The jUKEbox

Biography

Sabrina Hsia:

I am currently an undergraduate student at California State University, Long Beach pursuing a B.S. in Computer Engineering. Since I enjoy seeing my codes come to life, my area of interest is geared towards embedded systems and hardware development. One of my hobbies include researching new topics to gain exposure to various fields of study, which encouraged me in the past to work as a research assistant under Dr. Perla Ayala, a Biomedical Engineering professor at California State University, Long Beach, to explore different solutions for repairing the damage from cardiac arrest. When I have free time, I typically spend it traveling, trying out new restaurants, or playing with my pets.

Jackie Huang:

I am a first-generation college student currently pursuing a bachelor's degree in Computer Engineering at California State University Long Beach. Throughout the course of my studies, I found my interest lies in hardware development. My main field of interest is embedded systems, and I am currently working with Professor Min He on testing and developing labs for future students in the embedded series courses at Cal State Long Beach. Additionally, I am also working as a teaching assistant under Professor Hailu Xu for the Operating Systems course. When I'm not at school, I work on repairing computers and consulting customers on their technological needs. During my free time, I enjoy working out, going out for late night drives, and kicking back with my friends.

Jordan Maynard:
I am a computer engineering honors student at Cal State Long Beach. During my undergraduate studies, I have worked as a teaching assistant, tutor, competition robotics instructor, and research assistant. Currently, I have two academic papers published in the ACM and IEEE Xplore digital libraries (with a third in the works!). My research focus is on hardware security, with an emphasis on hardware Trojans, logic locking, and logic obfuscation. Outside of school, I spend my time playing with my three cats, skateboarding, and working on my super cool car.

Project Overview:

Our project is a self-playing ukulele that can also play music back to the user. We achieved this goal by mounting motors to the neck and strumming hole of a ukulele and managing the timing with the TM4CGH6PM microcontroller. We are incorporating a microphone to listen to audio and identify ukulele chords to try and replicate it through a software run on the Raspberry Pi 3 Model B. The ukulele will use this information to play any audio within its range back to us.

The intended audience of our product is anyone who doesn't want to spend countless hours learning and practicing the ukulele. Instead, get a jUKEbox and sing along to the song of your choice!

Project Implementation:

In terms of computational hardware, our project is run entirely on a Raspberry Pi model 3B and a TM4CGH6PM microcontroller. The Pi is used for UI and audio processing, while the TM4C is used for servo control and chord information storage. The computing components are connected via UART and information about listening vs playing mode is transmitted between the two.

Other electronic aspects of our project include 9 HS-311 servo motors, 8 mounted to the neck of the ukulele and 1 mounted above the mouth to create a strumming mechanism. The user interface is accessible through a touch screen which mounts on top of the Pi. Additionally, all electronic components are powered by a 5v 12a power supply which plugs directly into an outlet.

The mechanical aspects of our project handle the way the ukulele is automatically played. Strumming the strings and holding down the strings on the frets were the two challenges to be solved. For strumming, a servo is mounted above the mouth and a dowel is fitted to the servo with a pick attached at the end. When the strumming servo turns, the pick strums each string. The neck required further mechanical prowess. Eight servos were attached to the neck, and each was fitted with a custom camshaft to press down two strings on the fret it is attached to in any combination.

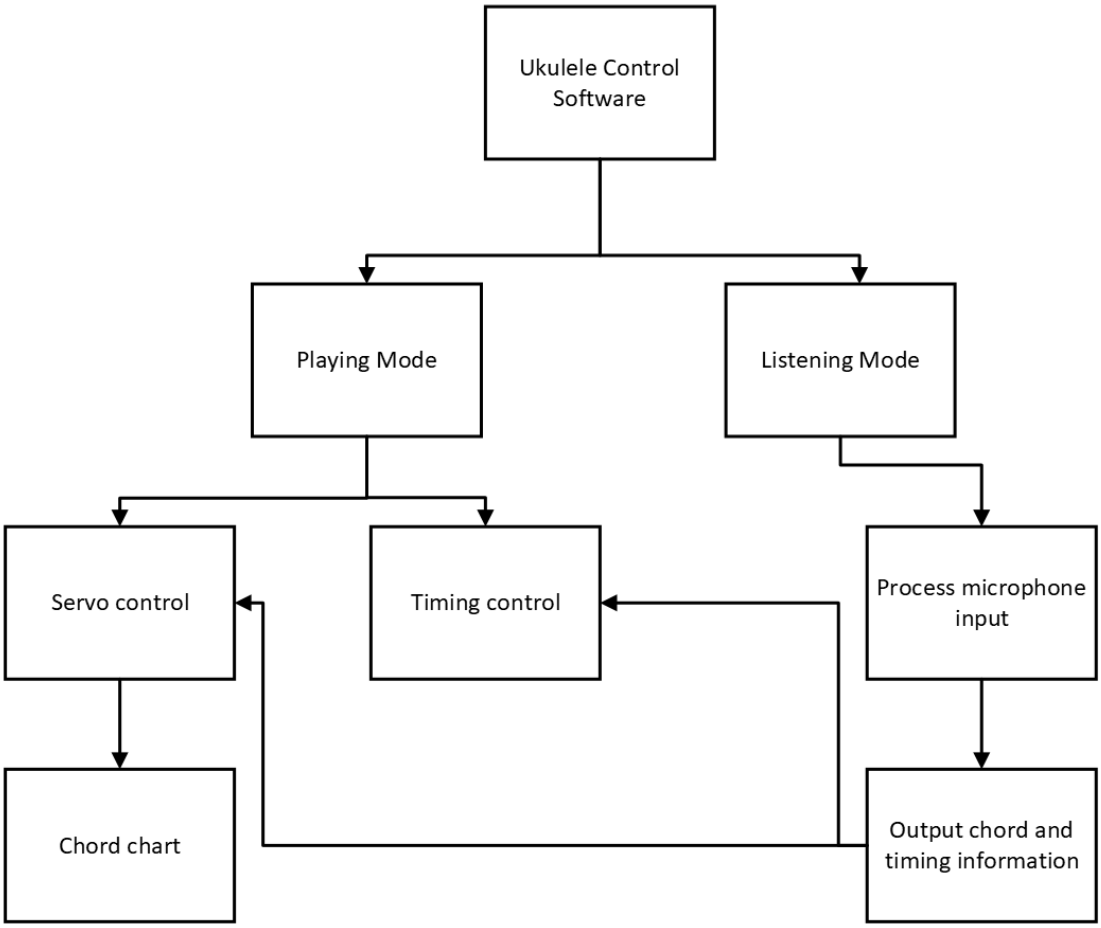
Almost all of the software in our project is written by us and customized to our exact needs and specifications. Our code has two modes: the playing mode and the listening mode.

The playing mode outputs PWM signals to control the 1 strumming servo and 8 neck servos. We have a list of all the possible chords we can use, which is 324 chords, and each one configures the neck servos to hold the strings for each chord. The strumming is controlled by SysTick interrupts to give accurate timing to each strum.

The listening mode consists of a USB microphone input to the Raspberry Pi which stores input in a .wav file. Using the information extracted, a music analysis software is run to find the chords played at a set interval of time. At this point, we use an algorithm to convert the chords and timing into a readable format for the playing mode. This information is sent to the servo controller via UART to make the ukulele play songs back to us.

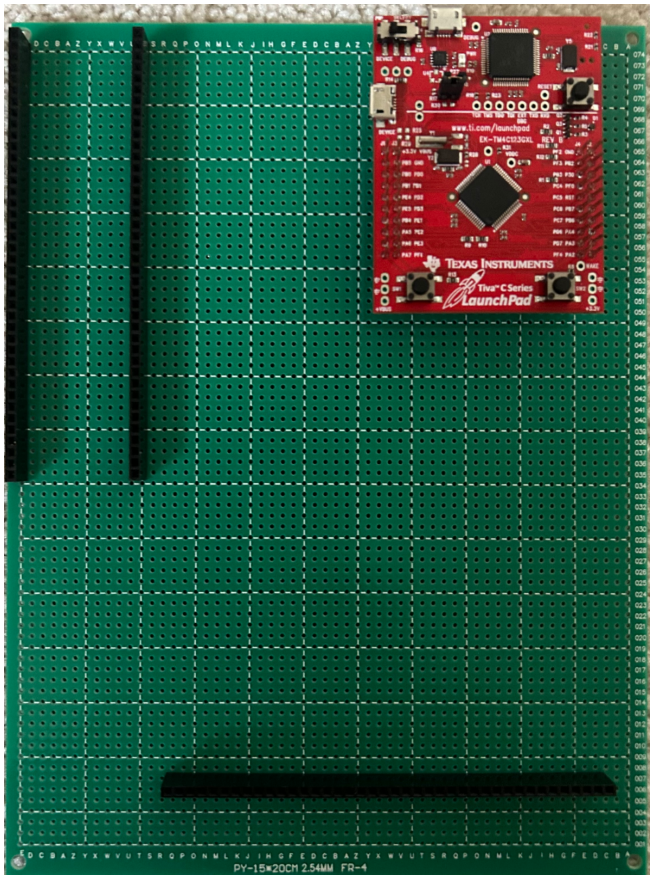
Initially, to implement the audio processing, we ran a bandpass filter to attenuate tones outside of our playing range. Then, we ran a short-time FFT to extract spectrogram information. From this point, we ran an algorithm which returned the highest 4 peaks from the frequency matrix at each time interval. From this point, we were able to match these tones to the closest notes and use these four notes to construct a chord. This was a harsh implementation of chord detection, and returned inaccurate results. The problems mostly arose from harmonics and noise, both of which could introduce large spikes in the target frequency range, introducing false notes and causing incorrect analysis results. Because of this, we switched to using a collection of powerful music recognition algorithms called Essentia.

Essentially, Essentia is a full-flow music analysis software. It parses input from the file, runs a bandpass filter, runs short-time FFT, and runs a number of algorithms to extract information from the audio data. The results are written to a .json file which we parse in software to use the chord data. A large array of detected chords is returned, with each chord representing the strongest chord detected in an interval of 0.045 seconds. From this array, we find the length of time at which each chord is detected and only store the ones which are longer than 0.5 seconds (the minimum time at which our ukulele can strum and change chords). The chords and timing are sent to the ukulele, which then plays each back and strums.



For the user interface, our team selected the Raspberry Pi 3 Model B and the ELECROW 5in LCD display. The LCD display allows us to provide a GUI and has touch input capabilities, which helps the client interface with the project. The Raspberry Pi 3 Model B is compatible with the LCD display we selected. We are using Glade and GTK to create the graphical user interface that the users will interact with to either select a pre-programmed song or have the ukulele be in free listen mode. In the pre-programmed song option, there are 5 songs displayed. The user has a choice between the Adventure Time intro, Riptide by Vance Joy, Somewhere Over The Rainbow by Israel Kamakawiwo'ole, Can't Help Falling In Love by Elvis Presley, and Looking Out For You by Joy Again. In the free listening mode, the user has the option to choose between 5 seconds, 10 seconds, 15 seconds, 20 seconds, 25 seconds, or 30 seconds of microphone input to process, which is used to feed into the Essentia algorithm to detect chords.

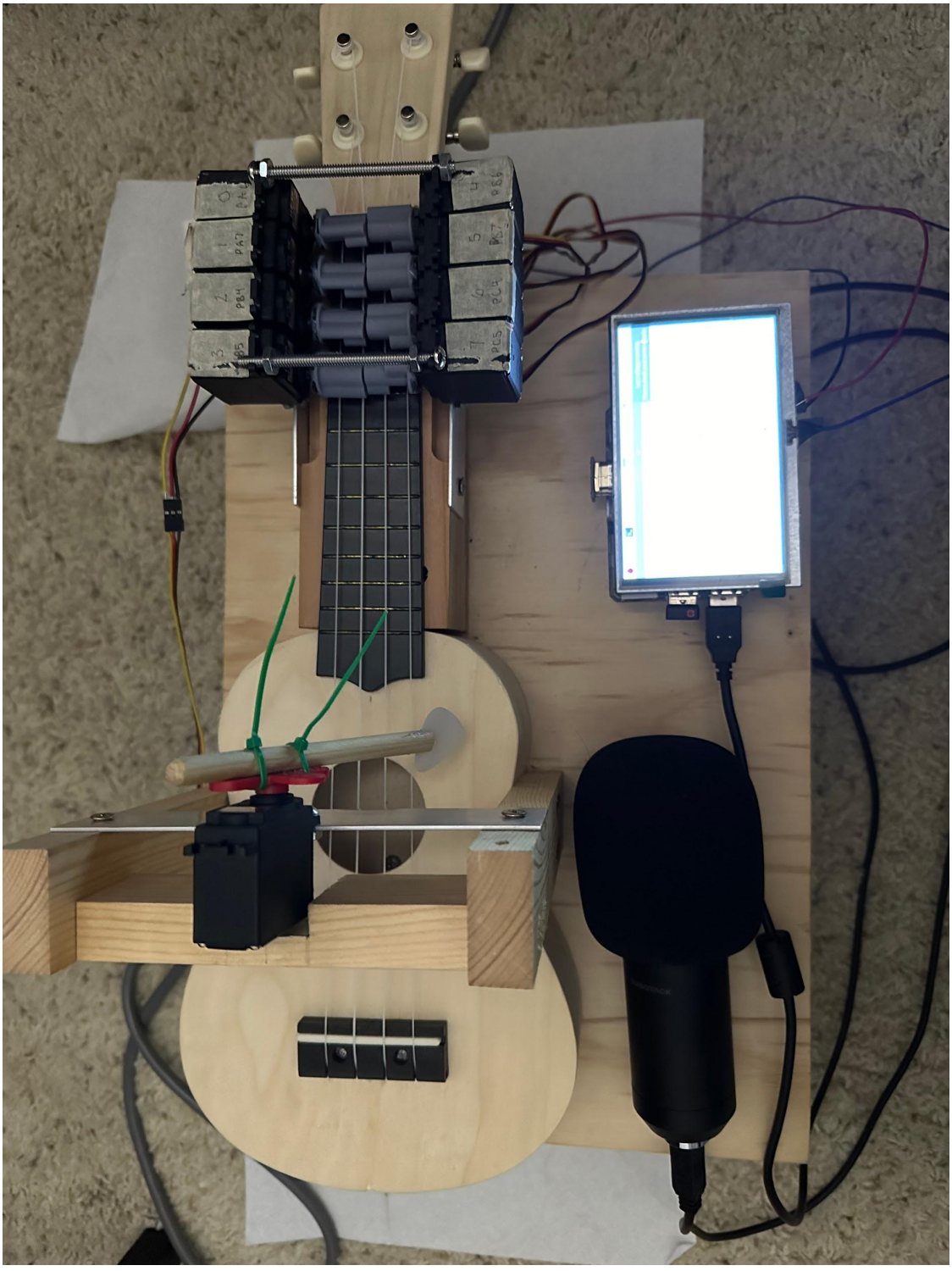
In terms of the hardware aspects of the project, a 15x20cm perfboard is used for the purpose of having a centralized host for all the connections necessary for the hardware.



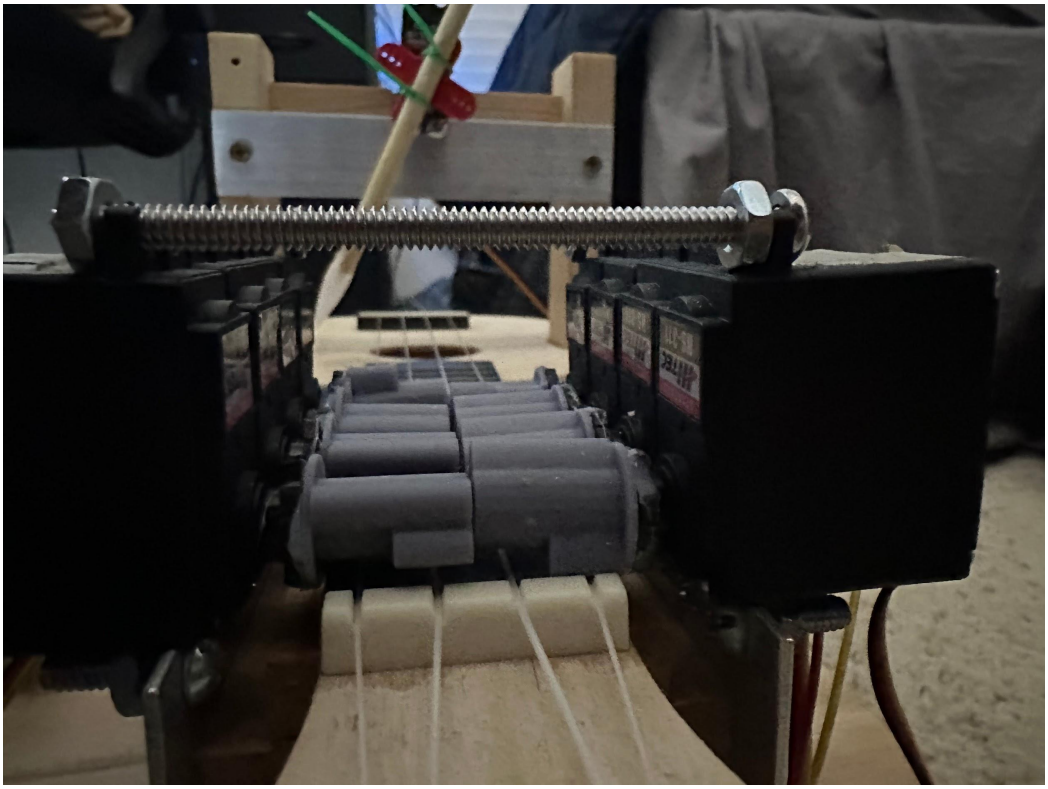
In terms of the mechanical aspects in our project, we used a 2x4 wood to mount the servo motors onto the ukulele neck. The 8 chord servo motors are held against the wood using metal brackets. The custom camshafts are attached to the servo motors and the tops of the servos are screwed together to support the weight of the camshafts by preventing them from tilting away due to the tension of the strings. The nut and saddle of the ukulele is filed down to lower the strings, making it so that the camshafts would not have to press as far down. For the single

strumming servo motor, we used wood to create a bridge that goes from both sides of the ukulele body. On that bridge, we placed a metal plate to mount the strumming servo. A dowel is secured onto the strumming servo with a flexible pick attached to it.

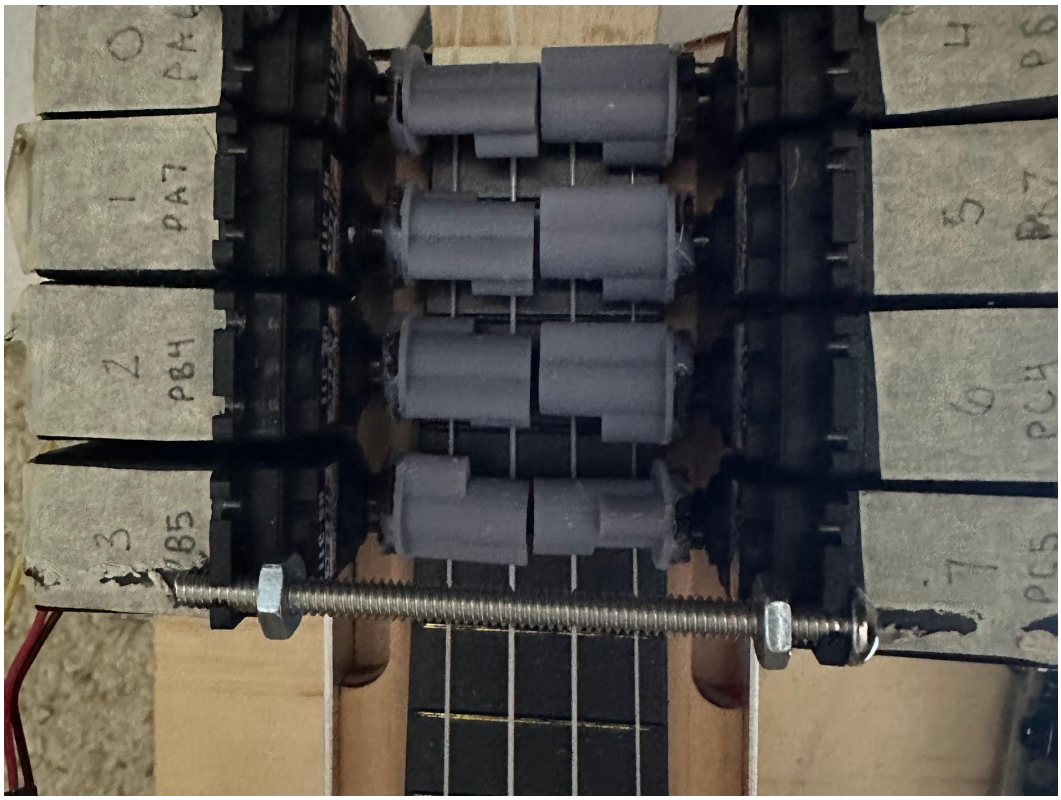
Current Top View



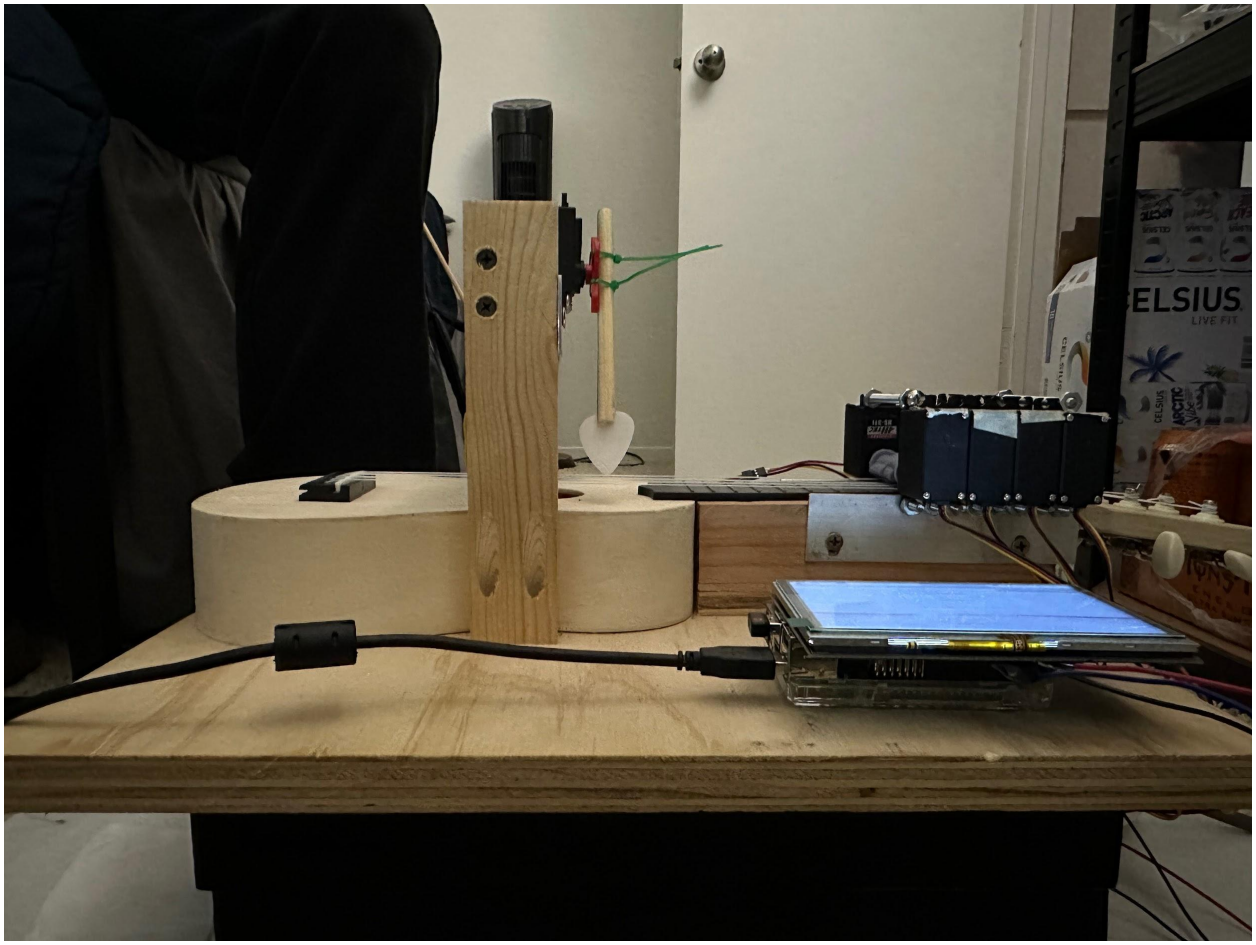
Picture of Servo Mounting



Top Down View of Chord Playing Mechanism

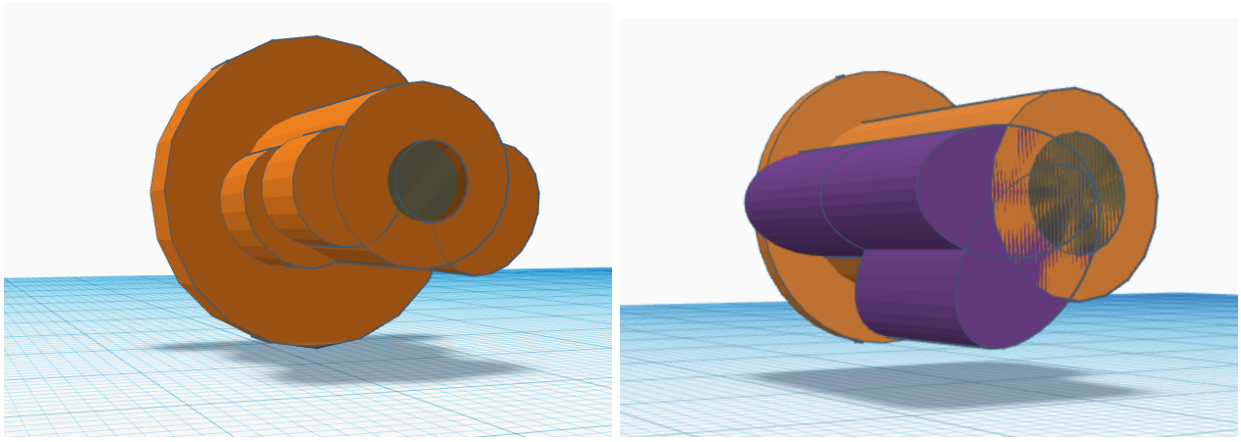


Side View of Chord Playing Mechanism

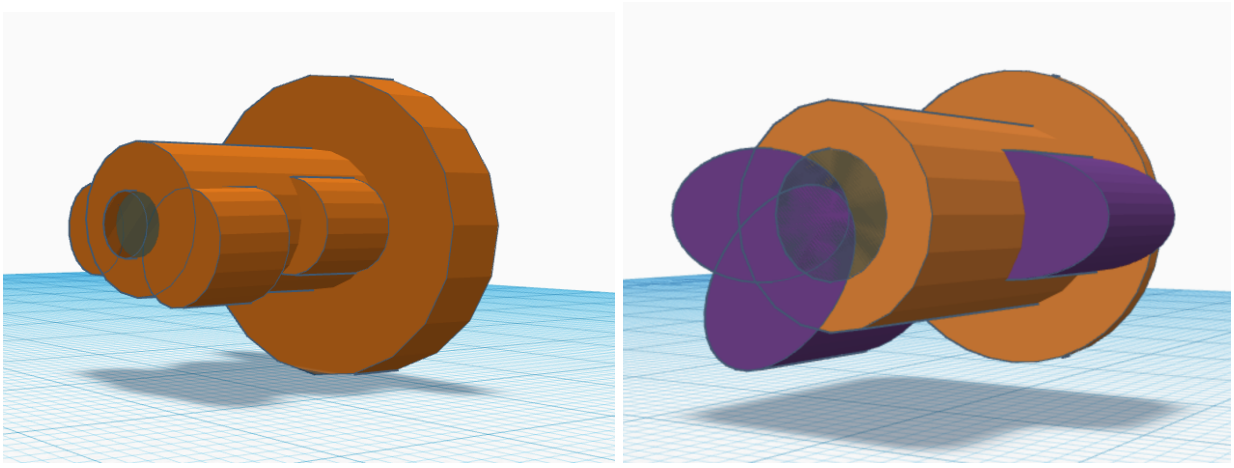


The camshafts have been redesigned with five major changes. Firstly, the height of the cylinder was reduced to prevent rubbing with the camshaft on the opposite side. Second, the cams were extended to remove the extra space between them on the shaft, ensuring that the strings cannot slip out from the side of the cam. Third, the width of the baseplate was reduced in order to prevent rubbing the baseplate on adjacent camshafts. Fourth, a hole was placed all the way through the middle to make space for a screw to be driven into the motor, fastening the camshaft more securely than just hot glue. Lastly, the width of the cams have been reduced slightly to reduce the possibility of touching strings when in the open position. The pictures below depict the difference between the old and new designs.

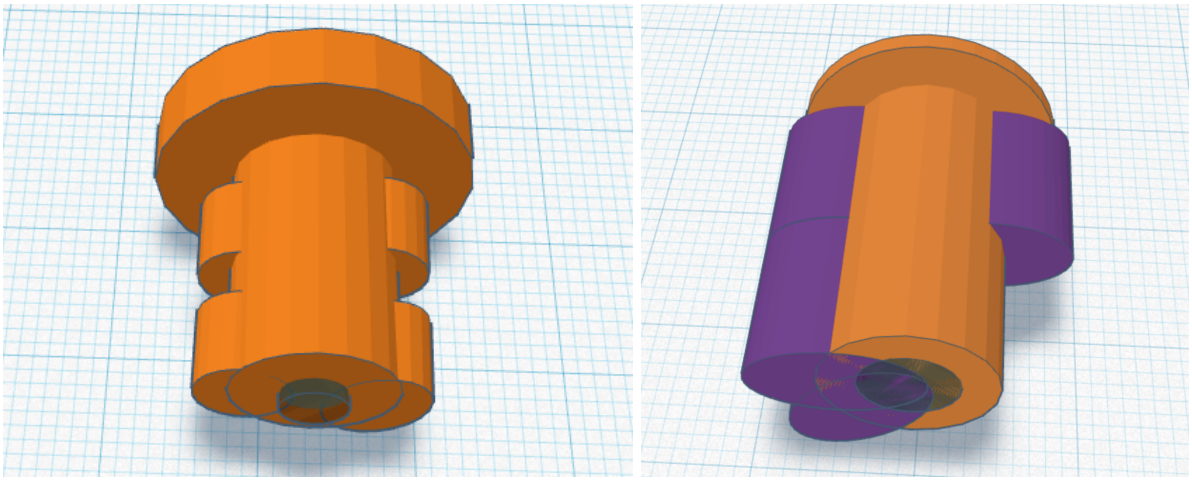
Left View Camshaft (old vs new)



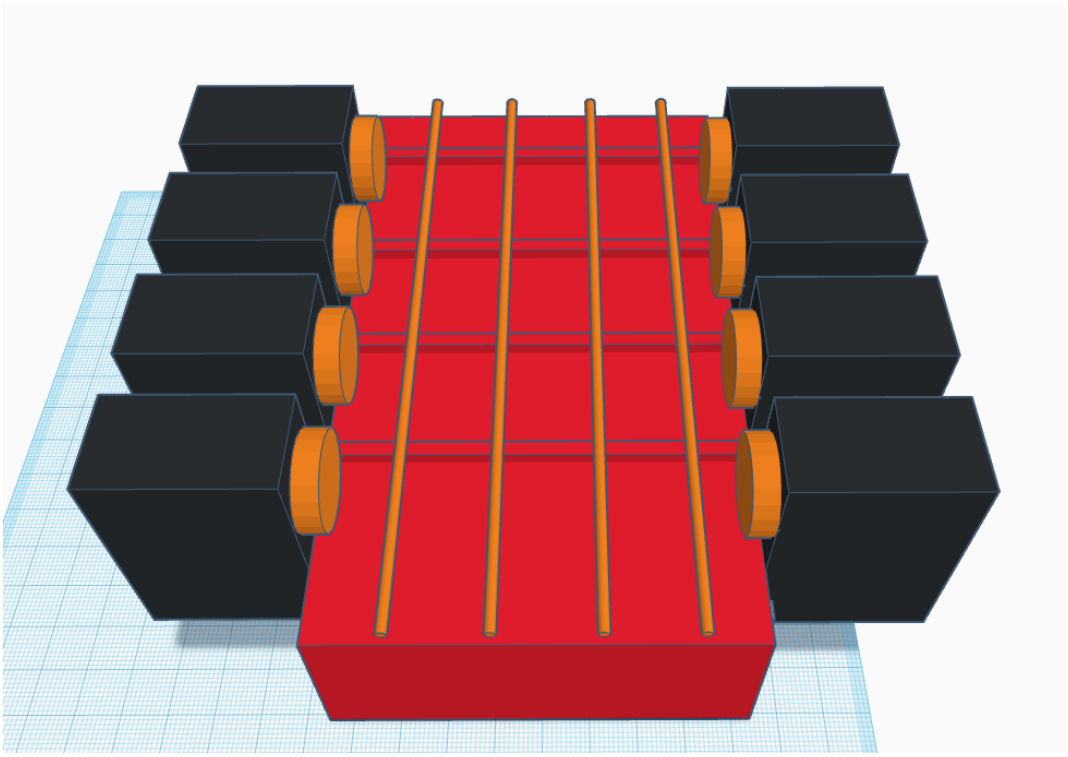
Right View Camshaft (old vs new)



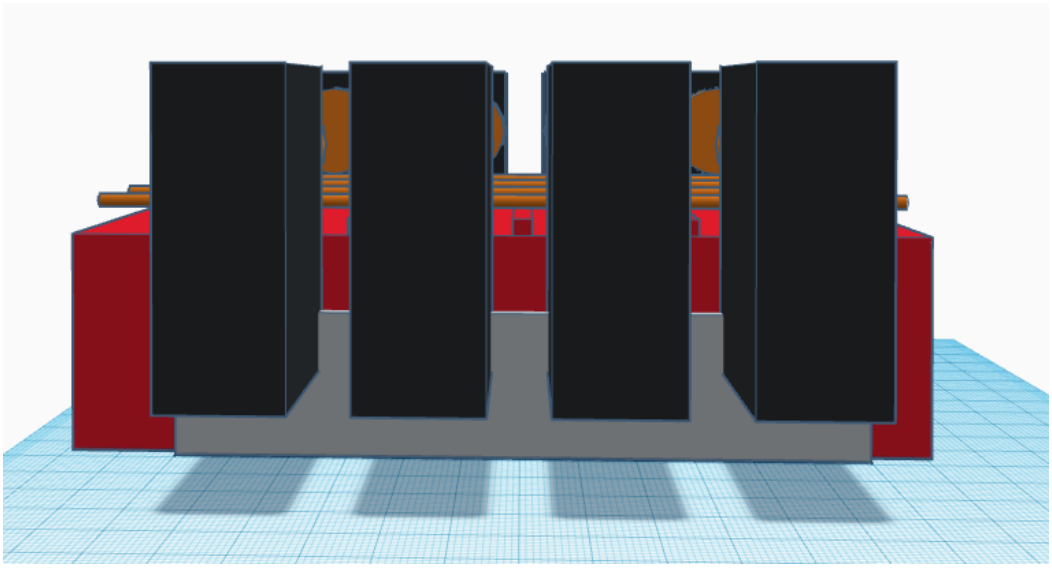
Top View Camshaft (old vs new)



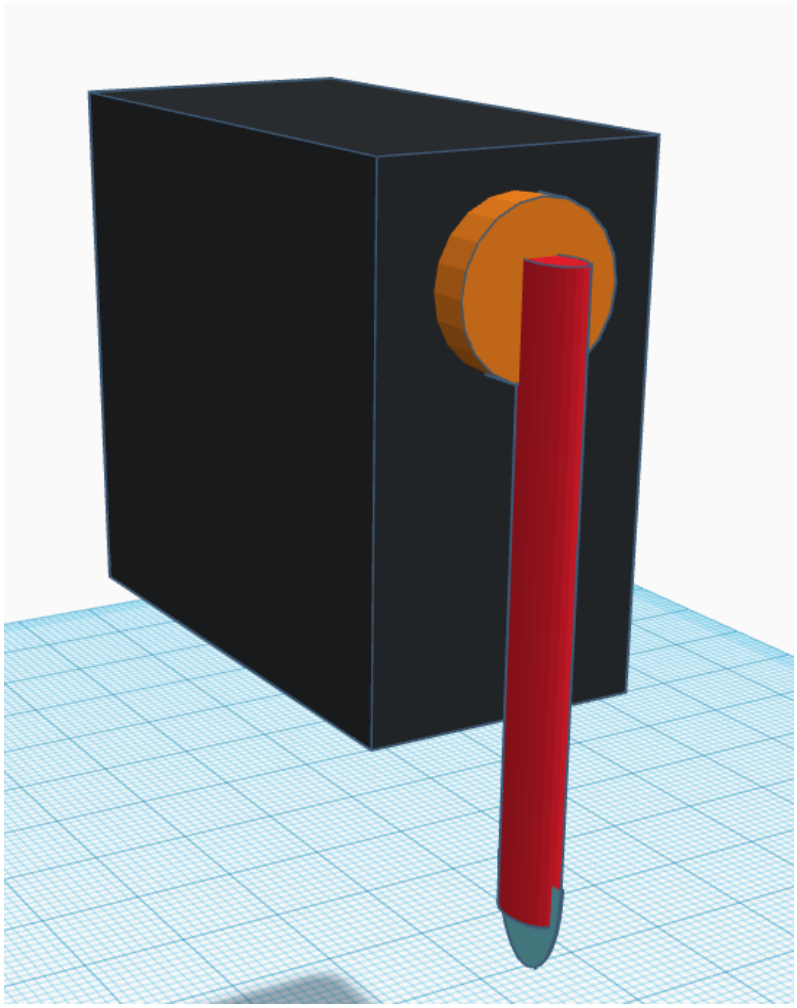
Mounting Top View



Mounting Side View



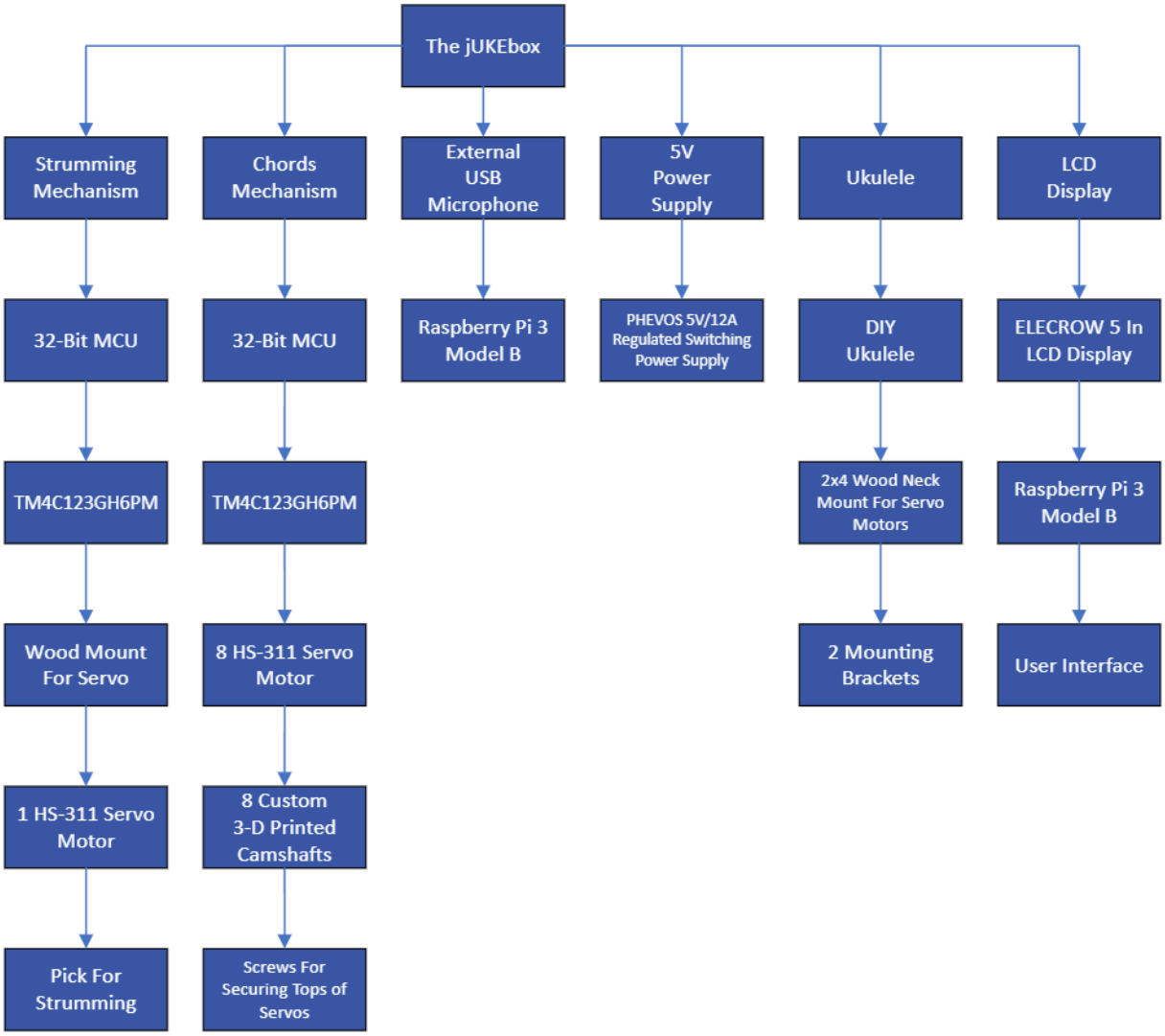
Strumming Mechanism



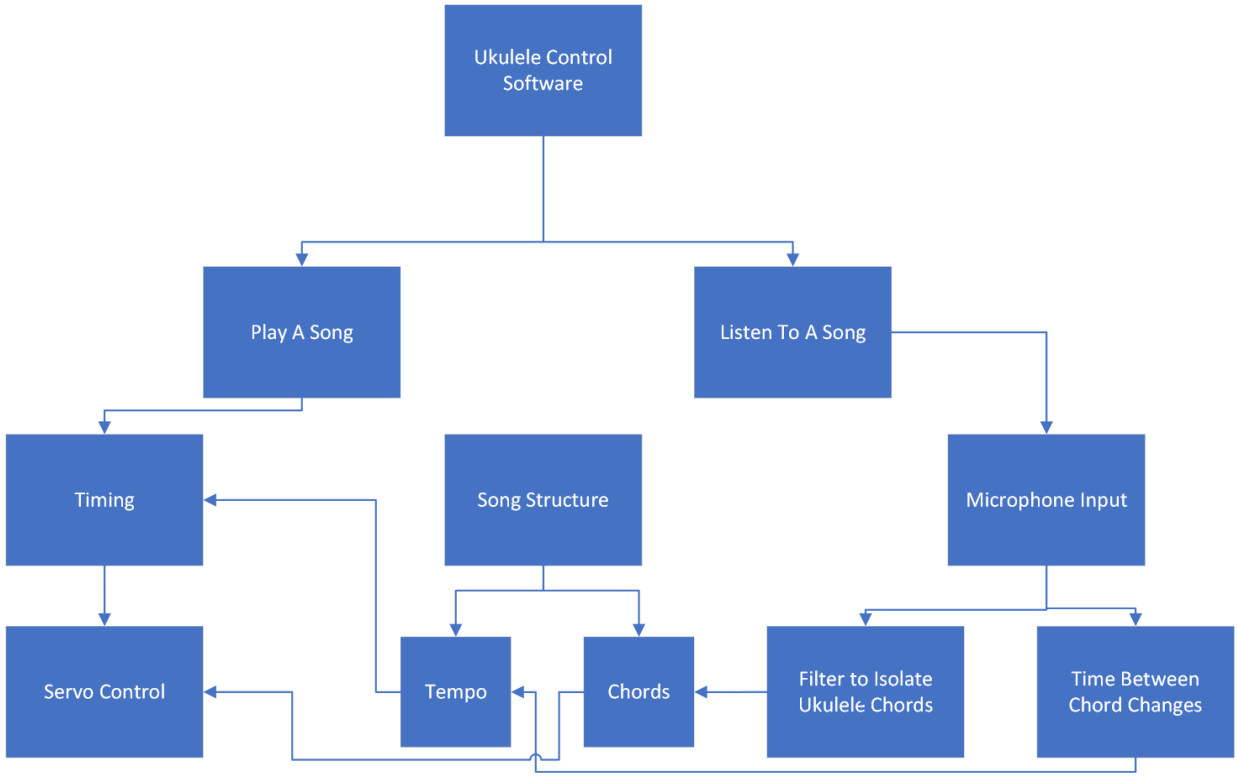
Project Constraint:

Our project is not able to play every song back to the user perfectly. We decided to cut out any chords detected from the Essentia algorithm that have less than 10 samples due to the amount of noise that were converted into chords. Based on the sampling rate of 0.045 seconds, we based the tempo off of multiplying that value by the amount of samples for the specific chord detected, meaning the tempo is not accurate. There is a physical constraint with the ukulele because it will not be able to play certain chords due to songs being in a lower or higher frequency than the ukulele range. The strumming mechanism will only be able to strum in a down, up pattern. This is a limitation of using a servo to strum the chords, as a servo has a limited 180 degree range of motion. The strumming mechanism is also limited on how fast it can strum because the servo requires enough time to go through all the strings.

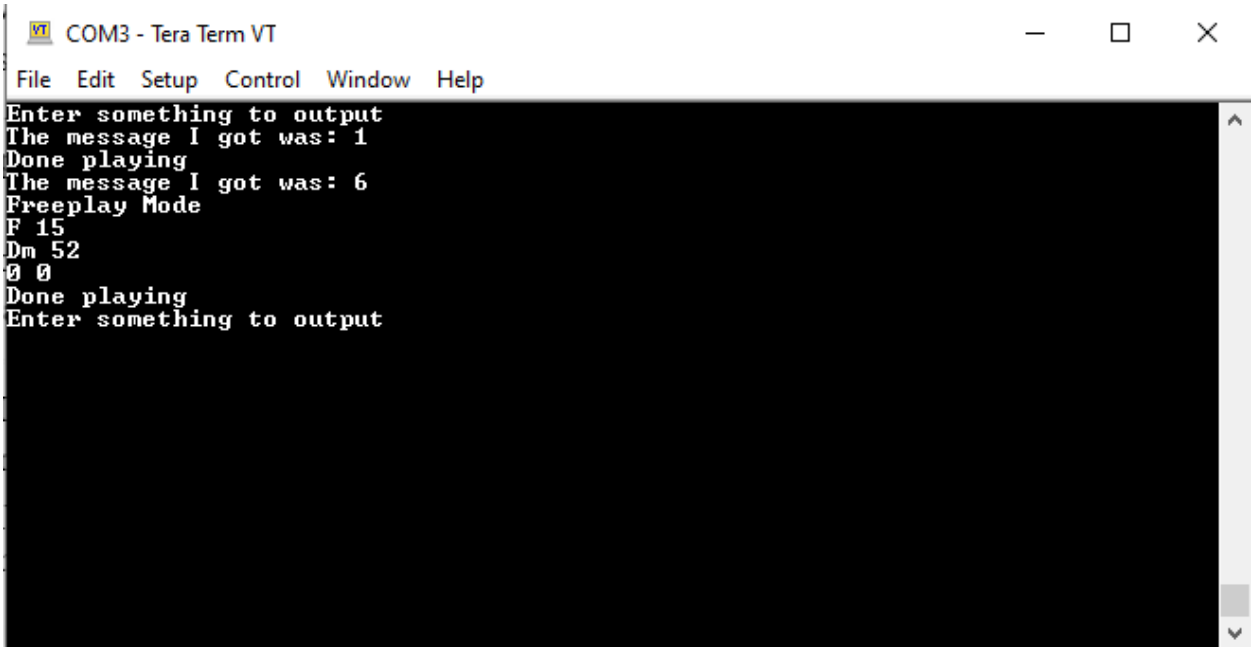
Work Breakdown Structure:



Software Verification:



UART Communication between Raspberry Pi and TM4C:



The screenshot above shows the UART communication received by the TM4C from Raspberry Pi and displays it on a serial terminal that is connected to the TM4C.

Essentia Chord Extraction Algorithm Output:

```
ameter 'minTempo' with a parameter of type 'REAL' but the required parameter typ
e is 'INT. Losing resolution while truncating to integer.
Process step 2: Low Level
Process step 4: Mid Level
Process step 5: High Level
Process step 6: Aggregation
Writing results to file audio.txt
Line:
    chords_progression: ["Am", "Am", "Am", "G#", "G#", "F", "F", "F", "F", "F",
"F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "G#",
 "F", "G#", "G#", "G#", "G#", "G#", "G#", "G#", "G#", "G#", "G#", "G#", "G#", "G
#", "G#", "G#", "G#", "Fm", "Fm", "Fm", "F", "F#m", "F#m", "F#m", "F#m", "F#m",
"F#m", "F#m", "F#m", "F#m", "F#m", "F#m", "F#m", "F#m", "F#m", "F#m", "F#m", "F#
m", "F#m", "F#m", "F#m", "F#m", "F#m", "F#m", "F#m", "Bm", "Bm", "F#m", "F#m", "
F#m", "F#m", "F#m", "F#m", "F#", "F#", "F#", "F#m", "F#", "F", "F", "F", "F", "F
", "F", "F", "F", "F", "Cm", "Cm", "Cm", "Cm", "Cm", "Cm", "Cm", "Cm", "Cm", "Cm
", "Cm", "Dm", "Cm", "Cm", "Cm", "Cm"]
F 20 :i=0
G# 16 :i=1
F#m 24 :i=2
Cm 11 :i=3
```

The chords detected by the algorithm after an audio file is processed. Only chords with 10 or more samples in a row are preserved and outputted to the TM4C.

Example of Song Structure:

```
21
22  Can't Help Falling In Love - Elvis Presley
23  ruct Song CHFIL[] = { {"C", {8,5,1,4,3,-1}}, {"Em",
24                        {"G", {8,5,1,4,3,-1}}, {"G", {
25                        {"G7", {8,5,1,4,3,-1}}, {"C",
26                        {"Am", {8,5,1,4,3,-1}}, {"F",
27                        {"G", {8,5,1,4,3,-1}}, {"Am",
28                        {"C", {8,5,1,4,3,-2}} };
29
```

The first element contains the chord and the following contains an array of timing for that specific chord.

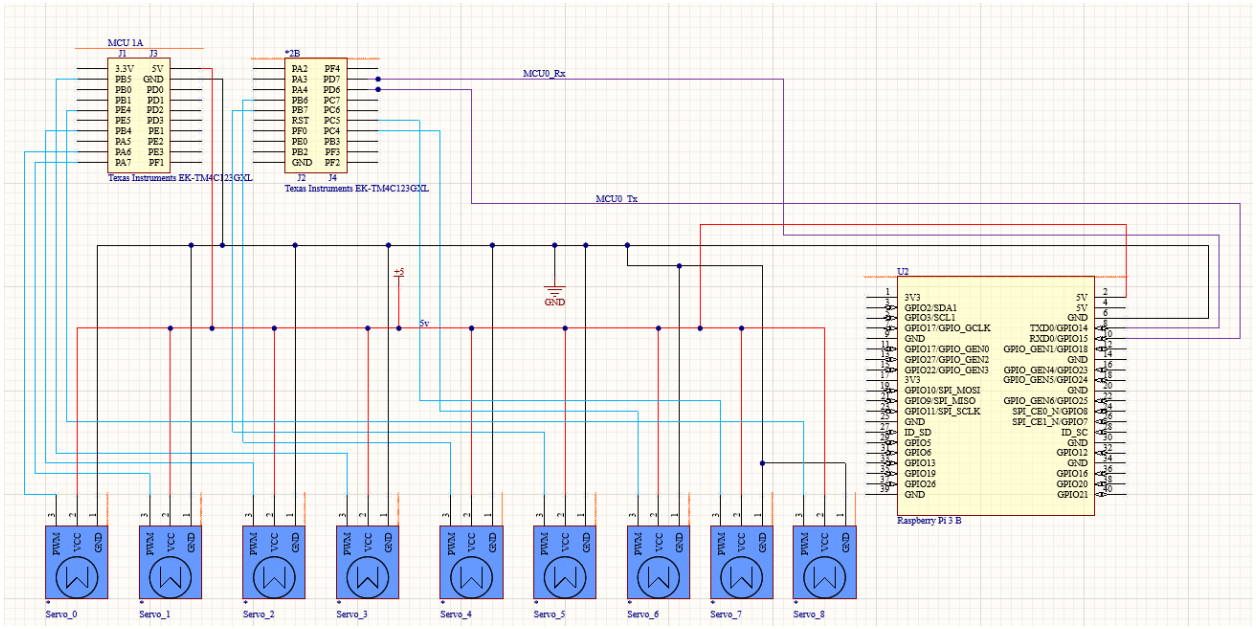
Example of Chord Structure:

```
// All the chords for A
const struct chord_positions chords_A[] = { {"A", 0x5552}, {"Am", 0x5551}, {"Aaug",
                                            {"A7", 0x5556}, {"Am7", 0x5555}, {"Amaj
                                            {"Asus2", 0x9462}, {"Asus4", 0x555D}, {
                                            {"A9", 0x6152}, {"Am9", 0x6154}, {"Amaj
                                            {"A5", 0x5591}, {"A6", 0x1911}, {"Am6",
```

The first element contains the chord and the second element contains the servo position information required to play that chord. With eight motors, we need 16 total bits, or four hexadecimal digits, to represent any chord configuration. Using this approach, a decoder sets the positions of the camshafts based on the configuration number it receives. We transcribed all 324 possible chords into this format, and are able to play them by simply passing the string name of each chord to a function.

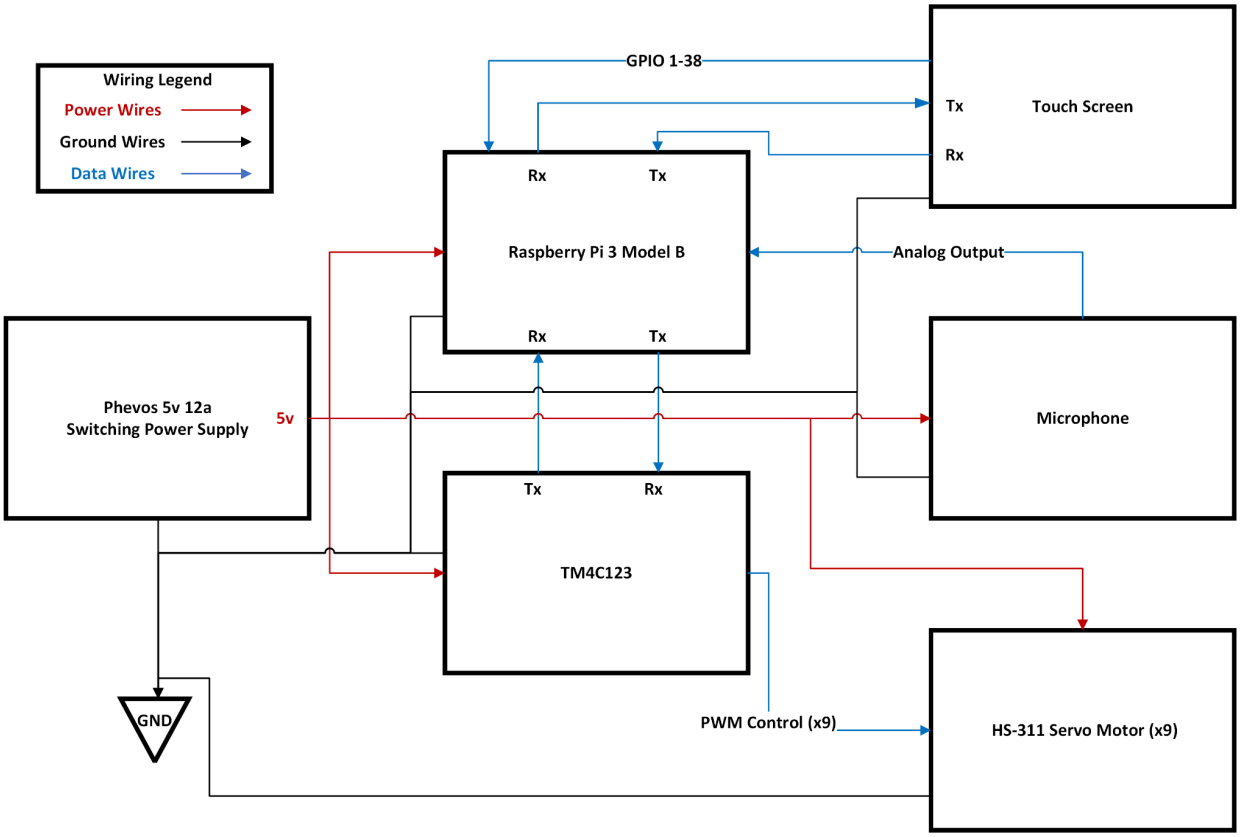
The strumming works by setting the servo to the opposite side and passing a delay value to wait after the strum. In this scheme, the timing and strums are integrated into the same function, creating an easy way for us to define songs. We have 5 songs defined at the moment: the Adventure Time intro, Riptide by Vance Joy, Somewhere Over The Rainbow by Israel Kamakawiwo'ole, Can't Help Falling In Love by Elvis Presley, and Looking Out For You by Joy Again. Our song definition format consists of structures containing a chord and a sequence of timing numbers to denote the strumming pattern. Each structure defines the strumming pattern for a single chord, so a list of these structures is able to define a full song.

Complete Schematic:



Bill of Materials (Electronic Components)							
Part	Part #	Description	Package Used	Manufacturer	Distributor	Cost	Quantity
LaunchPad TM4C123G Evaluation Board	EK-TM4C123GXL	32-Bit Arm Cortex-M4F based MCU	64-pin LQFP	Texas Instruments	Digi-Key	$21	2
HS-311 Standard Economy	31311S	DC Motor	Bulk	Hitec Commercial Solutions	Digi-Key / Amazon	$15	9
PHEVOS 5V/12A Regulated Switching Power Supply	N/A	DC Power Supply	N/A	PHEVOS	Amazon	$20	1
YUNGUI PCB	N/A	15x20cm PCB Protoboard	N/A	YUNGUI	Amazon	$15	1
ELECROW LCD Display	QD050001C0-40	5in Raspberry Pi Screen	N/A	ELECROW	Amazon	$45	1
Raspberry Pi 3 Model B	Raspberry Pi 3	SBC 1.2GHz 4 Core 1GB RAM	Bulk	Raspberry Pi	Digi-Key	$35	1

Overall System Functional Block Diagram:



HS-311 Servo Table x9						
Signal Name	Net label	Type	Direction	Voltage	Current	Description
Power 5v	5v	Power	Input	5 V	800mA	Servo Power
Ground	Gnd	Ground	Input	0 V	0mA	Servo Ground
PWM	Servo_PWMx	Digital	Input	0 to 5 V	<1mA	Servo

MCU 1 (Servo Control)						
Signal Name	Net Label	Type	Direction	Voltage	Current	Description
PA 6	PWM Signal	Digital	Output	0 - 3.3 V	>1mA	Controls Servo 0

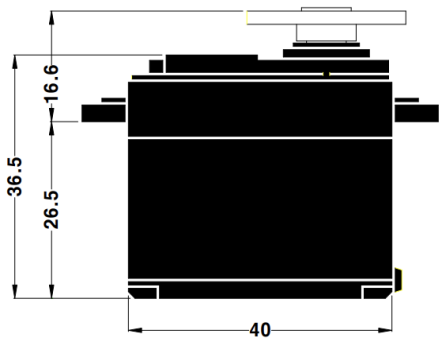
PA 7	PWM Signal	Digital	Output	0 - 3.3 V	>1mA	Controls Servo 1
PB 4	PWM Signal	Digital	Output	0 - 3.3 V	>1mA	Controls Servo 2
PB 5	PWM Signal	Digital	Output	0 - 3.3 V	>1mA	Controls Servo 3
PB 6	PWM Signal	Digital	Output	0 - 3.3 V	>1mA	Controls Servo 4
PB 7	PWM Signal	Digital	Output	0 - 3.3 V	>1mA	Controls Servo 5
PC 4	PWM Signal	Digital	Output	0 - 3.3 V	>1mA	Controls Servo 6
PC 5	PWM Signal	Digital	Output	0 - 3.3 V	>1mA	Controls Servo 7
PE 4	PWM Signal	Digital	Output	0 - 3.3 V	>1mA	Controls Servo 8
PB 0	URx	Digital	Input	0 - 3.3 V	>1mA	Receives data from Raspberry Pi

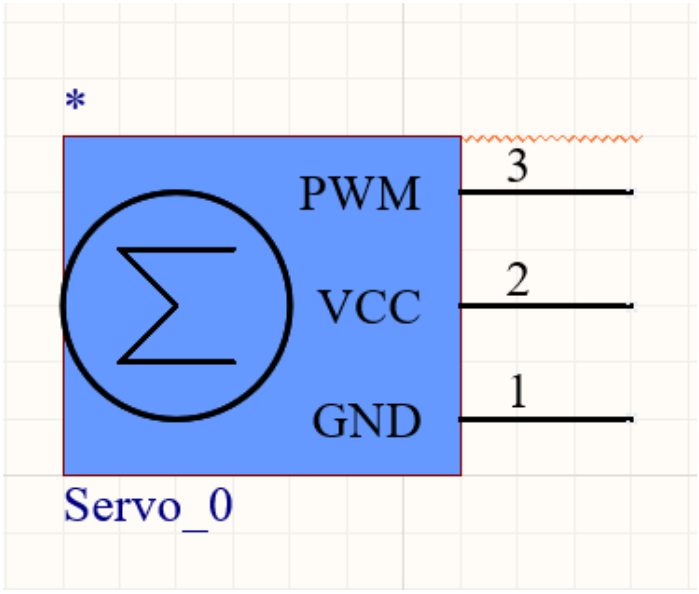
Raspberry Pi						
Signal Name	Net Label	Type	Direction	Voltage	Current	Description
Pin 2	5V	Power	Input	5V	2.5A	Powers the Pi
Pin 3	5V	Power	Input	5V	2.5A	Powers the Pi

Pin 6	GND	Ground	Input	0V	2.5A	Ground for the board
Pin 8	UTx	Digital	Output	0-3.3V	<1mA	UART transmit to MCU
Pin 10	URX	Digital	Input	0-3.3V	<1mA	UART receive from MCU
HDMI	Video Signal	Digital	Input	5V	<55mA	HDMI input from the LCD display
USB Connection	Touch Inputs	Digital	Input	0-5V	<0.5A	Takes in touch nput from LCD display
USB Connection	USB	Digital	Input	5V	<500mA	Takes in audio input from microphone

Subcomponent Descriptions and Operation:

HS-311 Servo Motor:



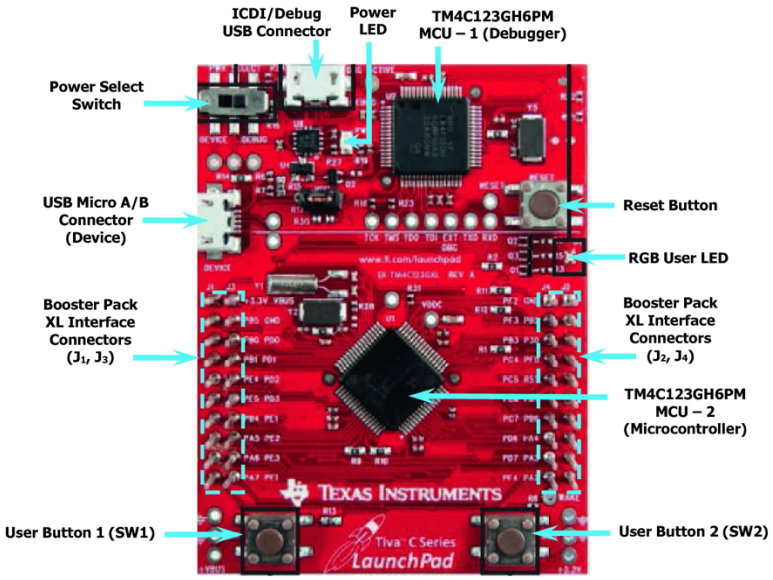


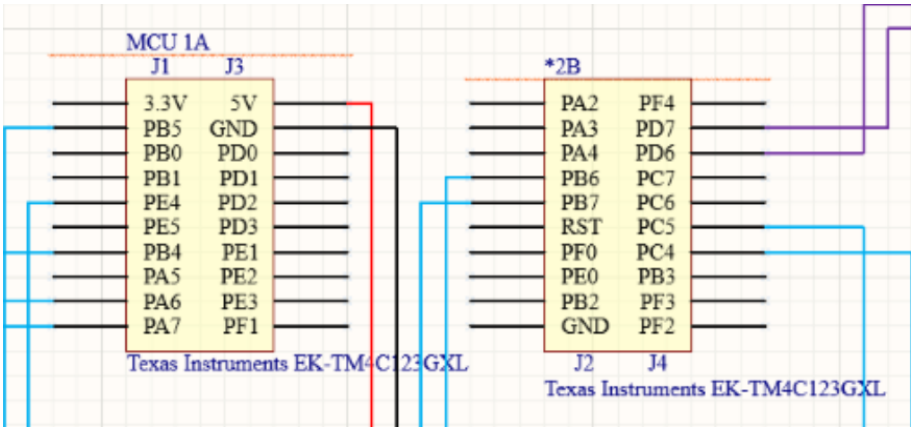
- Operating Voltage 4.8v-6v, will be operated at 5V
- Max Amp draw of 800mA at 6v
- PWM Controller with 1500usec for center position, TM4C123GXL microcontroller will be providing all the PWM signals.
- A camshaft will be attached to the ends of 8 of the servos that will be used to compress the strings of the Ukulele used to create the desired chords on the first four frets.
- A single servo will be used to strum the Ukulele strings.
- All of them will be powered by PHEVOS 5V/12A Regulated Switching Power Supply.

PHEVOS 5V/12A Regulated Switching Power Supply:

- Input Voltage 110-220V AC at 50-60Hz
- Output voltage 5V DC
- Max Current output 12A
- Max Power output of 60 Watts
- Will power all the Servos, TM4C Microcontrollers, Microphone, LCD Screen, Raspberry Pi
- Can not be used to provide multiple voltages at once. Can only be used to deliver constant 5V.

TM4C123GXL ARM Cortex-M4F:





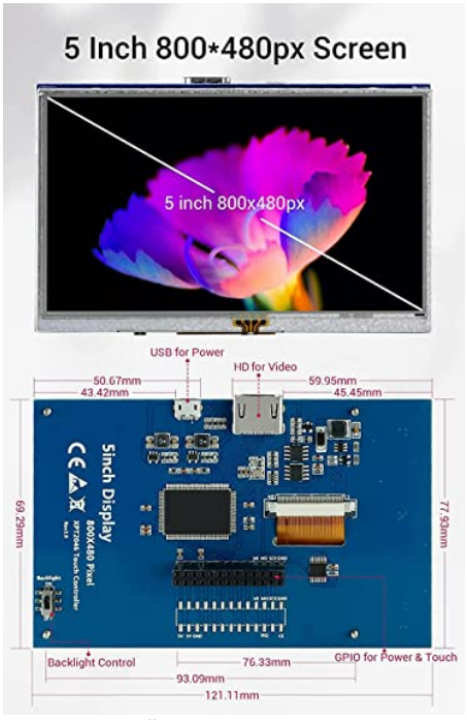
- Will provide the PWM signals to control all nine of the Servos
- Input Voltage is 5V

- Logic is 3.3V

USB Condenser Microphone:

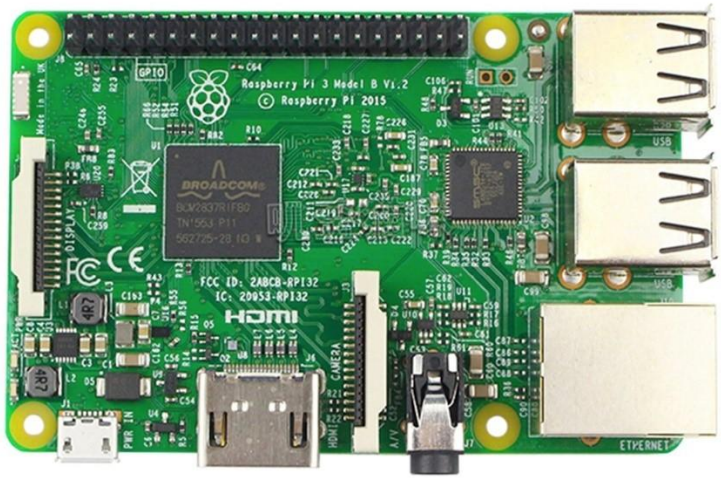
- Input Voltage: 5V
- Input Current: < 500mA
- Will interface directly with the Raspberry Pi and take audio input
- Allows for capturing better audio quality

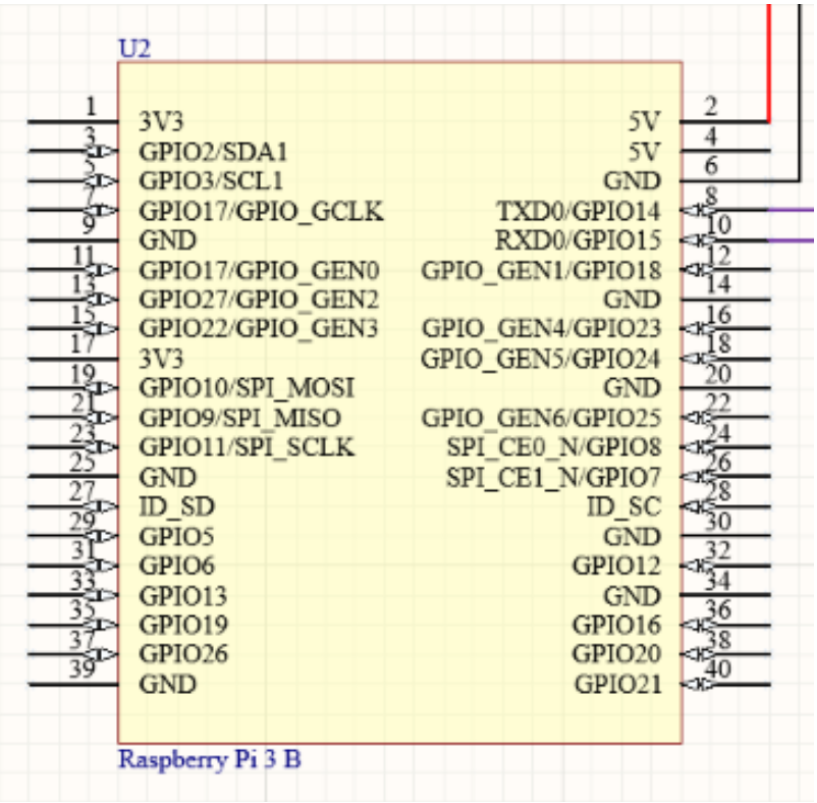
ELECROW 5 In LCD Display:



- 5in LCD display
- Input Voltage: 5V
- Input Current: 2A
- Will be used to display the mode the system is in. This will allow the user to select if it would like the ukulele to play pre-programmed songs or if the user would like the ukulele to play back an audio sample.
- Will allow users to interact with the display with touch
- Connects to the TM4C123GXL and sets the mode that it should be in.

Raspberry Pi 3 Model B:





- Input Voltage: 5V
- Input Current: 2.5A
- Power Consumption: 12.5W
- Will be used to allow the users to interface with the product with touch capabilities
- Is compatible with the ELECROW 5in LCD Display
- Will be using a HDMI to connect with LCD display
- Will run the chord extraction algorithm and send out the information collected to the TM4C

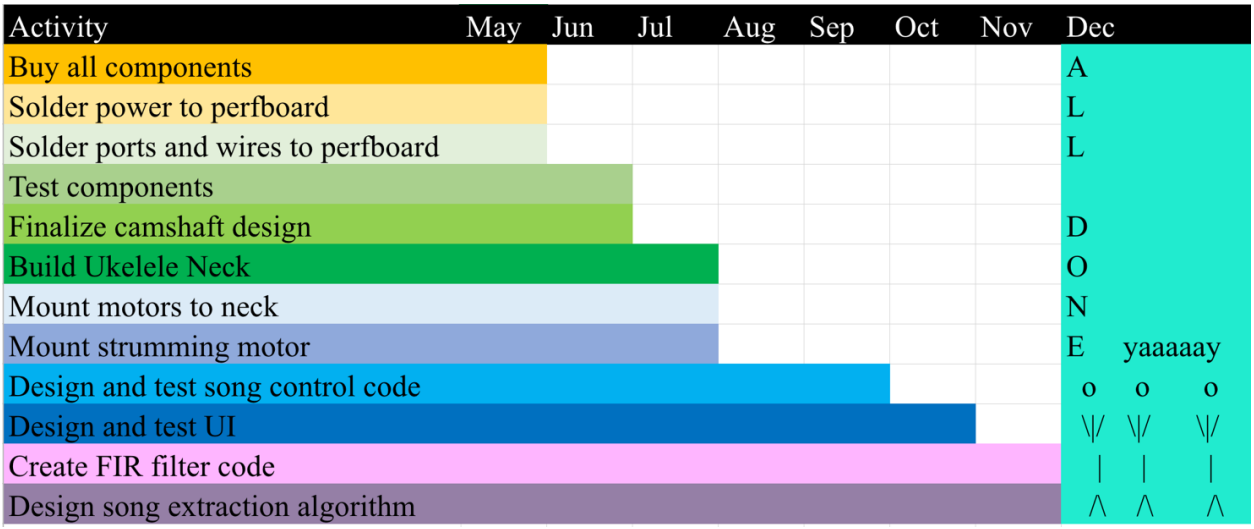
Power Management:

Component	Voltage	Current	Power
TM4C123	5V	300mA	1.5W
Servo Motor(x9)	5V	4.5A	22.5W
LCD Display	5V	2A	10W
Raspberry Pi	5V	2.5A	12.5W

USB Condenser Microphone	5V	<500mA	2.5W
PSU Rating	5V	12A	60W
	Total: 5V	Total: ~10A	Total: ~49W

These are preliminary power estimates. Based on calculations of power drawn from datasheets, we should be within specifications of our given power supply.

Gantt Chart:



5 Demos Contract:

Demo 1) SHAKE TEST / SERVOS MOVING

Objective: The objective of demo 1 is to make sure that most of the mechanical/physical components (excluding finalized camshafts and strumming mechanism) are mounted properly and securely with all the servo motors being able to move. There will also be a simple C++ demo code on the TM4C123 launchpad that can move the servo motors. This demo's purpose is to put us in a good position to finalize the camshaft design.

Verification: This test will include a shake test to show that the assembly is secure and that nothing will break or fall apart. The servo motors will be able to simultaneously move independently of each other.

Measurable Outcome: To constitute a successful demo, nothing will fall out or break from the assembly and all the motors will move at full range, independently and simultaneously. An acceptable error is if we are not able to acquire every single servo motor that we need by this

demo due to being out of stock. That servo motor will most likely be the strumming servo motor if we are not able to acquire it on time.

Demo 2) PLAY ANY CHORD

Objective: The objective of demo 2 is to make sure that the 3D printed camshafts can press on the strings to play any chord and that the strumming mechanism is working properly. Demo code will include a set of all chords in our ukulele's range playing continuously at a fixed tempo.

Verification: This test will demonstrate the finalized camshaft design being able to press on the strings and play all the chords within the first 4 frets of the ukulele. The strumming mechanism will be working to produce the strumming for each chord and UART between the two microcontrollers will be used to send the information regarding the different chords to be played.

Measurable Outcome: To constitute a successful demo, none of the camshafts will be interfering with one another and will be able to press on every chord within the first 4 frets that we are using. The strumming servo motor will be able to strum up and down the strings. UART will send over any chord we choose through the microcontrollers, adjust the camshafts according to those chords, and the strumming servo will strum once it receives the information that the camshafts are in place. An acceptable error is that it may take a while for everything to execute due to the exchange of information between the microcontrollers.

Demo 3) PRE-PROGRAMMED SONGS

Objective: The objective of demo 3 is for the jUKEbox to be able to play two or more pre-programmed songs as one of the modes of our final design. All motors for the neck and strumming should be controlled at different tempos and the camshaft design should be finalized. In software, we should be able to control the tempo of the chord changes and strumming.

Verification: This test will show that pre-programmed songs are able to be played fully. The pre-programmed songs may include songs like "Twinkle Twinkle Little Star", "Mary Had a Little Lamb", and "Somewhere Over The Rainbow." At least two songs will be played to verify that the design is able to play pre-programmed songs.

Measurable Outcome: To constitute a successful demo, the user will be able to choose any of the pre-programmed songs and the jUKEbox will be able to play it. The user will be able to switch to a different song while a song is playing. The song that is currently playing will stop and the different song will start to play. This means that any song will be able to be interrupted and the camshafts will return to neutral position. An acceptable error is that the song may be played at a

slower tempo or sound a little different than the original due to strumming patterns and UART communication to the microcontrollers. Another acceptable error is that some chords may be played at different octaves due to our design only using the first 4 frets of the ukulele.

Demo 4) MICROPHONE INPUT AND PLAYBACK

Objective: The objective of demo 4 is to have the microphone be able to take audio input from the user, distinguish chords and tempo, and play back the chords on the robo-ukulele. This will demonstrate that the FFT is working.

Verification: This test will show that any song that the user plays through the microphone will be played back to the user through the mechanisms of the jUKEbox. The instructor will be able to see that the FFT is working properly and should be able to identify which song is being played back through the jUKEbox.

Measurable Outcome: To constitute a successful demo, the jUKEbox will be able to take in microphone input of any song from the user and be able to play it back. The song that is played back should be recognizable in comparison to the original song. An acceptable error for this demo is that some of the chords may not be played back due to sampling speed of the FFT and the tempo may be slower than the original song because of all the processing that is required to play back the song. Similarly to the previous demo's acceptable error, some of the chords may be played at a different octave than the original due to our design only using the first 4 frets of the ukulele.

Demo 5) INTEGRATION HELL

Objective: The objective of demo 5 is to complete the UI for users to be able to select between mode 1, where the user can select a pre-programmed song, and mode 2, where the user can play a song into the microphone for the jUKEbox to play back. At this stage, the entire project should be near completion. The chords and strumming should be working and able to play songs, and the FFT should be able to distinguish chords and tempo from audio input and process them into a jUKEbox-playable format.

Verification: This test will demonstrate that the jUKEbox is able to switch between mode 1 and mode 2 through the LCD display. Mode 1 will be able to play through various pre-programmed songs that we put in and mode 2 will be able to play back any song that the user plays through the microphone. The user will be able to interrupt the song when switching to a different mode or a different song.

Measurable Outcome: To constitute a successful demo, all of the functional requirements of the jUKEbox should be complete and working properly. This includes being able to switch between the modes of playing pre-programmed songs or songs that the user plays through the microphone. This also includes the UI displaying the option choices for the user to select between modes and songs. To be considered a successful demo as well, the user will be able to go back and switch between modes and songs while the jUKEbox is playing. The acceptable error is like the previous demos' acceptable errors, where there may be a lag when switching between modes and songs, tempo differences, some chord differences due to only using the first 4 frets and missing some of the notes from the songs due to sampling speed of the FFT.