavier board was in Chris' possession after the class went online.

The software category primarily refers to the integration from data from the classification and the glove into Ableton Live. The Max4Live system will be written by Chris since he has the most experience with Ableton Live. The translation of data into Max4Live readable data will be done by Mark and Tony, with Mark and Tony being responsible for glove data and classifier data respectively.

6.3 Budget and Parts

A bill of materials (BOM) is displayed at the end of this document in Figure 9. The group was allocated approximately 600 dollars to spend on this project. However, the more expensive components in our project are borrowed from the ECE department rather than bought, so they are not contributing to our overall costs. It is important to note that the costs displayed in Figure 9 are round number estimates, and fall slightly over the raw costs for the parts, as shipping costs and taxes were included. Additionally, some parts like wire and soldering equipment are provided through the various ECE lab spaces in Hammerschlag Hall.

6.4 Risk Management

In the design document, we identified many potential risks and ways to mitigate them. The most important was balancing latency and accuracy in the classification algorithms and reducing latency on the Xavier.

6.4.1 OpenPose Latency

We ran OpenPose with finger keypoints enabled at an average of 10 frames per second on the Xavier board, which posed an issue because this did not meet our latency requirement of 80 ms. To remedy this, we sped up OpenPose by reducing resolution as far as we could while keeping the accuracy above requirements for the editing gestures. Another solution to this problem may have been to attempt to use the Windows Kinect SDK for pose estimation. This may be less accurate than OpenPose but may also run at a higher frame rate. Given more time, we may also have branched out to other lightweight implementations of pose estimation that can run more quickly [1] [8].

6.4.2 Classification Latency

Implementing LSTMs proved to be difficult. Since none of the LSTMs we used met our accuracy goals, we did not have to optimize our system to account for their latency. Instead, the simple spatial-only algorithm for the music creation gestures allowed us to achieve our latency goal. For the editing gestures, the framewise image classification likely has less latency than a LSTM.

6.4.3 Classification Accuracy

Although we initially aimed for no false positives, this proved to be ambitious. Since we were forced to reduce OpenPose's resolution and resort to simpler classification schemes to reduce latency, we could not uphold our original goal of detecting no false positives.

6.4.4 Difficult Gestures

Difficult gestures refers to gestures that are essential to the success of the project yet are difficult to classify. More concretely, this refers to claps, stomps, and hits that are more subtly performed. For example, a user clapping fast will not separate their hands very much and similarly someone trying to stomp fast will not lift their feet very far off the ground. Our solution was to not classify these gestures and require the user to make more sweeping versions of these gestures for accurate classification. This meant increasing reset thresholds so that the user had to, for hits, fully extend the elbow; for stomps, lift their heel high enough off the ground before stomping; for claps, bringing their palms far enough apart before clapping. However, this sacrificed ease of use and restricted the way the user could operate.

6.5 Bluetooth Latency

Since the glove is connected to the laptop via Bluetooth, there is a risk that the input from the glove will not be registered in time to supplement the output from our classifier. However, due to us working from home in the second half of the semester, we did not integrate the glove with the other components of the system and.

7 RELATED WORK

One system that is similar to CV Studio is GECO [6]. This attempts to provide MIDI expression to DAWs to control the feeling of a sound. This is a bit different from our work to provide a studio setting for our users.

Another system, perhaps the most complete, is the Mi.Mu gloves developed out of London in collaboration with the famed Imogen Heap [5]. A very impressive system that is integrated well into Ableton Live, it is a well proven system to make music with gestures. Some advantages that CV Studio offers over these is the ability to use more percussive elements and to utilize the entire body as input rather than just hand motion.

8 SUMMARY

Unfortunately, our system was not able to reach all of our specifications. False positives were very difficult to eliminate, especially on the time-distance gestures. In order to improve performance of OpenPose to get very low latency on time-distance, we decreased the net-resolution of OpenPose, reducing the accuracy of joint detection, meaning the thresholds we set would not be consistent for every time the gesture was used. If we continued with this project, we would revise our thresholding to track more keypoints that are interrelated with each gesture. For the image classified gestures, we were able to hit our accuracy requirement of 90%, achieving 92% accuracy; however, there were a few false positives. This is mainly due to the limitations of our feature extraction method. When moving quickly, accuracy of the mask around the hand was quite limited due to motion blur, resulting in masks that may look similar to those of our gestures. With more time, we would change the feature extraction algorithm completely to a convolutional-neural network to get much more accurate masks. Lastly, we were not able to collect any ease of use data due to COVID-19.

Through this process, we learned quite a bit about computer vision systems that would attempt to recognize gestures. First off, gathering data should be one of the first things done. Because we tested on a limited set of data, we missed some cases that should be accounted for. Second, if using machine learning, it is critical to have good and consistent features to pass to the classifier. Much of our error on the image classification side was not due to the classifier itself but rather the way we extracted features. Lastly, make sure that there are at least basic functional components working in the whole system before trying to make each one perfect. This way, testing the system as a whole is much easier. Furthermore, the way components fit together is already in place so that when refining parts, code will not have to be rewritten at the end to fit everything together.

8.1 AWS Credit Usage

AWS credits were used to speed up processing the videos we took before we received the Xavier. Videos were taken on our personal devices, transferred to AWS and pose information returned and processed using our classification algorithms. This was used to test our algorithms on non real-time videos since our machines were not powerful enough to run OpenPose at an acceptable frame rate. We accidentally used all 100$ of credits allocated from two team members due to confusion about how to deactivate instances. We did not deposit the 50$ worth of credits from the third team member.

Thank you to AWS Education and to the whole capstone staff for helping us acquire these credits.

References

[1] https://github.com/Daniil-Osokin/lightweight-human-pose-estimation.pytorch. Accessed: 2020-03-02.

[2] Francois Chollet. *Building powerful image classification models using very little data*. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html. Accessed: 2020-03-23.

[3] *Communication with Bluetooth*. http://www.python-exemplary.com/index_en.php?inhalt_links=navigation_en.inc.php&inhalt_mitte=raspi/en/bluetooth.inc.php. Accessed: 2020-04-01.

[4] CMU Perceptual Computing. *OpenPose*. `https://github.com/CMU-Perceptual-Computing-Lab/openpose`. Accessed: 2020-03-02.

[5] *Mi.Mu Gloves*. `https://mimugloves.com`. Accessed: 2020-03-02.

[6] Leap Motion. *Geco MIDI*. `http://uwyn.com/geco/`. Accessed: 2020-03-02.

[7] Jakob Nielsen. *Usability Engineering*. Jan. 1993. URL: `https://www.nngroup.com/articles/response-times-3-important-limits/`.

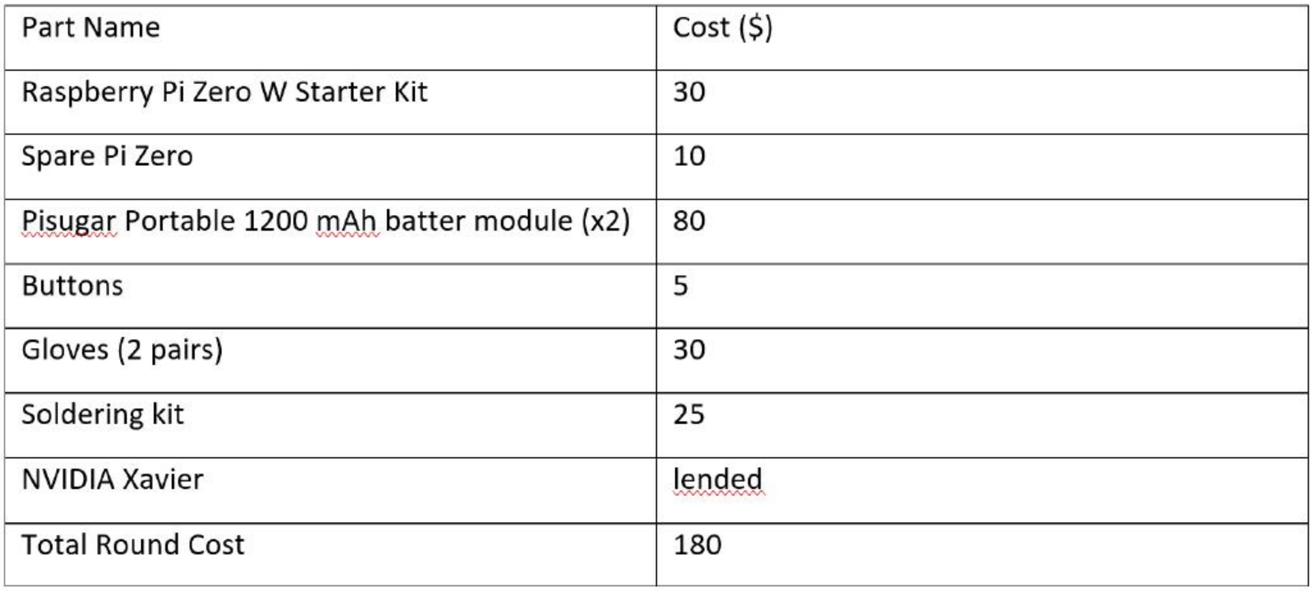
[8] Daniil Osokin. `https://github.com/Daniil-Osokin/gccpm-look-into-person-cvpr19.pytorch`. Accessed: 2020-03-02.

[9] Amanda Storey. *What Are Beats Per Minute and What Can BPM Tell You?* `https://blog.storyblocks.com/tutorials/what-are-beats-per-minute-bpm/`. Accessed: 2020-03-02.

[10] Andy Swanson. *Latency and Its Effect on Performers*. `https://www.churchproduction.com/education/latency-and-its-affect-on-performers/`. Accessed: 2020-03-02.

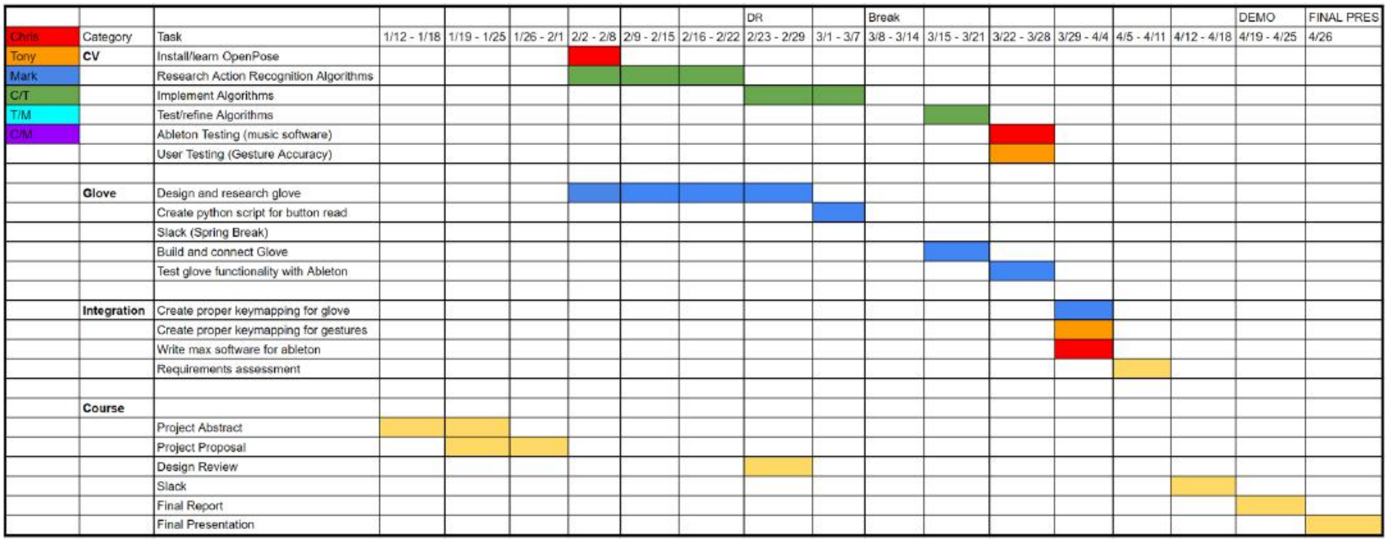
[11] *Which are the 15 most common time signatures?* `https://music.stackexchange.com/questions/14415/which-are-the-15-most-common-time-signatures`. Accessed: 2020-03-02.

Appendix A



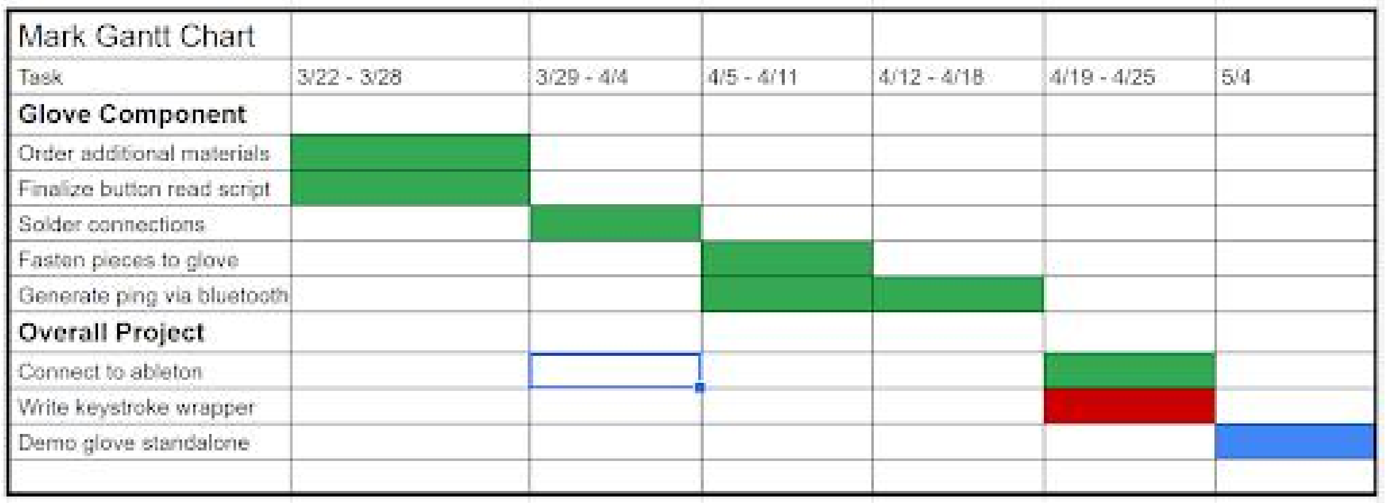
Part Name	Cost ($)
Raspberry Pi Zero W Starter Kit	30
Spare Pi Zero	10
Pisugar Portable 1200 mAh batter module (x2)	80
Buttons	5
Gloves (2 pairs)	30
Soldering kit	25
NVIDIA Xavier	lended
Total Round Cost	180

Figure 9: Bill of materials including costs.



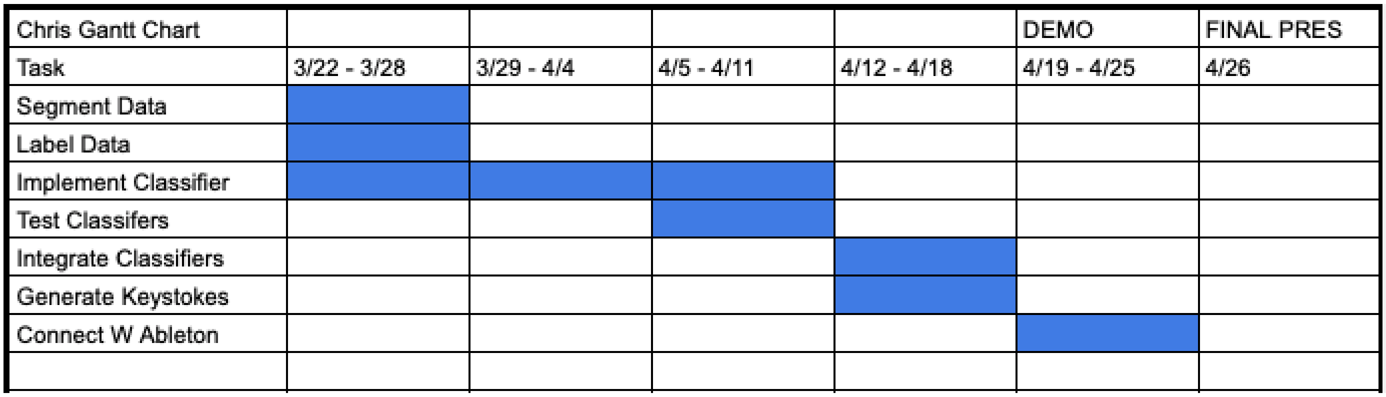
									DR		Break						DEMO	FINAL PRES
🟥 Chris	Category	Task	1/12 - 1/18	1/19 - 1/25	1/26 - 2/1	2/2 - 2/8	2/9 - 2/15	2/16 - 2/22	2/23 - 2/29	3/1 - 3/7	3/8 - 3/14	3/15 - 3/21	3/22 - 3/28	3/29 - 4/4	4/5 - 4/11	4/12 - 4/18	4/19 - 4/25	4/26
🟧 Tony	CV	Install/learn OpenPose				🟥												
🟦 Mark		Research Action Recognition Algorithms				🟩	🟩	🟩										
🟩 C/T		Implement Algorithms							🟩	🟩								
🟦 T/M		Test/refine Algorithms										🟩						
🟪 C/M		Ableton Testing (music software)											🟥					
		User Testing (Gesture Accuracy)											🟧					
	Glove	Design and research glove				🟦	🟦	🟦	🟦									
		Create python script for button read								🟦								
		Slack (Spring Break)																
		Build and connect Glove										🟦						
		Test glove functionality with Ableton											🟦					
	Integration	Create proper keymapping for glove												🟦				
		Create proper keymapping for gestures												🟧				
		Write max software for ableton												🟥				
		Requirements assessment													🟨			
	Course																	
		Project Abstract	🟨	🟨														
		Project Proposal		🟨	🟨													
		Design Review							🟨									
		Slack														🟨		
		Final Report															🟨	
		Final Presentation																🟨

Figure 10: Gantt chart displaying original semester schedule.



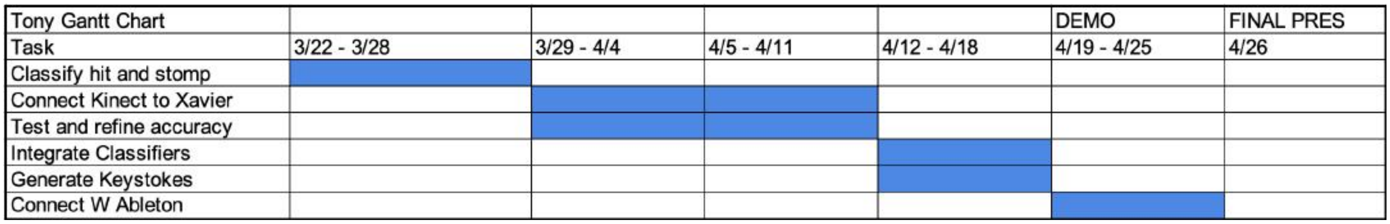
Mark Gantt Chart						
Task	3/22 - 3/28	3/29 - 4/4	4/5 - 4/11	4/12 - 4/18	4/19 - 4/25	5/4
Glove Component						
Order additional materials	🟩					
Finalize button read script	🟩					
Solder connections		🟩				
Fasten pieces to glove			🟩			
Generate ping via bluetooth			🟩	🟩		
Overall Project						
Connect to ableton					🟩	
Write keystroke wrapper					🟥	
Demo glove standalone						🟦

Figure 11: Gantt chart displaying Mark's updated schedule.



Chris Gantt Chart					DEMO	FINAL PRES
Task	3/22 - 3/28	3/29 - 4/4	4/5 - 4/11	4/12 - 4/18	4/19 - 4/25	4/26
Segment Data	🟦					
Label Data	🟦					
Implement Classifier	🟦	🟦	🟦			
Test Classifers			🟦			
Integrate Classifiers				🟦		
Generate Keystokes				🟦		
Connect W Ableton					🟦	

Figure 12: Gantt chart displaying Chris's updated schedule.



Tony Gantt Chart					DEMO	FINAL PRES
Task	3/22 - 3/28	3/29 - 4/4	4/5 - 4/11	4/12 - 4/18	4/19 - 4/25	4/26
Classify hit and stomp	🟦					
Connect Kinect to Xavier		🟦	🟦			
Test and refine accuracy		🟦	🟦			
Integrate Classifiers				🟦		
Generate Keystokes				🟦		
Connect W Ableton					🟦	

Figure 13: Gantt chart displaying Tony's updated schedule.