**TABLE OF CONTENTS**

[**Phase I: Getting used to python: 2**](#_tk2iekexesng)

[**Phase II: Brainstorming ideas… 2**](#_s9oozzafb67b)

[Priority: 2](#_6eoo3qlcymlz)

[Finalized Idea 3](#_mflen9saga6t)

[Project Plan 3](#_oigl6m196kbp)

[**Phase III: The Hand Tracking Model 3**](#_t7di2011ri9v)

[**Phase IV: Deciding What Hardware to Buy/Use: 4**](#_uixhwzwp70l7)

[**General Weekly/Monthly Updates 4**](#_2qa7z2tuol84)

[4/12/24 4](#_9xn6nt2kkjlj)

[4/26/24 5](#_q7hclvqae9dq)

[4/27/24 5](#_xc6m0rpzgnm0)

[5/11/2024 6](#_40p48dv1fk5c)

[5/12/24 6](#_kq9ep848h1jf)

[5/16/24 7](#_2ebyiuz41csx)

[5/17/24 7](#_wuk9tdubrwud)

[**Post-Nationals: Brainstorming for Internationals- 5/26 8**](#_6wt8osjxygq6)

[Week 1- Testing Servos + Middle Octave- 6/3 8](#_9bj0zvx6gs4x)

[Week 2- Ticks- 6/10 9](#_jxj359n8k2y3)

[Weeks 3 & 4- Lights & Performance- 6/17 & 6/24 9](#_77d6gxcryn5r)

[Week 5- Integration + Video/Poster- 7/1 9](#_z4ftl7ahxwn6)

[**Future Enhancements 10**](#_btgcqvizr12y)

# 

# 

# Phase I: Getting used to python:

* we planned on using the Raspberry Pi to power most of our electronics
* all three of our prior experience in programming was mainly in C
  + because C is a low level language, it wasn’t too difficult for us to pick up python
* # is comments, triple quotes (single or double) is for chunk of comments
* floor division is using two slashes for division instead of 1
  + this turns it into an integer instead of a float (in python, numbers automatically become floats in division)
* two stars for exponents (\*\*) instead of using pow function (like double multiplication)
* when there's brackets around an argument in a function, it means they are optional
* instead of else if, python uses elif
* dictionaries are extremely fast and a very good alternative for if-statements
  + dictionaries contain and key and value
  + the key is used to access the value
  + the value can be a tuple, containing multiple things
* variables are initialized differently, a type is not assigned
  + the type is determined based on what is stored in the variable
* return statements are extremely convenient
  + you can return multiple variables of different types simultaneously from a function

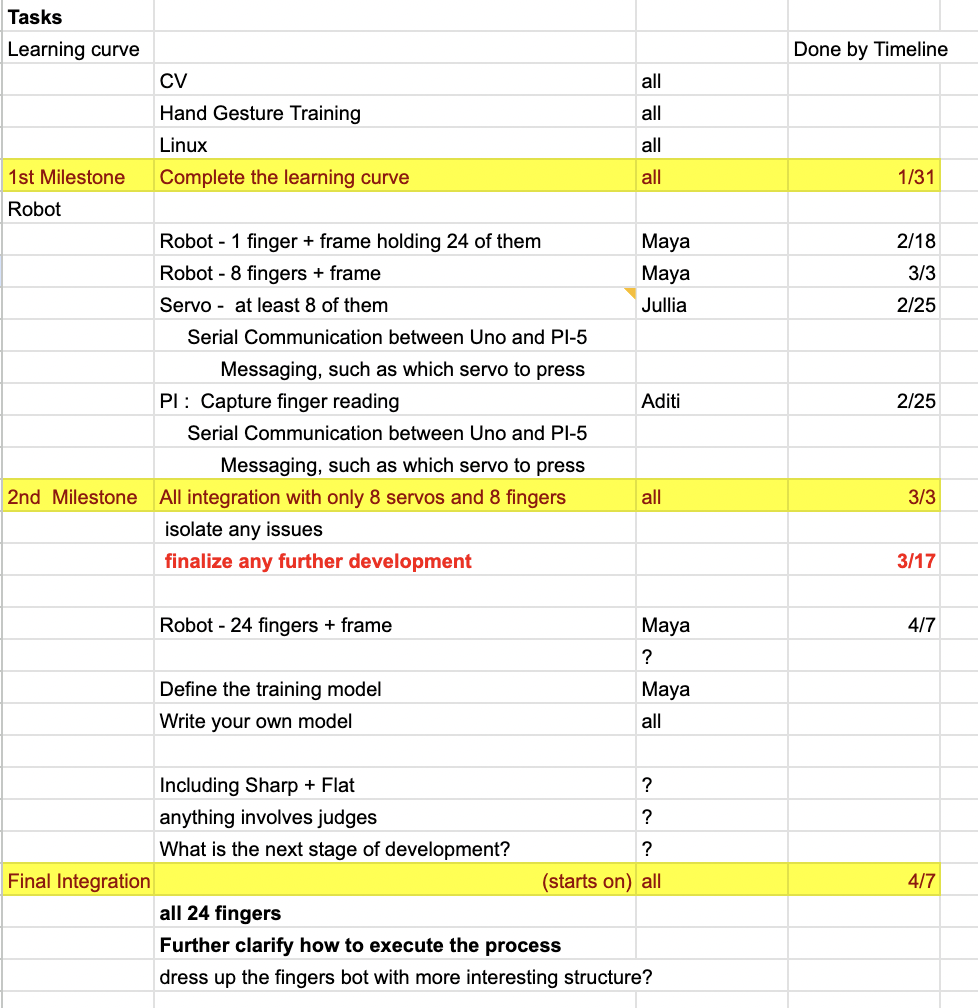
# Phase II: Brainstorming ideas…

## Priority:

* must be practical in the real world
* Entertainment was secondary
* Knew we wanted to include sign language
  + Signing the alphabet?
  + Signing signals?
  + Signing with both hands?
* How we could incorporate entertainment:
  + Robot plays a full song from one signal
  + Play a simple game with the robot
  + Have the robot play an instrument
  + Robot starts a dance from what we sign it

## Finalized Idea

* Robot plays notes on a piano based on the *letters* that we sign it.



## Project Plan

Created a project plan with tasks, assignment to the tasks, and milestones/the date we must finish by.

# Phase III: The Hand Tracking Model

* 4 different parts

1. **collect\_imgs.py**: collects each “frame” of the “video” and stores them into arrays and folders
2. **create\_dataset.py**: takes the array and finds the “landmarks” on each hand (coordinates of where the point/coordinate is), storing them into an array. this array is converted to a dictionary when placed into data.pickle
3. **train\_classifier.py**: takes the dictionary from data.pickle and converts it to a numpy array. uses 80% of the data to train the hand classifier, and the other 20% to test
4. **detect\_and\_send.py (DAS)**/**inference classifier (IC):** DAS will take in an image and check if it is equal or close to one of the signals that it knows (through the training). IC does all of that but also does communication between the pi and arduino.
   1. Typically, we use IC so that we can use serial to see what is printing, and also so that we can incorporate WaitforNote and WaitforPause, etc.

# Phase IV: Deciding What Hardware to Buy/Use:

* needed a portable piano keyboard
* servo motors to move each of the robot “fingers” playing the notes
* 3D printed fingers
* multiple channel servo driver, so that we could hook up multiple
* frame to hold the fingers on top of the keys
* camera to see us signing letters
* microprocessor (raspberry pi) to hook up the camera to
* a microcontroller (arduino uno) that can communicate with the pi, where we

While we were working on the connection to the camera, we created a function to use for testing purposes. Essentially, it hardcodes the data input that is supposedly from the camera.

# General Weekly/Monthly Updates

## 4/12/24

* **Problem**: camera was taking in too many frames at once and therefore taking in too many instances of the character it was supposed to play; the servos were lagging and repeating the note too many times
  + tried adding in a half second delay, but the camera became laggy and hard to signal with because it would only take a “picture” every half a second
* **Brainstorming**: needs a different way to send information over to the arduino
  + option 1: bidirectional communication where the arduino sends back information (ex. “im done playing, send me the next note now”
  + option 2: have a signal to tell the cam that the next note is happening (constantly waiting for this “next” symbol)
  + option 3: cv contour stuff, see colors (hold up piece of paper) means the next note etc.
* **Solution**:
  + Created WaitForNote and WaitForPause
  + Trained camera to only send over one signal, look for the “pause” symbol (an open hand) and then send over the signal that appeared right after the “pause”, and repeat for the duration of the code

## 4/26/24

* **Problem:** realized that to present on stage, we would need extremely long jumper wires, adapters, long wires to power sources, etc.
  + We also realized that our entire hardware could be simplified down to two boards: the Raspberry Pi and PCA9685 servo driver
* **Brainstorming:** 
  + Option 1: We could keep the system the same and just buy the long jumper wires
  + Option 2: Find a way to shrink down our hardware and reduce our code
* **Solution:** we went with option 2, in order to keep everything more compact
* **Problem:** as a result of our decision to remove the Arduino Mega and only use the Raspberry Pi, we realized that all our other code for the Arduino Mega (which was coded in C), would have to be transferred over onto the Raspberry Pi in python
  + None of us 3 had much experience or exposure to the new language, so this posed a new challenge
  + Additionally, we would have to run the servos from the Raspberry Pi as well
* **Solution:**
  + With some help from instructors, google, and each other, we were able to find a library that moved the servos from the Pi, and transfer all the C code over to the Pi
  + Additionally, in order to have easy access to the note letter we had to play and corresponding servo to press, we created a dictionary containing both of those bits of important information
  + Our key for the dictionary was the class number that the signal was classified to be (ex. C = 0, D = 1, E = 2, etc.)
  + The value was a tuple, containing the note value and corresponding servo number (ex. one key value pair of the dictionary would look like (0: [C, servo[1])])

## 4/27/24

* watched videos of RCJ Onstage Videos
* got an entire octave working well with the camera, mega-pi, and servos
* picked what song we wanted to do (you are my sunshine)
* discussed how we are going to present on stage
* hoping to build a camera mount and shelf to hold hardware underneath the servo mount (wooden frame)

## 5/11/2024

* Made layout and content for poster

**1)**

**Problem**

* The camera was lagging due to the time.sleeps we were using to move the servo motors
* So, it was hard to see the display, because the camera would pause displaying frames while the servo motors ran

**Solution**

* Multithreading with the Raspberry Pi’s multi core
* Each character that was classified was sent into a queue, where another thread would grab this character and play the servo from that thread.
* This way, the camera would constantly be displaying frames, and the servo motors wouldn’t lag either

**2)**

**Problem**

* Notes would sometimes be inaccurate, because as we transitioned from one note to another, our hand signals would sometimes be classified as another note

**Solution**

* + Continuously grab most frequently seen note
  + Create a list initialized to 0 with 8 elements
  + When a note was classified, we would increment the value in the list
  + At the end, we would check which character had most occurrences, and send that one into the queue
    - * We also ensure that this character was accessed at least two times

## 5/12/24

* final preparations for competition
* putting together the poster and video

## 5/16/24

**1)**

**Problem**

* Servo accuracy was extremely poor, the fingers were not lining up well

**Solution**

* Hot glued extra extensions that were thinner and could press the keys with much higher and more reliable accuracy
* Downside: we were unable to add the felt covering the piano “fingers” to prevent harsh clicking noises

## 

## 5/17/24

* camera mounts were created
* tried testing the entire project, but many of the servo motors were failing (ex. pressing too much/too little, some broken/not working entirely, others making buzzing noises)
  + spent a lot of time trying to fix these values

**1)**

**Problem**

* realized the entire frame didn’t fit in a minivan
* ended up accidentally breaking a side of the wooden frame off, and breaking a Pi Cam

**Solution**

* + Ended up having to take apart the frame to be able to move it
  + Also had to drive to an in-person shop to buy more Pi-Cams
  + Learned to be more careful when transferring equipment and to always have extra spares on hand

**2)**

**Problem**

* Raspberry Pi was unable to find the I2C address of the servo driver
* Wasted a lot of time trying to solve this issue

**Solution**

* Simply bad wires (everything was plugged in correctly, but the wires that we were using were faulty

# Post-Nationals: Brainstorming for Internationals- 5/26

* Optimized wooden frame
  + hinges
  + folded up
  + transportability
  + covering up the servo wires
* Fix misalignment issue with servos
  + Reprint servo fingers (potentially longer arm)
  + Use the smaller Adafruit servos to our advantage
* Gliss with stepper motor
* Metronome “Ticks”, so that 3 notes can all sync up together, keeping in mind human error
* Lights for extra visual effect
* Character
  + Arnold the alien
  + “Conversation” with Arnold
  + Googly Eyes with 360 degree servo motors
  + line tracing robots on the bottom of the wooden frame
* Better Servo Motors (Adafruit Tower Pros)
* Adding Middle Octave
* Find a speaker for piano so it can be heard without having to hold up a mic
  + just bring a mic stand

### Week 1- Testing Servos + Middle Octave- 6/3

**Problem**

Our servos would constantly break, lock up, or need to be re-calibrated. This made our consistency highly decrease. Damage to the servos was often caused by repeatedly hitting a piano key or hitting the keys too hard.

**Solution**

We decided to run tests to determine the best brand of servo. We tested the Tower Pro SG92R servos from Adafruit and the 9g Micro Servos. After leaving each running for 15 minutes, the 9g Micro Servos were broken, while the Tower Pro servos held up. We decided to switch our servo brand, and buy several extras just in case.

We also added a middle octave to our piano- the assembly of which required another pi, camera, servo driver, and 7 new servos.

### Week 2- Ticks- 6/10

**Problem**

When we tried to sync notes across multiple Pis, they would often play asynchronously, which caused a discordant noise.

**Solution**

We created a rounding system that would ensure notes only played every whole second, and nicknamed this interval of time one “tick”.

We did this by using truncation to find the nearest whole second in the future, and delaying the program for that amount of time

timeToWait = currentTime - int(currentTime + 0.999999)

**Problem- 7/7/2024**

The one second “tick” was too long, and disrupted the flow of the song

**Solution**

We changed one “tick” to be 0.5 seconds instead, by checking if the timeToWait was greater than 0.5 seconds. If it is, we subtracted 0.5 seconds from the timeToWait.

timeToWait = currentTime - int(currentTime + 0.99999)

if currentTime > 0.5: currentTime -= 0.5

play note

### Weeks 3 & 4- Lights & Performance- 6/17 & 6/24

We integrated lights. We used the Adafruit NeoPixel lights due to the fact that they are easily programmable.

**Problem**

The RP chip in the Raspberry Pi 5 was not compatible with the NeoPixel lights.

**Solution**

We connected the Pi 5 to the Cytron MakerPi RP2040, and used UART to send communications that conveyed which symbol had been detected, in order to activate the corresponding color of lights.

**Problem**

The lights glitched and flashed different colors too frequently, despite being coded to display a single solid color.

**Solution**

We added a capacitor across the power and ground terminals of the lights to guard against sudden power changes.

### Week 5- Integration + Video/Poster- 7/1

**Problem**

We had code working for the NeoPixels, and we had code working with the ticks. We decided to integrate the two. And it failed- the servos would not move, but the lights flashed.

**Step 1**

We removed all the updates and added the NeoPixel code back in, line-by-line. It worked.

**Step 2**

We thought that the issue might be where we are calling the serial function to control the lights. Currently, we call it in the main loop after putting the object in the queue, while we control the servos in the worker\_thread. Therefore, we will now try to control the lights in the worker\_thread as well.

**Problem**

As we played the piano, it would repeat notes 2-3 times extra.

**Solution**

Added a “timer” that ensured no two notes could be played within 0.5 seconds of each other. The exact time value will most likely require modification.

# Future Enhancements

* use computer vision to train at least one of the symbols ourselves (example hold up a peace sign facing to the side to show to play the following note twice)
* expand from taking just picture frames, to video captures in order for the computer to be able to learn words and phrases in ASL, not just letters