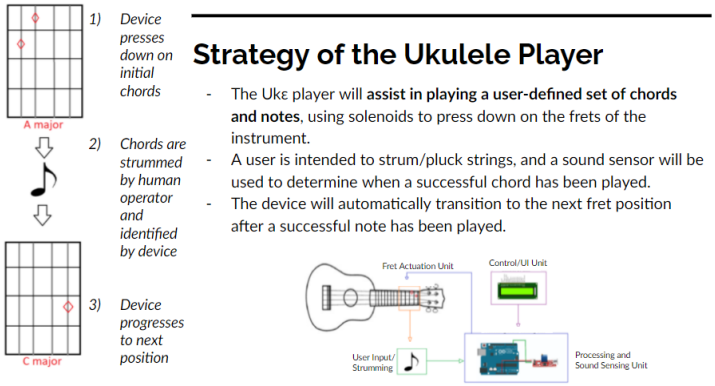
UkƐ MANUAL

John John Huddleston, Hadar Gamliel, Shaan Jagani

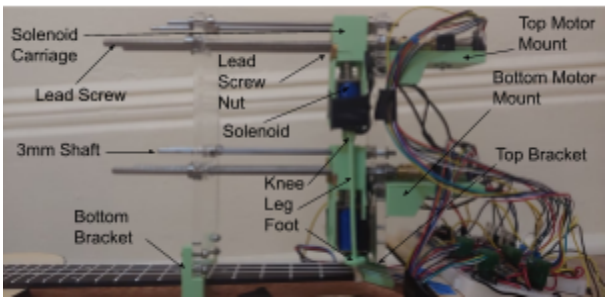
The opportunity that we converged on was based on the personal aspirations and learning goals we had for this class; we wanted to build something creative and fun, as well as create a design challenge for ourselves. The solution we selected through the decision matrix was a string instrument player. Briefly, the idea is to build a device that will automate some portion of string instrument playing. For practicality, we narrowed our solution to one that will assist in playing the ukulele: The UkƐ.

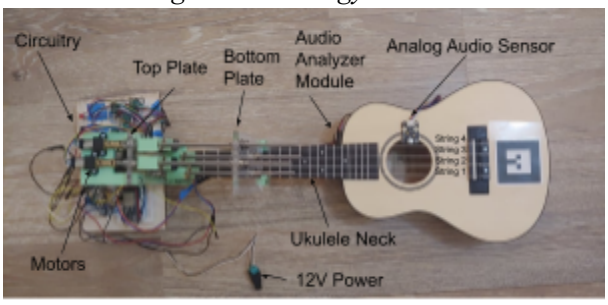
Previous devices that have approached this include the Guitar Machine, created by the MIT Media Lab. However, this device is oriented at producing novel ways of producing music, instead of assisting the player. To our knowledge, this is the only realization of a product that assists a user with little to no understanding of music theory.

The adjacent image explains the high-level strategy of how our device works. Something that was very important to us was that the device could play any combination of chords. Because of this, we centered a design that allows most frets on the ukulele to be pressed. Additionally, we wanted to make it such that the device is just as fast as a human user. Hence, we determined that it should take about a second to switch to any chord. At the longest, it should take about .8s to switch to a different chord.



High level strategy





Device overview

Our approach was to create a device where 4 independent “carriages”, carrying solenoids, travel along the length of the instrument to depress frets, while the user strums. The carriages actuate via lead screws mounted to DC motors. This automates the majority of the instrument playing. At the same time, the device detects when a user has completed strumming for a particular chord, and progresses to the next position automatically. This functionality was fully realized in the final prototype, with it taking ~0.8 seconds to transition chords. The song “Riptide” by Vance Joy was successfully programmed into the prototype and played.

Constraining the device to the ukulele was challenging, but was achieved using two 3D printed brackets, one at the top of the neck and one at the fifth fret, which were clamped down. Several iterations were tested before a rigid design was produced. Most other major design decisions were about component selection. The tension of the ukulele string was a point of consideration in determining a solenoid. The required solenoid strength was determined empirically through testing. Solenoids with a form factor that sat neatly along the width of the ukulele did not produce enough force for the string to be adequately depressed, and solenoids that did so with a reasonable margin were too large to use. The sweet spot was found to be a 5N solenoid that enabled us to implement a staggered design, as seen on the final prototype.

The primary structural load consideration was the suitability of the four bearing/lead-screw assemblies. The primary load (the solenoid) is between two bearings, drastically reducing the radial load on the motor shaft, and the load on either bearing. The solenoid force was approximated as a point load on a beam with a magnitude of 5N. The reaction forces at either bearing for the worst case, where the solenoid carriage was adjacent to one of the plates, were calculated as follows:

$$\Sigma F_y = R_{Bearing1} + 5N + R_{Bearing2} = 0$$

$$\Sigma M = 5N * 10mm + R_{Bearing2} * 100mm = 0$$

$$R_{Bearing\ 2} = -0.5\ N,\ R_{Bearing\ 1} = -4.5N$$

The bearings are rated for a 300N dynamic load and flexure of the shaft was presumed to be negligible due to the minimal transverse loads and steel lead screw. Therefore, the lead screw assembly was presumed to be adequate. Due to the unusual geometry of the ukulele, the acrylic mounting plates were iteratively designed to ensure proper alignment of the bearings.

Motor selection was the function-critical decision which required the greatest effort. The primary constraint for this actuator is the torque required to accelerate the solenoid carriage up and down the length of the ukulele neck. The minimum acceleration requirement was determined to be 0.08 $\frac{m}{s^2}$after considering the desired tempo of playing (from P2).

For lead screws with a square thread profile, the torque required to exert a load applied to a nut can be calculated as:

$$T_R = \frac{F*d_m}{2}\left(\frac{l+f\pi d_m}{\pi d_m + fl}\right)$$

where F is the linear load on the lead screw nut, f is the friction coefficient, d_m = 5mm is the mean diameter, and l = 2mm is the screw lead.

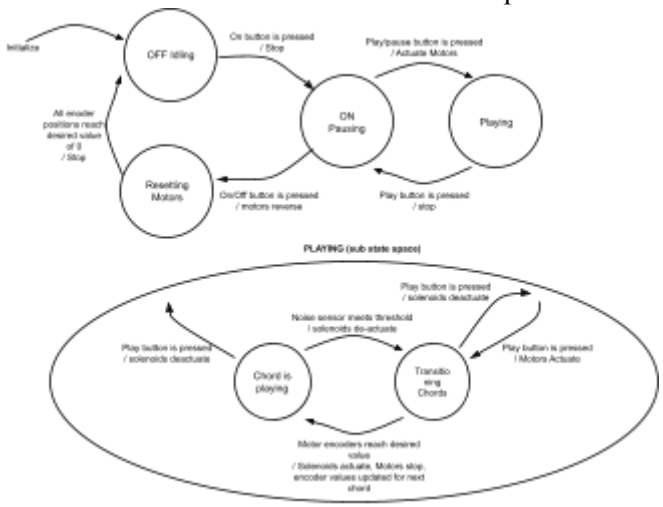
Given a conservative 100g carriage mass, a max torque of 0.25 kg·cm (0.0245 N·m), and a well-lubricated lead screw (f ~ 0), the max acceleration of the carriage is therefore:

$$A_{max} = \frac{2*0.0245\ Nm}{0.1\ kg*0.005\ m}\left(\frac{\pi*0.005\ m}{0.002\ m}\right) = 769.69\ m/s^2$$

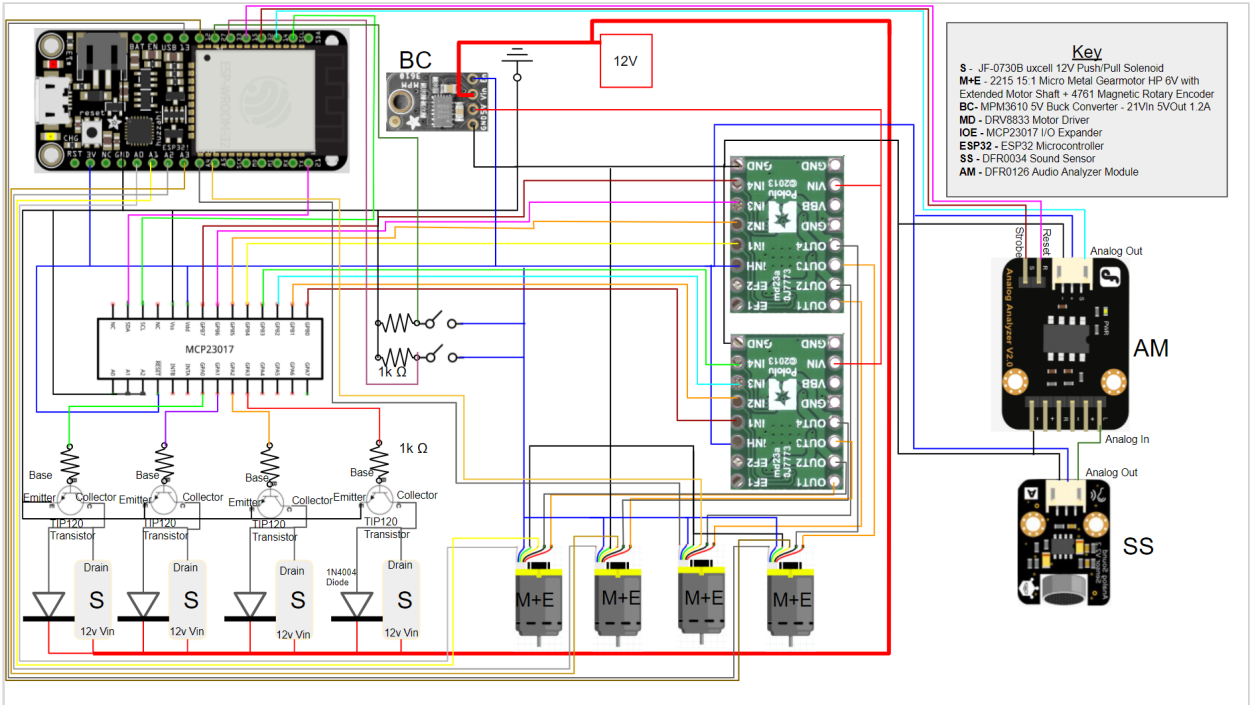
The velocity of the carriage in steady state is a function of the lead screw and motor specifications, determined to be:

$$v = 2\ \frac{mm}{rev}(lead) * 2200\frac{rev}{min}(motor) * 1/60 = 73.33\ mm/s$$

This proved to be sufficient. The 12v power supply and buck converter were sized to be able to apply stall current to all motors and actuation current to all solenoids. This also proved to be effective.



State Space Diagram



Updated Circuit Drawing *(linked for zoom in ability)*

The design for the Ukɛ was very iterative - our group met often to discuss and prototype different components, mechanisms, and designs to see what worked. Working on multiple avenues of design until it became clear that one design was superior was a good way to ensure everyone's design input was considered, and that the final design was optimal. Time permitting, the sunken-cost fallacy should be avoided - it often saves effort down the line to address problems as they arise. This is especially true for problems and geometries that are loosely or unclearly constrained, such as the ukulele.

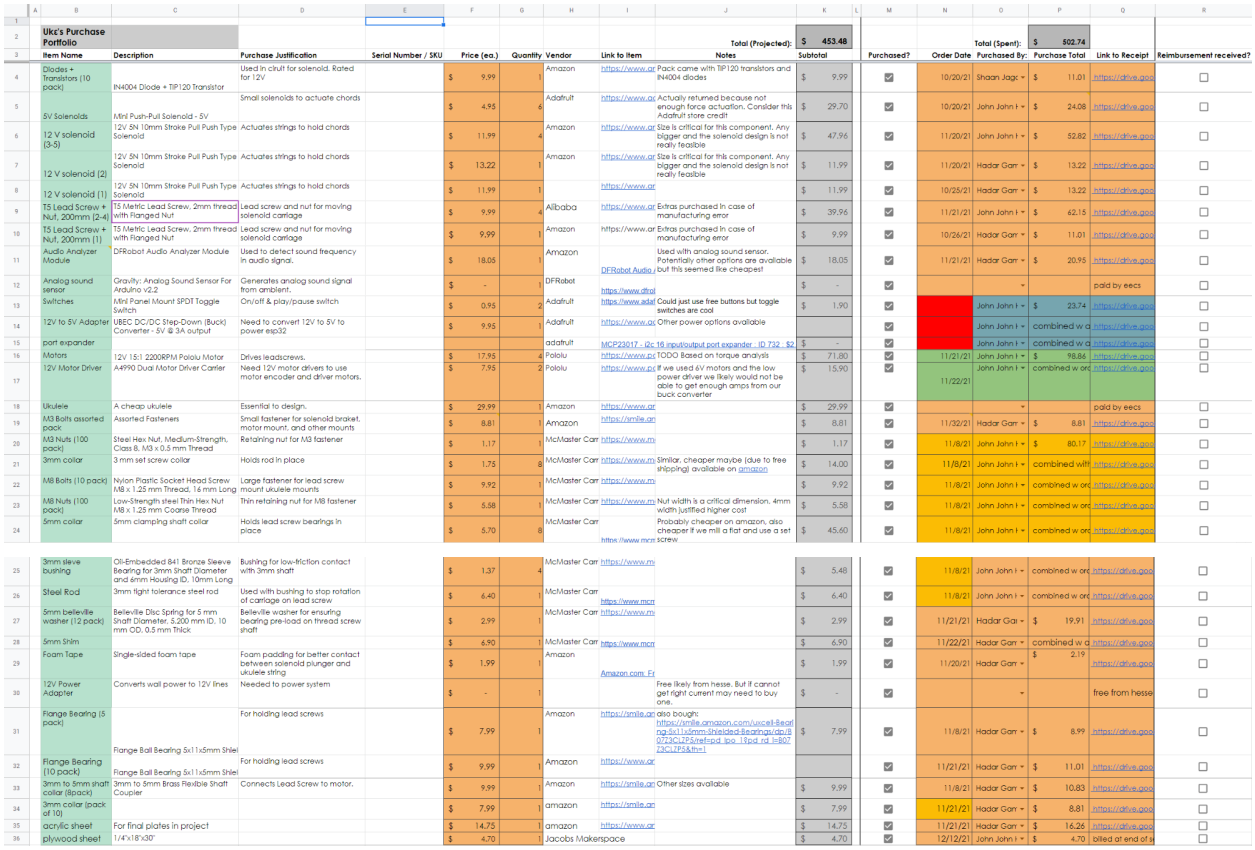
Allocating a portion of the budget for extra components was also important. Component failure occurred often in the design and assembly process, and the project would not have been completable without a safety margin. Electrical component failure and debugging took a significant amount of time. Caution when handling and testing sensitive equipment should be taken.

Supply chain issues resulted in time pressure that could have been avoided. Acquiring components, especially specialty components that aren't likely to always be in stock, should happen as early as possible.

A larger budget for this project in particular would have improved the prototype drastically. Smaller solenoids that produce the same force were out of our budget, but would have resulted in a sleeker design that would be easier to assemble. While it was easy to overlook in our initial design phases, considering manufacturability and ease-of-assembly in designing components will save a lot of effort overall.

For future students, dream big! With the mentorship from the course staff, a team of a few dedicated students can make anything become a reality.

Appendices:
Appendix A: BOM

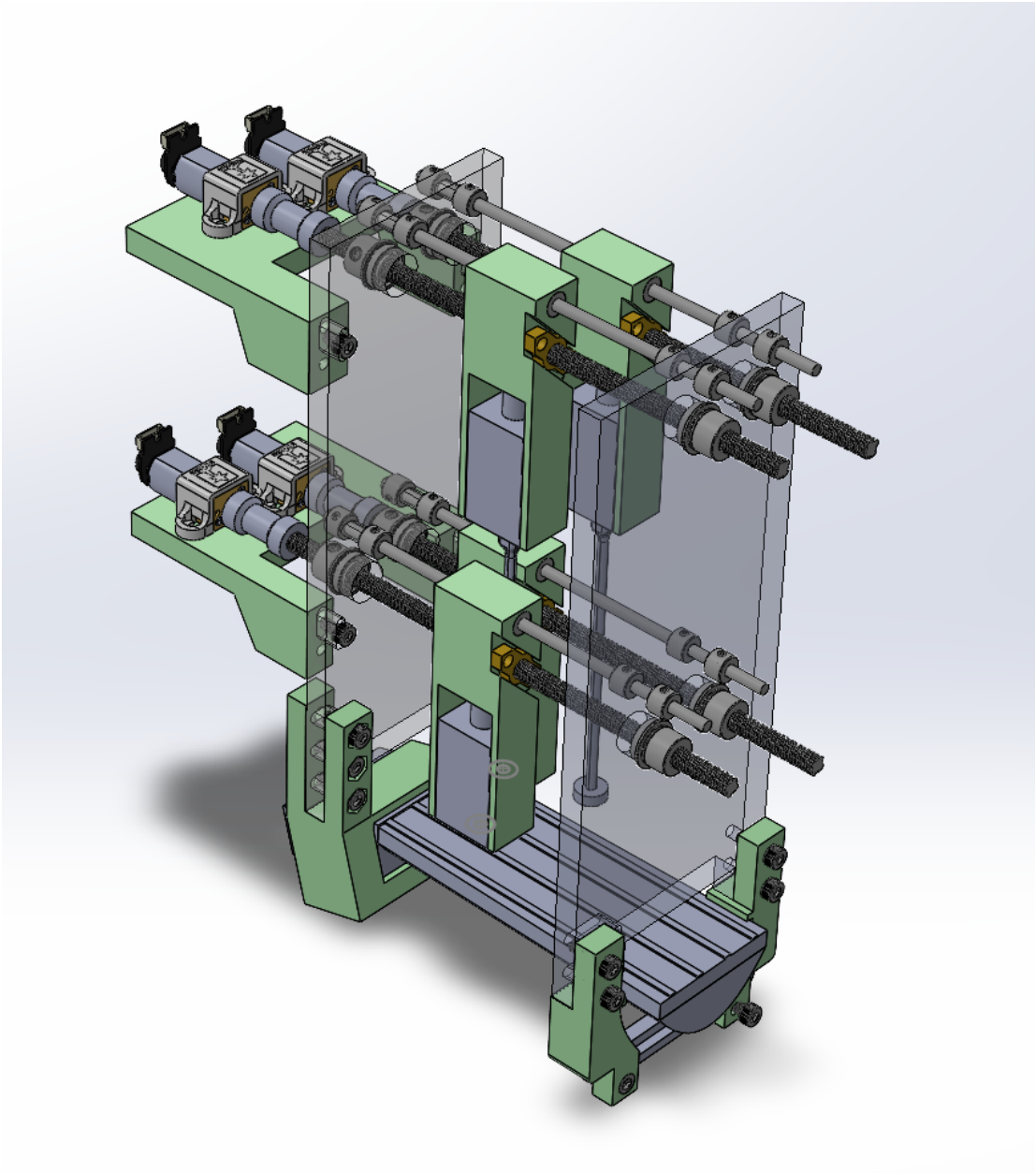


	Ukc's Purchase Portfolio								Total (Projected):	$ 453.48				Total (Spent):	$ 502.74		
	Item Name	Description	Purchase Justification	Serial Number / SKU	Price (ea.)	Quantity	Vendor	Link to Item	Notes	Subtotal	Purchased?	Order Date	Purchased By:	Purchase Total	Link to Receipt	Reimbursement received?	
	Diodes + Transistors (10 pack)	IN4004 Diode + TIP120 Transistor	Used in ciruit for solenoid. Rated for 12V		$ 9.99	1	Amazon	https://www.ar	Pack came with TIP120 transistors and IN4004 diodes	$ 9.99	☑	10/20/21	Shaan Jagc	$ 11.01	https://drive.goo	☐	
	5V Solenoids	Mini Push-Pull Solenoid - 5V	Small solenoids to actuate chords		$ 4.95	6	Adafruit	https://www.ad	Actually returned because not enough force actuation. Consider this Adafruit store credit	$ 29.70	☑	10/20/21	John John H	$ 24.08	https://drive.goo	☐	
	12 V solenoid (3-5)	12V 5N 10mm Stroke Pull Push Type Solenoid	Actuates strings to hold chords		$ 11.99	4	Amazon	https://www.ar	Size is critical for this component. Any bigger and the solenoid design is not really feasible	$ 47.96	☑	11/20/21	John John H	$ 52.82	https://drive.goo	☐	
	12 V solenoid (2)	12V 5N 10mm Stroke Pull Push Type Solenoid	Actuates strings to hold chords		$ 13.22	1	Amazon	https://www.ar	Size is critical for this component. Any bigger and the solenoid design is not really feasible	$ 11.99	☑	11/20/21	Hadar Garr	$ 13.22	https://drive.goo	☐	
	12 V solenoid (1)	12V 5N 10mm Stroke Pull Push Type Solenoid	Actuates strings to hold chords		$ 11.99	1		https://www.ar		$ 11.99	☑	10/25/21	Hadar Garr	$ 13.22	https://drive.goo	☐	
	T5 Lead Screw + Nut, 200mm (2-4)	T5 Metric Lead Screw, 2mm thread with Flanged Nut	Lead screw and nut for moving solenoid carriage		$ 9.99	4	Alibaba	https://www.ar	Extras purchased in case of manufacturing error	$ 39.96	☑	11/21/21	John John H	$ 62.15	https://drive.goo	☐	
	T5 Lead Screw + Nut, 200mm (1)	T5 Metric Lead Screw, 2mm thread with Flanged Nut	Lead screw and nut for moving solenoid carriage		$ 9.99	1	Amazon	https://www.ar	Extras purchased in case of manufacturing error	$ 9.99	☑	10/26/21	Hadar Garr	$ 11.01	https://drive.goo	☐	
	Audio Analyzer Module	DFRobot Audio Analyzer Module	Used to detect sound frequency in audio signal.		$ 18.05		Amazon	DFRobot Audio /	Used with analog sound sensor. Potentially other options are available but this seemed like cheapest	$ 18.05	☑	11/21/21	Hadar Garr	$ 20.95	https://drive.goo	☐	
	Analog sound sensor	Gravity: Analog Sound Sensor For Arduino v2.2	Generates analog sound signal from ambient.		$ -		DFRobot	https://www.dfrol		$ -	☑				paid by eecs	☐	
	Switches	Mini Panel Mount SPDT Toggle Switch	On/off & play/pause switch		$ 0.95	2	Adafruit	https://www.ad	Could just use free buttons but toggle switches are cool	$ 1.90	☑		John John H	$ 23.74	https://drive.goo	☐	
	12V to 5V Adapter	UBEC DC/DC Step-Down (Buck) Converter - 5V @ 3A output	Need to convert 12V to 5V to power esp32		$ 9.95	1	Adafruit	https://www.ad	Other power options available		☑		John John H	combined w d	https://drive.goo	☐	
	port expander						adafruit	MCP23017 - I2c 16 input/output port expander - ID 732 - $2		$ -	☑		John John H	combined w d	https://drive.goo	☐	
	Motors	12V 15:1 2200RPM Pololu Motor	Drives leadscrews.		$ 17.95	4	Pololu	https://www.pc	TODO Based on torque analysis	$ 71.80	☑	11/21/21	John John H	$ 98.86	https://drive.goo	☐	
	12V Motor Driver	A4990 Dual Motor Driver Carrier	Need 12V motor drivers to use motor encoder and driver motors.		$ 7.95	2	Pololu	https://www.pc	If we used 6V motors and the low power driver we likely would not be able to get enough amps from our buck converter	$ 15.90	☑	11/22/21	John John H	combined w ord	https://drive.goo	☐	
	Ukulele	A cheap ukulele	Essential to design.		$ 29.99	1	Amazon	https://www.ar		$ 29.99	☑				paid by eecs	☐	
	M3 Bolts assorted pack	Assorted Fasteners	Small fastener for solenoid braket, motor mount, and other mounts		$ 8.81	1	Amazon	https://smile.ar		$ 8.81	☑	11/32/21	Hadar Garr	$ 8.81	https://drive.goo	☐	
	M3 Nuts (100 pack)	Steel Hex Nut, Medium-Strength, Class 8, M3 x 0.5 mm Thread	Retaining nut for M3 fastener		$ 1.17	1	McMaster Carr	https://www.m		$ 1.17	☑	11/8/21	John John H	$ 80.17	https://drive.goo	☐	
	3mm collar	3 mm set screw collar	Holds rod in place		$ 1.75	8	McMaster Carr	https://www.m	Similar, cheaper maybe (due to free shipping) available on amazon	$ 14.00	☑	11/8/21	John John H	combined with	https://drive.goo	☐	
	M8 Bolts (10 pack)	Nylon Plastic Socket Head Screw M8 x 1.25 mm Thread, 16 mm Long	Large fastener for lead screw mount ukulele mounts		$ 9.92	1	McMaster Carr	https://www.m		$ 9.92	☑	11/8/21	John John H	combined w ord	https://drive.goo	☐	
	M8 Nuts (100 pack)	Low-Strength steel Thin Hex Nut M8 x 1.25 mm Coarse Thread	Thin retaining nut for M8 fastener		$ 5.58	1	McMaster Carr	https://www.m	Nut width is a critical dimension. 4mm width justified higher cost	$ 5.58	☑	11/8/21	John John H	combined w ord	https://drive.goo	☐	
	5mm collar	5mm clamping shaft collar	Holds lead screw bearings in place		$ 5.70	8	McMaster Carr	https://www.mcm	Probably cheaper on amazon, also cheaper if we mill a flat and use a set screw	$ 45.60	☑	11/8/21	John John H	combined w ord	https://drive.goo	☐	
	3mm sleve bushing	Oil-Embedded 841 Bronze Sleeve Bearing for 3mm Shaft Diameter and 6mm Housing ID, 10mm Long	Bushing for low-friction contact with 3mm shaft		$ 1.37	4	McMaster Carr	https://www.m		$ 5.48	☑	11/8/21	John John H	combined w ord	https://drive.goo	☐	
	Steel Rod	3mm tight tolerance steel rod	Used with bushing to stop rotation of carriage on lead screw		$ 6.40	1	McMaster Carr	https://www.mcm		$ 6.40	☑	11/8/21	John John H	combined w ord	https://drive.goo	☐	
	5mm belleville washer (12 pack)	Belleville Disc Spring for 5 mm Shaft Diameter, 5.200 mm ID, 10 mm OD, 0.5 mm Thick	Belleville washer for ensuring bearing pre-load on thread screw shaft		$ 2.99	1	McMaster Carr	https://www.m		$ 2.99	☑	11/21/21	Hadar Garr	$ 19.91	https://drive.goo	☐	
	5mm Shim				$ 6.90	1	McMaster Carr	https://www.mcm		$ 6.90	☑	11/22/21	Hadar Garr	combined w d	https://drive.goo	☐	
	Foam Tape	Single-sided foam tape	Foam padding for better contact between solenoid plunger and ukulele string		$ 1.99		Amazon	Amazon.com: Fr		$ 1.99	☑	11/20/21	Hadar Garr	$ 2.19	https://drive.goo	☐	
	12V Power Adapter	Converts wall power to 12V lines	Needed to power system		$ -	1			Free likely from hesse. But if cannot get right current may need to buy one.	$ -	☑				free from hesse	☐	
	Flange Bearing (5 pack)	Flange Ball Bearing 5x11x5mm Shiel	For holding lead screws		$ 7.99	1	Amazon	https://smile.ar	also bough: https://smile.amazon.com/uxcell-Bearing-5x11x5mm-Shielded-Bearings/dp/B0723CLZF5/ref=pd_lpo_19?pd_rd_i=B07Z3CLZF5&th=1	$ 7.99	☑	11/8/21	Hadar Garr	$ 8.99	https://drive.goo	☐	
	Flange Bearing (10 pack)	Flange Ball Bearing 5x11x5mm Shiel	For holding lead screws		$ 9.99	1	Amazon	https://www.ar			☑	11/21/21	Hadar Garr	$ 11.01	https://drive.goo	☐	
	3mm to 5mm shaft collar (8pack)	3mm to 5mm Brass Flexible Shaft Coupler	Connects Lead Screw to motor.		$ 9.99	1	Amazon	https://smile.ar	Other sizes available	$ 9.99	☑	11/8/21	Hadar Garr	$ 10.83	https://drive.goo	☐	
	3mm collar (pack of 10)				$ 7.99	1	amazon	https://smile.ar		$ 7.99	☑	11/21/21	Hadar Garr	$ 8.81	https://drive.goo	☐	
	acrylic sheet	For final plates in project			$ 14.75	1	amazon	https://www.ar		$ 14.75	☑	11/21/21	Hadar Garr	$ 16.26	https://drive.goo	☐	
	plywood sheet	1/4"x18"x30"			$ 4.70	1	Jacobs Makerspace			$ 4.70	☑	12/12/21	John John H	$ 4.70	billed at end of s	☐	

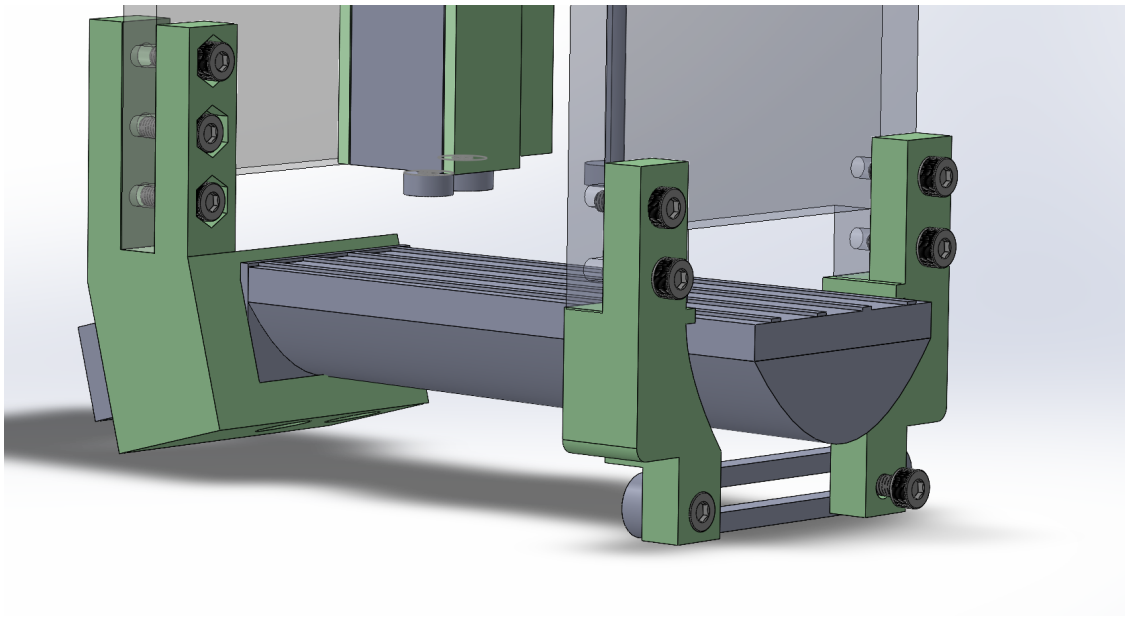
ASME Maker Grant Purchase Portfolio - Group 23

https://docs.google.com/spreadsheets/d/1uyjSxqoBfZSKuT4vhcuw_a5z0_TwZLJevIdevdjqj3I/edit?usp=sharing

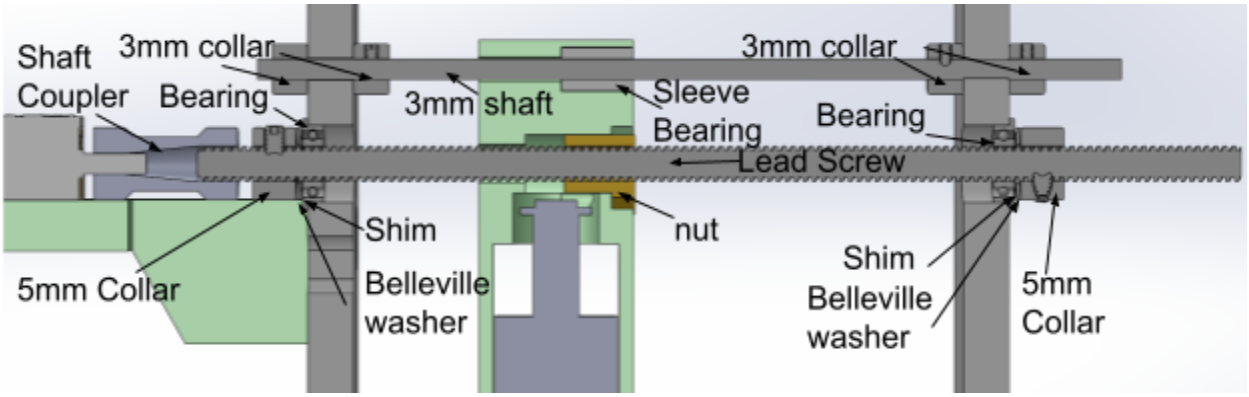
Appendix B: CAD w/ mechanical transmission



Overview



Top and bottom mounts



Lead screw mechanical transmission

Appendix C: Screenshots of code

```
#include <Adafruit_MCP23017.h>
#include <AudioAnalyzer.h>
#include <ESP32Encoder.h>

#define BTNon 15
#define BTNplay 13
#define SOL_PIN0 7
#define SOL_PIN1 6
#define SOL_PIN2 5
#define SOL_PIN3 4

#define motor1_channel1 8 //blue
#define motor1_channel2 9
#define motor2_channel1 10 //green
#define motor2_channel2 11
#define motor3_channel1 12 //yellow
#define motor3_channel2 13
#define motor4_channel1 14 //brown
#define motor4_channel2 15

Adafruit_MCP23017 mcp;
ESP32Encoder encoder1;
ESP32Encoder encoder2;
ESP32Encoder encoder3;
ESP32Encoder encoder4;

Analyzer Audio = Analyzer(14, 32, 12); //Strobe pin ->14  RST pin ->32 Analog Pin ->

int nut = 0;
int fret0 = -15;
int fret1 = 610;
int fret2 = 1620;
int fret3 = 2460;
int fret4 = 3366;
int fret5 = 4194;

int chords[10][4] = {
  /*A*/{fret2, fret0, fret0, fret0},/*G*/{fret0, fret2, fret3, fret2},/*C*/{fret0, fret0, fret0, fret3},
  /*A*/{fret2, fret0, fret0, fret0},/*G*/{fret0, fret2, fret3, fret2},/*C*/{fret0, fret0, fret0, fret3},
  /*A*/{fret2, fret0, fret0, fret0},/*G*/{fret0, fret2, fret3, fret2},/*C*/{fret0, fret0, fret0, fret3},
  /*F*/{fret2, fret0, fret1, fret0}
};

bool actuate[10][4] = {
  /*A*/{true, false, false, false},/*G*/{false, true, true, true},/*C*/ {false, false, false, true},
  /*A*/{true, false, false, false},/*G*/{false, true, true, true},/*C*/ {false, false, false, true},
  /*A*/{true, false, false, false},/*G*/{false, true, true, true},/*C*/ {false, false, false, true},
  /*F*/{true, false, true, false}
};

int w = 0;
int x = 0;
int y = 0;
int z = 0;

int state = 0;
int FreqVal[7]; //set array of values of size 7
int highthresh = 3100;
int lowthresh = 750; //adjust depending on ambient sound levels
bool highpass = false;


volatile bool motor1on = false;
volatile bool motor1onreverse = false;
volatile bool motor2on = false;
volatile bool motor2onreverse = false;
volatile bool motor3on = false;
volatile bool motor3onreverse = false;
```

```
volatile bool motor4on = false;
volatile bool motor4onreverse = false;
volatile int des_enc1_val = chords[w][0];
volatile int des_enc2_val = chords[x][1];
volatile int des_enc3_val = chords[y][2];
volatile int des_enc4_val = chords[z][3];


//Setup interrupt variables ----------------------------

volatile bool ONtimerDone = false;
volatile bool PLAYtimerDone = false;
volatile bool ONButtonIsPressed = false;
volatile bool PLAYButtonIsPressed = false;

//interrupts init
hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

//Initialization -----------------------------------
void IRAM_ATTR onTime0() { //ON BUTTON DEBOUNCING
  portENTER_CRITICAL_ISR(&timerMux0);
  ONtimerDone = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
  timerStop(timer0);
}
void ONTimerInterruptInit() {
  timer0 = timerBegin(0, 80, true);
  timerAttachInterrupt(timer0, &onTime0, true);
  timerAlarmWrite(timer0, 1000000, true);
  timerAlarmEnable(timer0);
}
void IRAM_ATTR ONbtntmr() {
  if (ONtimerDone && (state == 0 || state == 1))
  {
    ONButtonIsPressed = true;
  }
  ONTimerInterruptInit();
}

void IRAM_ATTR onTime1() { //PLAY BUTTON DEBOUNCING
  portENTER_CRITICAL_ISR(&timerMux1);
  PLAYtimerDone = true;//the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
  timerStop(timer1);
}

void PLAYTimerInterruptInit() {
  timer1 = timerBegin(0, 80, true);
  timerAttachInterrupt(timer1, &onTime1, true);
  timerAlarmWrite(timer1, 1000000, true);
  timerAlarmEnable(timer1);
}
void IRAM_ATTR PLAYbtntmr() {
  if (PLAYtimerDone)
  {
    if ((state == 0) || (state == 2)) {
      PLAYButtonIsPressed = false;
    } else if ((state == 1) || (state == 3) || (state == 4)) {
      PLAYButtonIsPressed = true;
    } else {
      PLAYButtonIsPressed = false;
    }
  }
  PLAYTimerInterruptInit();
}
```

```
137
138 //Setup ----------------------------------------------------------
139 void setup() {
140   // put your setup code here, to run once:
141   pinMode(BTNon, INPUT); // configures the specified pin to behave either as an input or an output
142   pinMode(BTNplay, INPUT);
143   attachInterrupt(BTNon, ONbtntmr, RISING);  // set the "BTN" pin
144   attachInterrupt(BTNplay, PLAYbtntmr, RISING);  // set the "BTN" pin
145
146   Serial.begin(115200);
147
148   Audio.Init();//Init audio module
149   mcp.begin();// "Start" the mcp object
150   mcp.pinMode(motor1_channel1, OUTPUT); //set motor pins
151   mcp.pinMode(motor1_channel2, OUTPUT);
152   mcp.pinMode(motor2_channel1, OUTPUT);
153   mcp.pinMode(motor2_channel2, OUTPUT);
154   mcp.pinMode(motor3_channel1, OUTPUT);
155   mcp.pinMode(motor3_channel2, OUTPUT);
156   mcp.pinMode(motor4_channel1, OUTPUT);
157   mcp.pinMode(motor4_channel2, OUTPUT);
158   mcp.digitalWrite(motor1_channel1, LOW); //sets motor to low by default
159   mcp.digitalWrite(motor1_channel2, LOW);
160   mcp.digitalWrite(motor2_channel1, LOW);
161   mcp.digitalWrite(motor2_channel2, LOW);
162   mcp.digitalWrite(motor3_channel1, LOW);
163   mcp.digitalWrite(motor3_channel2, LOW);
164   mcp.digitalWrite(motor4_channel1, LOW);
165   mcp.digitalWrite(motor4_channel2, LOW);
166
167   mcp.pinMode(SOL_PIN0, OUTPUT);
168   mcp.pinMode(SOL_PIN1, OUTPUT);
169   mcp.pinMode(SOL_PIN2, OUTPUT);
170   mcp.pinMode(SOL_PIN3, OUTPUT);
171   mcp.digitalWrite(SOL_PIN0, LOW); // sets the initial state of SOL as turned-off
172   mcp.digitalWrite(SOL_PIN1, LOW); // sets the initial state of SOL as turned-off
173   mcp.digitalWrite(SOL_PIN2, LOW); // sets the initial state of SOL as turned-off
174   mcp.digitalWrite(SOL_PIN3, LOW); // sets the initial state of SOL as turned-off
175
176   ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resistors
177   encoder1.attachHalfQuad(26, 25); // Attache pins for use as encoder pins
178   encoder2.attachHalfQuad(34, 39);
179   encoder3.attachHalfQuad(36, 4);
180   encoder4.attachHalfQuad(33, 27);
181   encoder1.setCount(0);  // set starting count value after attaching
182   encoder2.setCount(0);
183   encoder3.setCount(0);
184   encoder4.setCount(0);
185   encoder1.clearCount( );
186   encoder2.clearCount( );
187   encoder3.clearCount( );
188   encoder4.clearCount( );
189
190   // initilize timer
191   timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
192   timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggered
193   timerAlarmWrite(timer0, 10000, true); // 2000000 * 1 us = 10 ms, autoreload true
194
195   timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns * TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
196   timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
197   timerAlarmWrite(timer1, 10000, true); // 10000 * 1 us = 10 ms, autoreload true
198
199   // at least enable the timer alarms
200   timerAlarmEnable(timer0); // enable
201   timerAlarmEnable(timer1); // enable
202 }
203
204 ////////////////////////////////////////////////////////////MAIN LOOP ////////////////////////////////////////////////////////////----------------------------------
```

```
void loop() {
  // put your main code here, to run repeatedly:
  Audio.ReadFreq(FreqVal);

  switch (state) {
    case 0 : // CURRENTLY IN OFF STATE
      Serial.print("Encl Val: ");
      Serial.print(((int32_t)encoder1.getCount()));
      Serial.print("Enc2 Val: ");
      Serial.print(((int32_t)encoder2.getCount()));
      Serial.print("Enc3 Val: ");
      Serial.print(((int32_t)encoder3.getCount()));
      Serial.print("Enc4 Val: ");
      Serial.print(((int32_t)encoder4.getCount()));
      Serial.print(" State: 0, OFF\n");
      if (CheckOnButtonPressed()) {
        state = 1; // SWITCH TO PAUSE STATE
      }
      StopActuatingService(1);
      StopActuatingService(2);
      StopActuatingService(3);
      StopActuatingService(4);
      break;

    case 1 : // CURRENTLY IN PAUSE
      Serial.print("Encl Val: ");
      Serial.print(((int32_t)encoder1.getCount()));
      Serial.print("Enc2 Val: ");
      Serial.print(((int32_t)encoder2.getCount()));
      Serial.print("Enc3 Val: ");
      Serial.print(((int32_t)encoder3.getCount()));
      Serial.print("Enc4 Val: ");
      Serial.print(((int32_t)encoder4.getCount()));
      Serial.print(" State: 1, PAUSE\n");
      if (CheckPlayButtonPressed()) {
        state = 3; //Chord transitioning state
        NextChordService(1);
        NextChordService(2);
        NextChordService(3);
        NextChordService(4);
      }
      if (CheckOnButtonPressed()) {
        state = 2; // SWITCH TO RESETTING MOTORS
        ResetMotorsService(1);
        ResetMotorsService(2);
        ResetMotorsService(3);
        ResetMotorsService(4);
      }
      break;

    case 2 : // RESETTING MOTORS STATE
      Serial.print("Encl Val: ");
      Serial.print(((int32_t)encoder1.getCount()));
      Serial.print("Enc2 Val: ");
      Serial.print(((int32_t)encoder2.getCount()));
      Serial.print("Enc3 Val: ");
      Serial.print(((int32_t)encoder3.getCount()));
      Serial.print("Enc4 Val: ");
      Serial.print(((int32_t)encoder4.getCount()));
      Serial.print(" State: 2, RESETTING MOTOR\n");
      if (CheckEncAtZero(1)) {
        StopActuatingService(1);
      }
      if (CheckEncAtZero(2)) {
        StopActuatingService(2);
      }
      if (CheckEncAtZero(3)) {
        StopActuatingService(3);
```

```
    }
    if (CheckEncAtZero(4)) {
      StopActuatingService(4);
    }
    if (!motorlon && !motorlonreverse && !motor2on && !motor2onreverse && !motor3on && !motor3onreverse && !motor4on && !motor4onreverse) {
      //if for some reason this isn't working try to fix the tolerence in check enc at zero
      state = 0;
      Serial.print("\n RESET MOTORS DONE -- DONE \n");
    }
    break;

  case 3 : // CHORD TRANSITIONING STATE
    Serial.print("Enc1 Val: ");
    Serial.print(((int32_t)encoder1.getCount()));
    Serial.print("Enc2 Val: ");
    Serial.print(((int32_t)encoder2.getCount()));
    Serial.print("Enc3 Val: ");
    Serial.print(((int32_t)encoder3.getCount()));
    Serial.print("Enc4 Val: ");
    Serial.print(((int32_t)encoder4.getCount()));
    Serial.print(" State: 3, TRANSITIONING\n");
    if (CheckEncAtDes(1)) {
      stopMotorResponse(1);

    }
    if (CheckEncAtDes(2)) {
      stopMotorResponse(2);

    }
    if (CheckEncAtDes(3)) {
      stopMotorResponse(3);

    }
    if (CheckEncAtDes(4)) {
      stopMotorResponse(4);

    }
    if (!motorlon && !motorlonreverse && !motor2on && !motor2onreverse && !motor3on && !motor3onreverse && !motor4on && !motor4onreverse) {
      //also may need to check tolernces with new motor
      Serial.print("\n NEXT CHORD SERVICE -- DONE \n");
      BeginChordService(1);
      BeginChordService(2);
      BeginChordService(3);
      BeginChordService(4);
      state = 4;
    }

    if (CheckPlayButtonPressed()) {
      StopActuatingService(1);
      StopActuatingService(2);
      StopActuatingService(3);
      StopActuatingService(4);
      state = 1; //Chord transitioning state
    }
    break;

  case 4 : // CHORD PLAYING STATE
    Serial.print("Enc1 Val: ");
    Serial.print(((int32_t)encoder1.getCount()));
    Serial.print("Enc2 Val: ");
    Serial.print(((int32_t)encoder2.getCount()));
    Serial.print("Enc3 Val: ");
    Serial.print(((int32_t)encoder3.getCount()));
    Serial.print("Enc4 Val: ");
    Serial.print(((int32_t)encoder4.getCount()));
    Serial.print(" State: 4, CHORD PLAYING\n");
    if (CheckAudioAtThresh()) {
      state = 3;
```

```
341         NextChordService(1);
342         NextChordService(2);
343         NextChordService(3);
344         NextChordService(4);
345       }
346       if (CheckPlayButtonPressed()) {
347         state = 1; //Chord transitioning state
348         StopActuatingService(1);
349         StopActuatingService(2);
350         StopActuatingService(3);
351         StopActuatingService(4);
352       }
353       break;
354
355   }
356 }
357 ///////////////// EVENT CHECKERS BASED OFF PSEUDO CODE ////////////////////
358
359 bool CheckOnButtonPressed() {
360
361   if (ONtimerDone && ONButtonIsPressed && (state == 1 || state == 0) ) {
362     ONtimerDone = false;
363     ONTimerInterruptInit();
364     ONButtonIsPressed = false;
365     return true;
366   }
367   else {
368     return false;
369   }
370 }
371
372
373 bool CheckPlayButtonPressed() {
374
375   if (PLAYtimerDone && PLAYButtonIsPressed) {
376     PLAYButtonIsPressed = false;
377     PLAYtimerDone = false;
378     PLAYTimerInterruptInit();
379     return true;
380   }
381   else {
382     return false;
383   }
384 }
385
386 bool CheckEncAtDes (int encnum) {
387   //Did all Encoders reach the desired value?
388   //  If MotorsAtDesValue is true
389   //    Turn flag off and return true
390   //  Else, return false
391   //  Serial.print("Check ENCODER");
392   if (encnum == 1) {
393     if ((motor1on == true) && ((int32_t)encoder1.getCount() < des_enc1_val - 75) ) {
394       return false;
395     }
396     if ((motor1onreverse == true) && ((int32_t)encoder1.getCount() > des_enc1_val + 75)) {
397       return false;
398     }
399     else {
400       return true;
401     }
402   }
403   if (encnum == 2) {
404     if ((motor2on == true) && ((int32_t)encoder2.getCount() < des_enc2_val - 75) ) {
405       return false;
406     }
407     if ((motor2onreverse == true) && ((int32_t)encoder2.getCount() > des_enc2_val + 75)) {
408       return false;
```

```
    }
    else {
      return true;
    }
  }
  if (encnum == 3) {
    if ((motor3on == true) && ((int32_t)encoder3.getCount() < des_enc3_val - 75) ) {
      return false;
    }
    if ((motor3onreverse == true) && ((int32_t)encoder3.getCount() > des_enc3_val + 75)) {
      return false;
    }
    else {
      return true;
    }
  }
  if (encnum == 4) {
    if ((motor4on == true) && ((int32_t)encoder4.getCount() < des_enc4_val - 75) ) {
      return false;
    }
    if ((motor4onreverse == true) && ((int32_t)encoder4.getCount() > des_enc4_val + 75)) {
      return false;
    }
    else {
      return true;
    }
  }
}

bool CheckEncAtZero (int encnum) {
  //Did all Encoders reach the zero value?
  //  If MotorsAtDesValue is true
  //    Turn flag off and return true
  //  Else, return false
  //  Serial.print("Check ENCODER");
  if (encnum == 1) {
    if ((int32_t)encoder1.getCount() > 75) {
      return false;
    }
    if ((int32_t)encoder1.getCount() < -75) {
      return false;
    }
    else {
      return true;
    }
  }
  if (encnum == 2) {
    if ((int32_t)encoder2.getCount() > 75) {
      return false;
    }
    if ((int32_t)encoder2.getCount() < -75) {
      return false;
    }
    else {
      return true;
    }
  }
  if (encnum == 3) {
    if ((int32_t)encoder3.getCount() > 75) {
      return false;
    }
    if ((int32_t)encoder3.getCount() < -75) {
      return false;
    }
    else {
      return true;
    }
  }
```

```
  }
  if (encnum == 4) {
    if ((int32_t)encoder4.getCount() > 75) {
      return false;
    }
    if ((int32_t)encoder4.getCount() < -75) {
      return false;
    }
    else {
      return true;
    }
  }
}


bool CheckAudioAtThresh () {
  //Did the Audio Sensor reach the threshold?
  //  If AudioAtThresh is true
  //    Turn flag off and return true
  //  Else, return false
  //  Serial.print("Check AUDIO THRESH");

  int audio_val = FreqVal[1];
  Serial.print("Audio value = ");
  Serial.println(audio_val);
  if (state == 4 && audio_val > highthresh)
  {
    highpass = true;
  }
  //Serial.print(audio_val);
  if (state == 4 && highpass && audio_val < lowthresh) {
    highpass = false;
    return true;
  }
  else {
    return false;
  }
}

//////////////////////  MOTOR HELPER FUNCTIONS //////////////////////////
void stopMotorResponse(int motornum) {
  //ledcWrite(ledChannel_2, 0);
  //ledcWrite(ledChannel_1, LOW);
  if (motornum == 1) {
    mcp.digitalWrite(motor1_channel1, LOW);
    mcp.digitalWrite(motor1_channel2, LOW);
    motor1on = false;
    motor1onreverse = false;
  }
  if (motornum == 2) {
    mcp.digitalWrite(motor2_channel1, LOW);
    mcp.digitalWrite(motor2_channel2, LOW);
    motor2on = false;
    motor2onreverse = false;
  }
  if (motornum == 3) {
    mcp.digitalWrite(motor3_channel1, LOW);
    mcp.digitalWrite(motor3_channel2, LOW);
    motor3on = false;
    motor3onreverse = false;
  }
  if (motornum == 4) {
    mcp.digitalWrite(motor4_channel1, LOW);
    mcp.digitalWrite(motor4_channel2, LOW);
    motor4on = false;
    motor4onreverse = false;
  }
}
```

```
544
545 void startReverseMotorResponse(int motornum) {
546   if (motornum == 1) {
547     mcp.digitalWrite(motor1_channel2, LOW);
548     mcp.digitalWrite(motor1_channel1, HIGH);
549     motor1onreverse = true;
550   }
551   if (motornum == 2) {
552     mcp.digitalWrite(motor2_channel2, LOW);
553     mcp.digitalWrite(motor2_channel1, HIGH);
554     motor2onreverse = true;
555   }
556   if (motornum == 3) {
557     mcp.digitalWrite(motor3_channel2, LOW);
558     mcp.digitalWrite(motor3_channel1, HIGH);
559     motor3onreverse = true;
560   }
561   if (motornum == 4) {
562     mcp.digitalWrite(motor4_channel2, LOW);
563     mcp.digitalWrite(motor4_channel1, HIGH);
564     motor4onreverse = true;
565   }
566 }
567
568 void startMotorResponse(int motornum) {
569   if (motornum == 1) {
570     mcp.digitalWrite(motor1_channel1, LOW);
571     mcp.digitalWrite(motor1_channel2, HIGH);
572     motor1on = true;
573   }
574   if (motornum == 2) {
575     mcp.digitalWrite(motor2_channel1, LOW);
576     mcp.digitalWrite(motor2_channel2, HIGH);
577     motor2on = true;
578   }
579   if (motornum == 3) {
580     mcp.digitalWrite(motor3_channel1, LOW);
581     mcp.digitalWrite(motor3_channel2, HIGH);
582     motor3on = true;
583   }
584   if (motornum == 4) {
585     mcp.digitalWrite(motor4_channel1, LOW);
586     mcp.digitalWrite(motor4_channel2, HIGH);
587     motor4on = true;
588   }
589 }
590
591
592 /////////////// EVENT SERVICES BASED FROM PSEUDO CODE ///////////////////////
593
594 void StopActuatingService(int motornum) {
595   //Turn all solenoids to LOW & Stop all Motors
596   if (motornum == 1) {
597     stopMotorResponse(motornum);
598     mcp.digitalWrite(SOL_PIN0, LOW);
599   }
600   if (motornum == 2) {
601     stopMotorResponse(motornum);
602     mcp.digitalWrite(SOL_PIN1, LOW);
603   }
604   if (motornum == 3) {
605     stopMotorResponse(motornum);
606     mcp.digitalWrite(SOL_PIN2, LOW);
607   }
608   if (motornum == 4) {
609     stopMotorResponse(motornum);
610     mcp.digitalWrite(SOL_PIN3, LOW);
611   }
```

```
  //Serial.print("\n stop actuating service -- done \n");
}

void ResetMotorsService(int motornum) {
  //All desired encoder values set to 0
  //Turn all solenoids to LOW
  //Turn all motors on in reverse
  w = 0;
  x = 0;
  y = 0;
  z = 0;
  if (motornum == 1) {
    des_enc1_val = chords[w][motornum-1];
    mcp.digitalWrite(SOL_PIN0, LOW);

    if (((int32_t)encoder1.getCount() > 0 )) {
      startReverseMotorResponse(motornum); //MOVING BACKWARDS
    }
    if (((int32_t)encoder1.getCount() < 0 )) {
      startMotorResponse(motornum); //MOVING FORWARDS
    }
  }
  if (motornum == 2) {
    des_enc2_val = chords[x][motornum-1];
    mcp.digitalWrite(SOL_PIN1, LOW);

    if (((int32_t)encoder2.getCount() > 0 )) {
      startReverseMotorResponse(motornum); //MOVING BACKWARDS
    }
    if (((int32_t)encoder2.getCount() < 0 )) {
      startMotorResponse(motornum); //MOVING FORWARDS
    }
  }
  if (motornum == 3) {
    des_enc3_val = chords[y][motornum-1];
    mcp.digitalWrite(SOL_PIN2, LOW);

    if (((int32_t)encoder3.getCount() > 0 )) {
      startReverseMotorResponse(motornum); //MOVING BACKWARDS
    }
    if (((int32_t)encoder3.getCount() < 0 )) {
      startMotorResponse(motornum); //MOVING FORWARDS
    }
  }
  if (motornum == 4) {
    des_enc4_val = chords[z][motornum-1];
    mcp.digitalWrite(SOL_PIN3, LOW);

    if (((int32_t)encoder4.getCount() > 0 )) {
      startReverseMotorResponse(motornum); //MOVING BACKWARDS
    }
    if (((int32_t)encoder4.getCount() < 0 )) {
      startMotorResponse(motornum); //MOVING FORWARDS
    }
  }
}

void BeginChordService(int encnum) {
  //  Turn desired solenoids to HIGH
  //  Set desired encoder values to next chord positions
  //  Set desired solenoids to next chord solenoids
  if (encnum == 1) {
    if (actuate[w][encnum-1]){
      mcp.digitalWrite(SOL_PIN0, HIGH);
    }
    //change to set desired next chord position
    w++;
    if (w >= 10) {
```

```
      w = 0;
    }
    des_enc1_val = chords[w][encnum-1];
    Serial.print("\n begin chord service -- done \n");
  }
  if (encnum == 2) {
    if (actuate[x][encnum-1]){
      mcp.digitalWrite(SOL_PIN1, HIGH);
    }

    //change to set desired next chord position
    x++;
    if (x >= 10) {
      x = 0;
    }
    des_enc2_val = chords[x][encnum-1];
    Serial.print("\n begin chord service -- done \n");
  }
  if (encnum == 3) {
    if (actuate[y][encnum-1]){
      mcp.digitalWrite(SOL_PIN2, HIGH);
    }

    //change to set desired next chord position
    y++;
    if (y >= 10) {
      y = 0;
    }
    des_enc3_val = chords[y][encnum-1];
    Serial.print("\n begin chord service -- done \n");
  }
  if (encnum == 4) {
    if (actuate[z][encnum-1]){
      mcp.digitalWrite(SOL_PIN3, HIGH);
    }

    //change to set desired next chord position
    z++;
    if (z >= 10) {
      z = 0;
    }
    des_enc4_val = chords[z][encnum-1];
    Serial.print("\n begin chord service -- done \n");
  }
}

void NextChordService(int encnum) {
  //  Turn all solenoids to LOW
  //  Compare current encoder positions to desired positions
  //  Actuate motors forward/reverse based on their positions
  if (encnum == 1) {
    mcp.digitalWrite(SOL_PIN0, LOW);
    if(des_enc1_val == fret0){
      //do nothing
    }
    else if (((int32_t)encoder1.getCount() < des_enc1_val )) {
      startMotorResponse(encnum); //MOVING FWD
    }
    else if (((int32_t)encoder1.getCount() > des_enc1_val )) {
      startReverseMotorResponse(encnum); //MOVING BACKWARD
    }
  }
  if (encnum == 2) {
    mcp.digitalWrite(SOL_PIN1, LOW);
    if(des_enc2_val == fret0){
      //do nothing
    }
    else if (((int32_t)encoder2.getCount() < des_enc2_val )) {
```

```
    if(des_enc2_val == fret0){
      //do nothing
    }
    else if (((int32_t)encoder2.getCount() < des_enc2_val )) {
      startMotorResponse(encnum); //MOVING FWD
    }
    else if (((int32_t)encoder2.getCount() > des_enc2_val )) {
      startReverseMotorResponse(encnum); //MOVING BACKWARD
    }
  }
  if (encnum == 3) {
    mcp.digitalWrite(SOL_PIN2, LOW);
    if(des_enc3_val == fret0){
      //do nothing
    }
    else if (((int32_t)encoder3.getCount() < des_enc3_val )) {
      startMotorResponse(encnum); //MOVING FWD
    }
    else if (((int32_t)encoder3.getCount() > des_enc3_val )) {
      startReverseMotorResponse(encnum); //MOVING BACKWARD
    }
  }
  if (encnum == 4) {
    mcp.digitalWrite(SOL_PIN3, LOW);
    if(des_enc4_val == fret0){
      //do nothing
    }
    else if (((int32_t)encoder4.getCount() < des_enc4_val )) {
      startMotorResponse(encnum); //MOVING FWD
    }
    else if (((int32_t)encoder4.getCount() > des_enc4_val )) {
      startReverseMotorResponse(encnum); //MOVING BACKWARD
    }
  }
}
```

Appendix D: Additional functions