Senior Design 22'-23'
Self-Playing Guitar

ECE Group 30 / CS Group 22

Pedro Contipelli	CS
Blake Cannoe	CpE
Ethan Partidas	CpE
Jonathan Catala	EE
Kyle Walker	EE

Interdisciplinary Self-Sponsored Project

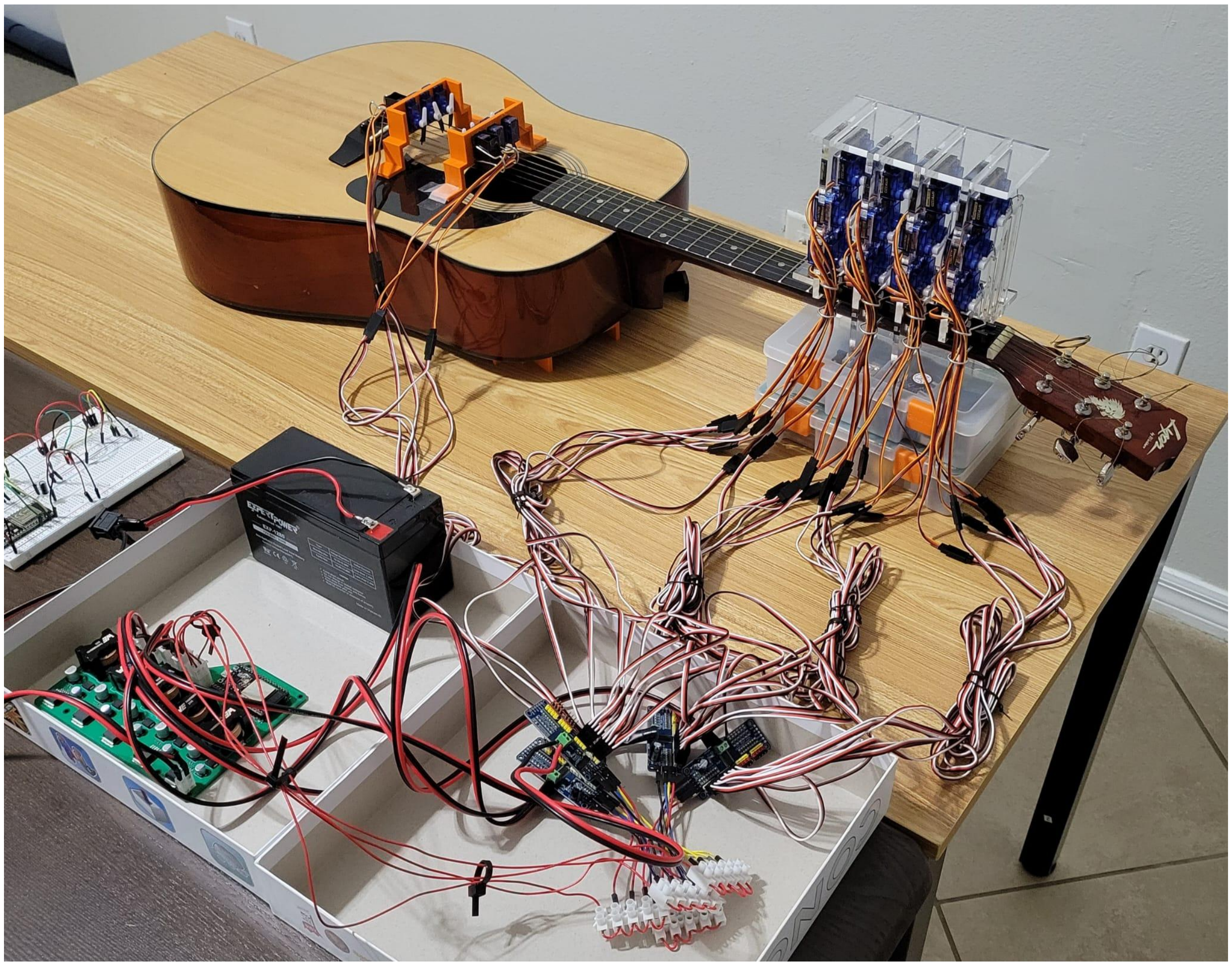


Table of Contents

1. Executive Summary **1**
2. Project Description **2**
2.1. Overview 2
2.2. Student Motivations 3
2.3. Design Goals 4
2.4. Objectives 4
2.5. Requirement Specifications 5
2.6. House of Quality 7
2.7. Initial Block Diagram 9
2.8. Final Block Diagram 9
2.9. Software Use Case Diagram 10
3. Technology Investigation **11**
3.1. Related Works 11
3.1.1 Demin Vladimir's Guitar Robot 11
3.1.2 TECHNICally Possible's Lego Mindstorms Guitar 12
3.1.3. MegCell's Guitar Robots 13
3.2. Software Investigation 19
3.2.1. Viable Time Complexity Calculation for Algorithm 19
3.2.2. Programming Language Investigation 21
3.2.2.2. C++ 21
3.2.2.3. Java 21
3.2.2.4. Python 21
3.2.2.5. MicroPython https://micropython.org/ 22
3.2.3. Parsing MIDI File Bytes Data Stream 22
3.2.3.1. Mido Library 22
3.2.3.2. Pygame 22
3.2.3.3. PythonInMusic Documentation 23
3.2.3.4. MIDIFile 23
3.2.4. Microprocessor Communication 23
3.2.4.1. Serial Communication 23
3.2.4.2. Serial Communication Comparison - BC 29
3.2.4.3. WIFI 30
3.2.4.4. Bluetooth 30
3.2.4.5. Wireless Comparison 30
3.2.4.6. PWM 31
3.3. Hardware Investigation & Considerations 31
3.3.1. Guitar 31

3.3.2. Single Board Computers 32
3.3.2.1. Raspberry Pi 3 Model B 32
3.3.2.2. Onion Omega 2 32
3.3.2.3. Odroid XU4 32
3.3.2.4. Asus Tinker Board 32
3.3.2.5. Libre Computer Board 33
3.3.3. Microcontrollers 33
3.3.3.1. Arduino Due 33
3.3.3.2. MSP430FR6989 34
3.3.3.3. ESP32 34
3.3.3.4. STM32 34
3.3.4. Comparison Table 34
3.3.5. Motor Control 35
3.3.5.1. 74HC595N (Shift Register) 35
3.3.5.2. L293D (Motor Driver) 36
3.3.5.3. PCA9685 (PWM/Servo Driver) 36
3.3.6. Motors 36
3.3.6.1. Servos 36
3.3.6.1.1. Servos Linear Actuator 37
3.3.6.2. Solenoids 38
3.3.6.3. Lead Screw Drive 38
3.3.6.3.1. Lead Screw Drive Linear Actuator 39
3.3.7. Power 39
3.3.7.1. Power Demand 39
3.3.7.1.1. TalentCell Rechargeable 12V 6000mAh 40
3.3.7.1.2. Amazon Basics 9 Volt Everyday Alkaline Batteries 40
3.3.7.1.3. Lead Acid - 12V 8A ExpertPower 40
3.3.7.2. Linear Voltage Regulator 41
3.3.7.2.1. LP2985 41
3.3.7.2.2. LP5912-EP 41
3.3.7.2.3. Custom Linear Regulator 42
3.3.7.2.1. Diode protection 44
3.3.7.2.2. Heat sink 44
3.3.8.3. Switching Regulator 45
3.3.8.3.1. TPS62992-Q1 46
3.3.8.3.2. TPS568231 46
3.3.8.3.3. Custom Switching Regulator 47
3.3.8.3.4. Boost Converter 48

3.3.8.3.5. Buck Converter 49
3.3.8.4 DC Power supply 49
3.3.8.5 AC Power supply 50
3.3.8.6 Op-Amp as a Regulator 50
3.3.9. Signals Technologies 50
3.3.9.1 BJT/MOSFET Amplifier Circuits 51
3.3.9.1.1. MJD44H11AJ 51
3.3.10. 3D Printed Parts 52
3.3.10.1. CAD Software 55
3.3.10.1.1. SOLIDWORKS 55
3.3.10.1.2. Fusion 360 55
3.3.10.1.3. FreeCAD 55
3.3.10.1.4. OpenSCAD 56
3.3.11. PCB Design 56
3.3.11.1 Circuit Simulation Software 57
3.3.11.1.1 LTSpice 57
3.3.11.1.2 PSpice 57
3.3.11.1.2 Multisim 58
3.3.11.2 PCB CAD Software 58
3.3.11.2.1 Autodesk EAGLE 58
3.3.11.2.2 Fusion 360 59
3.3.11.2.3 KiCad EDA 59
3.3.11.3. Part Production 59
3.3.11.3.1. Digikey 60
3.3.11.3.2. PCBWay 60
3.3.11.3.3. OSH Park 60
3.3.11.4 Assembly & Final Implementation 60
3.3.11.4.1 - Wiring to/from/between PCBs 61
3.3.12. LCD 61
3.3.12.1 I2C LCD 16x2 Adapter 62
3.4. Possible Electronics Architectures 62
3.5. Possible Hardware Architectures 64
3.5.1 Hardware Option 1 64
3.5.2 Hardware Option 2 64
3.5.3 Hardware Option 3 65
3.5.4 Hardware Path Moving Forward 65
3.6. Plug in Power 66
3.7. Guitar Mount Building Material 66

3.7.1. Lego Brick Mount 66
3.7.2. K’Nex Mount 67
3.7.3. Wooden Mount 69
3.7.4. 3D Printed Mounts 71
3.7.5. VEX Robotics Mounts 71
3.7.6. Final Material Choice 72
4. Constraints & Standards 73
4.1. Constraints 73
4.1.1 Economic/Time Constraints 73
4.1.2 Environmental Constraints 73
4.1.3 Social Constraints 73
4.1.4 Political Constraints 74
4.1.5 Ethical Constraints 74
4.1.6 Health and Safety Constraints 74
4.1.7 Manufacturing Constraints 74
4.1.8 Sustainability Constraints 75
4.2. Standards 76
4.2.1. Design Standards 76
4.2.1.1. MIDI Audio Storage Standard 77
4.2.1.2. UART Communication Protocol Standard 78
4.2.1.3. Bluetooth Standard 78
4.2.1.4. WIFI Standard 78
4.2.1.5. C Standards 79
4.2.1.6. Python Standards 80
4.2.1.7. Guitar Tuning Standard 81
4.2.1.8. Electrical Component Standards 81
4.2.1.8.1. PCB Standards 81
4.2.1.8.2. Soldering Standards 83
4.2.1.9. Power Supply Standards 85
4.2.1.10. 3D Modeling, Printing, and Laser Cutting Standards 87
4.2.1.10.1. STEP File Standard 87
4.2.1.10.2. STL File Standard 87
4.2.1.10.3. SOLIDWORKS File Standards 87
4.2.1.10.4. DXF File Standards 88
5. Final Design 89
5.1. Hardware Design & Part Details 89
5.1.1. Mechanical Components 89
5.1.2. Power Supply 89

5.1.3. PCBs 90
5.1.3.1.1. Component Size 91
5.1.3.2. PCB Size 91
5.1.3.3. PCB layers 92
5.1.3.4 Voltage Regulation 92
5.1.3.5. Wire Selection 94
5.1.3.5.1. Resistivity 94
5.1.3.5.2. Gauge of the Wire 94
5.1.3.6. Passive Component Supplier 96
5.1.3.6.1. Power Components 96
5.1.3.6.2. Low Noise Components 96
5.1.3.6.3. Power Supply Switch 96
5.1.3.7. Power Configuration 96
5.1.3.8. Signal Configuration 97
5.1.4. Microcontroller 100
5.1.4.1 Wireless Communication for Microcontroller 101
5.1.5. Circuit Design 101
5.1.6. 3D-Printed Mounting Hardware 102
5.2. Software Design 108
5.2.1. Algorithm 108
5.2.1.1. Range Compression 109
5.2.1.2. Handling Self-Overlapping Notes 110
5.2.1.3. MIDI to 6-string and fretting conversion 110
5.2.2. User Interface or UI 110
5.2.3. Cross-Platform Mobile Application 111
5.2.4. Development Environment 111
5.2.4.1. uPyCraft IDE 111
5.2.4.5. Diagrams.net 112
5.2.4.6. Thonny 112
5.2.5. Microcontroller Code 112
5.2.5.1. Micropython and C integration 114
5.2.5.2. Microcontroller libraries 116
5.2.5.2.1. UMidiParser Library https://github.com/bixb922/umidiparser 117
5.2.5.2.2. PCA9685 Library https://github.com/adafruit/micropython-adafruit-pca9685 118
5.2.5.2.3. Servo Library https://github.com/adafruit/micropython-adafruit-pca9685 118
6. Prototype Construction & Coding 119

6.1. Hardware 119
6.1.1. Linear servo motor 119
6.1.2. 3D Designing the Prototype 119
6.1.3. Physical Prototyping 119
6.1.3.1. Mounting System 120
6.2. Electronics 120
6.2.1. Power Supply 120
6.2.2. Circuit Prototyping 121
6.3. Software 122
6.3.1. Algorithm Prototype and Construction 122
6.3.2. User Interface Prototype and Construction 122
6.3.3. Mobile App Prototype and Construction 123
6.4. Integrated Prototyping 124
6.4.1. Controlling Servos with Micropython, ESP32, Shift Register, and Servo Shield 125
6.4.1.1. Micropython Code 125
6.4.1.2. ESP32 Programming 126
6.4.1.3. Breadboard Wiring 126
6.4.1.4. Prototype Takeaways 127
6.4.2. Controlling Servos with Arduino C, ESP32, and Shift Register 127
6.4.2.1. Arduino C Code 127
6.4.2.2. ESP32 Programming 127
6.4.2.3. Prototype Takeaways 129
7. Prototype Testing & Evaluation 130
7.1 Hardware Testing 130
7.1.1. Initial Motor Testing 130
7.1.1.1 Strumming Assembly Testing 131
7.1.1.2. Fretting Assembly Testing 131
7.1.2. Power Supply Testing 131
7.1.3. Integrated Circuit Testing 131
7.1.3.1. Shift Register Testing 131
7.1.3.2. Motor Driver Testing 132
7.1.3.3. WiFi/Server Testing 132
7.1.3.4. LCD Testing 132
7.1.3.5. PCB Testing 133
7.1.5. ESP32 Benchmarking 133
7.1.6. Note Pitch/Frequency Testing 133
7.2. Hardware Evaluation 134

7.2.1. Note Pitch/Frequency Evaluation 134
7.3 Software Testing 136
7.4 Software Evaluation 136
8. Project Operation 137
8.1. Operating Modes 137
8.2. Error Correction 138
9. Administrative Content 139
9.1. Division of Labor 139
9.2 Project Milestones 141
9.2.1. [Fall] Senior Design I 141
9.2.2. [Spring] Senior Design II 142
9.3 Gantt Charts 144
9.3.1. Senior Design I [Fall 2022] 144
9.3.2. Senior Design II [Spring 2023] 145
9.4 Budget/Financing 145
9.4.1. Original Prototype Bill of Materials 145
9.4.2. Final Expenses and Budget Distribution Agreements 147
10. Project Summary & Conclusions 149
10.1. Project Summary 149
10.2. Design Summary 149
10.3. Planning / Engineering Design Conclusions 150
10.4. Philosophical Conclusions 150
11. Broader Impacts 152
12. Legal, Ethical, and Privacy Issues 153
12.1. Legal Issues 153
12.2. Ethical Issues 153
12.3. Privacy Issues 153
13. Facilities and Equipment 154
13.1. Facilities 154
13.2. Equipment and Materials 155
13.2.1. George Washburn Lyon Acoustic Guitar 155
13.2.2. Miuzei Micro-Servo Motors 155
13.2.3. ESP32 2.4 GHz Dual Core WLAN WiFi + Bluetooth Microcontroller 38PIN Narrow Version 156
13.2.4. Breadboard 157
13.2.5. Custom-Designed PCB 157
13.2.6. Shift Registers (74HC595N) 157
13.2.7. Power Supply 158

13.2.8. Oscilloscope 158
13.2.9. Digital Multimeter 158
13.2.10. Other Miscellaneous Electrical Components 159
13.2.11. Universal Laser Cutter & Dimension 3D Printer 159
14. Consultants, Subcontractors, and Suppliers 160
Appendices 161
Appendix A: References 161
15. Section 4.2.1.1. MIDI Audio Storage Standard 162
16. Section 5.2.5.1. Micropython and C integration 162
Appendix B: Copyright Permissions 162
Appendix C: Purchase Links 162
Appendix D: Datasheets 162

List of Tables

Table 1: Correlation between Objectives & Goals
Table 2: Requirement Specifications
Table 3: Table of Quality Model
Table 4: I2C and SPI comparison
Table 5: Comparison between WiFi, Bluetooth, and BLE
Table 6: Single Board Computer and Microcontroller Comparison
Table 7: Comparison Between Hardware Design Options
Table 8: Constraints
Table 9: Standards
Table 10: Clearance Tolerances Outlined in ICP-2221 (via protoexpress.com)
Table 11: Power Supply Circuit Definitions (via CUI)
Table 12: Component Voltage and Current Demand
Table 13: Motor Assembly Component Values
Table 14: LCD Screen States
Table 15: Project Milestones for Senior Design I (Fall 2022)
Table 16: Project Milestones for Senior Design II (Spring 2023)
Table 17: Bill of Materials

List of Figures

Figure 1: Playable Note Range for Requirement Specifications
Figure 2: Functional Class Diagram
Figure 3: Functional Block Diagram
Figure 4: Software Use Case Diagram
Figure 5: Denim Vladimir's Guitar Robot
Figure 6: TECHNICally Possible's Lego Mindstorms Guitar

Figure 7: MegCell First Prototype
Figure 8: MegCell 2nd Iteration Prototype
Figure 9: MegCell Strumming Arm Mechanism
Figure 10: MegCell Difficulty Fretting Close Together
Figure 11: MegCell Updated Strumming Assembly
Figure 12: MegCell Updated Fretting Assembly
Figure 13: Fret Override
Figure 14: UART Structure For Autonomous Guitar
Figure 15: SPI in Parallel
Figure 16: SPI in Daisy Chain
Figure 17: I2C Configuration for Electric Guitar
Figure 18: Assembly for Conversion of Rotational Servo Motion to Linear Actuation
Figure 19: Lead Screw Drive X-Ray Model
Figure 20: Custom Linear Regulator
Figure 21: Linear Regulator with Voltage Protection
Figure 22: Heat Sink Attached to a Regulator
Figure 23: Boost Converter
Figure 24: Buck Converter
Figure 25: Servo Placement In Strumming Assembly
Figure 26: Servo Placement in Fretting Assembly
Figure 27: Personal Computer Architecture
Figure 28: Single Board Computer Architecture
Figure 29: Motor Driver Architecture
Figure 30: Servo Architecture
Figure 31: Example Lego Mount Construction
Figure 32: Example K'Nex Mount Construction
Figure 33: Example Laser-Cut Wooden Servo Motor Mount Construction
Figure 34: 3D Printed Mount Construction
Figure 35: VEX Robotics Example Construction
Figure 36: Clearance vs Creepage on PCBs (via protoexpress.com)
Figure 37: Model of Exemplary PTH solder
Figure 38: Component Sizes with Dimensions
Figure 39: 4-layer PCB diagram
Figure 40: Motor Assembly Voltage Regulator
Figure 41: PCB Design Interface
Figure 42: PCB for Motor Assembly
Figure 43: Wire Gauge Diagrams with Corresponding Currents
Figure 44: Power Configuration
Figure 45: Example Signals using 800° Rotations for Strumming Assembly
Figure 46: Example Signals using 180° Rotations for Fretting Assembly
Figure 47: Example Signals using 90° Rotations for Fretting Assembly
Figure 48: Circuit Design Diagram Prototype
Figure 49: 3D Printed Servo Mount Prototype
Figure 50: Strumming Servo Mount Prototype
Figure 51: Fretting Servo Mount Prototype
Figure 52: MIDI Representation Visualized

Figure 53: Playable Note Range Visualized on Piano (Synthesia)
Figure 54: Microcontroller Code Block Diagram
Figure 54.5: C integration into micropython
Figure 55: Example Power Supply Used for Testing
Figure 56: MIDI file processing in Python using MIDO library
Figure 57: Simple Initial UI Prototype for Controller
Figure 58: Simple Song Selector Prototype for Controller
Figure 59: Cross-Platform Mobile App Example using Flutter
Figure 60: Breadboard Wiring Diagram
Figure 61: Arduino IDE Preferences Menu
Figure 62: ESP32 Package in Boards Manager
Figure 63: ESP32 Dev Module in Boards Manager
Figure 64: Pano Tuner App from Google Play App Store
Figure 65: Unicorn CPU Emulator Testing for MicroPython
Figure 66: File Upload in Project Operation
Figure 67: Senior Design I (Fall 2022) Tasks Gantt Chart
Figure 68: Senior Design II (Spring 2023) Tasks Gantt Chart
Figure 69: George Washburn Lyon Acoustic Guitar
Figure 70: Micro-Servo Motors Purchased
Figure 71: ESP32 Purchased
Figure 72: 3D-Printed Linear Actuator GrabCAD by Luca Delbarba Feb 28th 2022
Figure 73: Example Shift Register Render
Figure 74: Example Oscilloscope used for Debugging
Figure 75: TI Innovation Lab Equipment
Figure 76: TI Innovation Lab in Engineering Building at UCF

1. Executive Summary

This document outlines the ideas, motivations, design considerations, and implementation details of ECE Group 30 / CS Group 22’s Senior Design project. The end goal is to build a fully functional autonomous self-playing guitar using a combination of both hardware and software implementation. By nature, this is an interdisciplinary project. Our main objective is to be able to take any valid MIDI file and play those notes by fretting and strumming all 6 strings independently on an acoustic guitar (without self-interference). Specifications include being able to play all 29 valid notes between E2 and G#4, giving us a range of about 2.33 octaves.

We will accomplish this by using a custom algorithm written in Python to take MIDI data and convert it in realtime to playable notes on the guitar the exact start timing and duration of each note. This code will then run on a microprocessor that is connected to both the strumming motor assembly and fret motor assembly. The strumming assembly will consist of 6 servo motors positioned to pluck at each string. And the fret motor assembly will consist of servo motors with linear actuators positioned at fret locations 1-4 on each string. In total, giving us **30 unique playable combinations = 6 strings * (4 fret positions + 1 open string)** and totaling **29 unique notes** (B3 is repeated), as demonstrated in the below calculation.

$$29 = 6 * (4 + 1) - 1$$

As this is a student-led and self-sponsored project, our budget is mostly limited to what we as students can afford: which we estimate will be around $200 across the 5 of us. There are little to no consumer products available at the moment which do this. However, there are a few hobbyists who have built similar machines as personal projects and posted videos of it on YouTube. Essentially, all projects of this nature are simply built for fun, entertainment, and as a personal challenge rather than for mass consumption. We will have the final design document report completed by the end of Senior Design 1: December 5th 2022. We would like to have our minimum viable product ready by halfway through Senior Design 2: February 24th 2023. And have the final product ready and presentable by one month before the end of Senior Design 2: March 24th 2023.

2. Project Description

The below section serves as the detailing of our project. It provides an overview of the project and personal motivations before going into detail about its goals and the objectives we have laid out in order to achieve them. In order, the section provides an overview of the project and a summary of our motivations, which then leads into the specific goals that we wish to achieve to realize our design. Following that are the specific objectives we have laid out in pursuit of those goals before going into the specific requirement specifications that we will adhere to throughout our time in Senior Design in order to ensure that the project works as intended and is fully reproducible and testable. The section ends off with supplemental material in order to better elucidate the broad strokes in what the project will ultimately look like.

2.1. Overview

The goal of this project was to create an autonomous self-playing guitar that was able to produce its own music. We have seen and taken inspiration from videos on YouTube of similar projects such as Demin Vladimir's Guitar Robot and TECHNICally Possible's Lego Mindstorms Guitar, and intended to build on the design concepts utilized by these projects; improving upon them and creating a design that is our own.

The system takes in MIDI files and plays the notes on the guitar using separate mechanisms for strumming and pressing select strings against frets. Our original vision of the project was that it would be lightweight and maintain the general form factor of the guitar (i.e, fits closely to the body). The design would ideally be portable, and thus it would be powered by portable batteries. It should be responsive enough to accurately replicate the provided MIDI file compositions, comparable to - if not exceeding - the abilities of the average learnt guitar player. Not only should this design be lightweight and portable, an issue with similar concepts is the price and size. They are typically not an attachment for a guitar and are more commonly an entire unit within the guitar. They are also extremely expensive with some models going for up to $1,100. Our goal for this project was to bring this idea to reality for significantly cheaper.

One challenge that presented itself by nature of the project was the mechanical considerations that we had to tackle. Being an interdisciplinary group made up of Electrical Engineering, Computer Engineering and Computer Science majors with no significant mechanical engineering classes present within our curriculum, we had to perform significant investigations into how we will make the parts move in the way we want them to. However, we know that similar projects had been realized before and with dedication we saw to it that ours would be as well.

2.2. Student Motivations

Our time in Senior Design offered the unique opportunity to apply the techniques we've learned throughout our undergraduate programs into a system that interests us personally. In this way, we sought to pursue a project that would not only be technically challenging but also be creative and, ultimately, fun to put together.

Pedro Contipelli - My motivation for this project was that I wanted to bridge two very different fields that I am extremely passionate about: engineering and music. I felt this project would be the perfect choice for our team to put our knowledge and skills to the test, to create something that can be appreciated from both a scientific and artistic perspective.

Blake Cannoe - I used to play guitar in middle school. I thought this idea sounded like an interesting project. I'm very interested in low-level programming and design of embedded systems so I look forward to applying my knowledge in this project.

Ethan Partidas - I wanted to work on a project that has a large vertical scope, from the high-level software algorithm all the way down to the mechanical challenges of playing a guitar with motors. I find all of these fields interesting, which is part of the reason I chose to study Computer Engineering in the first place; it combines computer science and electrical engineering into one degree program. I have dabbled with 3D modeling in CAD software before, so I'm also excited to get more experience with it.

Jonathan Catala - I am interested in this project due to the multidisciplinary nature of this project. It gives us the best look at an actual work project we could do. Due to this I believe this project will be beneficial to develop my engineering skills.

Kyle Walker - My dad played guitar throughout most of his life & also motivated me to become an engineer, so it seemed like a cool way to combine those two things. It provides an opportunity to finally put my electrical engineering experience learned from class to work in the real world, and to come up with something that we can be proud of.

2.3. Design Goals

Our ultimate goal for this project going into Senior Design was to modify a guitar with electronics to be able to play itself. In pursuit of this we wanted our project to be:

- Able to reliably play digital audio data. This was the ultimate factor that will determine the functionality of our product. It should be able to take data in, interpret that data as musical notes, and then play those notes using the guitar that the design was built on.

- Portable. We want our design to fit the general form factor of a guitar, not departing too far from how a guitar looks; This would ultimately manifest itself in the form of minimal part count & mounting upon the guitar itself and a self-contained power supply (rather than wall outlet power, for example).

- Lightweight. We did not want the design to weigh too much. We still wanted the product to be held like a guitar, and too much weight would take away from that.

- Affordable. As we are a collection of undergraduate university students, we wanted to minimize the cost investment into the product - especially for prototyping and moving from the minimum viable product to final presentation.

- Reliable. Ideally, we would want our design to require as little maintenance as possible in order to present to the Senior Design board. A high enough fault rate would consume too much time to fix which would be better spent prototyping and refining the final product.

2.4. Objectives

To realize the above goals, our project had to:

1. Be built on an acoustic guitar
2. Be battery powered
3. Take MIDI file input over USB
4. Process MIDI files using a Microcontroller
5. Play a wide range of notes
6. Utilize servos for mechanical action (strumming and fretting)
7. Be able to strum all 6 strings either at once or independently
8. Be able to press frets to play individual notes
9. Use 3D Printed assembly parts

Table 1 below shows which objectives correlate to each goal organized by the index of each objective. In effect, we wanted to make sure that the meeting of each of these objectives during our project's development means that we reach each of the goals detailed in the section above.

Goals	1	2	3	4	5	6	7	8	9
Plays Audio Data			✓	✓	✓	✓	✓	✓	
Lightweight	✓	✓							✓
Affordability	✓					✓			✓
Portability		✓							

Table 1: Correlation between Objectives & Goals

2.5. Requirement Specifications

The following Table 2 outlines the specifications that we laid out for our project. It gave us targetable and testable metrics that we can use to gauge the success of the project. As our design began to take shape, we placed priority in maintaining these specifications through thorough testing of our minimum viable prototype and, ultimately, our final product.

As we began our implementation of our project we were also looking to see if our requirement specifications were feasible. The specifications of the design are ultimately a metric to determine how successful the project will be - as we have little to no experience with similar technically complex projects, our target conditions for project success naturally shifted and took new forms over time. Table 2 thus details the final design requirement specifications that we used to determine where our project should be at the time of completion.

Below Table 2 is a figure detailing all the notes that we will be aiming to play with our design. As detailed in our project overview, this gives us a range of about 2.33 octaves, however other octaves may be pursued as part of a stretch goal in the future.

Specification	**Measurement**
Electrical components weight	3 pounds
Battery weight	5 pounds
MIDI File Size	50 KB
Current Limit	< 8.6W
Playable notes	All notes between E2 and G#4 (see figure 1)
Minimum playing speed	2 notes per second per string
Maximum song length	3 Minutes
Battery Life	3.2 Hours
Max input voltage to Servos	5V
Max mounting system weight	20 lbs
Total cost	<$500
Max height of mounting system	1 ft
Power supply	DC battery(s)
Response times of servos	1ms

Table 2: Requirement Specifications

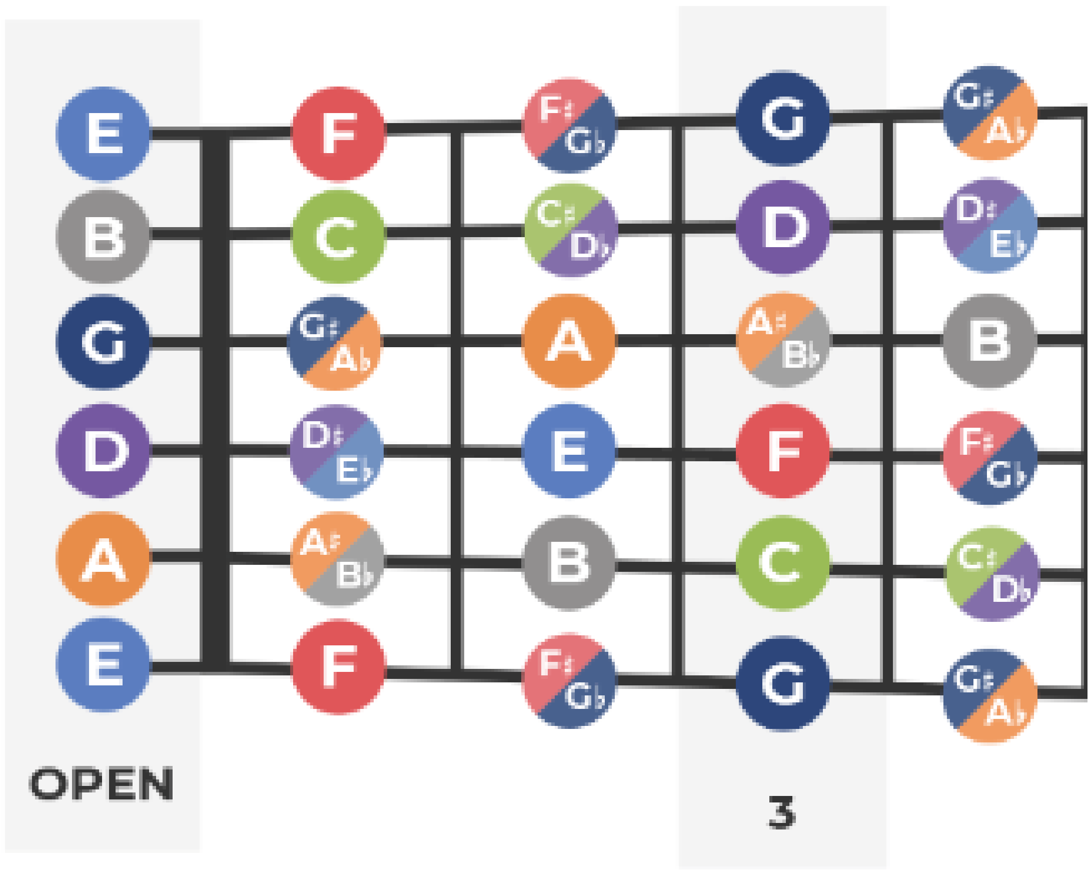


Figure 1: Playable Note Range for Requirement Specifications
https://yousician.com/blog/guitar-fretboard-learning-guide

2.6. House of Quality

Table 3 on the following page details the House of Quality model used to outline our project's specification requirements. The House of Quality model is used to weigh the effects that each engineering specification has on each market specification, alongside the effect that each engineering specification has with one another. Note that, as our project was self sponsored, it was ultimately us who determined the market requirements: We wanted our guitar to be able to play long, musically complex songs accurately while maximizing the potential volume of the notes. We also wanted our guitar to be lightweight and portable, while minimizing cost and power consumption.

As the implementation process began, we looked closely at all the relationships between all the different requirements, as we get further in the progress we will keep the house of qualities updates as there could be some unforeseen relationships that could have more or less relationship than previously assumed. Through keeping this table updated weekly and especially as we begin to order parts we will see if our original assumptions about the relationships between different requirements is true and verifiable through testing.

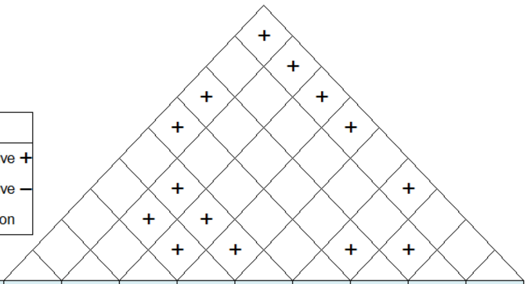
QFD: House of Quality C. Battles. (2011, May). QFD House of Quality Template. Schrodinger's Ghost.com [Online]. Available: http://www.schrodingersghost.com/?cat=54.

Project: Autonomous Guitar
Revision:
Date: 10/7/2022

Relationships	
Strong	●
Moderate	○
Weak	▽

Direction of Improvement	
Maximize	▲
Target	◇
Minimize	▼

Correlations	
Positive	+
Negative	–
No Correlation	



		Column #	1	2	3	4	5	6	7	8	9
		Direction of Improvement	▼	▼	▼	▼	◇	▼	▲	▼	▼
	Weight	Engineering Requirements / Customer Requirements (Explicit and Implicit)	MIDI file size	# of Fret motors	# of Strumming motors	Power Source	Output Power	# of microcontrollers	efficiency of code	timer speed	Cost
	8	Song Duration	●	○				●			●
	2	Sound Volume	▽	▽	●		●				
	4	Light Weight	▽	●	○	●		●			
		Song Complexity	●	●	●				●	●	
		Portable	▽	●	○						
		Low Power	○	●	●	●	●		●	●	
		Small install	▽	●	●						●
		accurate	○	○					●	●	
		Cost	●	●	●	●		●			●
		Target	>50 KB	> 29 motors	6 motors	Battery	> 9 Watt	>2	> 75%	1ms	>$200

Table 3: Table of Quality Model

2.7. Initial Block Diagram

Below is the first drafted figure for the hardware block diagram. In the initial design we were using 4 Shift registers and we had an LCD screen, we have since switched the shift registers for servo shields and the LCD screen was removed from the design. The shift registers were removed because of control issues and the LCD was removed because of time constraints.

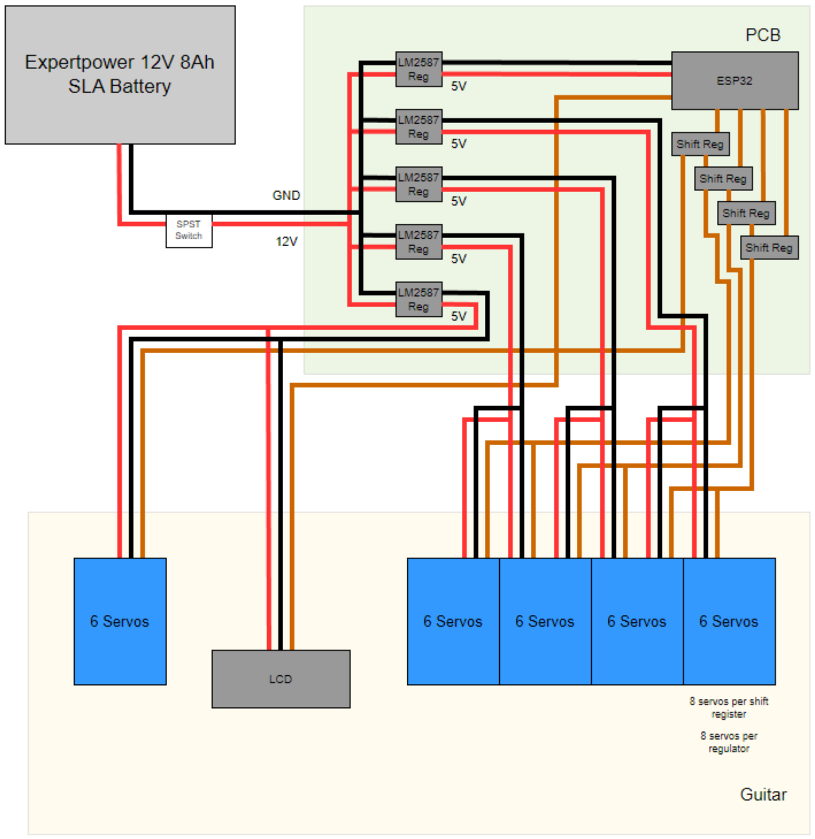


Figure 2: Initial Block Diagram

2.8. Final Block Diagram

Below is our mostly final hardware design except for the shift registers still being in the circuit those were replaced with servo shields, which function similarly however, are much easier to control due to each servo having its own PWM pin and instead of having a 3-wire connection to the ESP32 it has a 2-wire I2C connection.

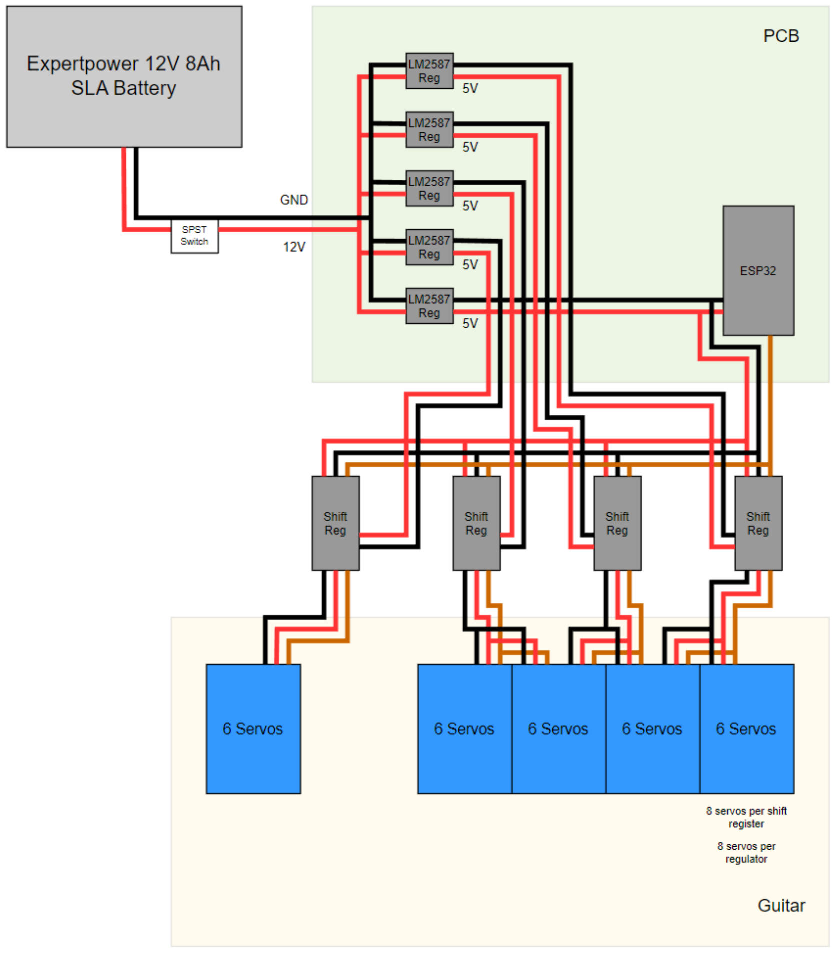


Figure 3: Final Block Diagram

2.9. Software Use Case Diagram

Provided here is the Software Use Case Diagram for the project. In effect, it outlines how we interact with the software of the guitar, providing input as well as state transitions through operating the guitar while it is in the process of performing a given song.

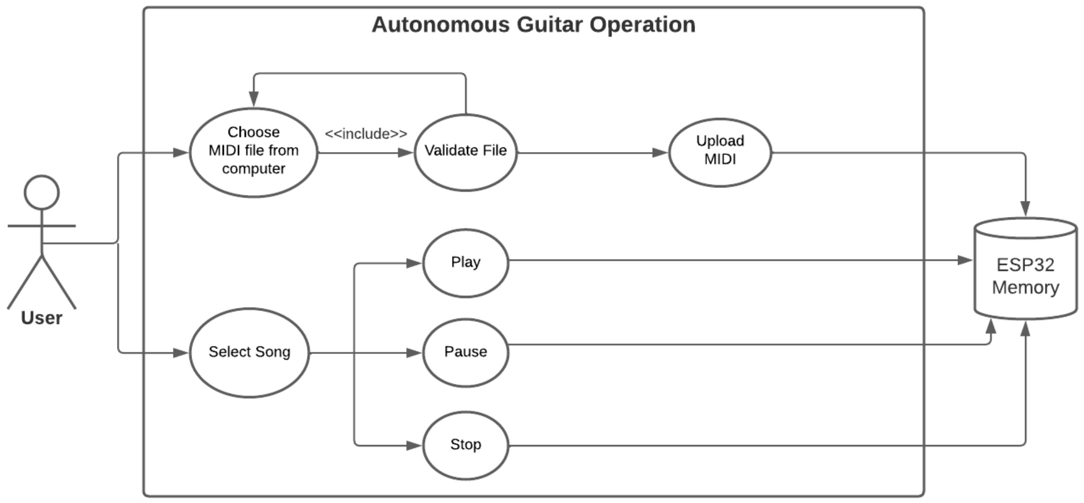


Figure 4: Software Use Case Diagram

3. Technology Investigation

The following section shows what technologies we investigated as we went through Senior Design 1 and before construction of the actual project began. It shows the thought process behind why our initial design was settled upon.

3.1. Related Works

As mentioned in our project overview, some examples of self-playing guitars already exist - In this section, we will be going over the design as evidenced by their appearance in their respective demonstration videos, weighing their pros and cons and what design philosophies we would ultimately integrate into our project.

3.1.1 Demin Vladimir's Guitar Robot

Demin Vladimir's Guitar Robot comprises a guitar with electronic components attached to it in order to facilitate its own strumming and fretting. It is similar to our design, and plays the entire possible range of notes - however, whereas we use servos to fret, this project uses many solenoids. It is much more versatile, but also more costly as a result of utilizing so many parts. The utilization of solenoids as frets is a simplistic and functional design that would likely serve our design well if integrated - However, solenoids prove to be much more expensive than servos. Hence, we elected for servos rather than solenoids. Figure 3 below is a picture of the robot in action as seen on Youtube.

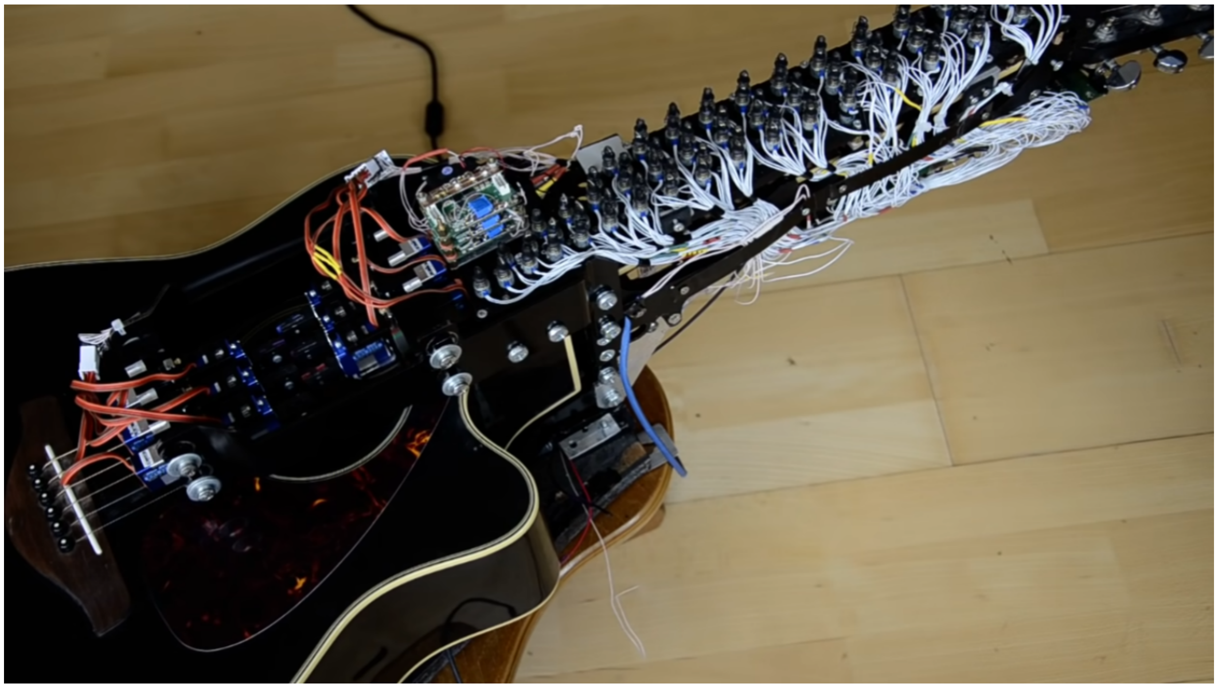


Figure 5: Denim Vladimir's Guitar Robot
https://youtu.be/n_6JTLh5P6E

3.1.2 TECHNICally Possible's Lego Mindstorms Guitar

The TECHNICally Possible's Lego Mindstorms Guitar is also reminiscent of our final design description, however what separates it from Demin Vladimir's guitar robot is that it utilizes Lego Mindstorms components. Lego Mindstorms is an extremely versatile project design kit that allows for the easy assembly of projects such as this one. The most notable downside to this configuration is that the combination of notes itself is mechanically integrated into the design of the project - pegs are used to control the fretting for specific chord combos, but in order to play another song these pegs must be changed out manually. This greatly hampers versatility, but it comes with the benefit of greatly reducing cost. The figure following on the next page is a picture of the Mindstorms Guitar in action.

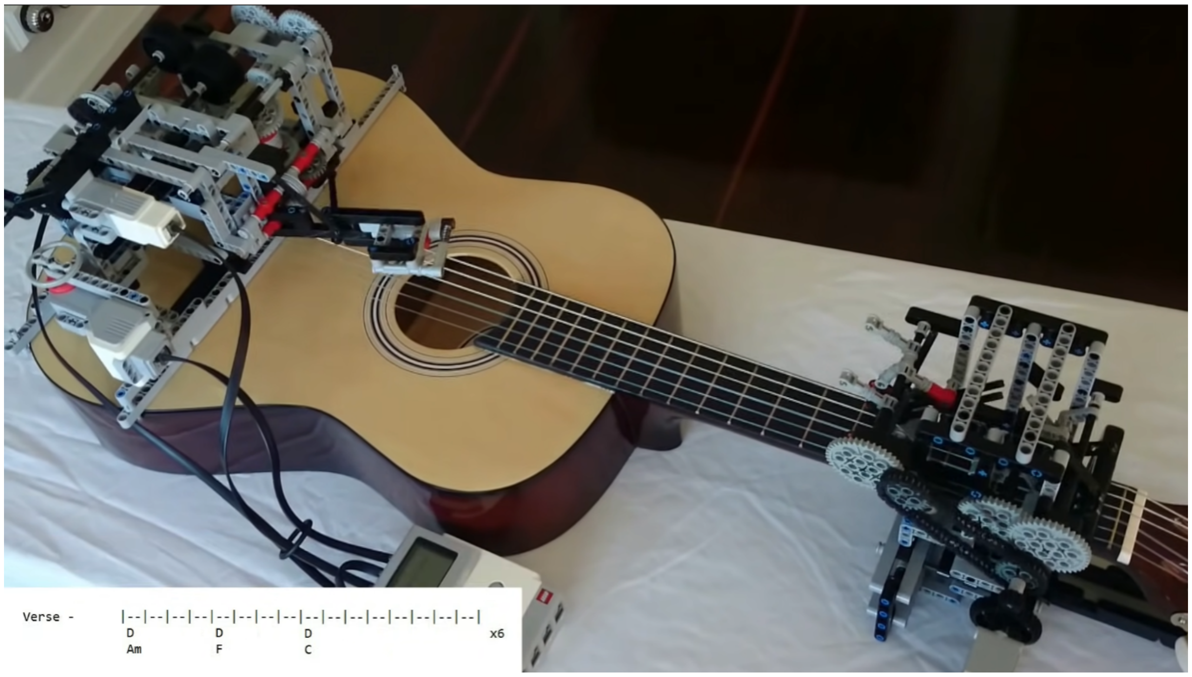


Figure 6: TECHNICally Possible’s Lego Mindstorms Guitar
https://youtu.be/cXgB3lIvPHI

3.1.3. MegCell’s Guitar Robots

MegCell on YouTube has recently been making big strides in the field of servo-controlled self-playing guitar robots. He started off by mounting servos to a guitar using popsicle sticks and controlling them with an Arduino. This already sounded great, but he didn’t stop there. He has continued to iterate and improve on his design over the last couple years, adding 3D-printed mounts and geared mechanisms to improve the actuation of the servos on the strings.

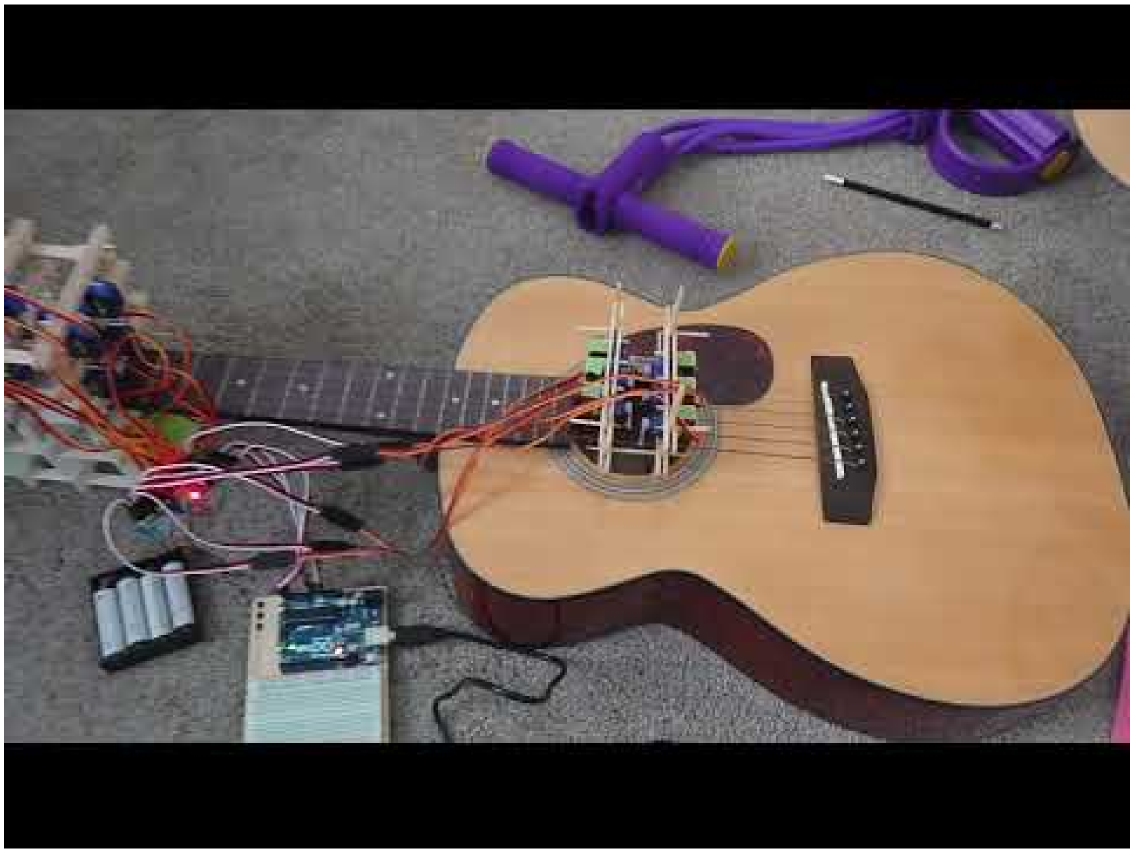


Figure 7: MegCell First Prototype
https://youtu.be/g_dBuR2GiTA

This is his first prototype. It is very low tech, which is a big relief for us as far as the feasibility of this project. It uses the same servos that we will likely end up using, and a microcontroller similar to the one we will likely use. Of course, we would like our final product to be more polished than this, and that's where MegCell's next iterations come into play.

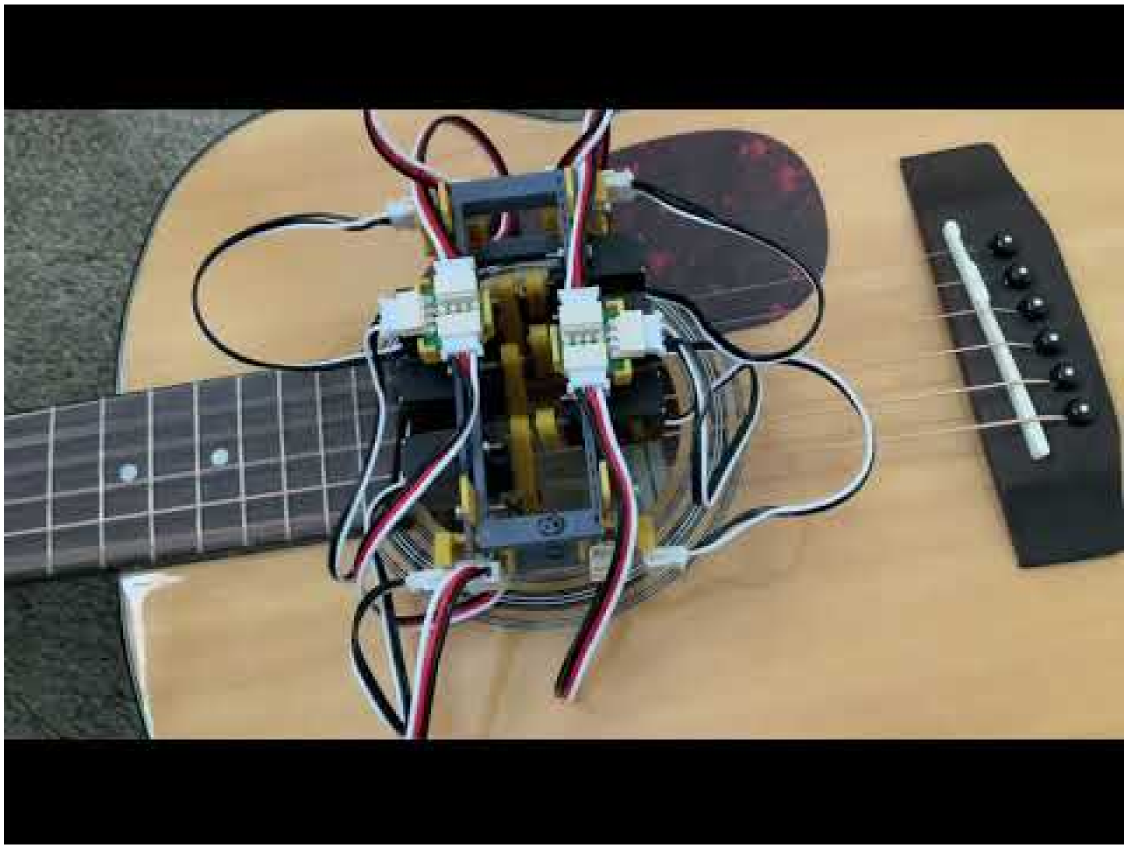


Figure 8: MegCell 2nd Iteration Prototype
https://www.youtube.com/watch?v=EFA9eQ3DfrU

This iteration has replaced the popsicle sticks on the strumming mount with 3D-printed supports and tidy wiring. We also get a closer look at how the servos are placed, front-to-front so that all 6 strings can be strummed as close to the center of the soundhole as possible.

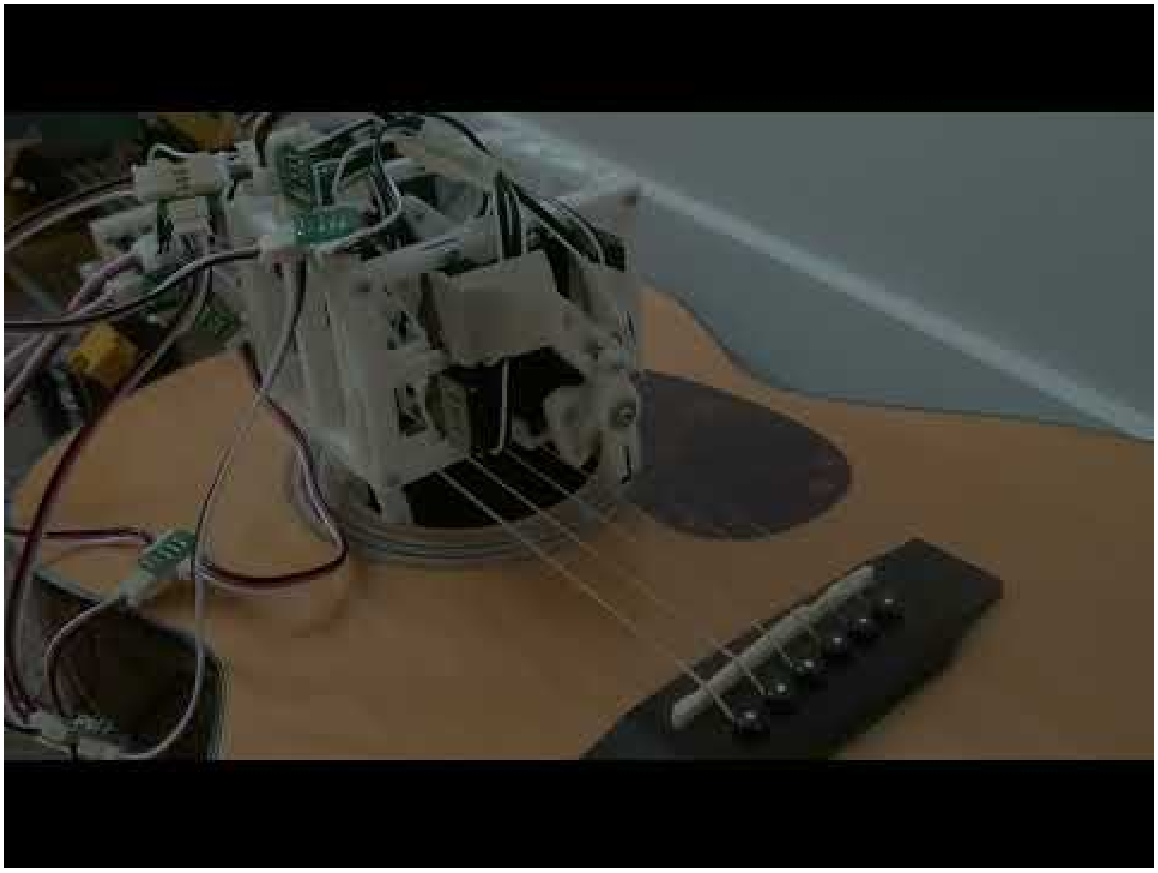


Figure 9: MegCell Strumming Arm Mechanism
[*https://www.youtube.com/watch?v=MpRi8xJrh24*](https://www.youtube.com/watch?v=MpRi8xJrh24)

This iteration adds a mechanism to the strumming arm. Instead of the servo strumming the string directly, an additional part is added which travels in a somewhat circular path, allowing the pick to move up and out of the way of the string once it has been strummed.

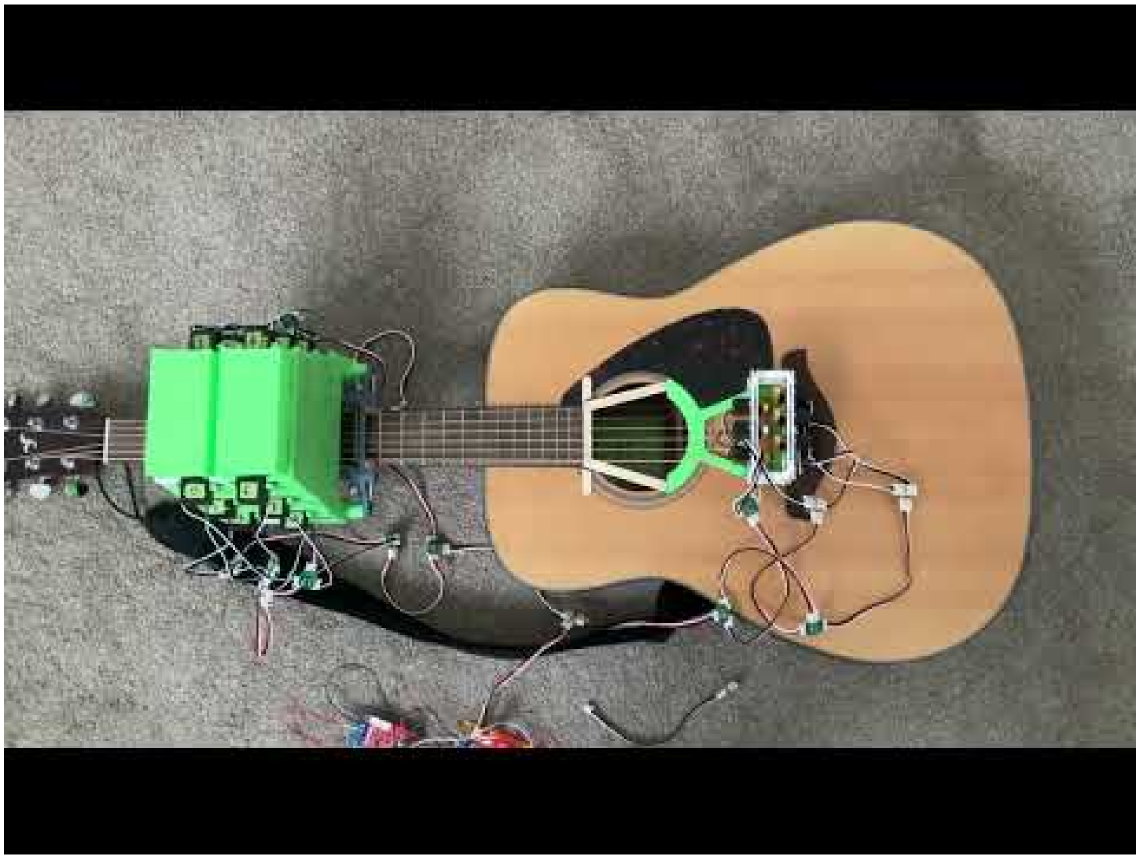


Figure 10: MegCell Difficulty Fretting Close Together
https://www.youtube.com/watch?v=JQZKGL_aIY8

This design shows the difficulty in fretting the strings when they are so close together. He has stacked the servos in 3 layers in order to be able to fret on such a small area.

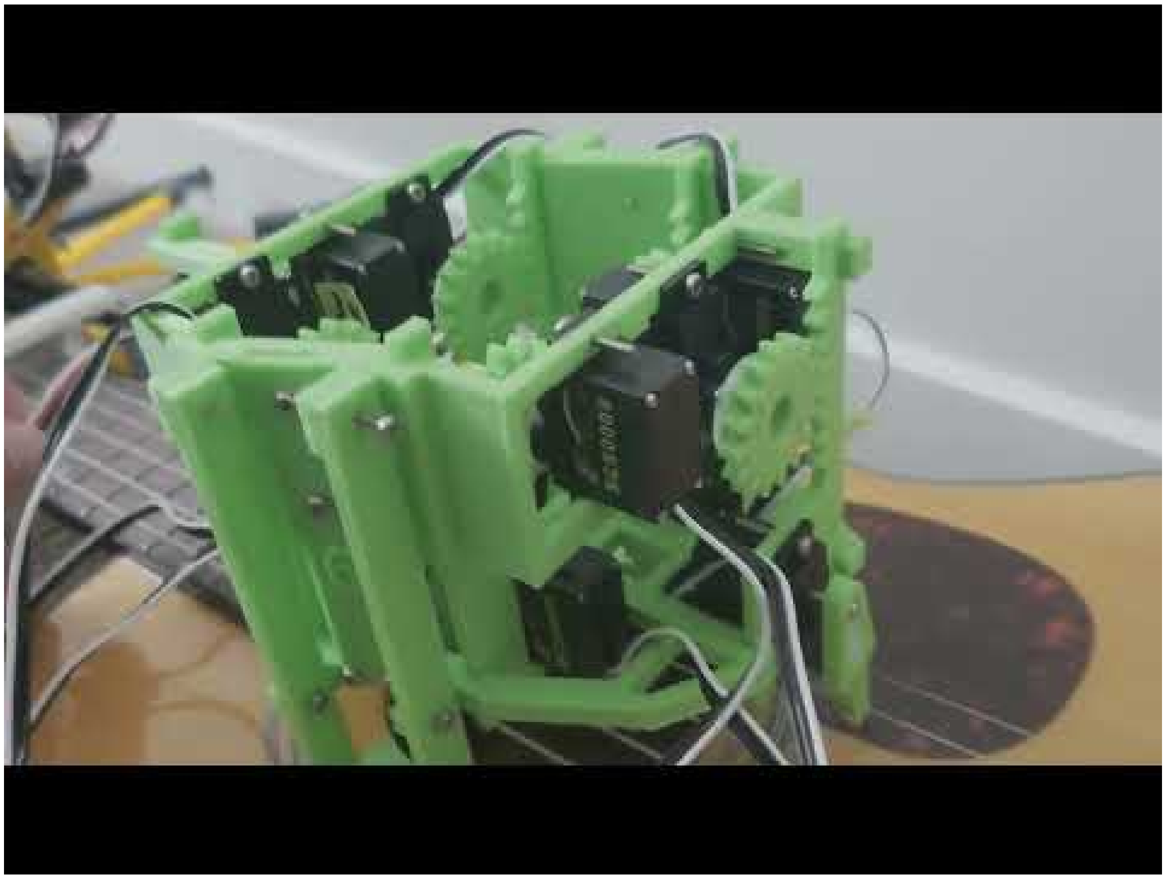


Figure 11: MegCell Updated Strumming Assembly
https://www.youtube.com/watch?v=yCJw_BHz3-o

This is a new design for the strumming assembly. Each string actually has 2 servos dedicated to it. One to strum the string, and one to move the other servo slightly upwards, so that the string is strummed in a linear path rather than a circular one.

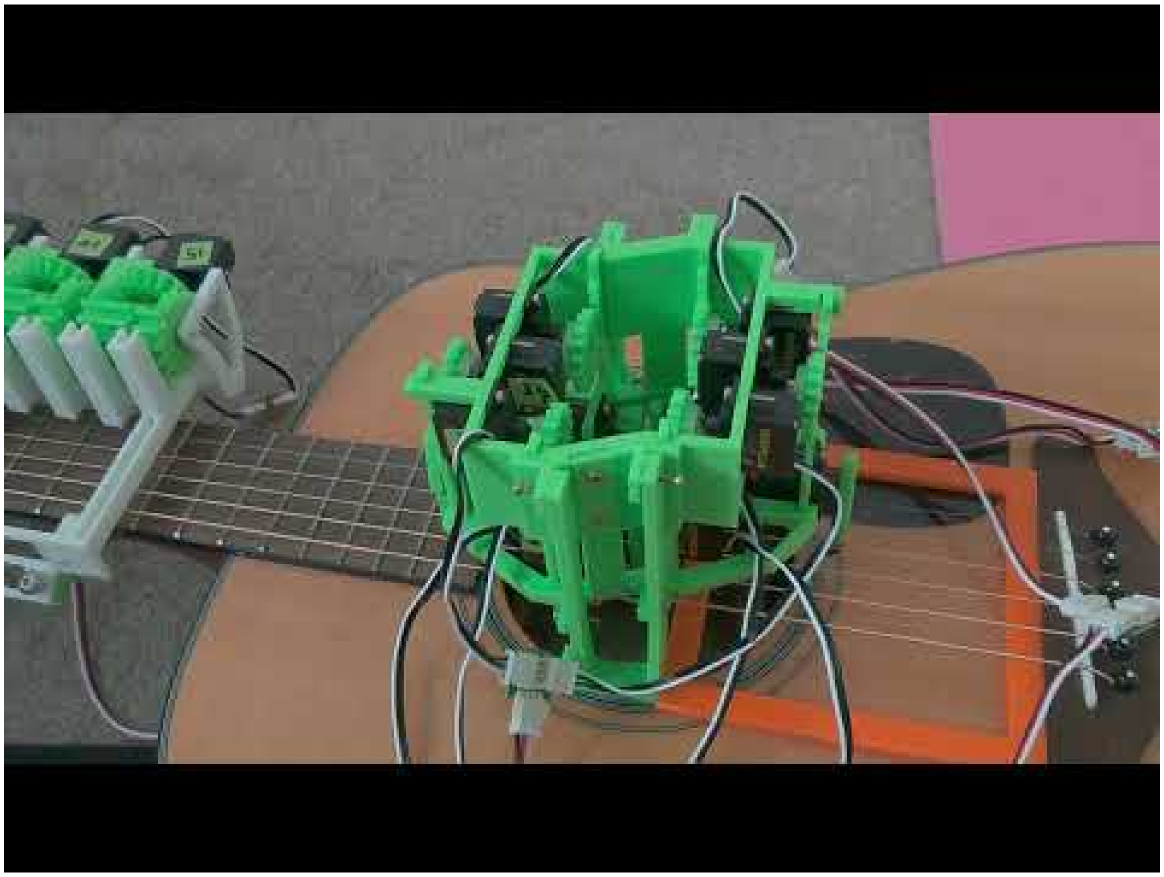


Figure 12: MegCell Updated Fretting Assembly
https://www.youtube.com/watch?v=MLX7lxt5xa8

Here we can see a very clever design for the fretting. Each servo is geared to two arms, so they can each control two notes.

3.2. Software Investigation

3.2.1. Viable Time Complexity Calculation for Algorithm

One fact that may not be well-known to the average layman but every beginner guitarist learns is that, on each string, only the fret position played that is furthest down the neck actually affects the pitch of the note when that string is plucked or strummed (frets are counted starting at the tip or head of the guitar and going “down” towards the sound hole and bridge, from the 1st fret to the 20th+ [depending on the guitar]). As shown in the diagram below, the blue ‘**x’**s are not actually affecting the pitch of the note played on those strings, as they are being “overridden” by the fingers pushing down on the higher frets.

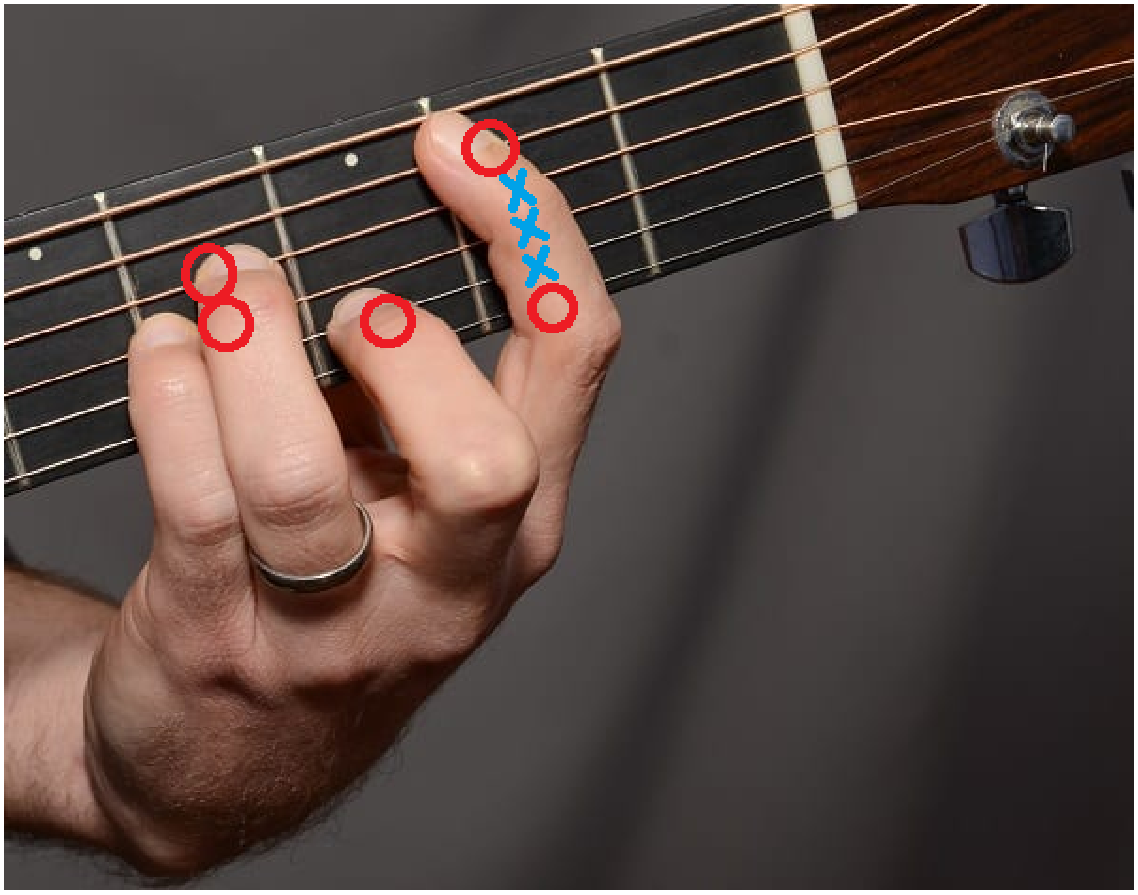


Figure 13: Fret Override
https://nationalguitaracademy.com/how-to-play-bar-chords/

What this means for the purposes of designing an algorithm to consider every possible valid yet unique-sounding combination of notes that can be made using the first four fret positions on each string is that we only need to consider the "highest" fret played on each string. This leads to a much smaller total number of combinations than if we had naively considered all **O(2^n)** technically possible up/down states for each fret on *every* string. So the calculation goes as follows: (4 fret positions + 1 "open" string a.k.a. no frets played)^(6 strings) = 5^6 = 15,625*. The * is one caveat because technically the note "B" played by the open 2nd or aptly-named "B" string is actually equivalent to the note played by the 4th fret position on the 3rd or "G" string. So the technically correct calculation gives us the even smaller 5*5*5*4*5*5 = 12,500 uniquely sounding combinations. However, since it would be programmatically nontrivial to actually only generate combinations which consider that one exception without any extra computation, we will choose to ignore it and maintain a still very viable upper bound at **O(5^6) = 15,625** combinations. Considering our requirement of a playing speed of 2 notes per second per string simply means that we'd like our algorithm to be able to go through approximately 15,625*2 = **31,250** combinations per second. Assuming that each combination takes approximately one CPU operation or **O(1)** time, this number gives us our minimum speed of runtime in operations per second that we will need for our algorithm, which as it turns out, is actually very realistically feasible for a microprocessor to carry out. The algorithm and hardware we used ended up taking raw note data in MIDIs as input so we were able to directly output the right motor activations for a given chord without having to search through all possible combinations.

3.2.2. Programming Language Investigation

3.2.2.1. C

C, being a relatively low level language is a good option as it could interface well with the hardware, providing an easy way to directly control our motor outputs via pins from the microprocessor. It can also provide very fast processing, allowing us to even write our algorithm to go through every possible fretting / open-string combination (5^6 = 15,625 combinations) in mere milliseconds (assuming it can process approximately 100,000,000 operations per second). In fact, with C, we may even consider a less efficient algorithm of exponential time complexity O(2^N) where N is the number of frets we plan on being able to press down on (24), since 2^24 = 16,777,216. Whereas with a slower language we wouldn't be able to process notes/chords fast enough for most songs. Some drawbacks of using C, however, would be developer cost / engineering velocity / debugging time as C doesn't have very verbose error logging, usually just throwing seg-faults and it is also difficult to manage memory correctly.

3.2.2.2. C++

C++'s advantages and disadvantages are similar to that of C's, unsurprisingly. However, the code is slightly less readable to developers without prior understanding of C++ and we think in this case, we can achieve good enough runtimes with a more verbose and easily understandable language, so it doesn't really provide the kind of tradeoff we would be looking for.

3.2.2.3. Java

Java provides a runtime in between that of C/C++ and Python, though it does inherently involve a large amount of boilerplate, and essentially forces an object-oriented structure which will most likely not necessarily be necessary for the purposes of our algorithm. What really matters to us the most in terms of choice of programming language would be that we can easily work together in both writing and debugging code while maintaining it very readable and following good standard practices in terms of coding style.

3.2.2.4. Python

Python offers us great libraries for interfacing with practically any microprocessor and very simple, clean, readable code that anyone on the team can read and understand, allowing for better communication and teamwork. In terms of efficiency, it *is* the slowest language we consider, however we strongly believe that we can write an algorithm that is efficient enough to do all the processing since with a conservative estimate Python can still handle about 500,000 operations per second without much trouble. This will be enough for our use case, since there are only about 5^6 = 15,625 unique finger-fret placement combinations (not to mention very few of those would actually be well-defined chords which create musically harmonious notes). An algorithm which goes through every single possible combination to find what the best one would be to play is viable in Python, at about 500,000 / 15,625 = 32 possible chords played ***per second***. We figure that the speed capacity of the actual motor assemblies and hardware would be much more limiting than this, essentially being the bottleneck of the actual output of the autonomous guitar much more than any programming language's floating operations per second speed would affect it. **This is the language we ended up using for the MIDI preprocessing algorithm in the final project.**

3.2.2.5. MicroPython https://micropython.org/

MicroPython is an implementation of Python 3 (including some parts of the Python standard library) specifically designed and optimized to run on microcontrollers. It would allow us to use a comprehensive list of many useful features such as an interactive prompt, arbitrary precision integers, closures, list comprehensions, generators, and exception handling. And would also be compact enough to fit and run within 256 kilobytes of disk space and 16 kilobytes of RAM (Random Access Memory). It should also aid with development speed in terms of the technical scope and reducing the overall learning curve as it is designed to allow transferring code very easily from a desktop computer to a microcontroller or embedded system. We can also very quickly simulate real MicroPython code on the CPU emulator provided by https://micropython.org/unicorn/ for starting software development and testing before the parts have actually arrived, reducing one of our main anticipated blockers on progression on the software side of things. **This is the language we ended up using for the ESP32 microcontroller song-playing algorithm in the final project.**

3.2.3. Parsing MIDI File Bytes Data Stream

3.2.3.1. Mido Library

We can use the Mido Library for parsing MIDI files using the documentation found at https://mido.readthedocs.io/en/latest/parsing.html. **This is the library we ended up using for the final project.**

3.2.3.2. Pygame

We can also use Pygame for parsing MIDI files as shown in this example https://www.daniweb.com/programming/software-development/code/216979/embed-and-play-midi-music-in-your-code-python. Although this approach seems to be more for playing the file rather than giving us the individual low-level control we'd like for parsing individual notes and sending them over to the guitar.

3.2.3.3. PythonInMusic Documentation

"This page is divided in three sections: Music software written in Python, Music programming in Python, and Music software supporting Python" https://wiki.python.org/moin/PythonInMusic
This is the main page we are referencing for all our music library needs. It contains a plethora of good resources that will be instrumental in building and debugging our project's operation as well as sound quality.

3.2.3.4. MIDIFile

https://pypi.org/project/MIDIFile/ provides a very simple MIDI file parser and interface for retrieving individual notes which would be a great tool. However, the developers have said "It is known to run on MacOS and Linux. It should run on Windows, but then, nothing is certain when Windows is involved, is it? Attempts to make it run on Windows are at your own risk." so we will have to be wary of any errors due to the operating system choice as we are all running Windows computers.

3.2.4. Microprocessor Communication

3.2.4.1. Serial Communication

The autonomous guitar needs to have a way to receive a MIDI file from the computer to the microcontroller attached to it where the user can specify the file to send and send a signal to start or to stop playing the song, There are multiple methods of serial communication including UART, SPI and I2C. For our purposes UART will be needed and SPI could potentially be used.

3.2.4.1.1. UART

Universal Asynchronous Receiver Transmitter (UART) is a serial communication module that is used to transmit and receive serial data; it can also take parallel data and turn it into serial data. It is built into most microcontroller devices, UART transmits 8 bits at a time where for every 8 bits there is a start and stop bit for the transmitter to read. UART is only capable of communicating with one other device but uses a full duplex communication system so it can send or receive data at the same time it also can use any other communication mode such as simplex and half duplex. Most of our communication with a PC and the microcontroller will be one way so the ability to send data back to the computer is not very important but it could be useful for debugging purposes. The receiver and transmitter for UART needs to have the same baud rate for them to be able to communicate with each other. The range of UART is not very far but we are expecting the PC transmitting data to be close by so it will not be a problem.

UART was meant for communication between two microcontrollers but a UART port can be simulated using a program such as putty. When we use bluetooth or wifi to connect the PC or smartphone to the MCU we can make this integrated into the program with bluetooth or wifi.

Most microcontrollers will have multiple UART ports on them; this is important because UART only supports communication between two devices and parses that data to send it serially instead of in parallel.

UARTs main use in this project would be to send the MIDI file over to the microprocessor and get the response from the user to start playing the song and send a bit back to the user when the song is completed. The user will first pick a MIDI file they want the guitar to play from their PC then the MCU will receive a bit to start taking in the MIDI file data from the computer. UART has a Rx and a Tx buffer for receiving and transmitting, the guitar only needs to receive bits from the computer for the main function of the guitar for the purposes of debugging we could send the MIDI data back to the computer to see if they were changed into the proper note. In figure 14, it shows what functionality UART will have with the autonomous guitar.

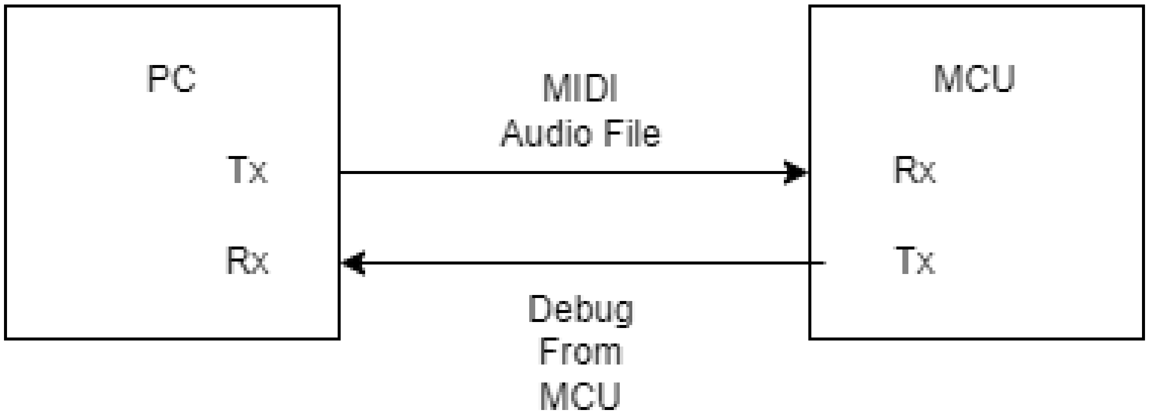


Figure 14: UART Structure For Autonomous Guitar

When debugging the code we will need to send a transmission from the MCU back to the PC to debug what is happening with the algorithm or anything that is not working correctly we will be using the software application PuTTy to read this information out from the microcontroller. We will be sending the MIDI data from the PC to the microcontroller, since all a MIDI file is, is a binary file to express which notes to play it will be simple to send it over to the MCU rather than if we did this with a more complex file format such as mp3 but in that case the same method still applies. The title of the MIDI file must also be sent over the PC as well as commands to start and stop playback and switching songs, but this can be done via sending an integer that will be the command being sent as an instruction.

3.2.4.1.2. USB

For the autonomous guitar, the only system that can perform the function of UART is USB, the benefit of USB over UART is that it is easier to debug the communication between the program on the PC or mobile phone since the communication protocol will be recognized by the computer, the downside of using USB is that it will not be able to connect to a smart phone this way which makes using USB connection for the final product not viable along with the fact that we want the autonomous guitar to be completely wireless. For these reasons USB will be required for testing and not for the final product. Most microcontrollers use the standard USB connector but some use a micro USB connector this will make no difference to the autonomous guitar project but micro USB connectors are not always as readily available as the standard USB connector.

Another thing to be aware of is not every connector for micro usb has a serial communication wire inside of it since many are used for chargers for devices that do not need serial communication functionality. That being said, finding a micro usb connector with a serial communication wire should not be a problem. Since Bluetooth cannot be used to establish a connection to the com port to upload code to the microcontroller using a usb connector will be necessary to upload the code to the microcontroller, as well as for purposes of debugging UART functionality before the Bluetooth functionality is added to the autonomous guitar.

3.2.4.1.3. SPI

SPI also has potential usage in the guitar, the microcontroller needs to control a total of 30 Servos for the guitar to be able to play the full range of notes, it also has to play them fast enough to keep up with the speed of the song but the song will come out slowed down from the original song and the microcontroller may not be able to control 30 Servos without the use of a shift register integrated circuit because of the lack of PWM pins. In this case we could use SPI to handle the communication between the microcontroller and the shift register. Unlike UART SPI can control multiple devices or slave devices for each master which would be the microcontroller in this case. SPI also uses 4 wires instead of 2 so it can transmit and receive data at the same time. SPI can only have one master and it is short range, even shorter than UART it is meant to be used between a master in this case an MCU and a slave which in the case of the autonomous guitar is would be the shift register ICs or it will be needed for a LCD screen which is one of our stretch goals. Also unlike UART SPI does not allow for error checking in the bits that it sends and it does not acknowledge the transmission when it is sent or received

The 4 wires available for use in SPI contain connections for MOSI (Master Out Slave Input) which is the wire that the Master uses to transmit data to the slave. MISO (Master Input Slave Output) which sends data from the slave to the master. SCLK (Serial Clock) is used to to schedule the serial clock from the master to the slave for reasons of synchronizing some process on the slave. CS (Chip Select) goes from the master to the slave to select which slave the master wants to communicate with.

There are multiple configurations to use with SPI master-slave communication for multiple slaves, because it is possible to use SPI with a potentially infinite amount of slaves, we need to be sure that the microcontroller has enough SPI pins on them to control all slaves. If there is enough SPI pins then all slaves can be wired in parallel to the master and then the chip select can be set up in a way where each slave is assigned to a different number and microcontroller can simply select the correct slave with the correct chip select. If there are not enough SPI pins for the system the slaves can be configured with the master in a daisy chain configuration in which MISO is sent to one slave and then that data is sent down to the other slaves until it gets to all of the slaves. The master is connects SCLK to all of the slaves so that their actions are synchronized and the master CS to all of the slaves to select all the slaves.

In Figure 15 and 16 below it shows both common configurations of SPI. Using SPI in a daisy chain requires less pins. Daisy Chain is more practical for our purposes since we will not have enough pins otherwise unless we use two microcontrollers.

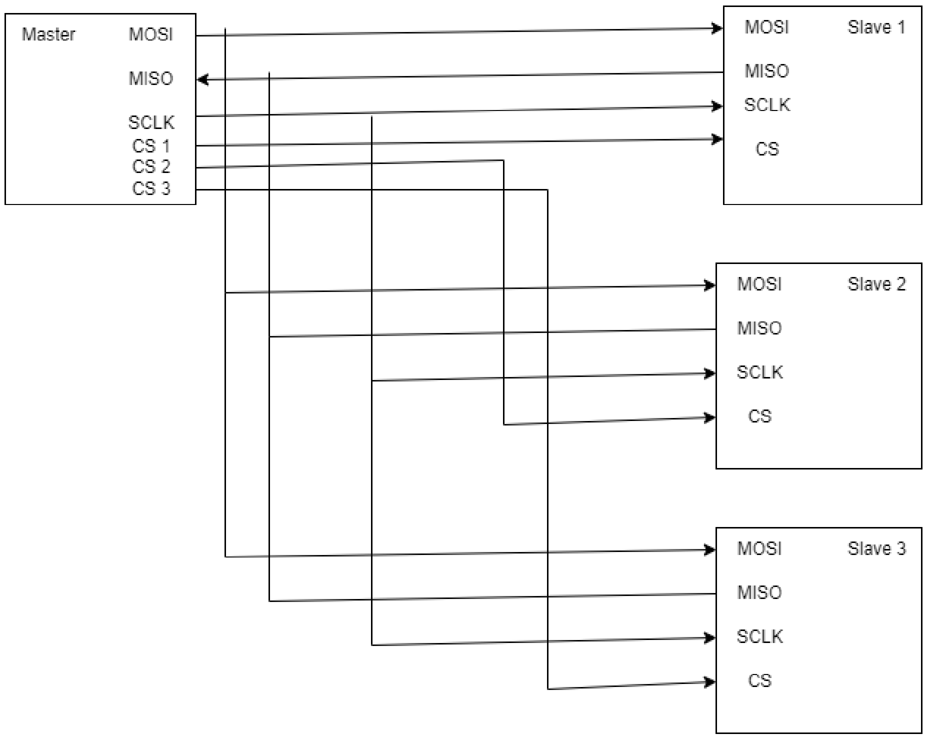


Figure 15: SPI in Parallel

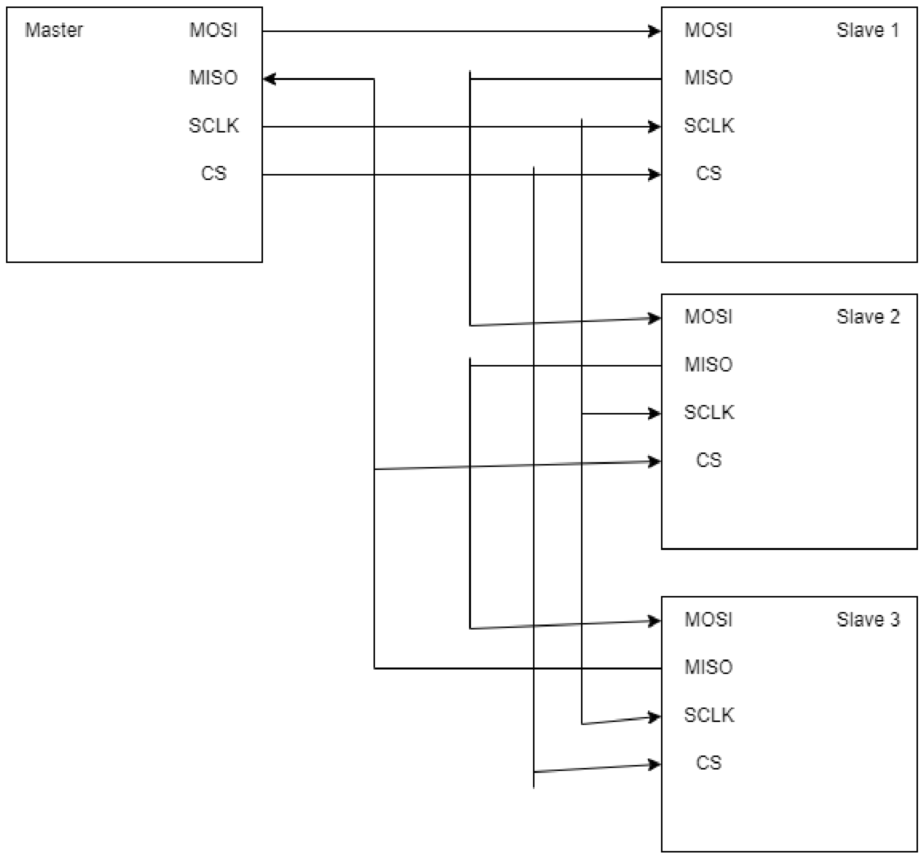


Figure 16: SPI in Daisy Chain

In the Autonomous guitar project daisy chaining has a potential usage in our shift registers if it is possible to make the response time faster than just using regular multiplexing to control the Servos, if we were to use SPI we would have to use the daisy chain configuration to connect the microcontroller to the shift registers because we will have 30 servos to control and each shift register will only be able to control 4 each which means the Autonomous Guitar will need 8 shift registers which means there will be 8 slaves. Also if we do implement our LCD screen stretch goal we will be using SPI to communicate with that as well. No microcontrollers that we have researched have had more than 3 or 4 chip select pins for SPI so daisy chain is the better configuration to use.

A shift register could also be used to implemented with a similar wire connection to SPI on the LCD screen but this configuration would require 3 pins on the microcontroller board this is much better than connecting it to the board directly but this is still not the minimum amount of pins we can connect the LCD screen with, and the LCD is not going to be demanding on data throughput. It could be used for the servos as well

3.2.4.1.4. I2C

The Autonomous guitar will have a LCD screen on the PCB however connecting an LCD screen without any serial communication protocols requires us to use many of the pins on our microcontroller. I2C allows us to do this with only 2 pins with its two wire interface; one of those wires carry data and the other carries a clock signal. The communication will be low range and the communication does not need to be fast as all that it is needed for is to display messages on the LCD screen. This will work well for controlling the LCD screen because models of LCD screens with a built in I2C model exist or a I2C module can be bought separately to control the LCD screen. I2C has a half duplex communication protocol which is not a problem for the autonomous guitar because the master will not be expecting any data back from the slave device which is the LCD screen.

The I2C module has an identical pinout to a 16x2 LCD screen and then the module would need to be connected to the SDA and SCLK pins of the microcontroller to use I2C and clear up pins which will be needed for the Autonomous Guitar servos. I2C also uses less pins than SPI does and we do not need the extra speed or range that could be given by SPI making I2C the better choice for the LCD screen. One potential consideration of using I2C versus using SPI is the power consumption because although I2C uses 2 pins instead of 4 with additional pins for extra devices unless the design is daisy chained, I2C uses more power than SPI despite being slower.

There are also modules so that we can use the Servo motors with I2C which would require the purchase of an additional module, using this method is more costly than a SPI solution with a shift register. The figure below demonstrates how we'll put the module into the project.

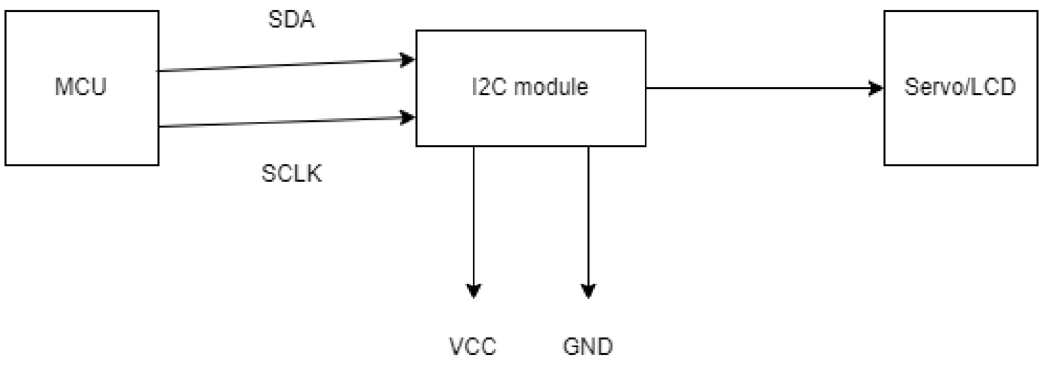


Figure 17: I2C Configuration for Electric Guitar

Either option for Servo or LCD I2C will require the purchase of an additional module, in our research we found no servo with built in I2C compatibility and the communication protocol is important for the servo because we need to control many of them and a communication method like SPI or I2C can help control these with the ability to save pins that would need to be used for the servos.

3.2.4.2. Serial Communication Comparison - BC

SPI and I2C both perform the same function but there are things to consider when implementing the protocol the things we need to consider include the following.

- Number of pins used is important since there are only so many pins on a microcontroller that can support the communication protocols however most of the time things can be done to reduce the pins needed.
- The data transfer rate is important because information needs to be sent to and from the master device at a reasonable speed.
- We need to know the price since that is one of the biggest factors in the feasibility of the implementation
- Our power consumption is limited by our requirements. It needs to stay inside the range of those requirements.

In the table below are comparisons between SPI and I2C, SPI uses 4 pins at minimum and that number can increase if slave devices are connected in the traditional fashion compared to daisy chain, with I2C the two pins will only be two pins regardless of how many devices are connected to it this can make setting up a I2C connection a bit more complicated than setting up SPI.

	SPI	I2C
Pin Count	4	2
Data transfer	Full-Duplex	Half-Duplex
Speed	100 MHz	Fast mode 400 KHz
Price	More expensive	Less expensive
Power consumption	Consumes less power	Consumes more power

Table 4: I2C and SPI comparison

The autonomous guitar is not expected to send any data back to the microcontroller, only from the microcontroller to the servos and the LCD. The speed of an I2C connection can also vary where with SPI the speed is fixed but can exceed the 100 MHz figure that is in the table. I2C could work in multiple speed modes but fast mode is there just for a benchmark, even at ultra-fast mode the speed is only 5 MHz. While I2C should be the more cost effective solution our research into parts have shown that this is not the case for our application, the I2C modules are regularly more expensive than buying shift registers to make an SPI connection because we can just use a daisy chain wiring configuration to mitigate the need for extra pins that would be needed to connect each shift register.

3.2.4.3. WIFI

Most microcontroller boards support wifi. We can connect the microcontroller for the guitar to transmit the MIDI file wirelessly. We need to create a wifi to serial bridge in the microcontroller. This would be done using UART since most microcontrollers already have functionality to use both but there are also many open source implementations that can create this connection. We can connect the microcontroller to the PC via LAN connection this way. This is a more complicated process then connecting the microcontroller through Bluetooth even though WiFi has a higher file transfer rate which could be used in our project, but WiFi has more power than the Autonomous guitar would need. WiFi would require more configuration on both the microcontroller side and on the PC side however we have been able to find an open source library that can implement a WiFi to UART bridge. For the Autonomous Guitar it is also being assumed that the PC transmitting the guitar is in a close proximity to the guitar that would not need a LAN network to cover. WiFi is capable of transmitting over the same frequency of 2.4 GHz as Bluetooth is capable of.

3.2.4.4. Bluetooth

Bluetooth is another way and probably the simplest way we can connect the microcontroller to a computer. Bluetooth can be used with UART or SPI, many microcontrollers have bluetooth capabilities built into them, Bluetooth modules can also be bought separately and placed in a PCB.

Bluetooth works on a personal area network (PAN) and is used for many common peripheral devices that would be used on a PC or a smart phone so most of these devices will have Bluetooth capability built in. This makes Bluetooth a good solution for getting the file and interacting with the electric guitar for the user. All the user would have to do is pair the guitar to the PC and then send the file to the guitar over bluetooth and that functionality would be built into the code, and the user will be able to initiate the playing of the guitar over bluetooth as well. BlueTooth is slower than WIFI when it comes to transferring files, this is because BlueTooth is low power and WIFI is not.

There is also Bluetooth low energy (BLE) this could be used to send commands to the guitar such as play and selecting a song but if we wanted to implement one of our stretch goals to allow wireless song uploads, this would not be a viable option due to micropython not having full support for BLE and having a low data rate.

3.2.4.5. Wireless Comparison

The following table is a comparison between numerous wireless communication protocols.

	WiFi	Bluetooth	BLE
Frequency	2.4-5 GHz	2.4 GHz	2.4 GHz

Standard	IEEE 802.11	Bluetooth SIG	Bluetooth SIG
Max Data Rate	2.4 Gbps	3 Mbps	200 Kbps
Channel Width	20, 40, 80, 160 MHz	1 MHz	2 MHz
Range	Long	Short	Very Short

Table 5: Comparison between WiFi, Bluetooth, and BLE

WiFi is the best option for the self playing guitar. It has a long range and making a user interface for the guitar is a simple task using HTML and Javascript compared to making an app to connect to the bluetooth and controlling it that way. Micropython does not support serial bluetooth which would work if we used C++. The ESP32 does not support 5GHz WiFi bands so it would need to connect to a dual band WiFi connection or a 2.4GHz WiFi connection.

3.2.4.6. PWM

All the microcontrollers that we looked at had dedicated PWM GPIO pins but a microcontroller with 30 PWM pins for all the required servos is not something that we are able to find. To solve this problem we could use a shift register or we could generate software PWM for our servo control where we artificially create a pulse and a duty cycle using the microcontroller clock and software interrupts to modulate the signal from the microcontroller to create an artificial PWM where the software interrupt will periodically cut off the connection to the servo according to the frequency that the clock will switch the voltage to the servo off and on like a duty cycle would. This does have the drawback of taking up more CPU power than hardware PWM would since the process is being done by software instead of just cutting the voltage to reach the desired force but whether that would be a problem for our application is something to consider.

PWM is also used for the purposes of controlling the servo rotation angle with the duty cycle time, where we can alter the duty cycle to change the angle of rotation.

It is also possible to use PWM for the LCD screen to denoise the screen without a potentiometer, this can be done by creating a low pass filter using a capacitor and resistor and charging the capacitor based on the PWM duty cycle to reduce the noise on the screen created by having too high of voltage.

3.3. Hardware Investigation & Considerations

3.3.1. Guitar

While not as technically complex as the rest of the project, the object of which guitar we will use was still important to consider. One of our biggest concerns was budget, and guitars have a massive price range. Electric guitars especially can be very expensive - thus, acoustic guitars looked like the appropriate choice. This came with the added benefit of less electromagnetic interference, as components like magnetic pickups are excluded. In this way we could reduce the noise that will be placed on signals, which can create issues for our servos by making the transition between duty cycles less clear and ultimately causing misreads. We ended up using an acoustic guitar for the final project.

3.3.2. Single Board Computers

Single board computers are an easy and versatile tool for any electronics project. They have tons of I/O, networking support, and processing power. All of these things would make our jobs easier. However, they tend to be somewhat costly, so they're not ideal for a finished product. Also, we will learn more from this project if we go with a simpler device that involves more challenges for connecting it with everything else. Still, we will investigate the specifics of a few single board computers and then compare them to some microcontrollers, to get a good idea of whether it would be feasible to use a microcontroller for this project.

3.3.2.1. Raspberry Pi 3 Model B

The Raspberry Pi runs a linux-based operating system called Raspbian which can run python code out-of-the-box. It has USB ports which can be used for quick file transfer, or can even be configured to take data and commands remotely via an internet connection. Raspberry PIs are also in very high demand and are out of stock in most places that we looked and were expensive where they were in stock.

3.3.2.2. Onion Omega 2

The Onion Omega 2 is an internet of things (IoT) module which functions mainly over WiFi. It has no USB or similar ports. It does however use a linux operating system, making it capable of running the high-level code. This is a lesser known module so it has good availability on Digi-Key. It is also very small, allowing it to be integrated into our PCB for maximum portability.

3.3.2.3. Odroid XU4

The Odroid XU4 is a very powerful computing board that can run full versions of Ubuntu and Android. It sports 8 cores, making it the most powerful option on this list. If we decide we need as much performance as possible based on our time complexity analysis, this is a very good contender.

3.3.2.4. Asus Tinker Board

The Asus Tinker Board is very similar to the Raspberry Pi 3 Model B. It has pretty much the same layout and many of the same features. The operating system it runs, Tinker OS, is very optimized, so it can outperform the Pi. It usually costs about twice as much as the Pi, but due to huge price inflation, it currently costs about half as much. This board should be preferred over the Pi if we choose to go with one of the two. We will likely only use the Pi if we happen to have an extra one on hand.

3.3.2.5. Libre Computer Board

This board is essentially a cheaper Raspberry Pi clone. The main difference to note is that it does not have built-in wifi, so if we want to add a wifi feature later in the project we'll need to purchase a dongle for at least $10. Even with that added cost, it's still competitively priced. If we need to buy a board and our budget ends up tight, this is a good option.

3.3.3. Microcontrollers

The microcontroller will run the high-level algorithm and output the control data for the motors. It also uses the clock on the microcontroller to synchronize the frets and the strings to play the full notes and properly time the motors to the flow of the song and to do this we need to use timer interrupts which require use of the system clock, the type of clock does not matter too much for this application but given the range of notes that guitars can play at any time it may need to be something that we consider.

To control the motors we also need to use PWM to change the angle of the servo's rotation because if the motor turns too far it could break the string, or not far enough and not make a sound on the string. The frets need to have PWM functionality as well to not break the string and to hold the string down firmly enough to play the note, for this we need the microprocessor to have enough PWM pins or have a way to implement the functionality in software.

It is preferable to our application that the microcontroller has enough pins to control all the motors on the strings and frets, but if not the motor control can be passed off to a motor control stage. Since we want the guitar to work without an external device the microcontroller must also be able to run the algorithm on the board itself.

Many microcontrollers can process MIDI files and play them if they have a module that can output sound, but our project still requires that this data get converted into notes to play on a guitar between 2 motors.

3.3.3.1. Arduino Due

The Arduino Due has a total of 54 GPIO pins of which 12 pins are analog pins and 12 are PWM pins the rest are digital, which would make it a good candidate for the motor control system however some of the servo's would need to be using a software PWM. It has an Atmel SAM3X8E ARM Cortex-M3 CPU, and a 3.3V power rail and a 5V power rail. I/O pins can only handle a maximum of 3.3V on this board which is something that should be considered.

3.3.3.2. MSP430FR6989

The MSP430FR6989 is more of a microcontroller than a microprocessor. It can run low-level C code. It is not designed to run a high level algorithm, if we were to use it, it would need to be connected to a computer via, bluetooth, wifi, or a microcontroller protocol like UART. since it is not optimal to run high-level code from this device it is not the ideal solution since we want this to be a contained system.

3.3.3.3. ESP32

The ESP32 is a Wi-Fi- and bluetooth-enabled microcontroller, making it suitable for IoT applications. It features up to 34 GPIO pins, meaning it could feasibly control all the motors without the need for a shift register. It has support for many languages, including Micropython.

3.3.3.4. STM32

The STM32 is a simple and cheap microcontroller that doesn't really stand out among the other options listed here.

3.3.4. Comparison Table

There are many options for the processor, so it helps to lay out the most important information in a table. The cost is an obvious factor. For the programming, the speed will determine how efficient the code needs to be. Pin count will determine whether we need to use the shift registers, or if we can control all 30 motors directly. A USB-A port is desirable for easily uploading MIDI files to the guitar. Wifi is an alternative option for this. The programming language will determine what libraries we will be able to use. The architecture may affect the difficulty of the programming. The pros and cons for each processor are provided below.

Microprocessor	Cost	Speed	Pins	USB	Wifi	Language	Architecture
Arduino Due	$46	84 MHz	54	No	No	C	32-bit ARM
Asus Tinker Board	$75	4 x 1.8 GHz	28	Yes	Yes	Any	32-bit ARM
ESP32	$15	160 MHz	34	No	Yes	Any	32-bit LX6
Libre Computer	$45	4 x 1.5 GHz	35	Yes	No	Any	64-bit ARM

Microprocessor	Cost	Speed	Pins	USB	Wifi	Language	Architecture
Board	🟨	🟩	🟩	🟩	🟥	🟩	🟩
MSP430FR6989	$20	16 MHz	48-83	No	No	C	16-bit ARM
Odroid XU4	$53	4 x 2.0 GHz 4 x 1.6 GHz	26	Yes	No	Any	32-bit ARM
Onion Omega 2	$29	580 MHz	18	No	Yes	Any	32-bit MIPS
Raspberry Pi 3 Model B	$156	4 x 1.2 GHz	26	Yes	Yes	Any	64-bit ARM
STM32	$12	240 MHz	37	No	No	C	32-bit ARM

Table 6: Single Board Computer and Microcontroller Comparison

3.3.5. Motor Control

Playing a guitar requires a high amount of dexterity. While this might be obvious to the layman, it becomes an especially challenging engineering problem with the inclusion of mechanical systems. Not only does a person have to either pluck strings individually or strum multiple strings at once for chords, but they also have to simultaneously fret (press down at the location corresponding to the note that needs to be played) which introduces an enormous number of possible configurations (5^6 = ~15,000 for the purposes of our project) for potential combinations of notes that can be played at once. In addition to this, guitars are designed such that some fret positions on one string overlap or correspond to the same note as playing a different fret position on a different string. Thus, fret combinations for each chord can exist at multiple points along the neck of the guitar, the only difference being a change in pitch between octaves. Ultimately this meant that we would have to utilize a number of motors in order to get the guitar to play. This could cause issues, since if the microprocessor does not have enough pins to control every motor, we will need an intermediate stage to take serial data from the processor and convert it into parallel control signals - This stage will be our Motor Control. This stage would also require the local placement of PCBs that would control power supply input so as to meet voltage requirements for individual parts, whether that be in the form of voltage regulators, transformers or amplifier circuits.

3.3.5.1. 74HC595N (Shift Register)

This serial-in, parallel out shift register would allow us to control all of the motors with just 3 pins from the microprocessor. One sends the serial data, the second shifts the data along the register, and the third forwards the data to the output pins. This configuration is perfect for our purposes, because servo motors require a PWM signal. We can shift the on and off states into the register and then forward them at the right times to get the correct PWM duty cycles. Also, these shift registers can be daisy-chained by connecting the last bit of one register to the serial input of the next. This is important because we expect to use about 30 motors, each needing independent control. Only about half of the considered single board computers and microcontrollers would have enough pins to control all the motors alone.

3.3.5.2. L293D (Motor Driver)

Most motors have just 2 connections: power and ground. However, microcontrollers are unable to supply enough power to drive a motor through their output pins. Because of this, we need some kind of switching component that separates the inputs into 3 connections: power, signal, and ground. The obvious component for this is a transistor, but we don’t have to go so simple. There exist integrated circuits with many transistors inside that are configured specifically for controlling motors. The L293D is one of them. This is the most common motor driver and we already have a few and we’ve confirmed that they would work for our project. However, if we choose to use only servo motors for our project, we will not be using this integrated circuit. The reason is that servo motors each have a driver built into the casing. They need this driver because they do not simply have on and off states, but can be configured to turn to any specific angle. The driver decodes a PWM signal and moves the motor to a specific angle based on the duty cycle of the input. As such, servos already have the separation of connections into power, signal, and ground.

3.3.5.3. PCA9685 (PWM/Servo Driver)

Instead of handling the PWM control of servos ourselves, we can use this board. Via I2C connection, our microcontroller can tell this board what angle to set each of the 30 motors and it will automatically generate the PWM signals for us. This greatly simplifies the design of both our circuit and our code. The I2C interface will likely have a library which both make the communication easy to work and and also faster than one we could write ourselves, as libraries are often implemented in a fast programming language like C in the backend. The only downside is the cost, which is $15 per board. Each board controls 16 servos, so we would need to purchase 2 of them. The two boards could be configured to different I2C addresses, so we could use a single bus to control all the motors.

3.3.6. Motors

The crux of our projects relies on the mechanical components that will work the guitar. There are a multitude of options on the market for turning electrical energy into mechanical movement, which will be discussed below.

3.3.6.1. Servos

Servos are the easiest motors to work with. They have separate connections for power and control, so it’s trivial to isolate the two. The power is simply 5V - likely, this will make up the bulk of our power demand. The control signal is a PWM (pulse-width modulation) signal with a period of 20ms and an on-time of roughly 0.6 to 2.3 ms. The duty cycle determines the angle that the servo will attempt to move towards. Servos are also very cheap, about $2 each.

3.3.6.1.1. Servos Linear Actuator

Servos rotational motion is good for strumming, but may not work well for fretting. If we were to use these for fretting, we would need to design some kind of mechanism—likely 3D printed—to convert the rotational motion into linear motion. An example found on the internet is given below.

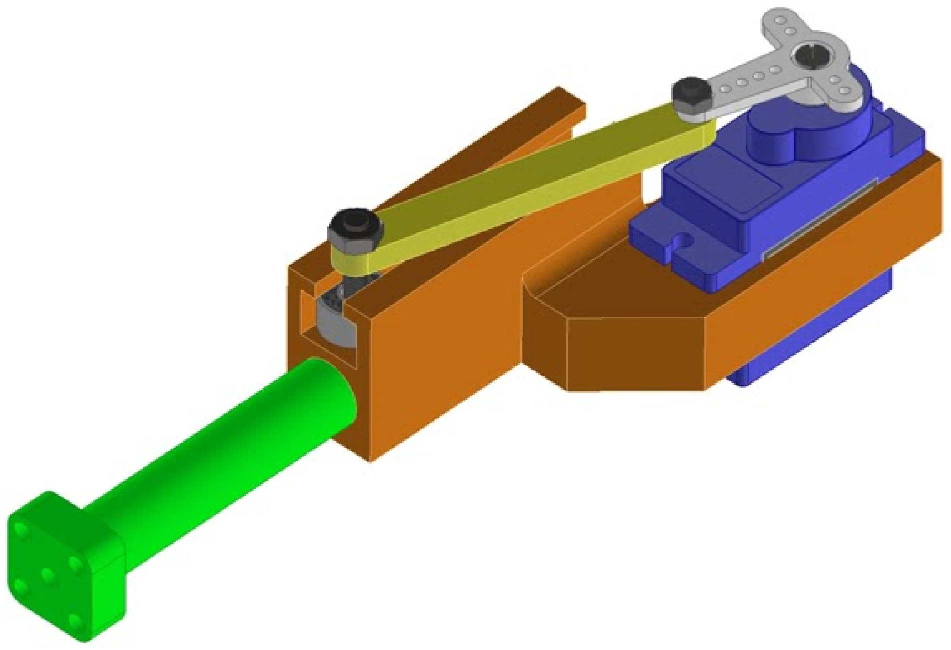


Figure 18: Assembly for conversion of rotational servo motion to linear actuation
https://www.youtube.com/watch?v=MeILaIGI1es

This design is very good for our project. It orients the servo vertically, which allows us to pack them closely together. This is important because the strings on a guitar are very close together. Even placed side-by-side, we can only fit 3 servos across the width of the neck of the guitar. The next advantage of this design is the mechanical advantage it provides to the servo. Near the bottom of the linear movement, the servo is essentially moving sideways, meaning it moves a large distance to create a very small movement near the guitar string. This means it has a mechanical advantage. The benefits of this are two-fold. Number 1, this will make it easier to achieve the proper amount of force on the string and to tune it finely based on the final position of the servo arm. Number 2, the decreased resistance on the servo motor itself will lower the passive power draw of the servo, which multiplied over 30 servos will be great for the demands on our power supply circuit. Our final model will likely be heavily based on this open-source design.

3.3.6.2. Solenoids

Solenoids essentially act as a push/pull arm, in this manner this is potentially the best choice for pressing down on frets. Solenoids would be more efficient than the servos due to the linear path they take, there is less room for error or chance it hits a neighboring fret. Some challenges that come with the solenoid is the cost and availability of them, they can vary from 5-20 dollars per solenoid which gets pricey with 29 needed in order to play frets. Additionally, solenoids typically have just 2 terminals, meaning we will need to use transistors or motor drivers like the L293 to isolate the control signals from the power.

3.3.6.3. Lead Screw Drive

The term “Lead Screw Drive” refers to the use of a screw assembly to drive the linked assembly along an axis. The lead screw is threaded through a hole in the assembly along the travel axis. The mechanical action can be described as the inverse of screwing in a bolt through a nut; rotational force is applied to the lead screw while the angle of the assembly is held fixed in relation to the screw (And thus, the travel axis). The assembly is forced along its movement axis by virtue of the threading linking it to the screw - in this way, rotational energy is converted to linear energy. Because of how the mechanism works, it would have to be used in tandem with a servo or another type of actuator which creates rotational force.

The following figure 6 is a rough model demonstrating the construction of a lead screw drive. The direction of linear travel depends on the direction of rotation of the leadscrew - one direction moves the assembly forward, and the other direction moves it backwards.

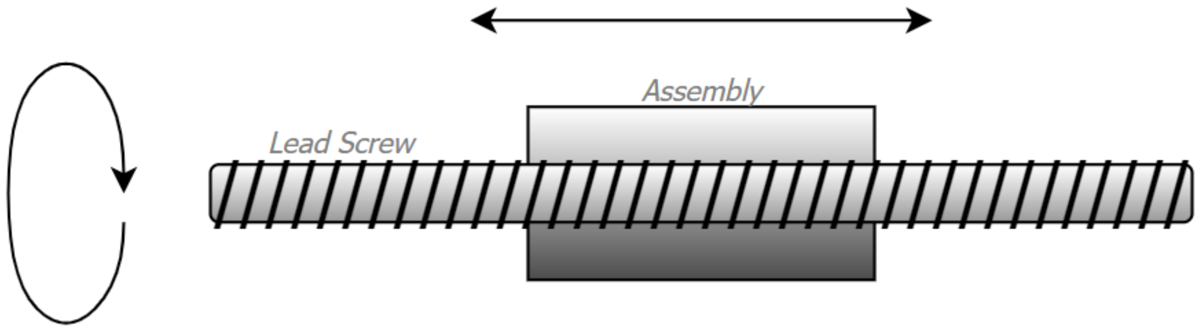


Figure 19: Lead Screw Drive X-Ray Model

3.3.6.3.1. Lead Screw Drive Linear Actuator

The most useful form that the Lead screw Drive can take for our project will be that of a Linear Actuator, which is a self-contained part designed for the end production of linear motion. These actuators vary in size, but are generally bulkier than solenoids. While their lengthwise design might be less practical for finer mechanical actions such as pressing frets, it would be useful as a slide for if we need to move a servo or servo assembly for flexibility in fretting for notes. We will keep this option in mind if we were to pursue a stretch goal involving more complex mechanical engineering for the project.

3.3.7. Power

Multiple considerations must be made when deciding which power source we will be using. Major design principles we will consider will be portability (i.e, weight and size profile) and charge duration. We also have to take into account what kind of impact the load will have on our parts; Voltage can be easily regulated using circuits to fit part specifications, but we must take care that not too much voltage is being dissipated across parts if we do not want to burn them out. Thus, it is vital that we find a voltage that minimizes power dissipation across parts. Noise will not be a major consideration, as in our utilization of our signals as a power supply we will only be dealing with DC signals. The impact of noise on our electronics will be further limited by usage of regulators for meeting part voltage requirements, which can be designed with minimal noise introduction to the system - however, as in any circuit, minimizing noise brings it closer to the ideal model which is preferable if we want to better predict the electrical behavior of the project and reduce time spent on trial-and-error.

3.3.7.1. Power Demand

As with all electrical applications, our project will require a power source. However, major considerations need to be made in our selection in order to conform to our primary objectives of sustainability and low failure rate. In our selection, we want a power source that has a high enough voltage output to supply every component within our project adequately (references for amplifier circuits, for instance), but we want to make sure that power output is not so high that it puts our components at risk.

To begin, a good starting point to look at for our voltage demands is the MSP430 family; The microprocessor will be key to the success of our project and likely one of our most sensitive components. Power failure or component damage would be intolerable, and the MSP430 being one of the most popular microprocessor family brands on the market gives us a good reference point - most microprocessors within the MSP430 family accept ranges from 1.8V to 3.6V.

In addition to this, most servo motors - the kind that we will be utilizing for our project - typically require 5V to operate, so that leaves us with a minimum floor value at 5 volts for the project. However, the large array of motors we will utilize means that considerations must be made for power tolerances - the combined power demand of the motors we will be utilizing means that it is a real possibility for our current to become out of control and destroy components. Running 30 servos which are rated at an input current of approx. 270mA max running input current means that, at most, we will see 8.1 Amperes of current if every servo in the system at once is turning. The input current of the microprocessor is 0.5A, leaving us with a total max running current of 8.6 Amperes (The current that the power supply will experience). At these currents, a fuse might be worth looking into to preserve our components (or ourselves) in the event of fault conditions.

Ultimately, we will likely set our supply voltage to be somewhere above 5 volts before we step down our voltage using DC-DC converters. We will supplement our power supply with supporting components such as switches to safely cut power to the project. Beyond this, various technologies that might prove themselves to be useful for power management will be discussed in the following sections, while specific examples for power supplies will be discussed here.

3.3.7.1.1. TalentCell Rechargeable 12V 6000mAh

Battery Packs refer to batteries that are often used to charge mobile devices. Specifically, the battery pack mentioned here features a 12 volt output with 6000mAh, which should yield ~42 minutes worth of use at max current draw. Realistically this current draw will be much lower (say, 25%), so we can estimate it to be around 3 hours worth of operation.

The issue arises with the observation that this, and all other mobile device battery packs that we could find, only output around 2A - 3A of current. Placing them in parallel would alleviate this but buying 3 of this brand would start to weigh heavily on our budget. This will not work for our project with the current demands of our motors and the scope of our budget, so we will have to take our search elsewhere.

3.3.7.1.2. Amazon Basics 9 Volt Everyday Alkaline Batteries

9 Volt batteries see use for applications that require high voltage and power, which accurately describes the power demands of our project. However, the cost-to-capacity ratio of these batteries are worth mentioning; alkaline 9 volts typically have a capacity of 550mAh, and assuming our average current draw to be approx. 25% of our max we have a demand of 2.15A. This yields 15 minutes of powering our device, which is extremely low - not to mention the excessive current could damage the batteries. Both of these problems could be solved by placing the batteries in parallel - although the max current characteristics of this battery were not able to be found, 8 9V batteries in parallel should be enough to maintain our average current demand for a reasonable amount of time.

3.3.7.1.3. Lead Acid - 12V 8A ExpertPower

The 12V 8A Emergency Light Battery from Mighty Max Batteries boasts an impressive 12V output voltage and 8A output current. This falls within our reasonably expected tolerances for current - while it would be nice to break 8.6A and be 100% certain, we can safely assume that the current will be well below this value. Of note is the 8Ah capacity, which falls within our expectations for power supply longevity in a single convenient package. This would yield us a little less than an hour's worth of operation at max current draw, and if we were to approximate the average current at 25% of the max as we have done for the 9 Volts this would yield around 4 hours of operation for the device. The price is also affordable, being $29.05 when purchased from Amazon. - The profile is small, being a rectangular design that is 3.94x5.94in with a depth of 2.56in. Ultimately, this is the most promising power supply that we will be considering.

3.3.7.2. Linear Voltage Regulator

Linear Voltage Regulators fall under one of the two major DC-DC converter components that we will be considering for powering our project. The following is a selection of Linear Voltage Regulators, with careful in-depth analysis of each component's pros and cons and whether or not they will be suitable for our purposes.

3.3.7.2.1. LP2985

The LP2985 is a linear voltage regulator made by TI that is characterized explicitly by its low-dropout voltage. It comes in 8 fixed output voltages - the ones we will be interested in are the 3.3V, 5V and 10V options. The low dropout characteristics (typically 0.27-0.28V at its max current of 150mA) means that we would be able to comfortably limit the output of our power supply to 10.5V minimum, with the extra ~.2V as additional tolerance. In addition, the pricing is remarkably affordable - From TI, the LP2985 is available by the reel in quantities of 1 to 99 for $0.359.

A discrepancy exists in our power output, however - The device is recommended to be run such that it outputs a 150mA continuous load current, and cursory research into the most popular servo on the market (SG90) indicates an approximate running input current of 270mA. Thus, this device will most likely fail to power even one motor before failure. This could be circumvented if we use multiple regulators for each component or implement current amplifier circuits - both of these implementations would be impractical due to space, cost and time restraints, so it would be best to take our search elsewhere.

3.3.7.2.2. LP5912-EP

The above linear regulator's current output is too low to support even a single motor's functioning at the desired voltage. The selection of the LP5912-EP is intended to solve this issue - As described in its datasheet from TI, the LP5912-EP exhibits similar characteristics to the LP2989, namely a low-dropout voltage - However, the major difference between the two is that the LP5912-EP is designed to supply up to 500mA of output current. The datasheet is gone into more detail in the next section.

From the perspective of voltage, it is specified that the model comes with fixed outputs from 0.8V to 5.5V in 0.025V steps. While the versatility is a benefit for our purposes, and it falls within our desired range for voltage outputs, custom ordering the part might lead to a significant lead time from purchase to arrival. This can be mitigated by punctual timing in ordering the part, but still is important to consider.

In addition to the aforementioned properties, we can pull from the data sheet that the part is built with a low quiescent current in mind. This would assist us in maintaining the primary objective of sustainability through power efficiency - Overall, the part seems to be more attractive than the LP2985. However, because cost is a major consideration for our project, other options will still be explored in order to maximize cost-effectiveness and minimize financial impact.

3.3.7.2.3. Custom Linear Regulator

Linear Voltage Regulators are a subject of study for Electrical Engineering students as part of Electronics 2, and thus it is within our capabilities to design our own Voltage Regulators. There are some benefits and drawbacks to going down this road, which will be discussed below.

The biggest benefit of realizing our own design for a regulator is the flexibility. We would have the most control over the input and output characteristics of the component and thus it could be built to fit into our project seamlessly. Below is a diagram realizing a linear voltage regulator design derived from our work in Electronics 2. Using this schematic, we could design the regulator such that it outputs all the current required to power each SG90 servo motor in parallel at max current draw by including our own parts rated at such currents.

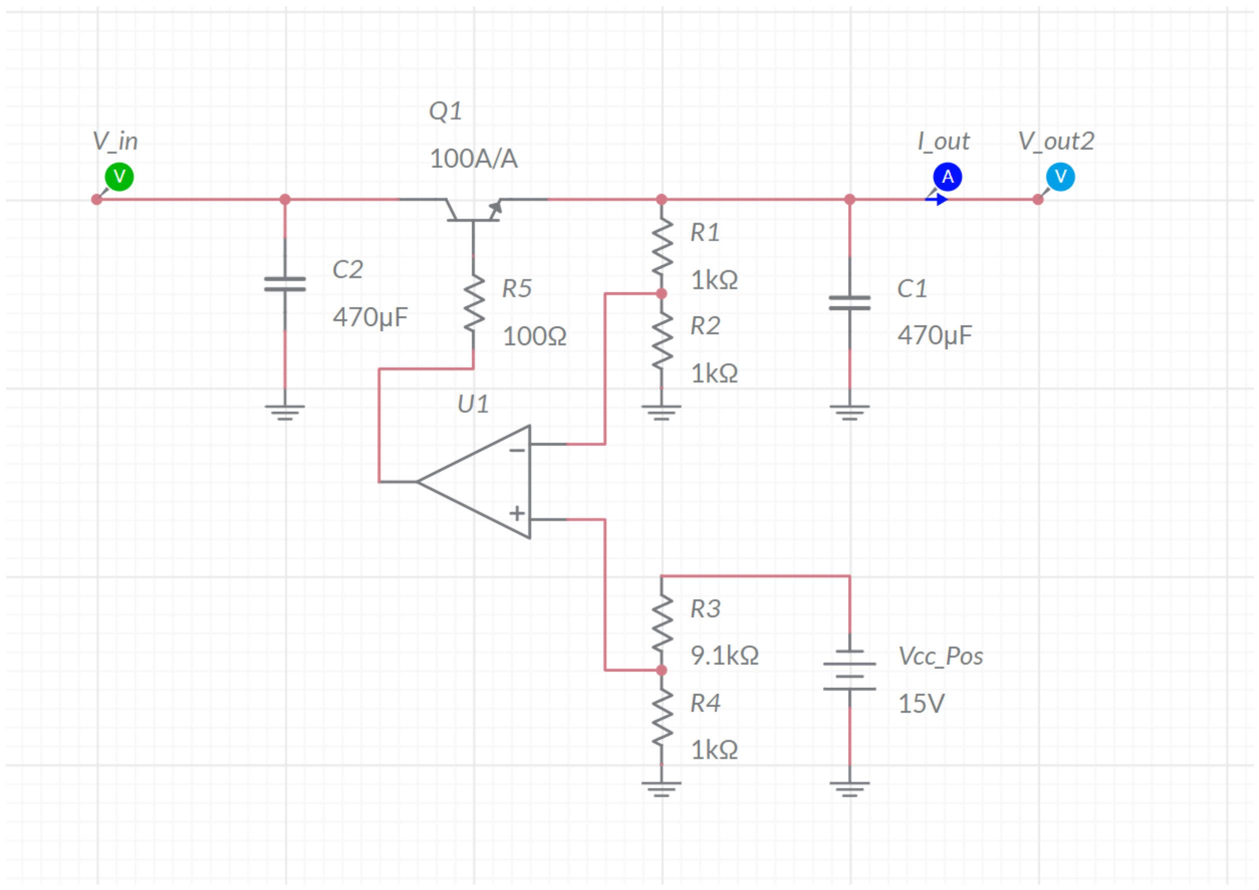


Figure 20: Custom Linear Regulator

In this configuration, the target voltage above which our input becomes clamped is $V_{out} = 1.485 * (R1 + R2)/(R2)$. For our project, we would require parts that can sustain high currents - namely, the Amplifier would have to be rated for approximately $270mA * 30 = 8.1A$. This is a massive current to run through a system, which would require wires with low enough resistivity to handle the total current before being split off to the servos in parallel. Ultimately however, these characteristics can be built for.

While the positives for designing our own circuit are apparent, the downsides are equally significant - as breadboards will not be allowed on our project, we would have to design a PCB to realize the circuit. In itself this is not a monumental task, although it can be time consuming to draft up a design in EAGLE (or a similar PCB program) and actualize it. This can be tackled either by having the design be built by a separate company or building it ourselves using equipment from the Senior Design lab - the discussion of how to go about including PCBs into the project warrants its own section, and so it will not be discussed in detail here as most of the considerations here would fall under that same umbrella. To summarize, PCBs represent a major time investment and, since at least one is required as part of our constraints, we would be better off limiting the number of PCBs in our design.

3.3.7.2.1. Diode protection

When using the linear voltage regulator we can ensure that our output voltage is what we are looking for but we need to be careful with our linear regulator in order to not burn out the regulator. Which would cause damages and need us to replace it, in one option we can use diode protection by positioning a diode between the input and output we can make sure that our linear regulator does not burn up when the voltage becomes to much and begins to burn up the diode draws some of the voltage in order to protect the regulator against any changes we make. There are different types of diodes we can use in order to implement diode protection. The schottky diode is commonly used in diode protection due to their fast forward action and low forward voltage drop. This is an option we would use if we were going to use some sort of individual linear voltage regulator for example if we used a LD1085V50, with this regulator it outputs 5 volts but the same issue we run into is we do not have enough current as it can only push 3 A in this situation we will need to create a current gain circuit. This is something that was especially covered when we were in electronics 1. When designing the current gain we will more than likely use a BJT as its characteristics are simpler to calculate. If we end up using some sort of singular regulator component this is a good idea to protect it but if we decide to go the route of creating a circuit regulator it would not be nearly as ideal to use this and more than likely a waste of resources. The figure below is a demonstration of implementing a schottky diode to protect a linear regulator.

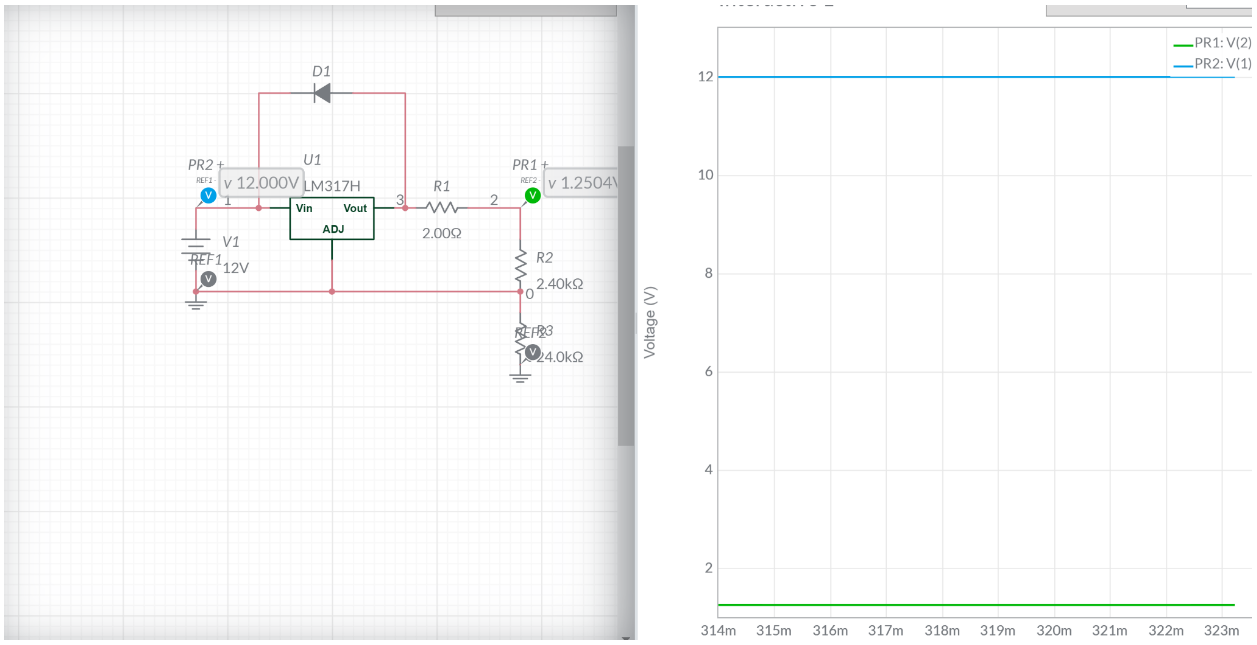


Figure 21: Linear Regulator with Voltage Protection

3.3.7.2.2. Heat sink

When the voltage gets stuck up in the regulator it causes the efficiency to decrease and the loss of efficiency is due to heat. This can become a problem as this is what is happening when the regulator is burning up. Something we can do in order to solve this problem is to attach a heat sink to the regulator that will allow the heat to transfer to material that will absorb the heat that is generated in order to keep our regulator intact.

If we were to use a linear voltage regulator with a heat sink this will allow us to use a higher voltage that would be provided from our power source. If we use a bigger power source the voltage would still come out as 5 V depending on the linear regulator we use but with a large power source we can get the large amount of current that we need. But with that we will experience more heat in the voltage regulator which will lead us to putting in our heat sink to absorb some of that heat. Some common materials we would use to bind the heatsink to the regulator would be silicon or materials which share similar properties to it. The heat sink is something that can come into play by looking at the diode protection circuit example. In that we are losing efficiency somewhere and it is most likely due to heat so in order to protect our regulator from burning up we can put a heat sink attached to it on top of the diode protection and moving forward we will have no issue with damaging parts. The following figure demonstrates a common design for heat sinks for use with integrated circuit components. This is something we realized we might need to think about after our breadboard circuit, when we would use the breadboard we would have our regulators get extremely hot at some points and it got to the point where it actually melted the breadboard and a small fire did occur. However we knew the PCB would provide some coverage as well as a more secure connection. Once we implemented the regulators onto the PCB we didn't have the issue anymore so we didn't need to implement it.

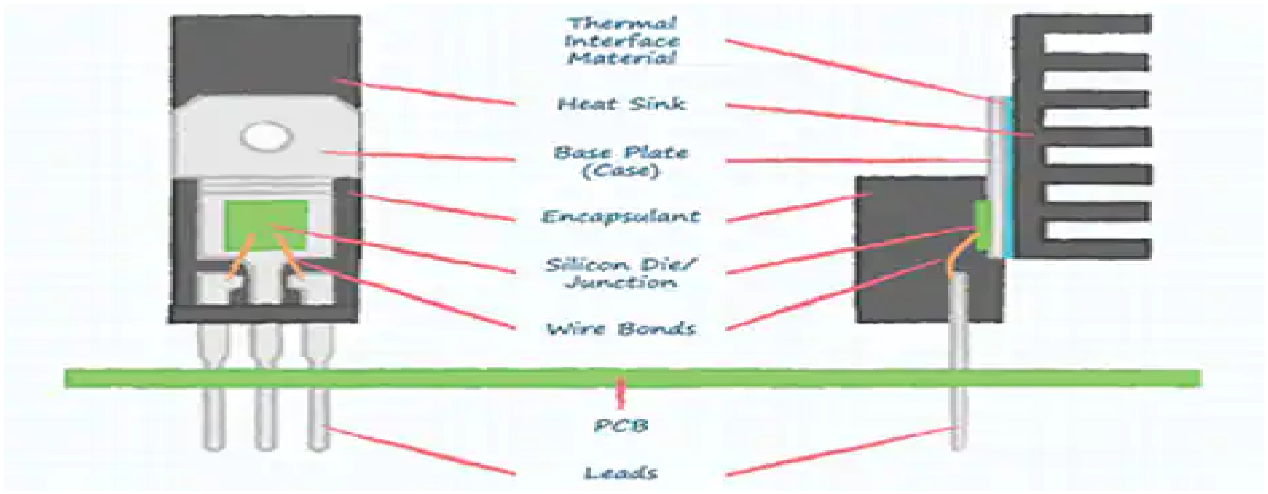


Figure 22: Heat Sink Attached to a Regulator

3.3.8.3. Switching Regulator

Switching Regulators are the other DC-DC converters we will be looking at for implementation into our project. It is generally understood that they see much more utilization in the tech world in comparison to linear regulators with regards to power supply management, and so we will be conducting more thorough research into the types of devices that are out there - As well, we will more extensively weigh the benefits between the purchasing of parts and the creation of our own.

As a foreword, because switching regulators are generally more efficient than linear regulators by virtue of the mechanism they use to clamp a given voltage we can assume that the following parts will be superior to linear regulators in terms of fostering longevity and duration of the power supply. Despite this, specific characteristics will be discussed so we can paint a complete picture of the technology that exists and how it can be used to benefit our project.

3.3.8.3.1. TPS62992-Q1

The TPS62992-Q1 is a newer model of Switching Regulator from TI. It is described by their datasheet as a “Highly efficient, small, and flexible step-down DC-DC converter that is easy to use.” Ease of use and implementation will be important for our project, as the limited time and cost scope means that utilizing components which take less time to accommodate is preferable. Step-down (or, “buck”) converters will likely be preferable for our motor assembly, as the 8.1A current draw is a considerable hurdle that can be solved by converting excess voltage to current.

The output voltage is defined as being from 0.4V to 5.5V, the range of which contains both desired values of 3.3V and 5.0V for the microcontroller and motor assemblies respectively. The issue arises when looking at output current; the device is rated for a maximum adjustable output current of 2A, which falls well below the demands for our motor assembly. Thus, should we implement this part, we will likely utilize it for our microcontroller which has significantly relaxed current demands.

The device features above-average safety characteristics, with overcurrent and over-temperature protection. This further cements the part as a good candidate for our microcontroller, as any failures that could damage the microcontroller would be catastrophic for our project and lead to significant time and cost penalties.

3.3.8.3.2. TPS568231

The TPS568231 is, as described by TI, a “Synchronous Buck Converter with D-CAP3 Control”. At first blush, this title offers up a few insights into the characteristics of the device; to start, some research into the topic indicates that synchronous converters are much more efficient than their asynchronous counterparts, so we can expect a considerable efficiency rating for power being transferred from the supply to the components. Next we see that the device utilizes D-CAP3, which is the most recent implementation of a series of TI circuits for pulse width modulation. This offers the cleanest transient performance characteristics which, for a switching regulator, translates to a cleaner operation of the switching component and ultimately less noise produced. As we have stated earlier, noise is not a major consideration for power characteristics. Nevertheless, less noise is never a bad thing and so we can make note of it during our part selection. In addition to the characteristics described above, we see from the datasheet that no external compensation is required, which will help considerably during the integration of the device into our project as less components will be required for our PCB.

The most important feature of the datasheet is the inclusion of an "Application and Implementation" section. In this section, it outlines a generic PCB implementation of a regulator using the part along with specific part selections. The notable features of this implementation will be discussed.

Most important of our considerations is the output Voltage and Current. The output voltage of the device is controlled directly by the ratio between the Upper and Lower Feedback resistors in the provided application schematic (Appendix D); the output voltage is calculated as $0.6 * (1 + (R_{Upper}/R_{Lower})$, similar to a non inverting amplifier with an input voltage of 0.6V. For our purposes, and approximating the given values for the datasheet's recommended component values, our R_{Upper} and R_{Lower} will be 82.5kΩ 10kΩ respectively. It can be observed in the performance specifications that the output current of the regulator typically lies at 8 Amps; this is considerable for our project, as the current draw of our motor assemblies in parallel call for a max of 8.1A during full operation of each motor - It is unrealistic to expect that our current will ever be maintained at this value as the motors will not be running continuously throughout their operation, and so 8A is acceptable.

The device features some additional characteristics that can be used to tailor the performance to our project; the MODE pin on the device can be adjusted by a voltage divider ratio which in turn governs Light Load Operation, Current Limit and PWM Frequency. Light Load Operation is defined by the device and effectively allows us to choose between more efficiency or less output ripple - for our purposes, we will likely choose the efficiency option.

In summary, this component seems like the best option for providing power from the supply to the motors. It is able to handle the sizeable current demands while also maintaining good voltage characteristics to maintain the motors well above the demands they likely will be operating at.

3.3.8.3.3. Custom Switching Regulator

As with the linear regulator, it is worth looking into the topic of creating a custom Switching Regulator that we could custom-tailor to the design parameters of our project. In order to accomplish this, we would have to design our own Pulse Width Modulator circuit which we would have to do using an Op-Amp on top of utilizing an AC-Source to create the modulated signal. In order to ensure the device operates without fault, we have to make sure that the Diode, MOSFET and Inductor are rated to maintain up to and including 8A of current. These assorted components can be found on websites such as Digikey or Mouser and so shouldn't pose a big issue.

The biggest issue we would face however is the utilization of a sawtooth or triangular AC signal. To do this would require separate equipment that would quickly evolve beyond the scope of the project. It would be preferable to look at specific parts rather than designing our own switching regulator circuit.

3.3.8.3.4. Boost Converter

Boost converters are way in order to get our voltage higher if it needs to be, it operates by switching between two modes of an open and closed switch and with an inductor that stores the energy resulting in the load output being a higher voltage than what was imputed. This would be good however this would require a control wire that opens and closes the switch, this could be doable but one of the issues is space on our board since we have 31 pins designated to the servos space. Which could get our board getting even messier than it already is. One reason we would like to use a boost converter is because we can use a small power supply and boost the voltage that's going to the circuit. The only thing is the amount of current that comes out of the smaller power supply leads to us needing a current gain circuit which we could implement using a BJT or a MOSFET. The following figure is a demonstration of a common Boost Converter setup which implements a switch.

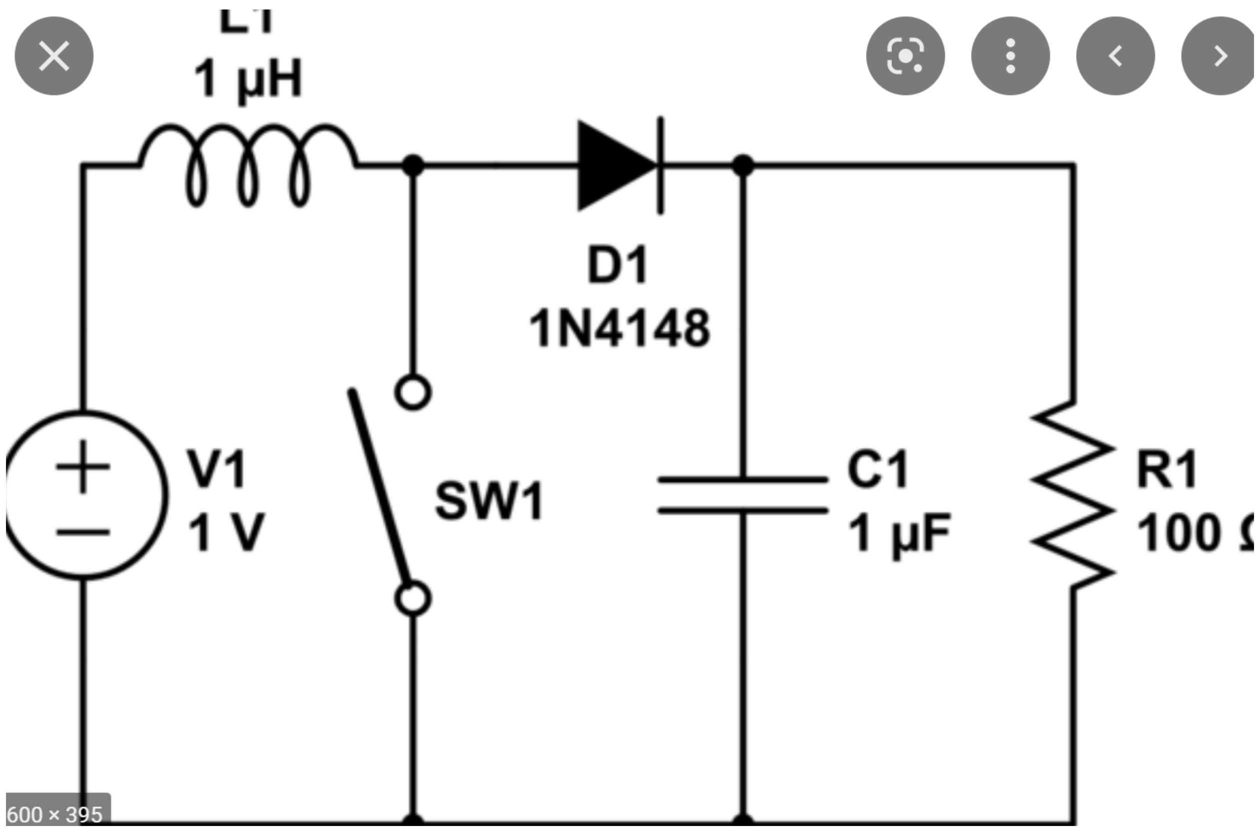


Figure 23: Boost Converter

3.3.8.3.5. Buck Converter

A buck converter would be good if we had too much voltage and didn't want to damage the servos or the boards at all. We can drop the voltage by switching a transistor open and closed and having the loan on an inductor. This could be an option if the power supply we get ends up being too much voltage. Like the boost converter this requires a control signal and has the same issue of space on the board being limited. A use we could have is if we need to have a larger power supply that can produce a lot of current that we need for our project. This is an ideal way for us to not only have a lot of current and at the same time limit the amount of voltage that we can use. An issue we could have with this is that the power supply we would use in order to draw a lot of current would be large and could take up a lot of space which ideally we would like to limit. Below is a common buck converter configuration realized using passive components and a high frequency switch, generally controlled by a PWM signal.

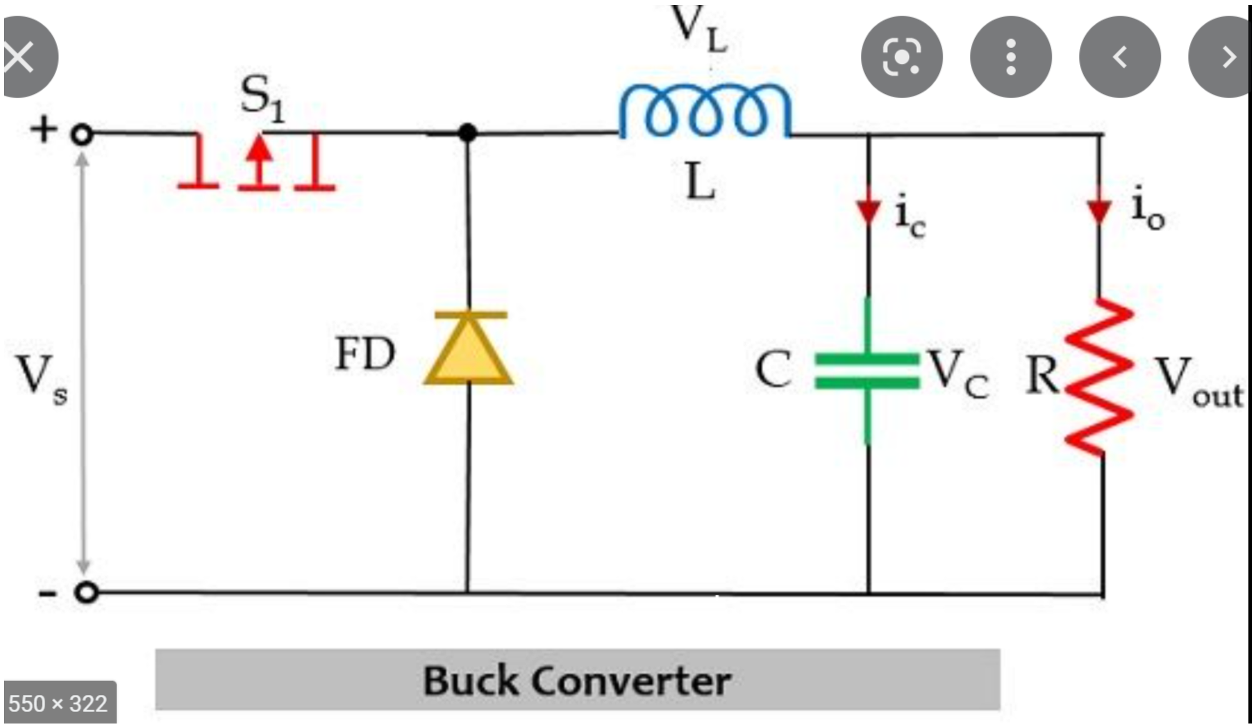


Figure 24: Buck Converter

3.3.8.4 DC Power supply

DC power is more than likely the simplest and easiest option in order to power our electronics. This is due to most microcontrollers using DC inputs in order to power them - the easiest way to apply DC input is to use batteries which will give us around 12 V.

Depending on the application we may need to drop or increase the voltage using a DC to DC converter or Op amp. This is the easiest option to use as well as affordable.

There are a variety of size and rating options available for batteries. The most common ones we see are AA and AAAs, which generally output 1.5V depending on the manufacturer. These voltages are insufficient by themselves - one option is to place the batteries in series so as to yield a higher total voltage by way of superposition, however there is the potential that we could run too much current through the batteries and damage them.

While it might not be preferable to connect individual batteries in series ourselves, there are battery packs that exist which are made with readily-available current thresholds. Most higher capacity and/or voltage batteries (such as lantern or car batteries) are excessively heavy which would go against one of the primary goals of being lightweight and portable. Thus the option of a medium voltage battery pack (i.e, 6V) would strike a balance between battery weight, duration and voltage output that would be favorable for our specifications.

3.3.8.5 AC Power supply

AC power is a commonly used power source and is most notable for being the voltage that comes out of the wall. However AC power does have a lot of negatives in respect to our project. The biggest issue is that in order to make it portable we need some sort of small power source like batteries which are a perfect option since they are DC in order to get some sort of AC battery or power source small like that which causes an unnecessary cost increase to our project. We could get AC power from an outlet however at that point our project no longer becomes portable. Another issue is that most of the microcontrollers work on DC inputs so in order to get DC power from an AC power supply we would need to use some sort of rectifier circuits that add unnecessary complications to the circuit design. For these reasons we will more than likely be using a DC power supply.

3.3.8.6 Op-Amp as a Regulator

Op Amps are a piece used in standard electronics especially in regulator circuits which act similarly to DC to DC converter. They have simple circuits in order to either limit or boost the voltage applied, however while it is a good electrical component using something more efficient is more preferable like a switching regulator. Also there are some small discrepancies on op amps like leaking current and offset voltage which can cause some areas of issue for us when powering the device.

3.3.9. Signals Technologies

Signals, for the purposes of this section, refers to those technologies relating to the use of current and voltage to relay information rather than strictly as a power supply. Not only do circuit configurations exist for the purposes of power management, but there are also documented integrated circuits with the express purpose of helping to amplify, filter and generally help take advantage of control signals.

3.3.9.1 BJT/MOSFET Amplifier Circuits

BJT and MOSFET amplifier circuits are a common staple for the implementation and functionality of devices that use analog signals. While not having much of a place for supplying power to the design, they can help immensely for amplifier control signals such as the PWM that we will be using to control our servos.

The option to include Transistor Amplifiers hinges on a couple factors. Transistor amplifiers could be utilized for satisfying both the power demands of the system as well as enhancing the signals that will be used to control our mechanical components. This section will go over applications of a transistor amplifier within the scope of this project, kind of parts that we could find useful for amplifying control signals, what makes them useful for this application, and ultimately whether or not the inclusion of amplifier circuits would be practical or necessary for the project.

The most useful form of transistor amplifier we could employ for our project is the Common Collector BJT Amplifier - Generally speaking, BJT amplifiers are employed for signal applications, such as audio interfaces. However, the circuit could work with DC signals as well, and the implementation of this amplifier circuit would allow us to relax our current demands for components governing the power supply of the project as Common Collector amplifiers are often employed for current amplification. In effect, this would allow us to expand our part selection for our voltage regulation circuits, as well as allow us to save on cost by using cheaper parts to circumvent the high current issue; A voltage regulator IC component is already much more expensive than a single transistor, and the cost of a voltage regulator with high power tolerances compounds upon this.

One downside we would have to consider, however, is the inclusion of a supply voltage for our amplifier. This would introduce a level of extra complexity into the design - nothing in our objectives or specifications requires that the power supply be a single unit, however it would still be convenient to lessen the amount of independent components in the system. Assuming the implementation of this circuit, the following section goes over a number of transistor models that would be suitable for this application.

3.3.9.1.1. MJD44H11AJ

The MJD44H11A family of transistors is a surface mount BJT transistor that is built for high power demands. It is marketed as having a “High power dissipation capability”, and “High energy efficiency due to less heat generation.” The biggest reason for this part’s inclusion for our research is its high collector current tolerance - the MJD44H11A is rated for a maximum collector current of 8A. The voltage tolerances as well are way beyond the values we will expect to put this transistor through, with a maximum lying at 80V. The part is available from Mouser, and would likely be easily found in parts libraries for software such as Multisim or LTSpice. The pricing is also affordable, with a single unit costing 60 cents, not including shipping costs. With the affordability of the part alongside its capability, we can say with confidence that this will be the part we would go for if including this technology. The question of including the circuit at all is mostly an analysis of cost to complexity - if we were to include this component then it would have to be to run less current through a linear voltage regulator - our selection of linear voltage regulators (and, generally, all linear voltage regulators) has a much lower tolerance for current than our switching regulators while being much cheaper. Overall, should cost become a concern, we will keep this option in mind.

3.3.9.2 Op-Amp Amplifier Circuits

Op-Amps are increasingly popular for use in amplifying signals. Because of their effectively infinite input impedance they do not suffer from the loading effect as BJT/MOSFET amplifiers do. They are also generally much more effective at amplifying gain than BJTs, and while the use of op-amps can introduce a noise component this noise can usually be calculated predictably. The biggest downside to including Op-Amp Amplifiers in our project however would have to be its even higher demand for supply voltage. Most amplifiers require both a positive and negative 15 Volt supply voltage; while the inclusion of a supply might be manageable for a transistor amplifier, implementing this would simply require too much accommodation to be worth using. Thus, we will probably not be using this anywhere in our project.

3.3.10. 3D Printed Parts

3D printing is an extremely powerful tool that has come into mainstream appeal as of recent years. In brief, it allows for the rapid and custom prototyping of parts for anything from individual projects to large-scale corporate-backed efforts. Naturally, as 3D printing is considerably popular with technical and hobbyist applications, it is a suitable addition to our project. Rather than having to purchase miscellaneous hardware such as bracket mounts, we can design these in a 3D modeling program and outsource it to be printed. For our purposes this will likely either be the Senior Design lab that we have access to as part of Senior Design 1, or simply printed ourselves using a 3D printer owned by one of the group members. The endless design possibilities leaves a lot to consider for how it should be applied to the project.

The strumming and fret assemblies will need to have 3D printed parts and it needs 3D printed picks that can fit on to the servos and parts to press down on the frets. The assemblies need to have the structural integrity so that it does not break while the servo plucks the string, the pick on the servo needs to be able pluck the string without breaking, and the parts on the fret motors needs to be able to hold the string down hard enough for the pick on the strumming assembly to play the note it has to be able to do this multiple times and quickly to play the song

The pick can be designed after a regular guitar pick but with a circular structure to where you would hold it, to attach it to the servo This also needs to have a buffered design on the guitar since the servo is larger than the gap between the strings. The figure below demonstrates how these servos would be laid out for the strumming assembly.

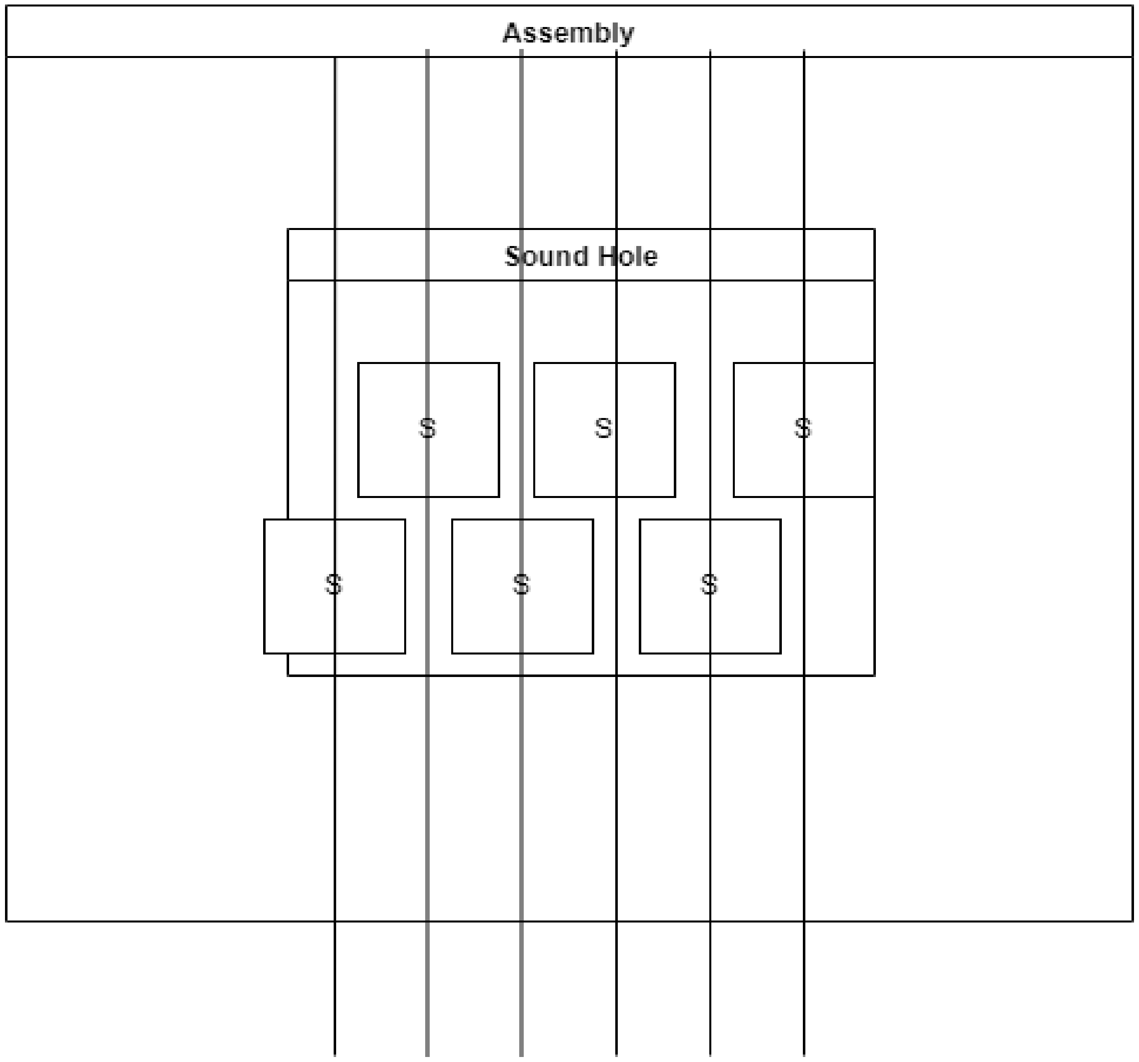


Figure 25: Servo Placement In Strumming Assembly

The assembly for the frets will not have this option since the and the motors for the frets all need to be on the same line, the fret assembly will hold the motors above the frets and be attached to another 3D printed part that will go on the back of the neck of the guitar. In addition to having space for the motors it will also need a way for the wires from each motor to go through it, there could be a gap between the neck of the guitar and the assembly where the wires could go to the back part of the assembly and out the back to the controller and power source.

One option for the fret assembly is to 3D print linear actuators to attach to the servos which would allow for the servos to have linear motion to press down on the strings although this is cheaper it is more complicated than using solenoids and the servos will be much slower than a solenoid would be to press down on the frets however at the frets the space between the strings is 6.35mm and the width of the servo is 12mm for this reason to have this work properly we would need to stagger the servos vertically for the fret assembly as well and some of the 3D printed stoppers on the higher part of the assembly would need to be longer than the ones at the lower part, since the SG90 servo that we are using is 32mm wide then the stopper needs to be that length plus the length of the lower stopper for it to be able to press down on the string at the same time as the lower one. The distance between the frets is 38.1mm and the distance across the servo is 32mm so this will not be a problem going down the neck only across. Shown in the figure is the general layout of the assembly looking down the neck.

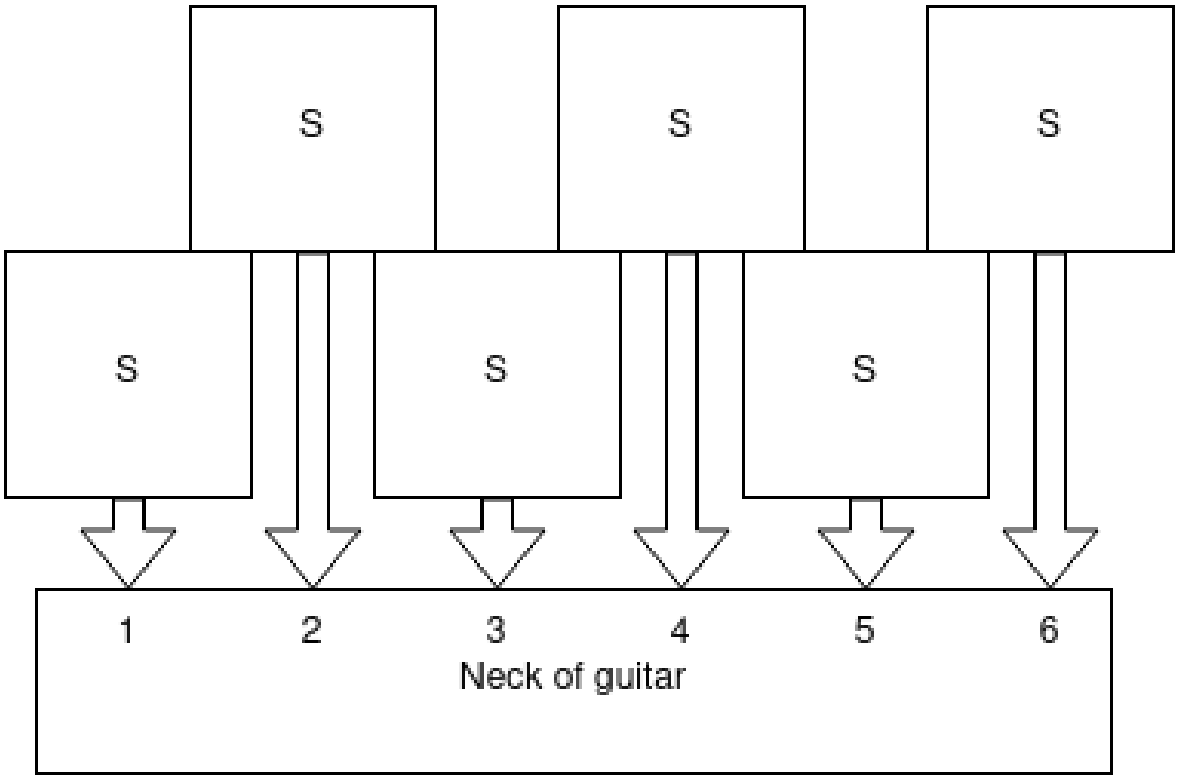


Figure 26: Servo Placement in Fretting Assembly

In the case that we do not use the linear actuator for the servos we can also set up the servos so that they rotate onto the strings on the neck this will simplify the design of the assembly as we are not mechanical engineers and the design for the linear actuator may not turn out the way we intend for it to, due to being too weak of a structure to support it or it just turns out to be too complicated to extend the length of the actuator to implement the staggered design vertically. This design can be implemented similarly to the design over the soundhole.

We are also considering a 3D printed housing for the microprocessor and PCB just for the purposes of presentation.

If issues arise with creating a 3D printed assembly there is also the possibility of making a Lego assembly as well, the assembly would need to be built to hold the servos securely in place while they strum the strings, the picks will still be 3D printed since that does not require as much design, and the assembly would need to be attached to the guitar in the same way it would be in the 3D printed design since using glue would mean that the assembly would not be able to be removed for maintenance of the strings. It would also be easier to fix minor issues with the assembly design than using the 3D printed design since the pieces can be removed and reassembled instead of needing to go back to the 3D printer and reprinting after the design is corrected, so this solution is much better from a trial and error perspective. Ultimately the flexibility of the 3D printed solution is what we want to focus on for the autonomous guitar, but if something goes wrong or for a prototype design it is worth considering the potential of the Lego assembly.

3.3.10.1. CAD Software

To model and print the 3D-printed parts for the project, we will need to use a 3D modeling software. This software should be able to export the design in STL format, which is the common format used for 3D printing.

3.3.10.1.1. SOLIDWORKS

SOLIDWORKS is the typical 3D modeling software used at UCF. It is installed on most of the computers in the engineering building, so we would have easy access to this software. It is also possible to get a student license, which would allow us to design the parts in this software from home. Additionally, since we are new to 3D modeling and 3D printing, it would be helpful to have assistance from someone who is familiar with the software. The technicians in the TI lab, where we would do the 3D printing, are familiar with the software and could give us some guidance in case we get stuck. The support structure from the university makes this option likely to be the best.

3.3.10.1.2. Fusion 360

Fusion 360 is another common CAD software for 3D modeling. Like SOLIDWORKS, it is possible to get a student license to install the software on a home computer. Additionally, CAD models for design and 3D printing are standardized, so there should not be any compatibility issues. In fact, Fusion 360 is compatible with SLDPRT and SLDASM files, so it has great interoperability with SOLIDWORKS. There are fewer people at the university that are familiar with the software, but, for example, the robotics club uses it often, so they could be of some help. There aren’t any big advantages to using this software over SOLIDWORKS, so we will probably not use this option.

3.3.10.1.3. FreeCAD

This is a free and open-source CAD software for 3D modeling. Because it is open-source, it is not as cleanly implemented and so would be slightly more difficult to learn and use. The main benefit of this software is that it is free, so if we have any issues with getting the licenses for the other software, we can just install this one. Again, it uses standardized file formats, so we could easily interop between this and one of the other professional softwares to get the best of both worlds.

3.3.10.1.4. OpenSCAD

This is a very unique CAD software. Instead of interacting with it via a mouse and tool ribbons, you write code that is executed to generate the design. This has a bit of a learning curve, but for those who are experienced with programming, it allows a level of fine control over the design that is not as easily achieved in other CAD software. This tool ended up being the main tool used for the 3D modeling portion of our project.

3.3.11. PCB Design

As our constraints outline, at least 1 PCB must be included as a part of this project. This is strictly dictated by our Senior Design guidelines - the inflexibility of this rule means that the research we'll be performing to understand how we will include PCBs into our project will be vital to the success of the project as a whole. However, even without the consideration of this requirement PCBs are themselves an incredibly powerful tool for the finalization of a circuit design which ensures its longevity and performance for years to come - One way or another, we will be using PCBs for our project. This section will go over in thorough detail the options we will face when traveling down the pipeline of taking a circuit schematic and turning it into a chip.

PCB design is largely governed by the IPC 2221 Standard. The fine details of this standard are discussed under the "PCB Standards" of our "Constraints and Standards" section, and so under this section we will forgo discussion of how the chip itself will be constructed and instead focus on our part of the production process for PCBs. we will go over every step of the process that goes into the implementation of PCBs, the types of challenges we face along the way, and the myriad of options we will ponder as we plan the finalization of our circuit design.

For our project we have some different routes we can go since we have two different areas that we need to address. For our motor assembly we need a lot more current in order to supply enough current to the different servo motors. Our other component is essentially the microcontroller which doesn't need to require nearly as much current as the servo motors need. To address this we have two options, we can design one PCB that has two switching regulators in parallel with each other since everything needs 5 volts but need different current amounts so in order to ensure that we don't burn up the microcontroller we will need to have them attached to two different regulators that work specifically for what we want them to do.

Another route we can go is we can create two different PCBs one would only handle the microcontroller and the other one would deal with supplying the adequate current to the servo motors and the other would contain a switching regulator which would be able to power the microcontroller and the shift registers. In doing this we will be able to make sure that we do not fry the microcontroller by providing it with too much current.

Pros of using the single PCB include

- One single device with all components included
- Less space we would account for on the guitar
- Cheaper to produce one PCB than two separate PCBs
- Only one PCB to troubleshoot if it isn’t working

Cons include

- More complex circuit design
- Chances of things becoming fried
- Have to order completely new one if issue with one part of the design

Pros of using two PCBs include

- We can make sure to not fry the Microcontroller
- Easier to design individual circuits
- Smaller boards will be needed
- Don’t need to change both for issues with one

Cons include

- Having to design two PCBs
- Cost would increase

3.3.11.1 Circuit Simulation Software

Our first step towards the implementation of PCBs in our project is the simulation of the circuits that they realize in order to ensure that they work as we expect them to. Note that this section will not go over the procedures we will use to test the circuits, but rather it serves as an investigation into the software that we can use to draft up schematic designs. Our main choice will be between one of a few software programs. and they will be discussed below.

3.3.11.1.1 LTSpice

LTSpice is a freeware Circuit Simulator, and is widely utilized; not only for hobbyist applications, but for educational and industrial applications as well. It is commonly touted as being very powerful, at the cost of less accessibility for newcomers. Utilization of LTSpice in the past demonstrates that it has a vastly superior circuit assembly to Multisim and that the simulator is superior in terms of calculation times - however, the interface can be described as “clunky” with parts that cannot be dragged errors that might not be faced during Multisim. LTSpice also has a vastly superior parts catalog, as it is hooked directly to Analogue’s repository of parts.

3.3.11.1.2 PSpice

PSpice is said to have even more capabilities than LTSPice - for example, Monte Carlo analysis which accounts for probability and random numbers. However, reports indicate that it is an expensive piece of software - A custom quote is required for specific pricing options. This can be circumvented by instead downloading Pspice-For-TI, which is a form of Pspice with exclusive integration with TI's part library - In exchange for this inflexibility, the software is provided for free. The downsides to this option are negligible, as all of the options for our linear and switching regulator options are from TI. Thus, PSpice for TI proves itself to be a preferable option for designing these two PCBs.

3.3.11.1.2 Multisim

Multisim is the circuit simulation software that we are most familiar with, due to its recommendation for most of our lab simulations throughout our undergraduate program. Multisim's strongest selling point, beyond the fact that it is completely free to download and utilize, is its GUI. It utilizes the same SPICE algorithms as LTSpice and PSpice, but has the added advantage of a much more accessible interface. Options such as being able to drag parts directly and attach probes as objects prove to be considerably useful, especially when having to go without by transitioning to another circuit simulator. The downside to this option is that it has a more restricted parts library than LTSpice, and less capabilities for circuit analysis than the other two options explored under this section. However, for the purposes of our assignment, Multisim would likely be extremely useful as the components that Multisim includes tend to be good enough for hobbyist applications. It also has the added benefit of being web-based, which means that files can be saved on the web server and accessed elsewhere as opposed to having to be downloaded onto a storage device and transported manually. Ultimately, it is likely that we will be utilizing Multisim for our schematic design.

3.3.11.2 PCB CAD Software

Before the PCB (or PCBs) can even be included in our design, we must first design how the circuit that they model will be implemented on the board. The actual construction of the circuits is a straightforward implementation of principles learned in Electrical Engineering, and has no place in this section. Rather, we have to consider what software we will use to generate the files needed for the PCB. Key considerations include the versatility of the software as well as how mainstream the software is - It is sensible to assume that the file types supported by well known CAD software will be more widely supported by companies specializing in PCB construction.

3.3.11.2.1 Autodesk EAGLE

Autodesk EAGLE is the most obvious choice for us just as a starting point, as it is included in every ECE senior design student's curriculum as it is included as a part of Junior Design. The most evident advantage to choosing Autodesk EAGLE is that we are guaranteed to have at least a baseline understanding of how to use EAGLE. This will provide a considerable advantage for our implementation stage as we will spend less time learning the software, thus leaving more wiggle room for designing the PCB, having it constructed and everything else that will go into the final implementation. Another upside is that the software is completely free to download and utilize, as most ECE students have already done. Further research indicates an above-average online support as well as a library for parts that proves to be rather extensive. Going forward, we will treat this option as the baseline and, if no better options are found, we will settle on Autodesk EAGLE.

3.3.11.2.2 Fusion 360

Fusion360 is another PCB CAD software that we have encountered, albeit more tangentially, in Junior Design. It was discussed more as an alternative to Autodesk for those individuals more familiar. However, cursory research into the program & comparisons to EAGLE seem to indicate that it has a number of issues and is generally considered inferior to EAGLE. Its advantage lies in its integration with the other mechanical engineering tools that are available on Fusion360, as the software was designed more with mechanical CAD in mind. However, for the purposes of this project, it would be best to look at other software solutions.

3.3.11.2.3 KiCad EDA

Research into KiCad indicates that it is similar to Autodesk Eagle. Like Eagle, KiCad is completely free and seems to be geared towards beginner-to-intermediate applications. The difference is that, while Autodesk Eagle is maintained by Autodesk, KiCad EDA is a crowd-funded open-source program maintained by volunteers and paid contributors. The general consensus seems to be that KiCad tends to be slightly less accessible than Autodesk EAGLE, but provides many features that prove themselves to be superior. KiCad is similar to Autodesk EAGLE in a number of ways - KiCad maintains an online network of parts, similar to EAGLE, as well as an Auto-route functionality that can significantly cut down on development time. Additionally, while both KiCad and Eagle export PCB designs as .brd files and any company that takes EAGLE files will also take files designed by KiCad, KiCad has the additional built-in functionality of exporting PCB files as Gerber Files. This allows extra flexibility, as some PCB manufacturers - especially hobbyist-centric ones - take Gerber Files more readily than .pcb files. Because of these similarities, we could leverage the experience we have with Eagle to learn KiCad in a much shorter time than most other PCB CAD programs to yield superior PCB designs and overall less headache with the PCB design process.

3.3.11.3. Part Production

Once we have the .brd file that realizes the various circuits we will need for our project, we will have to use those files to create the PCB. For hobbyist applications, this is typically accomplished by outsourcing the file to a manufacturing company. The company will take the design alongside a quote and manufacture the PCB before returning it to us. This is generally seen as the industry practice, and so there is no question that this is the avenue we will pursue. The challenge we are faced with then is which company to use for our PCB design. In this section, we will go over the find details behind individual companies and weigh them against each other to decide ultimately which company we will utilize.

3.3.11.3.1. Digikey

The first company we will be looking at is Digikey. The biggest upside to choosing Digikey is our familiarity with the company; we have obtained parts from them in the past during Junior Design, so we can trust that their service is adequate. A cursory look shows that Digikey accepts PCB files in the form of Gerber files. Gerber files are notable for our purposes in that they are able to be exported by most PCB CAD programs - this gives us a great deal of wiggle room in picking CAD software. Reports seem to indicate that Digikey is also cheaper than most other competitors for utilizing their PCB building software. This is a considerable boon for us, as our budget is limited. As this is the option that is most familiar to us, we will use this company as the baseline in our selection.

3.3.11.3.2. PCBWay

PCBWay is a popular PCB manufacturer that is based out of China. An upside to utilizing this company is that they have an instant quote generator which will save us the headache of requesting a quote and waiting for the price to be returned. A downside to this company is that, because they are located halfway around the world, shipping time will be a considerable burden on our schedule. On top of this, we have to worry about additional shipping costs which could further cut into our budget. Ultimately, compared to Digikey, this option seems to offer us more convenience at the cost of time and money - We will keep this option in consideration as we continue our search.

3.3.11.3.3. OSH Park

OSH Park is a company that markets itself as catering specifically to hobbyists. It does this with lower than average prices for smaller boards, and all accounts indicate that the boards are high quality. OSH Park also includes a partner company named OSH Stencils; They offer solder stencils for those same boards, which will cut down vastly on our assembly time by allowing us to utilize the reflow oven present in the senior design lab. In this way, our smaller parts can be included in our design without the time spent on a difficult soldering technique as well as cutting down on the possibility of injury. The downside to this option is that reports on delivery time seem to be inconsistent, which might put a hamper on our schedule. Overall, this service offers superior cost and quality at the cost of less consistency in part arrival; we will keep this option in mind as we go forward.

3.3.11.4 Assembly & Final Implementation

The last, and most hands-on component of the whole PCB implementation process, will be assembling the parts onto the PCB. This step is included with the general assumption that we will be soldering the parts ourselves, as opposed to having the parts included automatically on the board by the manufacturer - while enticing, this option severely limits our part selection as we would only be allowed to pick from the parts offered by the manufacturer in question. An alternative to this would be to send the parts to the manufacturer alongside the PCB design, but that would effectively double the time it'd take to get our PCB on our project. Thus, we will do the part assembly ourselves. Below outlines the concerns and considerations that we will have to keep in mind when it comes time to put our PCBs on the project. Note that this section does not cover our prototyping stage, but a consideration of parts that we will implement for our final project design.

3.3.11.4.1 - Wiring to/from/between PCBs

Generally speaking, our selection for wires would be negligible within the grand scheme of things. However, due to the high currents that are present in the project by way of the servo assembly, we must consider how our power will be transferred and dissipated as it is transported from the power supply and regulators to our components. Wires, from the viewpoint of current tolerance, are categorized by gauge (American Wiring Gauge, or AWG). A copper wire at an AWG gauge of 12 is rated for around 10A according to some sources, but the consensus seems to be split when looking across multiple websites. which should be sufficient assuming a worst-case scenario. This brand of wire is available from a variety of locations, and so it can be safely assumed that we will be able to procure it.

The solder used for attaching the wiring to PCBs is seen as negligible when viewed from the lens of current management, and thus we will not have to place a great deal of emphasis upon the solder selection for our higher current components.

3.3.12. LCD

One of The things that we want for the Autonomous Guitar is to have a LCD display on the PCB that will tell the user first what the Bluetooth connection status is after a song is uploaded to the MCU the LCD will display the name of the song. This can be done by sending the name of the file as well to the microcontroller along with the MIDI file itself. The LCD display will also show the amount of time has currently elapsed in the song. When the user chooses a different song from the songs uploaded on the autonomous guitar the LCD will display the song that the guitar will play.

The LCD screen needs multiple pins to control it and it will also require a potentiometer to control the voltage supplied to the screen to prevent noise from interfering with the display. However testing with multiple resistors between 1 and 10 K ohms will also work.

The LCD screen will serve no technical function and could be a potential limiting factor in the power consumption of the system but it is something that will be easy to add and can be very helpful for usability and testing to make sure everything is working properly. If we implement this with SPI with the motor controllers it is as simple as adding another shift register and adding it to the daisy chain with the other shift registers for the motor control. The other option is to implement it with I2C which will consume less power but then we will have to use an extra module to implement that feature.

3.3.12.1 I2C LCD 16x2 Adapter

In the case that we cannot find a 16x2 LCD with I2C capabilities we can use a I2C LCD adapter for connecting the LCD screen to the microcontroller. The adapter needs to connect all the LCD pins to the corresponding pins on the adapter then have the adapter connected to the data and clock pins on the microcontroller to get this to work. This module will also need to be connected to the power line to work. It is also possible that we could use a shift register instead of an I2C module and that would let us use the LCD screen in SPI and connect it with the already existing SPI connection with the Servo Control.

3.4. Possible Electronics Architectures

There are many ways we could structure this project. Each structure entails a different block diagram, and has far-reaching effects on which components we will use and how the end-user will interact with the device.

To get the output of the Algorithm to the controls for the motors there is the option to either transmit the data from an external device through methods, such as UART in the example given below, or we can run the Algorithm directly on the microprocessor. If we transmit the data from an external device, it can transmit the output of the algorithm which could be represented as a string to the microprocessor which then plays the note and repeats this until the song ends or the user stops the song. This method would also require a front end for the user to input their midi file and start and stop the song. There would need to be some method to wirelessly transmit the UART from the microprocessor to the computer. It could be transmitted over wifi or over bluetooth in this method. Most of our options for microprocessors have this functionality or are capable of having this functionality with an external module to provide wifi or bluetooth.

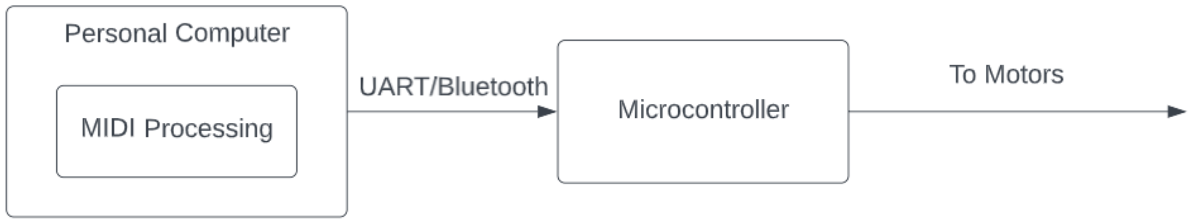


Figure 27: Personal Computer Architecture

The other option is to run the algorithm directly on the microprocessor, this would allow for the project to be more self-contained. Figure 28 shows how unlike in the previous architecture there are no PC or external devices involved, microcontrollers already take MIDI notes as individual pieces of data, our algorithm would just need to translate that into notes that would be played on a guitar within the range of notes that our device is accounting for. Using this method would require some way for the user to interface with the guitar directly to control starting, stopping and selecting a song to play, which could be done with a button to start a song, a button to stop a song, and an LCD display to display the name and time left of a song. it would limit our options for potential boards since some microcontrollers would not be able to run the algorithm or lack USB ports which would be necessary to store the MIDI file in cases where the microprocessor does not have onboard storage.

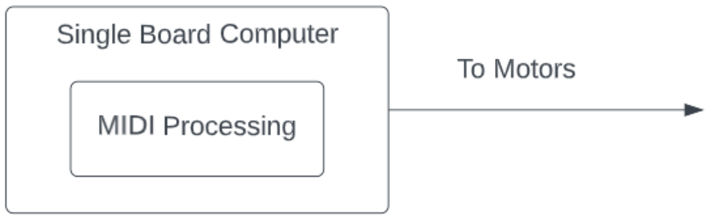


Figure 28: Single Board Computer Architecture

On the motor side of things, we need motor drivers for most motor types, which will take the 30 control signals from the shift registers and use them to switch the power for the motors as shown below.,

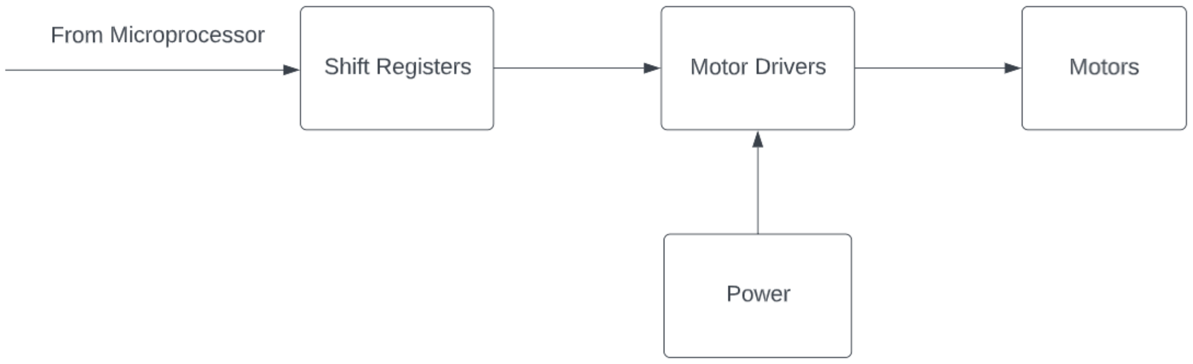


Figure 29: Motor Driver Architecture

Alternatively, servos have drivers built-in, eliminating the need for separate driver components.

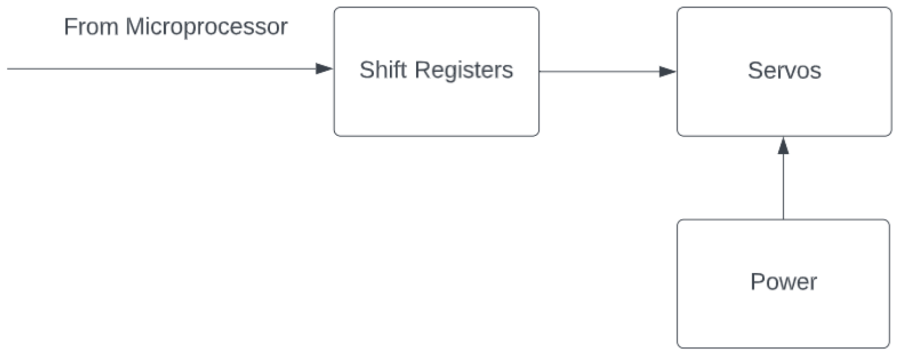


Figure 30: Servo Architecture

The LCD display will also need to be included in the architecture, this can be connected either by SPI with a shift register or by I2C with a I2C converter module, The benefit of using the shift register is that it has lower power consumption and it will have a faster refresh rate, but given that the servos are all going to be connected to a shift register already this could make add some unneeded complexity to the design since we will be using daisy chain configuration for the wires. Alternatively we can use I2C, this will be slower and more power consuming than using the shift register and it will require buying a module that will make the LCD screen compatible with the I2C protocol or an LCD screen that already has I2C protocol built into it.

3.5. Possible Hardware Architectures

3.5.1 Hardware Option 1

One thing that must be considered is the amount of motors that we plan on using. When using as many motors as we have initially thought of it will likely cause some careful planning to be done. To ensure that our motors are functional and operate independently we will likely need a microprocessor with enough pins to program them individually. This is something that must be looked at when selecting the correct microprocessor. In this scenario we would have the microprocessor program each motor individually. This solution has both pros and cons. Some pros include the repeatability of the code ideally, as if we can get the code working for one motor it is fairly similar between each motor with ideally some small tweaks. Some issues that arise from this solution is the pin outputs each microprocessor will need to have 29 different output pins or we may have to get multiple microprocessors that would increase the cost of our overall project. Another issue that will arise is space on the guitar for the individual motors as it may get potentially crowded along the neck of the guitar

3.5.2 Hardware Option 2

Another solution to our strumming problem is to create an xy plane that uses motors to control the position. In this scenario it would require us to have less motors in order to strum however these solutions bring up more problems. One major issue is it would only be able to only play one fret at a time which would severely limit the note range we would be able to play. It would require a lot more mechanical design in order to ensure that the xy system is calibrated correctly and some mechanical design we just don't have enough experience to create reliably. Also this solution will be much larger than we are aiming to achieve, as it would require a sort of mounting system in order to make sure it does not interfere with the frets. This idea is not ideal for us due to the amount of mechanical and structural design this option would require.

3.5.3 Hardware Option 3

Potentially an option would be to have different linear tracks which would run long ways along the neck of the guitar. This combines a little bit of both options above. This would be able to play more of a note range than option 2 but if we would try to match the note range in option 1 we would essentially have to put enough motors that would make this option useless. This would also cause the same mechanical issues as option 2 but possibly more difficult as it requires more motors and more pins which would lead to potentially more microcontrollers being involved in order to compensate for all the pins it would need to have. This may be our worst option as it could cause more headaches to pursue then worth.

3.5.4 Hardware Path Moving Forward

Through group discussion and feasibility as of now the option to be selected will be option 1 even though it could cause some increases in cost our group has decided to accept the risk and continue researching how to bring option 1 to life. This will include looking into the best programming language for the microcontroller as well as the algorithm to use midi files. We will look into which is the best motor type to use in order to press the frets down and strum accurately and quickly enough. With all these ideas in mind we believe that this will be our best option to achieve our goal and ambitions for this project. Below is a table which breaks down the thoughts that went into the decision of going with option A. Although the cost values may vary depending on availability, to our knowledge this is the best breakdown to suggest option A is our best option.

Option	# of motors needed	# of MCU	complexity	Note range	Mechanical requirements	Cost
1	29	1-2	High	High	Low	Med
2	1-7	1	Low	Very Low	Very High	Med
3	~12	1-2	Medium	Medium	High	High

Table 7: Comparison Between Hardware Design Options

3.6. Plug in Power

A stretch goal of ours can be to plug in the guitar to a wall outlet and run on corded power. Doing this we can ideally toggle between the Battery operated DC power and the wall outlet which runs on AC power. This will be considered a stretch goal since it will cause significant complication to our current design circuits.

The easiest way to implement this is to have a separate PCB and that will allow us to have a totally separate circuit, this is something that will lead to a pretty big increase in cost as well as time and resources. This would require us to essentially double our electronic devices. This would include having a second microcontroller since we have so many outputs we can not keep them on just one microcontroller unless we upgraded to a bigger controller with more output pins. The biggest issue with this design option that will make it difficult is we have two sets of wires which need to control one set of servo motors. In this scenario we would need to figure out how to either splice the wires or figure out some other type of to combine the control signal wires.

Another way we could go about and process this is to make one big PCB which will have both the DC and AC power circuits. This will make the circuit quite large and also very complicated. We will need to be careful of having voltage lines running next to each other and would have to deal with the AC power interfering with existing traces and components. This is why this is such a stretch goal since it would require a lot of extra time and money all though it is feasible it is extremely difficult.

Using AC power would give us additional standards that we do not need to currently worry about since we are not using AC power but using plugs it involves a lot more that is standard practice since every outlet is regulated causing it to be uniform.

3.7. Guitar Mount Building Material

The following section details various methods we could use to approach the challenge of prototyping our design. Specifically, we will be discussing strategies we could use to mount the various components to our guitar, such as our motor assembly.

3.7.1. Lego Brick Mount

We have considered the option of using LEGO bricks such as those in figure 31 to fully build out our guitar mount. Going this route / taking this path offers us the following advantages:

- We can easily customize, change, and adapt our mount to different needs as we see fit when actually setting up and prototyping the project, allowing us to account for unexpected variables

- It allows us to be very flexible with the shape of the actual mount with little effort and not much extra cost to us for building

While some disadvantages of using LEGO bricks would be

- Mount not being very sturdy

- Mount potentially breaking apart easily especially when being transported

Though, we also discussed using glue to ameliorate those disadvantages.

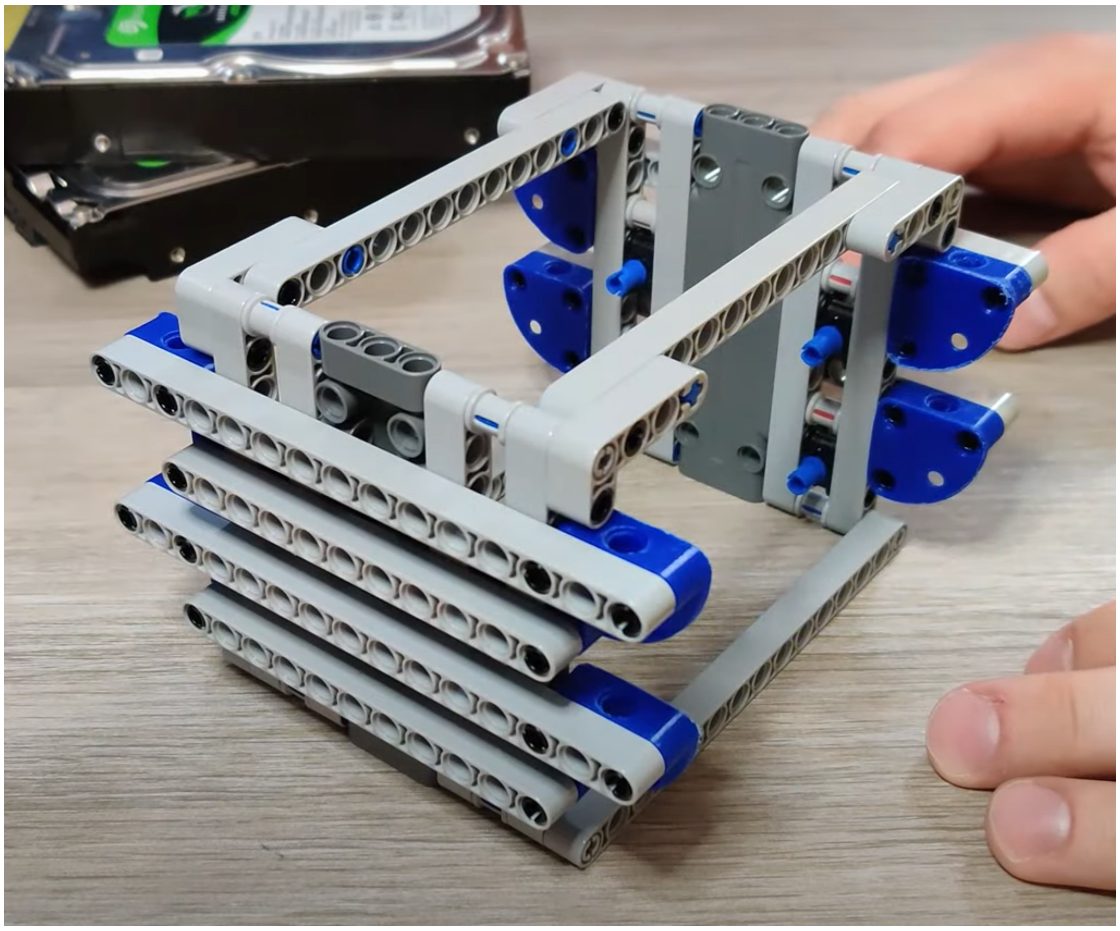


Figure 31: Example Lego Mount Construction

3.7.2. K'Nex Mount

Another option in consideration is using K'Nex pieces such as those in the figure below to build out the servo motor mount for the guitar. Going this route / taking this path offers us the following advantages:

- We can easily customize, change, and adapt our mount to different needs as we see fit when actually setting up and prototyping the project, allowing us to account for unexpected variables

- It allows us to be very flexible with the shape of the actual mount with little effort and not much extra cost to us for building

While some disadvantages of using K'Nex bricks would be

- No easy way of holding Servo Motors

- Mount potentially breaking apart easily especially when being transported

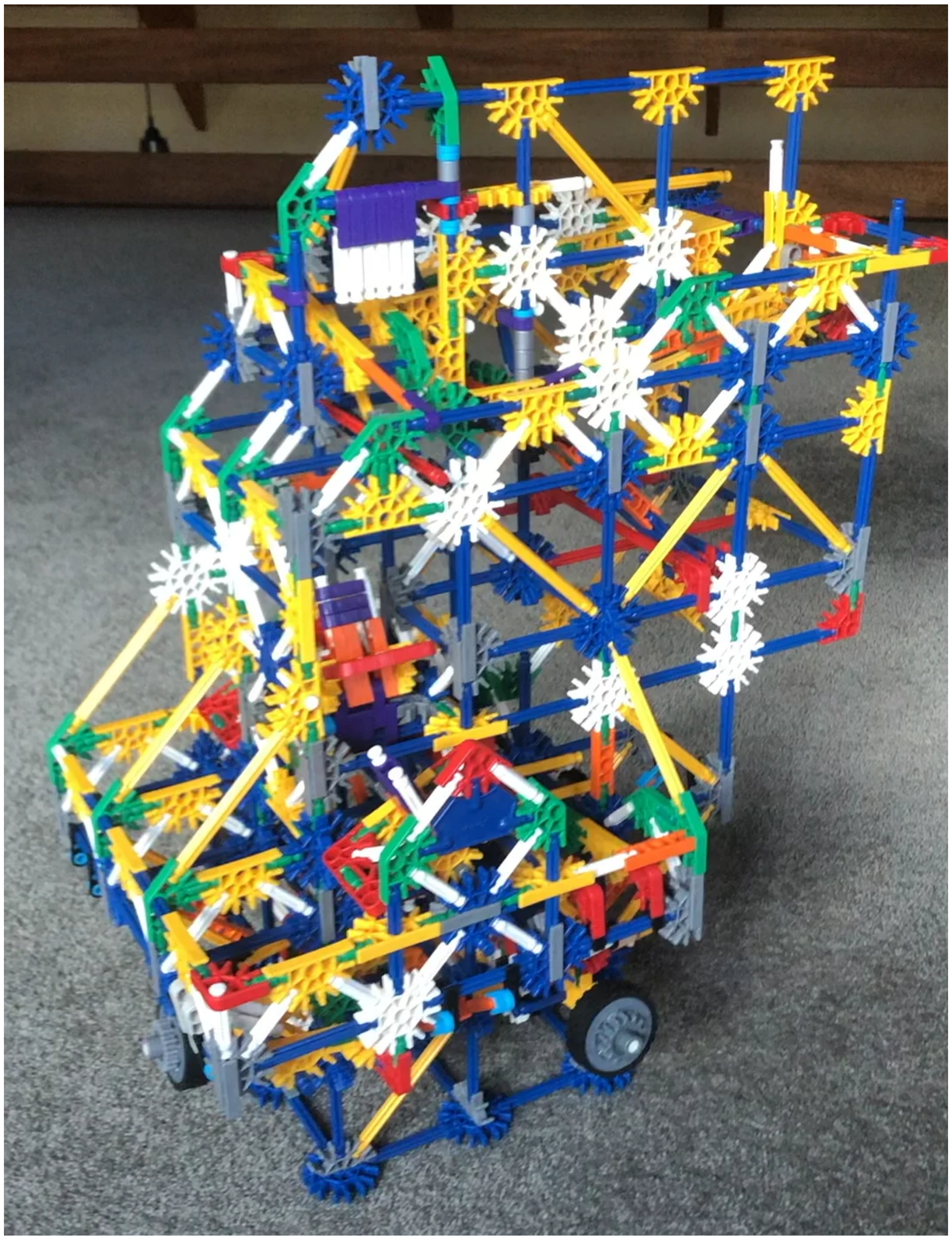


Figure 32: Example K'Nex Mount Construction
https://www.reddit.com/r/KNEX/

3.7.3. Wooden Mount

Wood is a very strong material and can be easily cut and shaped using a laser cutter. The TI lab at the university has one, along with technicians that are very experienced with using it and making files for it. With their help, we would likely be able to design a decent mount for the motors using this method. However, we would be limited by the flat nature of the wood, making our final mount very boxy. Below is an example of using wood to provide a mount for a servo.

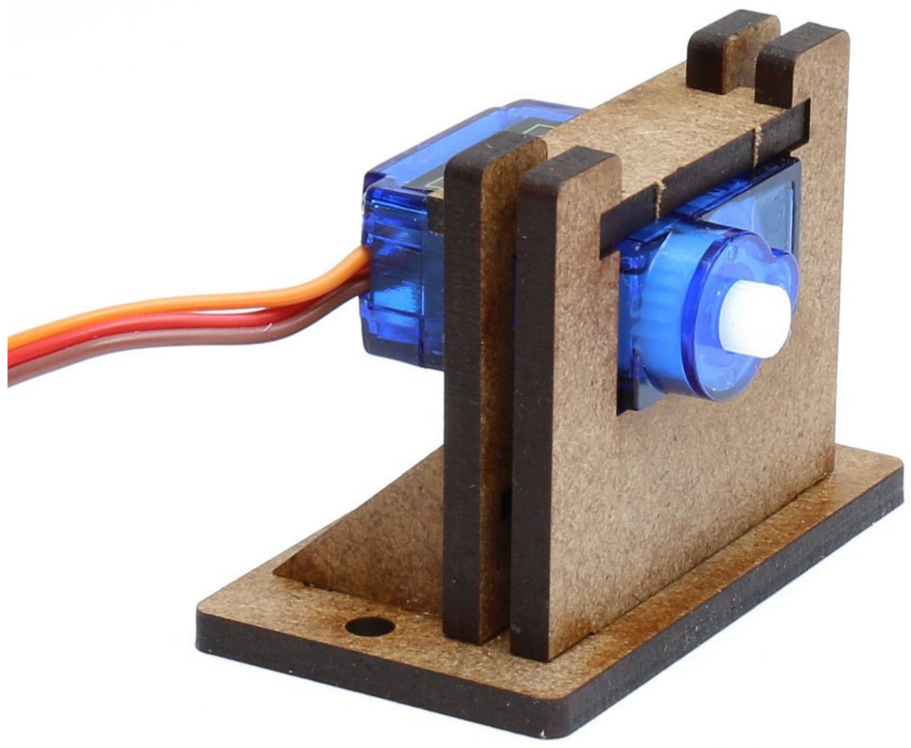


Figure 33: Example Laser-Cut Wooden Servo Motor Mount Construction
https://www.dm-toys.de/en/product-details/MU_N-A00108.html

Some advantages of using wood are:

- Very sturdy and robust material
- Supported by TI lab
- Cheap

Disadvantages include:

- Difficult to prototype, change, and adapt
- None of us have much woodworking experience or the necessary tools for creating intricate or complex designs using it

We have discussed the possibility of prototyping using one of the above materials and possibly making a stronger sturdier wooden frame to hold the larger parts in place. We can also manufacture some of the simple parts out of wood, such as the actuation arms, and then leave the more complicated parts to another method. We ended up using a laser-cut wooden mount for the first prototype of the fretting mount assembly, although for the final project we switched to laser-cut acrylic sheets.

3.7.4. 3D Printed Mounts

The below figure is a file for mounting a servo provided for public use on the Grabcad stl file repository.

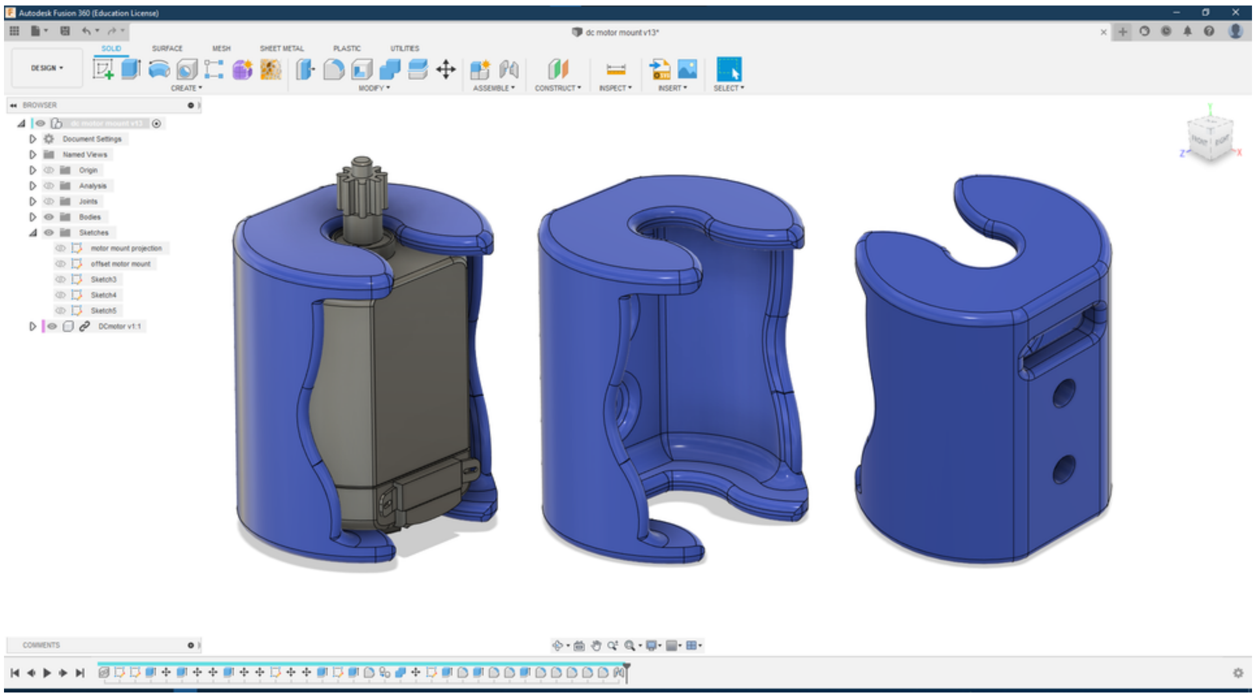


Figure 34: 3D Printed Mount Construction
https://grabcad.com/library/r130-3-6v-dc-hobby-motor-holder-mount-1

Some advantages of using 3D printed parts are:

- Extremely customizable to exact dimensions

Disadvantages include:

- Time-consuming to prototype, change, and adapt
- None of us have a 3D printer or much AutoCAD experience so we'd be very reliant on the Maker lab in the Engineering building

We have discussed the possibility of prototyping using one of the more easily adaptable materials and possibly using 3D printed plastic for the smaller custom parts needed such as the linear actuators for fretting. We ended up using 3D-printed plastic for the final project strumming mount assembly.

3.7.5. VEX Robotics Mounts

In this section we will discuss VEX Robotics Mounts. VEX Robotics gives us a good tradeoff between easy prototyping adaptability and customizability, while maintaining structural integrity and sturdiness of material since it mostly involves metal parts that can actually be bolted and screwed into each other. While some disadvantages would be that determining the necessary parts we'd need beforehand could be costly and time-consuming as well as take a while to ship. The figure below is an example of one configuration for VEX robotics mechanical assembly.

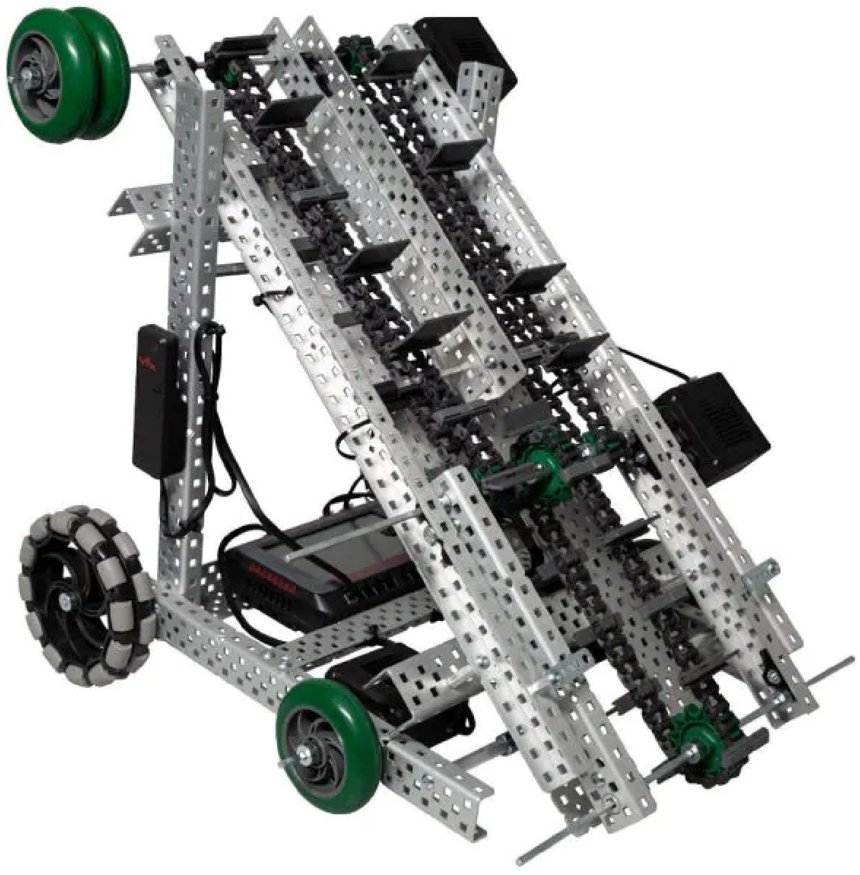


Figure 35: VEX Robotics Example Construction
https://www.vexrobotics.com/v5-competition-starter-kit.html

3.7.6. Final Material Choice

We have ultimately decided to go with laser cut wood for first prototype of the fretting mount and use acrylic for the final project. We decided to use 3D printed plastic for the strumming mount. The fretting mount is quite large and impractical to be 3D printed, and it needs to be pretty sturdy to handle the forces of the 24 motors and 6 strings that will be pushing back on them. The strumming mount, however, is small enough to be 3D printed, making it easier to prototype.

4. Constraints & Standards

The section below outlines the Constraints and Standards that we will have to work around. We need to consider constraints related to economic, environmental, social, political, ethical, health and safety, manufacturability and sustainability. It is important to consider these constraints because it will affect our design decisions.

4.1. Constraints

For our constraints as seen in table 5, per course requirement for senior design we need at least one PCB to turn in for the course. This will not be a problem because we will need power and shift registers to make our project work.

4.1.1 Economic/Time Constraints

We have a budget of around $500. We are paying for this out of pocket since we do not have a sponsor and we would prefer to keep the price to a minimum. This is flexible but $600 is the maximum, this will factor into the design and is the reason that we have decided to use servos on the fretting assembly instead of solenoids and using an acoustic guitar instead of an electric one which would put us at a very high price point. The most important standard is that the project must be operational by the end of april 2023. This also factors into our design and what parts we purchase, we can't purchase items on back order since that may take too long to arrive and we need to decide on parts quickly to make sure that the parts work, because if we buy parts that do not work or are not what we need in January and it will be 2 weeks before the new parts arrive, we will lose a significant amount of time to build the project. For this reason we will need to decide on the parts and purchase them by the end of our first semester of senior design.

4.1.2 Environmental Constraints

The Autonomous Guitar will not need any special considerations for the environment, it has no interaction with any environmental concerns except things like battery disposal. The project team will ensure that any batteries disposed of will be disposed of properly. Since we will not be using a battery that is larger than 9 volts we can just recycle them. We have not looked into any energy source such as solar power for our project since that is outside the scope of what we want to accomplish given the time that we have.

4.1.3 Social Constraints

The Autonomous Guitar has no actual use in the real world, if someone wanted to play a song but did not know how to play an instrument they could just use speakers. The Autonomous Guitar only exists as a fun project and most examples of self-playing guitars also seem to be hobbyist projects and not professional products. For these reasons there are not any social constraints for the Autonomous Guitar.

4.1.4 Political Constraints

The Autonomous Guitar does not have any constraints related to politics. The only potential problem is with DMCA during the final demonstration but this is not a design constraint and only royalty free music will be used during the demonstration.

4.1.5 Ethical Constraints

The Autonomous Guitar will not use any copyrighted software and any open source software used with this project or any figures used in this paper will be cited in the appendix. No designs for the guitar will be taken directly from other self-playing guitar projects, and care will be given to ensure that no patents to another product are infringed upon.

The Autonomous Guitar will not contain any substance or material that could be harmful to a human and any waste such as batteries or broken 3D printed parts will be recycled properly.

4.1.6 Health and Safety Constraints

The Autonomous Guitar will need to be designed with safety of the user in mind, the user must be able to use all the guitar's features without any risk to their safety. The assembly should be designed so that the user is not harmed if the strings break from the servos. There is also risk of electrocution to the user if the power supply is set up incorrectly in a way where the user can touch the electrical components. For this reason the PCB and the rest of the electrical components should be covered. Safety precautions also need to be taken for the design team when constructing the Autonomous Guitar, it is possible that a faulty battery may explode or someone may be shocked while debugging the system. To ensure that this does not happen, the team members should always turn off power to the device and disconnect it from the testing environment such as the computer. The design team should also make sure not to touch the guitar while it is playing, the person may be injured by placing their fingers between the strings while the guitar is playing and be injured by the guitar or the strings could snap while being played leading to injury and the best way to prevent this is for the team members to not touch the guitar during testing or demonstration.

4.1.7 Manufacturing Constraints

Due to supply issues affecting many electronics industries there are some things that need to be taken into account for the choices of electronics for the Autonomous Guitar. We needed to decide between using a single onboard computer and a microcontroller, due to the single onboard computer being way more powerful than the system would require and the price and availability of boards such as the Raspberry PI where we were unable to find one for under $150 even though the list price according to their website is $35 and all of the recommended retailers carrying the Raspberry PI were out of stock. For these reasons a microcontroller was chosen instead which we were able to find for $15.

Other parts required by the Autonomous Guitar do not seem to currently have any supply issues. As long as we know what we need before we begin running low on time we will be fine on acquiring the parts that we need.

Another manufacturing consideration to make is that the Autonomous guitar is going to have several 3D printed parts, and no members of the project team personally own a 3D printer, for this reason most 3D printing will need to be done in the TI innovation lab at University of Central Florida on the main campus.

4.1.8 Sustainability Constraints

The Autonomous Guitar is expected to be able to play a song on the guitar that is 3 minutes long, and it is expected to be able to do this reliably without breaking strings or wearing out it’’s guitar picks, it is unreasonable to expect the guitar picks to last forever especially 3D printed ones though so the guitar picks should last at least 1000 minutes of active use we can implement this by designing thicker guitar picks that can still gently pluck the string without breaking it if this is necessary, and there must be a way for the user to replace these guitar picks when they are worn down or break. Due to the fact that these guitar picks need to be 3D printed the user should have a 3D printable file available to them so that a new one can be printed. The file will be downloadable from the project's github repository.

The strings on guitars also wear down over time and the assembly will need to be removed so that the strings on the guitar can be replaced and then put back on the guitar. This process of removing the assembly and putting it back in place should be designed to be doable with just a screwdriver and should be documented for the users convenience further along in this paper.

Guitars also can get out of tune when not being played. Since the strings will not be accessible to the user the Autonomous Guitar will need to have a “Tuning Mode” of use where it will continuously pluck the string while the user turns the tuning dial until the user is satisfied with the sound. This action will need to be started from the PC connected to the Autonomous Guitar or it can be initiated from a button on the PCB, since the Autonomous Guitar will already have functionality on the PC end it is better to just include it there where the user can press a button to activate the tuning mode and then press another button to tune the next string in line until all the strings are sufficiently in tune.

The autonomous guitar is not meant for outdoor use, using it outside will be fine but the autonomous guitar is not waterproof in the case of rain and acoustic guitars in general are not waterproof since the wood can be warped and the strings can rust. The recommended operational area from the design team is indoors in a dry area.

Constraints
At Least 1 PCB required
Under $600
Algorithm written in python
Microprocessor code written in python
Only use guitar surface area
At least 1 Regulator
Only use DC power
Use Eagle for PCB design
Each Motor Individually wired
Must be completed by the end of April 2023

Table 8: Constraints

4.2. Standards

When designing the Autonomous Guitar we need to keep the design within certain standards to ensure that the device is properly designed by the most up to date standards on the technologies, items and methods being used so that it adheres with what will be expected with a finished product.

4.2.1. Design Standards

Standards	Official Name
Audio Storage Standard	Musical Instrument Digital Interface (MIDI)
UART Communication protocol	rs-232
Bluetooth Standards	Bluetooth SIG

WiFi Standards	IEEE 802.11
C standards	ISO9989
Python Standards	PEP 8
Standard Guitar Tuning	E A D G B E
Western Scale Tuning	A4 = 440 Hz
PCB Standards	IPC-2220
Soldering standards	IPC J-STD-001

Table 9: Standards

The table above shows the standards that we will be using as well as the official name of each standard.

4.2.1.1. MIDI Audio Storage Standard

MIDI does not have a standard defined by ISO or IEEE but it does have specifications for the MIDI that govern how it works that are outlined by the MIDI Manufacturers Association. For the purposes of the autonomous guitar MIDI 2.0 will be used which is an extension to MIDI 1.0 as most of MIDI 1.0 still applies. The MIDI file is a binary file with each chunk being 32 bits in length. There are two types of these chunks as they can either be a header chunk or a track chunk, a track chunk can use up to 16 MIDI channels.

Given that MIDI does have a way to separate channels it can be possible for the autonomous guitar to be able to separate the guitar track from the rest of the file for it to play but there is no standard piece of data that is specifically for a guitar note rather than any other instrument other than the byte “04” indicating that the channel is of an instrument.

MIDI files do not hold data for the actual sounds themselves inside the file. The file needs to go through a MIDI synthesizer whether that be implemented by hardware or software, without a synthesizer it is just a collection of bits. MIDI has a type of control change method called attack, decay and release time, these all of a default value of 64 according to the MIDI documentation, what these due is it determines the amount of time it takes for the note to reach its maximum volume then it will wait for the volume to lower again then release the note. Since the autonomous guitar does not use a synthesizer this will not be entirely necessary to keep in mind but it will let the guitar know how long to hold a note for.

4.2.1.2. UART Communication Protocol Standard

UART uses the rs-232 standard which applies to serial communication methods, This means will transmit a starting bit followed by 8-bits and then a parity bit before sending the next packet since this standard has been around for a very long time any time you code any UART functionality this is done for you in the library that is being used for UART transmission so this is not something that we need to be particularly mindful of since the libraries for the microcontroller will already have this done in the code library.. We will also be using standard baud rates which is 9600 baud, we will also be using UART in full duplex mode so that the computer can transmit and receive at the same time and the MCU can as well. We are doing this even though most functionality just involves transmitting to the guitar because it can help us debug the algorithm, so we can see if the MIDI data is being converted properly.

4.2.1.3. Bluetooth Standard

Bluetooth is Standardized by the Institute of Electrical and Electronics Engineers (IEEE) under the standard IEEE 802.15 Low Rate Wireless Networks. This standard covers both Bluetooth and Bluetooth Low Energy (BLE). Since bluetooth works on a PAN and PANs are affected by this standard the Autonomous Guitar will use it. Transmission of the MIDI file will not need a lot of power nor will it need to be fast since it will start playing the song only after the file is completely on board the MCU, after that all the Bluetooth is needed for is to tell the microcontroller to start playing the song or stop the song.

IEEE 802.15 covers all low rate wireless networks, of which Bluetooth is included and PANs are included as well. Bluetooth is also standardized by the Bluetooth Specification V4.2 by Bluetooth SIG, which states that Bluetooth and BLE operate at 2.4 GHz and uses Frequency-hopping spread spectrum (FHSS) to prevent interference in the connection with the device. Bluetooth low energy uses Frequency division multiple access and time division multiple access (TDMA). FDMA is used to separate the 40 physical access channels that BLE has and it uses a 2 MHz buffer to do this, 3 of these channels are used for advertising and 37 of them are data channels. TDMA allows one device to transmit to the other at a time and uses predetermined intervals to do this. Having advertising channels is important for the Autonomous Guitar since the PC or the phone will need to be able to find the microcontroller when trying to pair to the Autonomous Guitar.

4.2.1.4. WIFI Standard

WIFI is standardized by IEEE under the standard 802.11 Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. This technology can still fulfill our needs that the standard 802.15 can fulfill but it may be more than we need to make the Autonomous guitar work if during testing the bluetooth is too slow we may have to switch over to WIFI and the 802.11 standard. Since we did switch over to WiFi we found that our chosen microcontroller did not support WiFi 6 or 802.11ax but does support 802.11b/g/n which is what is used for 2.4 GHz WiFi 5GHz WiFi is not supported.

4.2.1.5. C Standards

If C is necessary for use in the autonomous guitar project we will need to follow some standard, C does not have any official ISO or IEC standards that are free however the 2017 draft of ISO9899 also known as C17 will be used since it is available for free.

Stated in the ISO9899 Standard is that preprocessing directives such as lines of code starting with #include should be at the top of the document before any functions are placed. When the program on the microcontroller begins executing it will begin with a function that is called main that is of type int, according to the ISO9899 standard the program can take arguments but for the purpose of the autonomous guitar this will have an argument of void. The main function should have a return value since it is of type int of 1 but since this is meant to loop indefinitely while the device is turned on there will be no return statement and there will be an infinite for loop.

As a general rule for writing in C for the Autonomous Guitar conventions that will be used are that variable names will be written in camel case and functions, loops, and if statements will have the curly brace be placed under the function call or statement instead of next to the function call or statement. Break commands will not be used outside of loops and switch statements in accordance with the ISO9899 standard. Statements such as GOTO will not be used in any C code in the project as it will not be necessary and adds unnecessary complexity to reading the C code for debugging and maintaining the code.

The general format of the source file will be at the top of the file will be calls to the header files that contain the libraries that will be used in the project starting with C standard libraries along with comments beside them which detail what functions from those libraries are needed. After the standard libraries the libraries that are for the chosen microcontroller are next not that much explanation is needed for this one since the microcontroller will not work without it. The next thing will be any structs that need to be created for user created data types if necessary. The main function will be after this, inside the main function the integer declarations will be made first then any initializations will come after that such as starting the watchdog timer. The function calls and the loop to wait for interrupts will come after that, however since we are writing code for microcontrollers we can initiate a low power mode instead of the loop.

Once the main function is written the other functions will come after that a brief commented out explanation will need to be above each function describing what it is for and the final thing that should be in the source file is interrupt vectors which above that should be a commented out brief explanation and what triggers the interrupt.

Another rule that can be considered is what to do with white space, in between types of headers will be a single white space, and in between functions will be a white space and one white space between the beginning of the function and the first line of code in the function. There should also be a space between the variable initialization and the next block of code, and there should be a single white space after an if statement or loop.

Detailed in ISO9899 there are several standard headers that can be used to implement standard libraries of which some will be used along with headers that will need to be included with the microcontroller library such as <time.h> and <math.h>.

4.2.1.6. Python Standards

For segments of the code that will be written in python such as the algorithm and the user interface we will need to use a different standard than the C standards because the syntax between the two languages and their uses are very different the active standard that we have found for python was a style guide known as PEP 8 on the python website.

According to the PEP 8 style guide when indenting code each level of indentation should be 4 spaces and wrapped code should be indented to the beginning of the parenthesis on the beginning line or it can be done with just the open parenthesis on the first line and then the following lines to define that function have an extra 4 spaces. This rule also applies to if statements and lists. According to this standard tabs should not be used unless tabs are already used in the file. If a line of code exceeds 79 characters then it should go to the next line in the method stated above and all following lines should not exceed 72 characters.

When importing libraries if two or more libraries are included in a file the libraries should be imported on separate lines for the purposes of readability, to go further imported libraries should be imported in the order of standard import libraries, third party libraries, and then local libraries and a blank space should be in between these groups. Not included in the PEP 8 standard but for simplicity in the autonomous guitar code, unless proven to be necessary, only full imports will be included in the code for readability purposes.

Comments should be used when appropriate, when something in the code cannot be intuitively understood then an inline comment should be made to give a brief sentence on the purpose of the line of code, above a loop, a function call, or a if statement, a block comment can be made to describe what is happening inside that block of code.

When it comes to naming conventions for variables and functions PEP 8 does have a set of rules but some freedom in naming styles. For the purposes of the autonomous guitar project any single letter variables will be lower case and all variable names that are longer than a single letter are completely lowercase. Use of single letter variables with lowercase l and uppercase I are prohibited by the PEP 8 standard as it can be confusing to read this. Naming conventions for variables should be only a single word to make typing them quickly easier; naming conventions for functions can be more than one word to provide a better description of the function but must have every word of the function start with a lowercase letter and have underscores in the place of spaces instead of doing it in a camelcase style as this is the general standard for python code. When naming constants the PEP 8 standard details that they should be named in all uppercase and in the case that the constants name is more than one word it should have an underscore instead of a space like a function name. Also every variable and function must be made with an ASCII character to avoid any complications or compatibility issues.

When writing exception handling it is important that the code inside the exception handler is specific enough to not catch more than a single exception in each statement so that it does not raise the flag for more than one possible error and the exception will make the user aware of the error. When making an if statement inside of a function the PEP 8 standard details that at the end of the possible decision if nothing changes from the function the final else statement should have a return value of None instead of just a return statement.

4.2.1.7. Guitar Tuning Standard

Standard tuning for the guitar will be used with notes on each string from top to bottom as usual: E A D G B E. They will be tuned using modern music's tuning standard of

A4 = 440 Hz

4.2.1.8. Electrical Component Standards

The integration of circuitry by way of component soldering and PCBs means that we would have to conform to a number of standards outlined within the technical field. This section will go over a multitude of standards that are relating to our project, of which our design would conform to - Not only are these standards designed for the smooth modularization and efficient production of parts, they are also written in order to insure the safety of us and those individuals who might have possession of the project in the future.

4.2.1.8.1. PCB Standards

The standards that govern the entirety of general PCB design are outlined within the IPC-2220 Family. The IPC (or, Institute of Printed Circuits) Is a Standards Development Organization (SDO) located in Bannockburn, Illinois that seeks the standardize the design and production of electronic equipment and assemblies. It is recognized world-wide, and thus whatever PCBs we order are likely to follow these standards closely - while our PCB will likely be routed by a third party company, the onus is on us to design our boards through whichever board capture software we utilize in such a way as to conform to these standards.

The IPC-2221 standard specifically outlines the generic standards for a PCB design, including specific parameters for component mounting and interconnections. For our project, this standard is also supplemented by the IPC-2222 standard which specifically refers to Rigid PCBs (as opposed to Flex, MCM-L or HDI boards). Below are some of the hallmarks of the IPC-2221 standard which our PCBs will adhere to. Note that these are the standards as outlined by Sierra Circuits PCB Manufacturing under their article “Applying IPC-2221 Standards in Circuit Board Design”, updated as of July 20th 2021.

Clearance is a major consideration for PCB design. Characteristics of schematic design such as parasitic capacitance and inductance mean that we will have to take into the account the gap between individual parts and copper runs - these considerations are well documented, and as a result have been included in the IPC-2221 standard. The different clearances within a PCB are categorized by what part is under consideration - Below is a table which outlines the various components on a PCB and their respective minimum tolerances as per IPC-2221.

Component Leads	0.13mm for up to a voltage of 50V
Uncoated Conducting Area	0.75mm
Test Probe Sites	80% of component height, 0.6mm min. & 5mm max.
Mounting Hardware	<6.4mm protrusion below PCB surface
PTH relief in heat sink	2.5mm larger than hole
Liquid Screenable Solder Mask	0.25mm (w/ 0.25mm dam)
Photoimageable Dry Film (<0.0635) solder mask	0.051mm (w/ 0.127mm dam)
Photoimageable Dry Film (0.066 to 0.1mm) solder mask	0.051mm (w/ 0.25mm dam)
LPI solder mask	0.051mm (w/ 0.1mm dam)

Table 10: Clearance Tolerances Outlined in ICP-2221 (via protoexpress.com)

Note the inclusion of dams for the standards outlining solder masks. it is especially important that this gap be maintained, as not only are we accounting for electrical characteristics but we also want to allow for the proper installation of surface-mount parts.

Creepage is another metric that is scrutinized as part of our PCB standards. Creepage is hard to distinguish from clearance for some metrics, and so a model has been included below to demonstrate the difference. It is included mostly for our benefit, so as to keep the standards within context while designing our PCB.

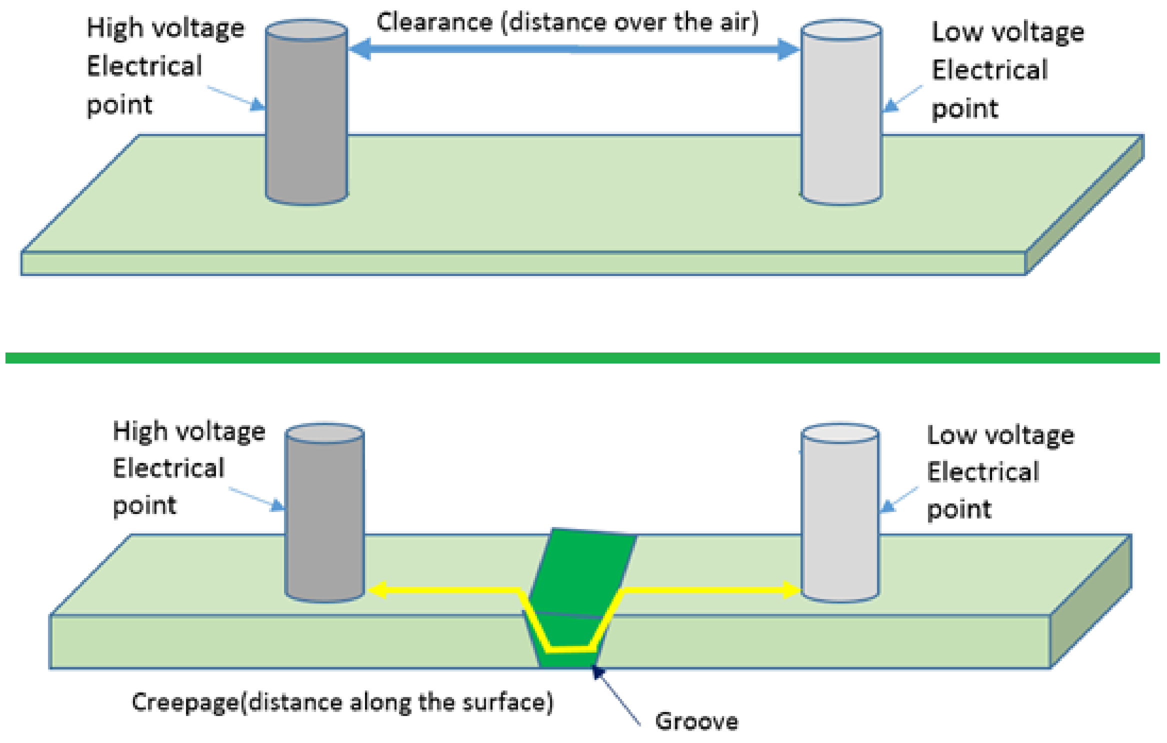

High voltage Electrical point
Clearance (distance over the air)
Low voltage Electrical point
High voltage Electrical point
Low voltage Electrical point
Creepage(distance along the surface)
Groove

Figure 36: Clearance vs Creepage on PCBs (via protoexpress.com)

By IPC-2221 guidelines, we should make it a priority to maximize the space between conductors such that creepage is large enough to accommodate etch compensation for all physical components on the board. This etch compensation is generally measured as twice the thickness of the etched copper on the board.

The third major component of the IPC-2221 standard lays out the recommended thicknesses for all copper traces on the board. The propagation of current through a copper trace could prove dangerous if the copper is too thin, and at high enough resistance and current magnitude, will burn the traces and damage the card. As part of the standard, our lowest tolerable copper trace thickness is calculated as a function of the current it is designed to sustain. Our thickness in oz (defined as 1.378 mils) is defined as $t = A / (w * 1.378)$ where w is width in mils and a is area, calculated as $a = (i/[k * (\Delta T^{0.44})]^{(1/0.725)}$. k is defined as 0.024 for internal layers and 0.048 for external layers.

4.2.1.8.2. Soldering Standards

The inclusion of PCBs into our project naturally leads us to the need for Soldering. The term "solder" refers to both the action of creating permanent electrical junctions by use of melting metal onto connections as well as the metal itself. For hobbyist applications it is most utilized for connecting PCBs to other components across a project - in order for our components to be powered and communicate with one another they need to be connected electrically, and by far soldering is the most efficient way to accomplish this as individuals. The standards that govern soldering components are largely defined under the IPC J-STD-001 Standard, and define what types of solder can be used for industrial applications as well as specify what an exemplary solder connection should look like. These standards are designed in order to ensure that connections made throughout a project last as long as possible with as little failure as possible, maximizing reliability of electrical components. In the following section we will go over the relevant components of the standards to our project, as outlined by Sierra Circuits.

Under the J-STD-001 standard, a number of general rules of thumb are outlined. These are more or less for industry practices, but apply to our project as well as good industry practices yield high quality results no matter who complies. It first states that cleanliness be emphasized to prevent the contamination of tools and surfaces - this is useful for any and all technical projects, but especially to ours as we will be working with small mechanical components which may be adversely affected by poor soldering techniques. Second it states that heating and cooling rates should be equivalent, and that multilayer chip capacitors are treated as thermal shock-sensitive to protect against thermal excursions. Adhering to this would be useful to prevent damaging components by heating or cooling them too fast.

The third specification is two-fold: it states that strands of the wires should not be damaged and that the solder must wet the tinned area of the wire. What this essentially states is that the solder must completely cover the junction, as wetting refers to the bonding of solder to metal and the tinned area refers to the area of wire we want soldered. These two standards together help ensure that the connections made between pins and wires or wire-to-wire junctions are sound and will not detach if non-excessive force is applied. The fourth, fifth, sixth and seventh statements apply mostly to packaging and manufacturing inspection, and thus can be safely disregarded as they are not relevant to our project.

The standard then goes on to define what a tolerable solder looks like. Specifically, the standard discusses soldering Header Pins. Header Pins are the pins that come out of integrated circuits and bare-PCB devices, such as most LCD screens. On a PCB, these header pins are utilized using through-hole mounting to the PCB (Often written as PTH, Plated Through-Hole mounting) - the connection is established by soldering the pin to the hole surrounding it, as the copper is located in and around the hole that the pin enters through. The copper surrounding the hole on the opposite side that the pin enters from is referred to as the pad. A number of characteristics make up an effective solder junction for PTH components. The most important characteristic of a good solder is that the PTH be entirely filled with solder. Additional specifications are that the meniscus (surface) of the solder when traveling from the tip of the pin to the pad be concave, and that the pad itself is well covered with solder, but not covered so much that it causes the solder to spill over to other pads. The following is a figure meant to demonstrate this criteria.

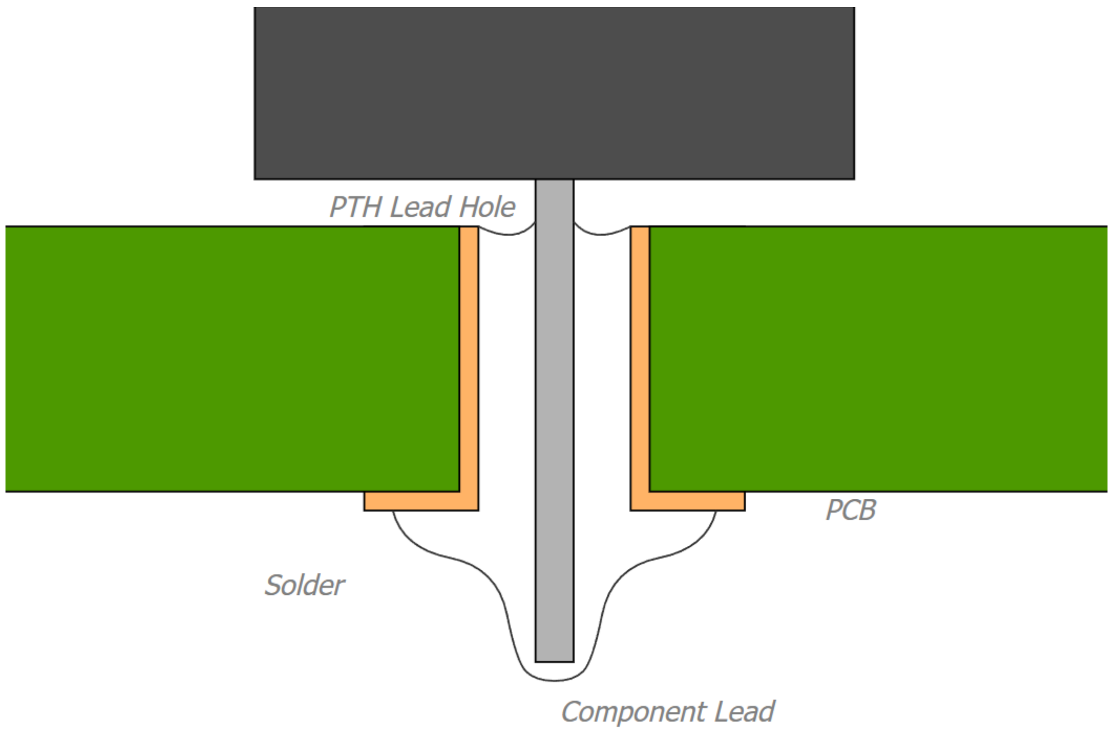


Figure 37: Model of Exemplary PTH solder

This specification is meant to ensure that the soldered lead stays strong throughout the component's lifetime. Thus, maintaining this standard will be helpful for the final design, as this type of connection is used for a number of connections - For example, the 78xx series family of Linear Voltage Regulators which utilize through-hole leads.

The standard also goes into detail about the characteristics of surface mount solder connections. Most base electrical components nowadays such as resistors or inductors use surface mount profiles - as such, it is important that we understand these standards so as to make sure the connections we make last as long or longer than the parts themselves. For rectangular or square end chip components, it is recommended that the fillet height (height that the solder is bonded up the end of the chip) be $\pm 25\%$ the height of the chip from the board.

4.2.1.9. Power Supply Standards

The power supplies that are used to power the devices we come across every day come in extremely varied capacities and outputs, and as such the standards that govern them are similarly varied. To help catalog these circuits and ensure that the standards are as thorough as possible, circuits are put into one of a few classes based on the power characteristics that govern them. Each circuit classification comes with a set of commonly accepted safety practices that pertain to the attributes of the class. Below is a table that establishes some circuit definitions which are important for standard classification and summarizes their characteristics. These definitions are provided by CUI Inc. and are generally accepted among the electrical engineering community.

Hazardous Voltage	>42.2 Vac, >60 Vdc, without current limited circuit
Extra-Low Voltage	<42.2 Vac, <60 Vdc, Separated from hazardous voltage by basic insulation or more
Safety Extra-Low Voltage	Can't reach hazardous voltages between any two parts or a part & ground, under fault condition doesn't exceed 42.4 Vac or 60 Vdc for more than 0.2sec. Two levels of protection from hazardous voltages
Limited Current Circuits	Under fault conditions, current drawn is not hazardous. Current from frequencies <1khz dont exceed 0.7mA ac or 2mA dc, frequencies >1khz currents dont exceed (0.7 * freq) or 70mA.
Limited Power Source	Inherent power limiting, linear/nonlinear impedance power limiting or regulating network power limiting power supply.

Table 11: Power Supply Circuit Definitions (via CUI)

Going by the definition given to us by the CUI, the sum total of our system will be defined as an Extra Low Voltage circuit - Our highest voltage draw will be as a result of the servo assembly that will be used for the mechanical action. The voltage draw, per component, of our servos is approximated at 5 volts. If we wanted to step down voltages in order to lessen current load on the components throughout the device, we could still maintain well below 60 volts for supplying the device with power.

Additionally, CUI outlines some safety standard classifications which help define how dangerous a piece of equipment or a project is to operate or maintain. Class I devices are protected from electrical shock with only basic insolation and grounding. These devices are required to have ground connections on components which could be expected to reach dangerous voltages in the event of insulation failure. Class II Devices are similar to Class I, only they are protected using double or reinforced insulation which removes the requirement for a ground connection. Finally, class III devices operate from a Safety Extra-Low Voltage supply circuit, which inherently protects them against hazards such as electric shock or dangerously high voltages.

4.2.1.10. 3D Modeling, Printing, and Laser Cutting Standards

For interoperability between different 3D modeling softwares and 3D printing devices, there are standard file types to store the data for the designs.

4.2.1.10.1. STEP File Standard

The ISO 10303-21 standard, also known as the STEP file standard, is the standard used for storing 3D models that can be transferred between different 3D modeling softwares. It is an ASCII-based file type, so it can be read by humans via notepad or some similar text file processing software. Unfortunately, the standard is not freely available and is quite complex. As a result, the standard may not be fully supported by every 3D design software, which will make it difficult to interoperate between different 3D modeling softwares. For this reason, each 3D modeling software usually has its own file format that works better for that particular software. It may be a good idea to stick to one CAD software in order to avoid file transfer issues.

4.2.1.10.2. STL File Standard

The STL file standard is a simplified 3D model standard typically used for 3D printing. Because it is so simple, it is not ideal for 3D modeling, as that typically entails complex operations that are not easily supported by STL. However, the simple description of the geometry of a design provided by an STL file is perfect for 3D printing, where the static geometry is all that is needed. Most CAD software is able to export to this file type, so whichever 3D modeling software we choose, we should be able to generate these files for the purpose of 3D printing. Afterwards, the TI lab technicians can handle the rest. **We ended up using STL for the 3D-printed strumming mount assembly in the final project.**

4.2.1.10.3. SOLIDWORKS File Standards

SOLIDWORKS has several file formats that it uses to store its designs. The SLDPRT format is used for storing the designs of individual parts. A single part is one cohesive block of material, and is typically manufactured separately from other parts. Once several SLDPRT files are made, they can be assembled together in a SLDASM file. This places all the parts together based on mating rules, to ensure that the parts will be able to interoperate correctly once they're assembled in the real world. For example, we can find existing files for the standard motors that we use, and then assemble that together with the 3D mounts that we design. Even better, we can actually design the 3D printed parts inside of the SLDASM file, and use the geometry of the motors as a reference. This will ensure that the final mounts will fit perfectly with the motors.

4.2.1.10.4. DXF File Standards

The DXF file standard was created by AutoCAD for interoperability with other CAD software. For us, it is a format that supports the encoding of a 2D drawing for the purpose of acting as laser cutter instructions. **We ended up using DXF for the laser-cut fretting mount assembly in the final project.**

5. Final Design

5.1. Hardware Design & Part Details

5.1.1. Mechanical Components

Thanks to their cheap cost and ease of use, we decided to go with servo motors to implement both the strumming and fretting actions on the guitar. The 30 servos will cost $60 on amazon, which is not a bad price. For mounting the servos to the guitar, we will be designing a 3D-printed mechanism based on a free and open-source file we found online.

5.1.2. Power Supply

In order to decide on a power supply, we needed to know the power demands for our project. The following is a table that summarizes all the power demands for the various components we utilized in our final design. These power demands are taken from the data sheets of each component included in Appendix D.

Component	Voltage Demand	Current Demand
ESP32 Microcontroller	Min: 1.8V Max: 3.6V Recommended: 3.3 (Regulated from 5V by devboard)	Recommended 0.5A
PCA9685 Servo Shield (x4)	Min: 2.3V Max: 5V Recommended: 5.5V	6 to 10mA
SG90 Servos (x30)	5V	~270mA per servo(moving) 10mA idle

Table 12: Component Voltage and Current Demand

As we have mentioned in the research section, a major consideration we had to make is how the high number of SG90 servos will affect the circuitry. Connecting each servo in parallel was the preferable option and will prevent us from reaching excessive voltages, but the current draw that these devices would collectively require could proved to be a heavy burden for consideration of power demands and supply.

When ensuring that all our electronics are adequately powered. The item that will cause the biggest issue will be the servo motors as they require a fair bit of current. Operating they need around 100mA idle they draw around 10mA so at a minimum we will need to have our power source be able to push 300mA for servos and that is assuming they are all off. If we go on the safe side and assume we have a maximum of 12 on at a time we will need to be able to produce around 12*250mA(Active servos) + 18*10mA(idle servos) + 80uA (Shift register) which is around 3.2 A. In this scenario when selecting our power supply it is important we get something that can provide the 5 V needed to power our electronics while also providing the current we need.

Based on our Power Supply selection, we elected to utilize the ExpertPower 12V 8AH Sealed Lead Acid Battery. The characteristics of the battery are favorable, outputting above our 5V regulation floor with a 12V output voltage along with being able to tolerate 8A of current. Additionally, the battery is rated to output 8AH, which should give us an impressive battery life when considering that the servos will only be at max current draw for a small amount of their total operating time. Finally, the battery is of sufficient size to implement into the form factor of the guitar that we will be modifying and, if not be hidden entirely, will not be easily noticeable when placed under scrutiny.

5.1.3. PCBs

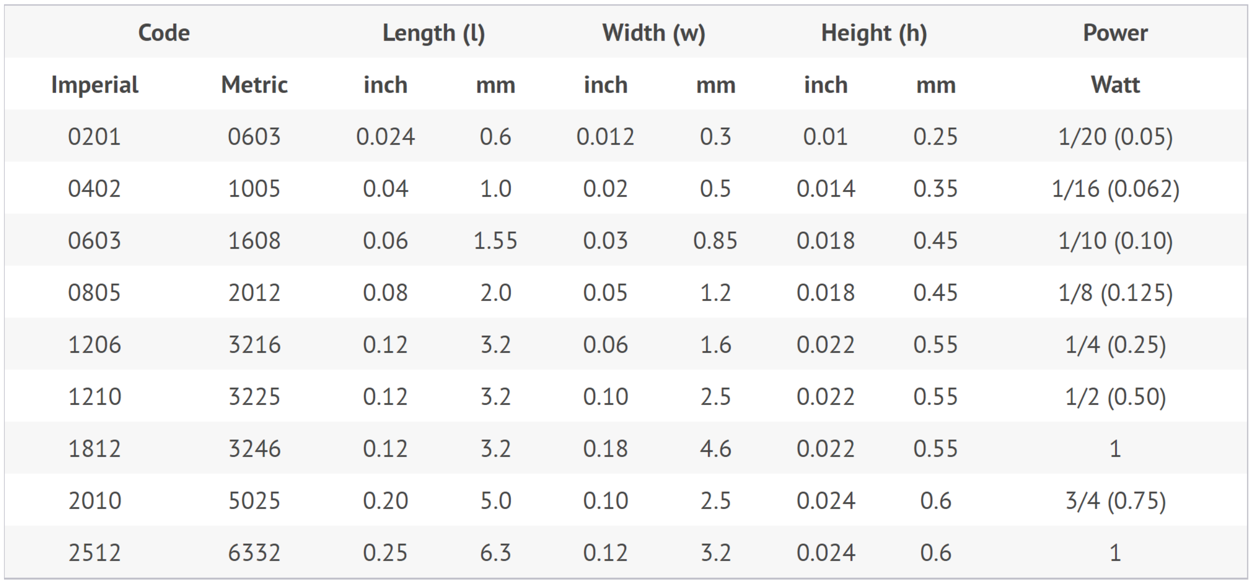
This section will go into detail about how each component will be consolidated onto PCBs. Primarily, this will encompass our microprocessor and how it will be integrated into the project so as to communicate with the mechanical components. For our project, we will also integrate voltage regulation in order to properly supply power to said components.

5.1.3.1. Surface Mount vs Throughput

When selecting the individual electrical components we will be using in the circuit design we will need to make a decision on what type of mounting we will choose to use. For us the easiest solution may be to use surface mount, for one we don’t have the most experience with soldering so this solution offers us an easy way to not only mount the components but also to remove or make any adjustments to the positioning of needed. The major drawback with this issue is that with our lack of soldering experience we can mess up the connection and connect them in a way that their connection is lacking since it connects in small areas on the board. One thing we can do to combat this issue is use throughput mounting components. This will make the component mount through the board onto the backside where they will be soldered. The benefit with this is that we will have a higher connectivity since the lead runs through the hole and makes sure our connection is more reliable. This will be a little trickier and we must have the manufacturer cut pre-cut holes into the PCB designated for the points. This solution also is a little bit harder to make adjustments and move things around to troubleshoot things. With this being said we will go forward using surface mount electronic devices and if we come into any issues with the PCB and not our design we will make a switch to throughput devices on our next PCB.

5.1.3.1.1. Component Size

When selecting the component size we want to use there are a couple different factors we need to take into account. From research it is important we do not make our components too small in size, with this it will make everything much harder to solder on and could lead to damaging our components. At the same time we don’t want to make all the components too big because this not only drives up the cost for the PCB but it also eats up more of our limited space for our microcontroller and other devices. The figure below is a catalog of component sizes. Ideally, we get a size in the middle which appears to be 1206 or maybe a 1210 depending what we decide when we actually receive them. For our final design we made one small mistake and that was choosing to use 0402 sized resistors in our circuit. It wouldn't have been too difficult but two of them fell off and this caused us to have to solder them by hand which was difficult due to the small size of the resistors. Looking back we could have gone with a bigger package or just used through hole resistors.



Code		Length (l)		Width (w)		Height (h)		Power
Imperial	Metric	inch	mm	inch	mm	inch	mm	Watt
0201	0603	0.024	0.6	0.012	0.3	0.01	0.25	1/20 (0.05)
0402	1005	0.04	1.0	0.02	0.5	0.014	0.35	1/16 (0.062)
0603	1608	0.06	1.55	0.03	0.85	0.018	0.45	1/10 (0.10)
0805	2012	0.08	2.0	0.05	1.2	0.018	0.45	1/8 (0.125)
1206	3216	0.12	3.2	0.06	1.6	0.022	0.55	1/4 (0.25)
1210	3225	0.12	3.2	0.10	2.5	0.022	0.55	1/2 (0.50)
1812	3246	0.12	3.2	0.18	4.6	0.022	0.55	1
2010	5025	0.20	5.0	0.10	2.5	0.024	0.6	3/4 (0.75)
2512	6332	0.25	6.3	0.12	3.2	0.024	0.6	1

Figure 38: Component Sizes with Dimensions

5.1.3.2. PCB Size

The size of Our PCB will be reliant on what we decide to put on it, essentially it can vary if we decide to mount some of our other electrical devices onto it such as the microcontroller and the power supply. While these are options we can implement for us it may be easier to keep them separate but all near each other in some sort of housing unit. For us our goal will be to minimize the size of the PCB since we are limited by the amount of space we have on the guitar. Even though we will minimize it as much as possible we need to take in account any wires that may interfere with each other if located too close together.

5.1.3.3. PCB layers

When picking out our custom PCB we need to decide how many layers we need to have. This is something that we will choose and it is dependent on how expensive and complex we want everything to be. The different layers consist of different properties. The most common is to have a 4-layer PCB with one layer being for our voltage which in our case it would be 5 volts and another layer would be the ground layer. Those two are middle layers; the top and bottom layer are used for soldering the electrical components onto the PCB. Another option would be a 2 layer which would allow us to have traces on both sides. The main difference is between the complexity in the circuits we are designing. In our case we have some complicated components and a lot of connections. In order to handle all the different connections we will use the 4-layer PCB for our custom design, with the topography described below.

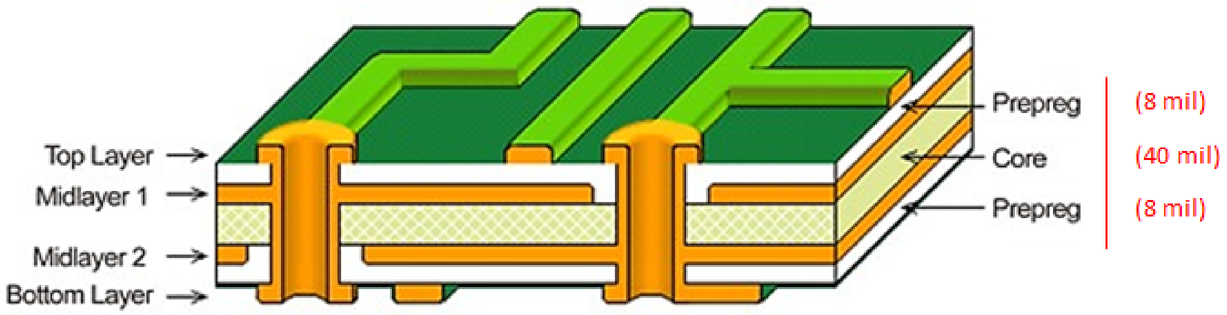


Figure 39: 4-layer PCB diagram

5.1.3.4 Voltage Regulation

The final Motor Assembly consists of 30 PWM-controlled SG90 Servos, split across the 6 servos used for strumming and the rest used for fretting the neck of the guitar. Each Servo has a 270mA max current draw, leaving us with a max theoretical current draw of 8.1A and minimum draw of 300mA from the servo assembly when looked at as a singular equivalent load. The motors will be attached in parallel, with each motor requiring a voltage of 5V to operate.

To make use of the SLA battery, as well as to step down its voltage, we decided to use five LM2576 Switching Voltage Regulators with isolated outputs. Specifically, we utilized the LM2576ADJ, which allows for a variable output by modifying the feedback voltage divider in the circuit, though for our purposes the outputs for each regulator will all be set to 5 volts. Each regulator is rated for outputting a maximum of 3 amp to a given load, thus we will have to utilize multiple regulators. Initially, we had planned on using more modern SMT regulators capable of handling more current - However, we decided on using multiple regulators of lower current rating rather than a single IC solution for a number of reasons. First and foremost, the LM2576 had multiple packages; one is a surface-mount design while two others are through-hole packages. On top of this, the LM2576 is already employed by UCF for our Electronics 1 labs - what this meant for us was that we could test the design effectively and quickly by obtaining the through-hole packages and using breadboards. We would also be able to refer back to our previous lab manuals, allowing us to troubleshoot the design in the event that faults are found. Another reason why using multiple regulators was favorable is that the thermal energy dissipated across the regulators can be spread out across the board rather than being centralized in one spot and burning a hole through the board or damaging individual chips. Lastly, utilizing a popular Texas Instruments component means that we have ready access to TI's software, such as WEBENCH and PSPICE for TI. This software is powerful, and would allow us to rapidly generate and simulate a ready-to-use circuit assembly for testing and later create a board file which we can send to a board house for production. The figure below is the schematic used to power the project.

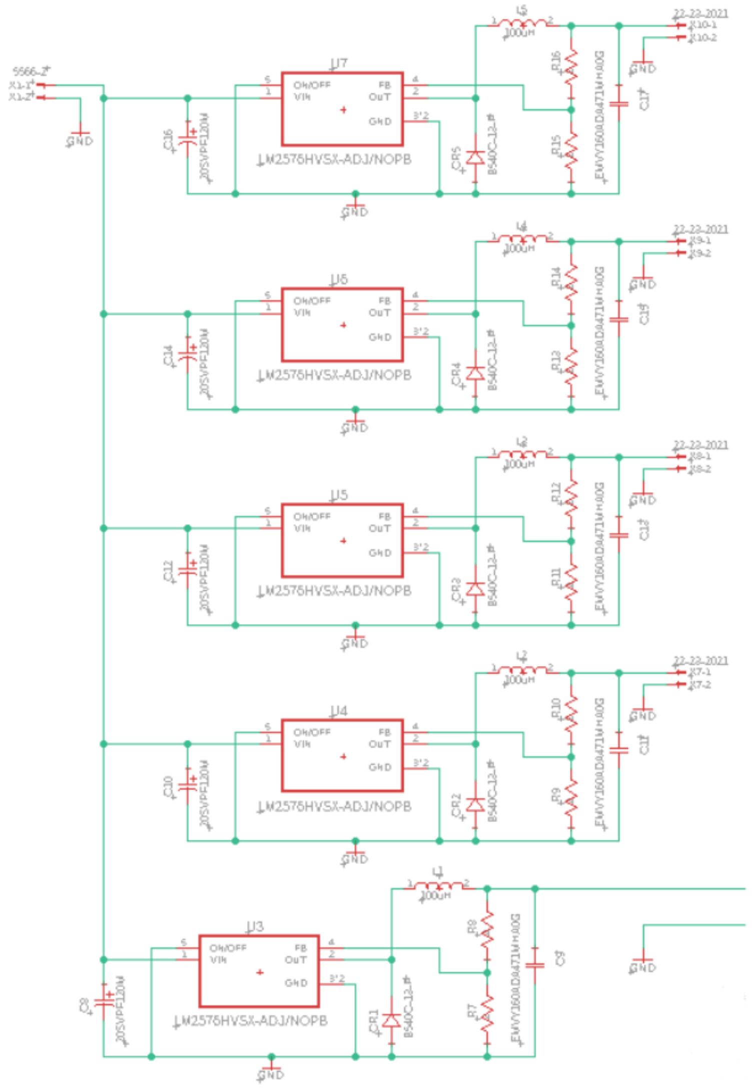


Figure 40: Regulator Schematic

The servos have been divided into 5 servo assemblies each containing 6 servos. As stated earlier, 4 of our regulators are dedicated to powering the servo assemblies, with

24 being located in the fretting assembly and 6 being located in the strumming assembly. All 30 of these servos are supplied power by 4 of our regulators, 3 of which are dedicated to the fretting assembly and the 4th regulator powering our strumming assembly. The final regulator will power our ESP32 Microcontroller. One thing to note is that originally the final regulator was also planned to power peripheral devices, however they did not make it to the final design. Additionally, the LM2576 is not safe for dual-regulation; this means that each regulator output must be its own node - otherwise, back current will damage the components. Thus each regulator used in powering the fretting assembly will supply power to 8 individual servos while the regulator used in the strumming assembly and peripherals will be totally separate. This power is passed through the servo shields that are also used to control the servos.

5.1.3.5. Wire Selection

It is very crucial for our project that we have the right wires running through our circuit. Our issues with wires can be split into two different categories one is resistivity of the wires and the other is the amount of current that can be passed through the wires to supply our servo motors.

5.1.3.5.1. Resistivity

When using smaller wires to jump breadboards we don't typically notice a voltage drop in the circuit and that is due to the length, for this project we will be having wires travel a decent distance on the guitar regardless where we put the PCB since we will have servo motors on both sides of the guitar. In this case we need to be aware of any resistivity the wire may possess. With this it could cause our servo motors to receive insufficient voltage which can be a headache to troubleshoot. So this potential issue may need us to adjust the regulator we go with in order to account for the resistive load of the wire. So instead of 5 volts coming out of the regulator we may need 6, that is within the operating range of the servo motors so there should be no issues with providing too much voltage. And if we need to drop the voltage going to the microcontroller we can put a resistive load to draw some of the voltage. While a solution to this problem would be to find a wire with the lowest resistivity we run into our next issue. Calculating resistance over the wire would be $R=\rho L/a$ in this formula the only thing to decrease our resistivity is to increase the thickness of the wire. This creates issues with space if the wire needs to be too big it will interfere with the space in the surrounding wires. Another issue we run into is the amount of current that is running through the wires, which we'll go into in the next section.

5.1.3.5.2. Gauge of the Wire

Since some of our project requires a fair bit of current to run we need to make sure we get an appropriate gauge of wire to handle all the current that can be run in it. If we do not this could cause issues with the current being received as well as damaging the wire internally. In order to increase the amount of current that can be passed through we need to make our wire thicker so if the wires in the lab do not work we will need to look for wires with lower gauges but that runs into the issue of space on the guitar. However in order to make sure the guitar functions as intended we will need to find a happy medium with the thickness of the wires. Typically the amount of wire used is around 20 gauge wire. Since we do have a higher amount of current we will be using we may have to go down in our gauge number to possibly 16 just to take care of the current amount. The price may increase ever so slightly based on the gauge size. The only other thing we may possibly need to worry about is the amount of space we need to have allocated depending on the thickness of the wires. We should be able to place all our wires with plenty of space due to the wires for 16 gauge being relatively thin. We also need to worry about soldering. On the ESP 32 the output pins are very close together and in order for them to not short themselves out we need to make sure our wire soldiers are precise and not excessive. If we solder we need to strip the wires very little amount as a lot of wires jumbled together like that they could have some excess wire exposed and short each other out. And we also can't cut too much since we need a fair bit of wire so we need to just cut a little bit and be precise with all our soldering and our wires. Below we can see the gauge sizing chart relatively and we can see the current amount that they can handle. This is something we focused too heavily on and after testing we noticed we didn't have as much current as we expected too. This is due to both us changing the control design but also our code was very efficient at reducing power consumption. We could have easily gotten away with using 24 gauge wire for the connection to the servo motors vs 16 gauge wire, we found the 16 gauge wire had some issue actually fitting into the connectors we had. However we did make it work but it's something that could've been more optimized and made things easier

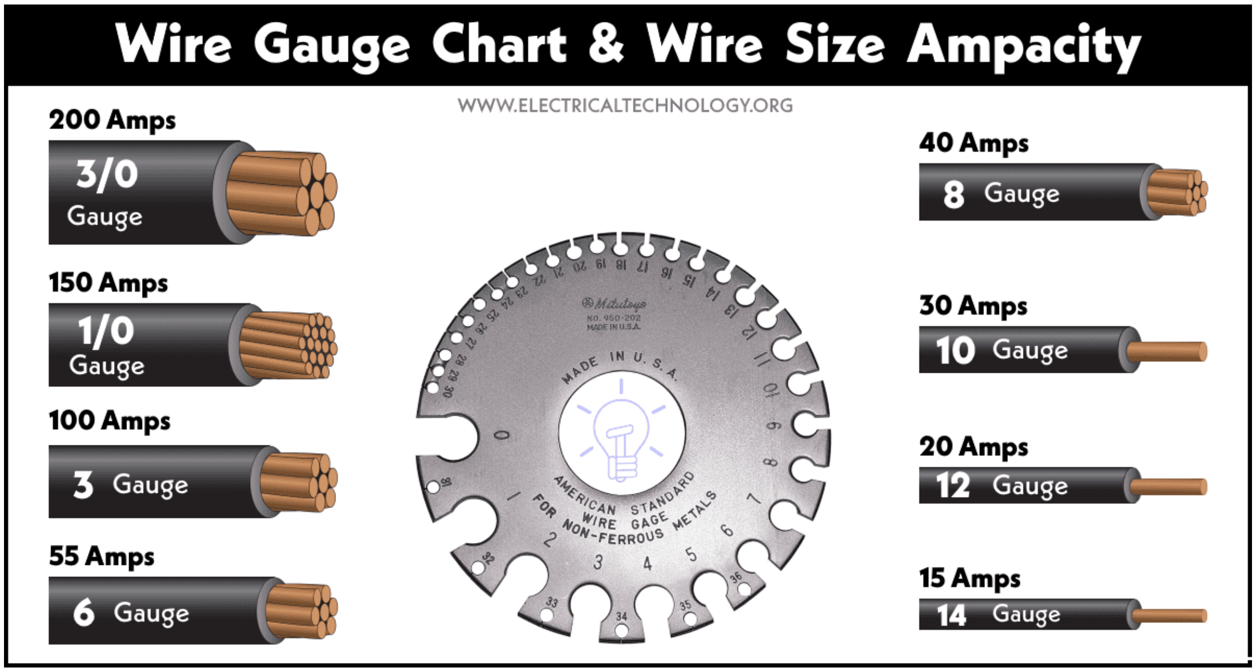


Figure 43: Wire Gauge Diagrams with Corresponding Currents
https://www.electricaltechnology.org/2022/04/american-wire-gauge-awg-chart-wire-size-ampacity-table.html

5.1.3.6. Passive Component Supplier

In order to supplement the above circuits, we needed to select a supplier for the various miscellaneous parts that will surround our main parts. Preferably, our selection should be diverse and not become a bottleneck in the future as we are forced to hunt for parts across other companies. To this effect, we leaned heavily on Digikey and Mouser as our primary suppliers - they have developed a reputation as having a wide selection of parts, and thus we will be happy to utilize them for our project. The following section details notable attributions for the parts we will order; as ordering individual parts that cost within the order of cents is generally negligible in the grand scope of the project, we will only go over the general considerations we will keep in mind as we go about transferring the schematic diagram to a real PCB.

5.1.3.6.1. Power Components

Power components are especially of note when we look at the operating conditions of the Motor Assembly Regulator Circuit. Because of the notable amount of current that will be flowing through the output of the regulator, we will need to look for components specialized in receiving high power.

5.1.3.6.2. Low Noise Components

As the signals traveling through our dc/dc converters lack a frequency component, noise will not play as big a factor in our power circuitry. However, the amount of noise generated by our passive components will matter for our microcontroller as we will be dealing with signals. In order to ensure we have as high a signal-to-noise ratio as possible, we will be utilizing components such as thin film or wire wound resistors where we can in order to make sure that our circuitry acts as expected.

5.1.3.6.3. Power Supply Switch

Because we are utilizing a portable power supply, we want to be able to turn it off for moments where we are not operating the system. Thus, we are utilizing a switch in order to cut power from the supply to the rest of the project. Specifically, we will be using the SPST Standard Toggle Switch from Parts Express.

5.1.3.7. Power Configuration

The below figure is a block diagram of the final design for the power system in our project. In this way we will ensure that each component is operational upon connection of the power supply in such a way as to be protected from overcurrent conditions Note that the switch will be a single part, chosen as discussed in section 5.1.3.6.3. The red wires indicate 5V power rails, while the black wires indicate ground connections.

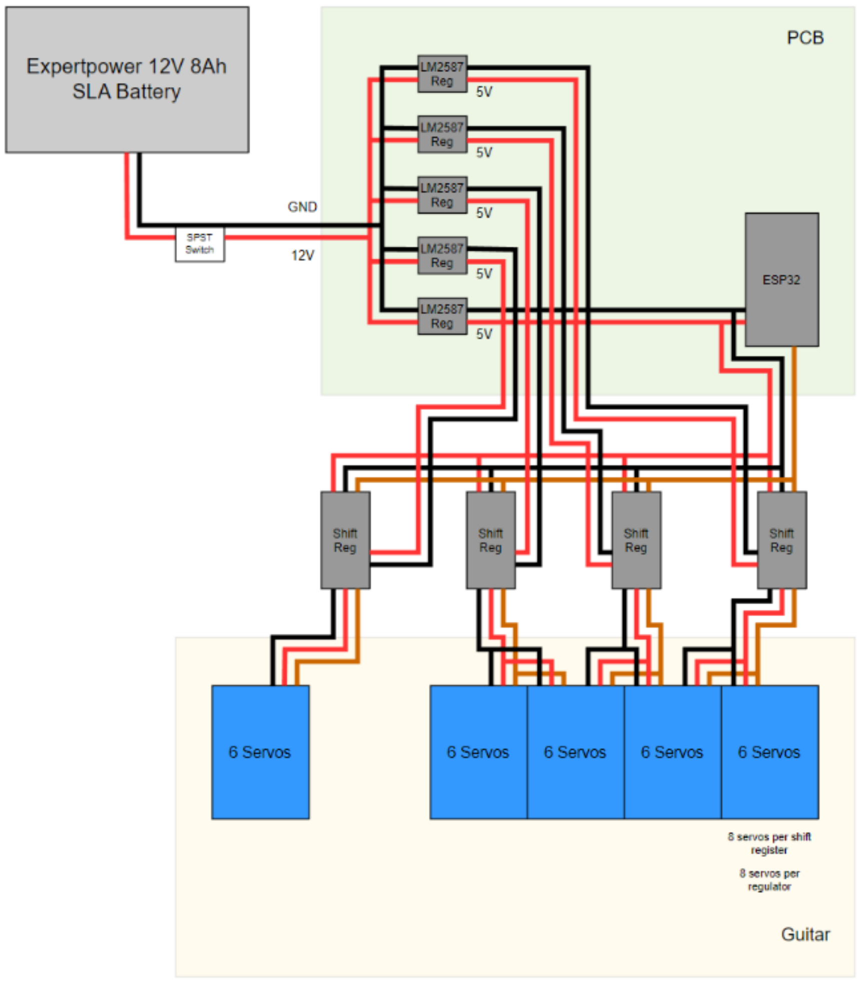


Figure 44: Power Configuration

5.1.3.8. Signal Configuration

When looking at the signals for the servo motors they will differ based on the purpose for that specific servo motor meaning if it is a strumming or a motor that is used to press the fret. They will differ most likely by the signal that is sent to them. Below we can see an example pulse of something we would want to see coming into the servo motors.

Servo motors that use strumming will need to be strummed across the strings for this we will need to give them a max pulse which allows them to rotate 180 degrees. This is because we need the servo to strum across the string in order to produce a note. While moving it 180 degrees will be our best decision, it is important to note that we will need to send another pulse that rotates it an additional 180 degrees back to its original starting position in order for it to get ready to strum again. When programming the signals we don't necessarily need a full pulse to get the string across but we do need to make sure we can get our servo motor to its original starting position or somewhere close to it so that we make sure we strum the guitar strings when we want to. Also it needs to be as quick as possible so that is something we will need to be aware of and we will need to practice the timing on the actual guitar in order to perfect our pulse signals. Below we can see an example pulse of something we would want to see coming into the servo motors.

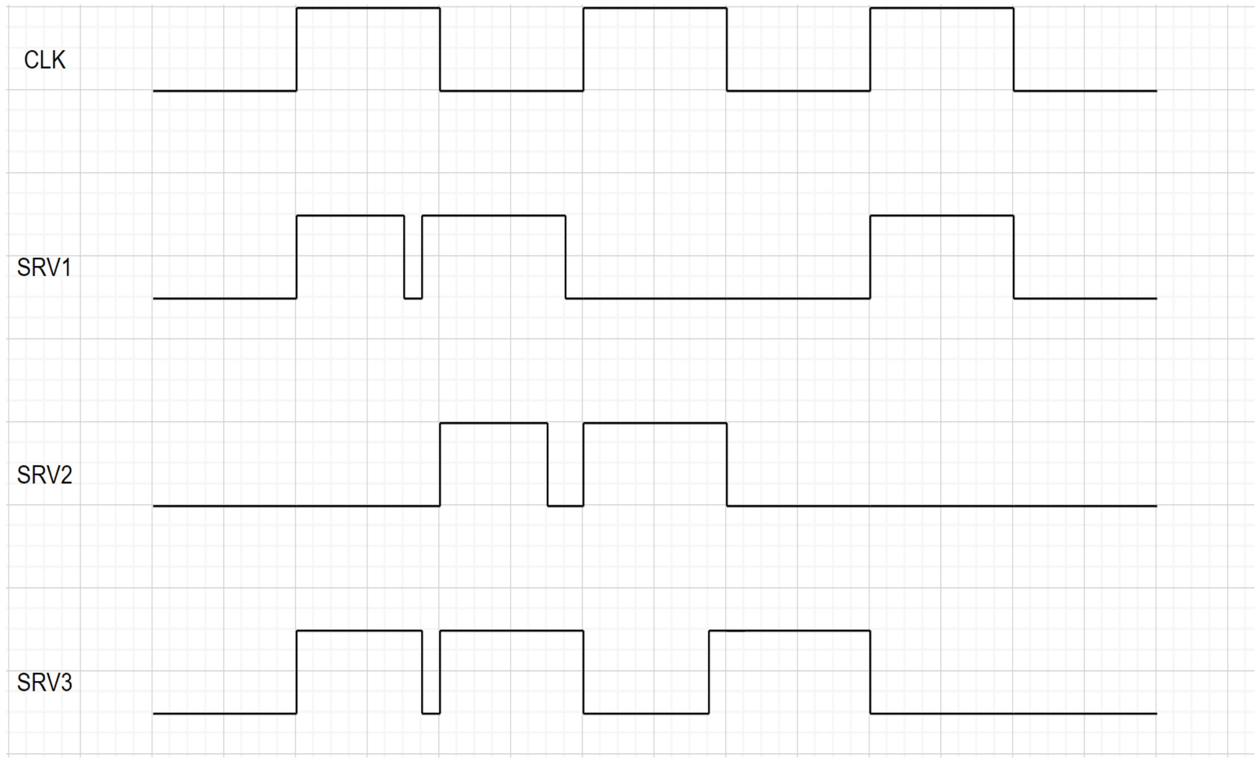


Figure 45: Example Signals using 800° Rotations for Strumming Assembly

The fret servo motors will be a little different as it wont need to be a full 180 degree rotation in order to press down depending on the route we take, if we go the route of using the 3D printed linear actuator we will need to use 180 degree pulses at a time in order to go up and down. This is due to taking advantage of the rotational motion in order to create linear motion using 180 degree turns. These signals will be different from the strumming signals as they would need to be held down for a longer time like a little bit before and after the strings are strung. That will be in order to make sure our note is clear and not skipping notes. So the time in between signals will need to be longer before resetting to their original positions of open. If we went a different route of pressing the frets our signals would look different as we more than likely would not need to go 180 degrees at a time and can take advantage of 90 degree rotations. If we go this route we would need to go back to the starting position each time with 2 pulses one being max pulse and the next being a half pulse to get it to the correct duration. Below we can see an example pulse of something we would want to see coming into the servo motors.

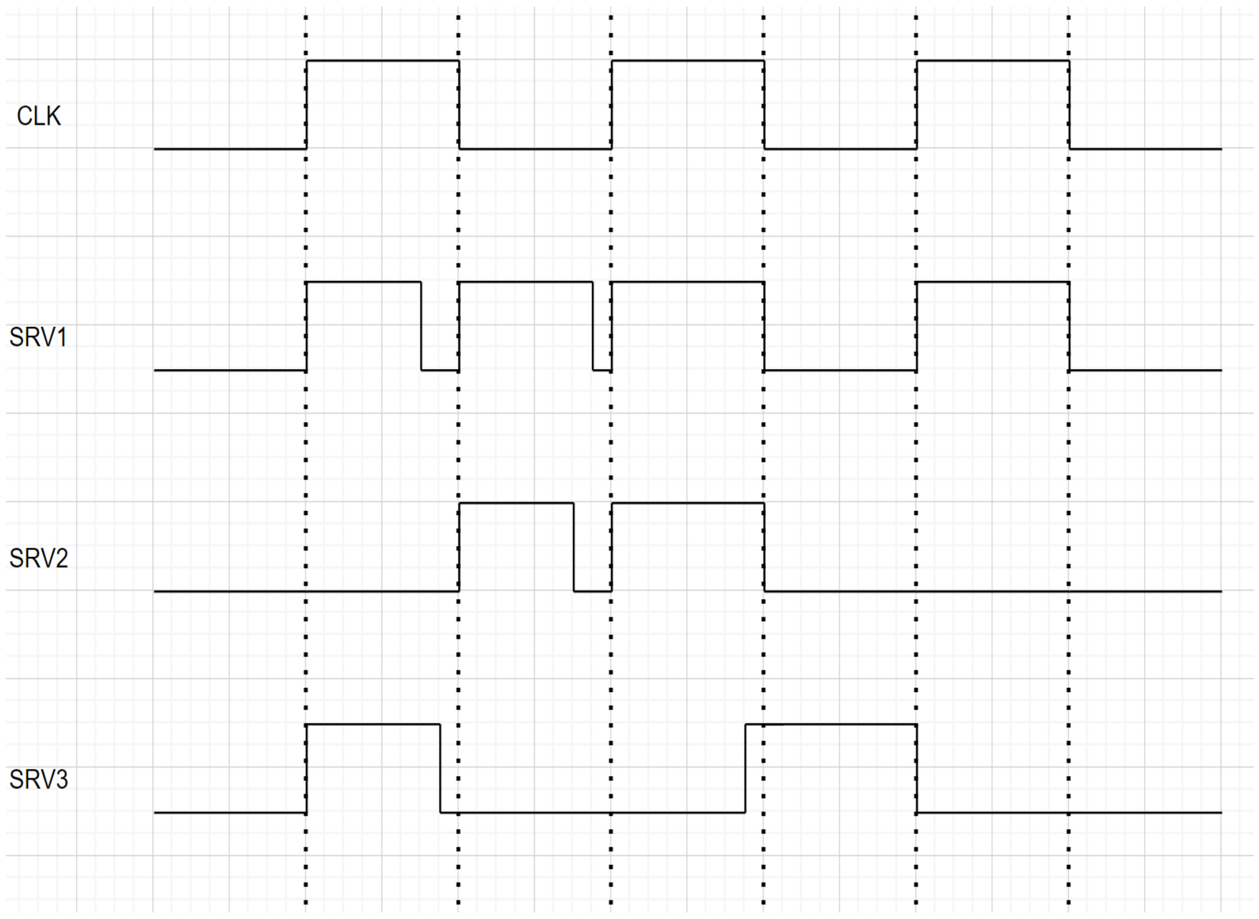


Figure 46: Example Signals using 180° Rotations for Fretting Assembly

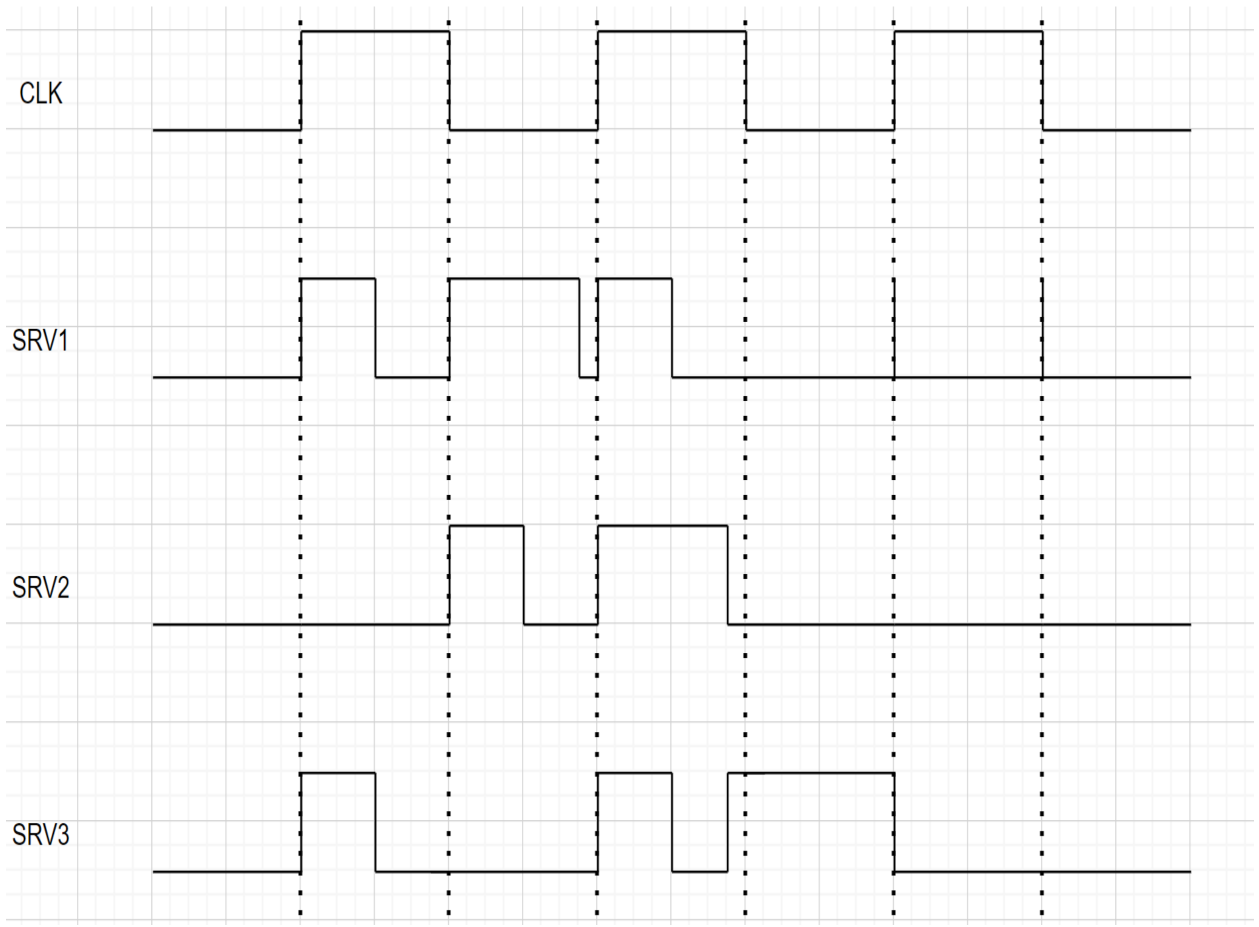


Figure 47: Example Signals using 90° Rotations for Fretting Assembly

5.1.4. Microcontroller

The microcontroller we are using is the ESP32. We initially wanted to use a single board computer to give ourselves more flexibility with the programming and to take advantage of the various types of I/O available on those boards, such as USB-A and Wi-Fi. When we found the ESP32 however, we were impressed by its low cost and Wi-Fi and Bluetooth support. This will make the programming more complicated in 2 ways: 1, we will need to use MicroPython to implement the MIDI processing code on the microcontroller, which may not be as flexible as a full implementation of Python, and 2, we will have to implement some kind of protocol to send the MIDI files over Bluetooth, as there are no physical ports that we can use for the file transfer. This Bluetooth protocol will run on an external computer, but will only be used for uploading files. The ESP32 has plenty of non-volatile flash memory that can be used to store the songs so that they can be played without a Bluetooth connection.

The ESP32 has 3 UART interfaces and each of these can receive and transmit at 5 Mbps. It also supports Bluetooth and Bluetooth low energy and is capable of having multiple channels and can interface with SPI and UART, It can only interface with UART up to 4 Mbps which given the size of the files that we are sending to the microcontroller this will not be a problem since the requirements for the autonomous guitar specify that the maximum MIDI file size is 50 KB. Bluetooth on the ESP32 is compliant with the Bluetooth 4.2 standard. The ESP32 has 2 sets of pins for SPI so the autonomous guitar will have to utilize a daisy chain configuration for SPI to make the shift registers work since it will need 8 shift registers. The ESP32 has 1 set of pins for the I2C protocol this is will work fine for the Autonomous Guitar since there is only one thing that needs to be connected via I2C.

The ESP32 has 5 power modes, they are Active mode in which everything on the microcontroller is turned on including all antenna functionality such as Bluetooth and wifi having this mode on can draw the largest amount of power depending on which transmitter/receiver is being used. Modem sleep mode will turn off all Wi-Fi/Bluetooth functionality. This method will not be able to be used for the Autonomous guitar because the user should be able to stop the song at any time from the user interface on an external device. Light-sleep mode will pause the CPU while RTC memory and peripherals connected to the RTC clock are still running, the system can be woken up via interrupts or the RTC timer, this could be used for the autonomous guitar. Deep sleep mode only powers on the RTC memory and the RTC peripherals Wi-Fi and Bluetooth connection will be stored while in this low power mode. Hibernation mode turns off everything but the RTC timer and a few RTC GPIO pins which can wake the microcontroller up in case of an interrupt.

In the Autonomous Guitar the use of the microcontroller is crucial to the entire design and is the main part of the Autonomous Guitar, it needs to communicate with the personal computer or smart device to get the MIDI file, once on the microcontroller then the MIDI file needs to go through the algorithm to be translated into the best interpretation of the guitar notes to play that note, then the microcontroller needs to decide which servos to activate to play that note, it also needs to control the LCD screen will display the current status of the guitar.

5.1.4.1 Wireless Communication for Microcontroller

Initially we wanted to use Bluetooth for our project but we discovered that micropython does not support serial Bluetooth. We have attempted to make a solution work with BLE but found that if we wanted to implement the Import file stretch goal we would not be able to use this solution due to BLE’s low data rate. We decided that using WiFi was the best course of action to connect to the front end because it was easy to use within our constraints and the front end could be viewed on any web browser.

5.1.5. Circuit Design

We can begin to construct our basic circuit design. The base of the circuit design can come from the three cables that are connected to the servo motor. We will have a power source that will be supplying 5 V(DC) to the servo motors. We will connect the power line to 5 V for the servo motor and connect all their grounds together and each control wire will be hooked up to an independent pin on our shift registers. This diagram will give us a good look into what the PCB will entail. As for the power source we will more than likely use some regulator in order to make sure we don't overload the circuit. Below is a diagram for what the circuit will look like.

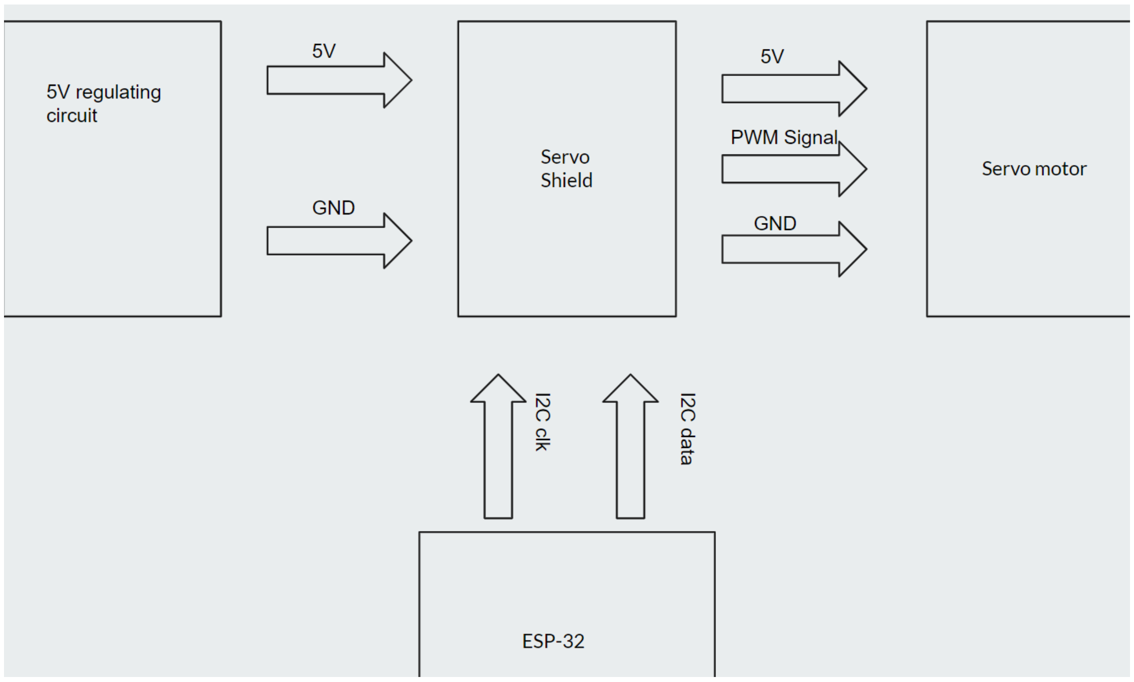


Figure 48: Circuit Design Diagram Prototype

This schematic demonstrates the daisy-chaining capability of the shift registers. The power and ground connections of nearby servo motors can be easily chained together, but the PWM signals need to be individually connected from the 4 shift registers.

5.1.6. 3D-Printed Mounting Hardware

The servo motors need to be very closely packed, so the mount will be sized such that three servos are placed as closely as possible, with a slight stagger so that the actuating arms don't hit each other. See the following drawing:

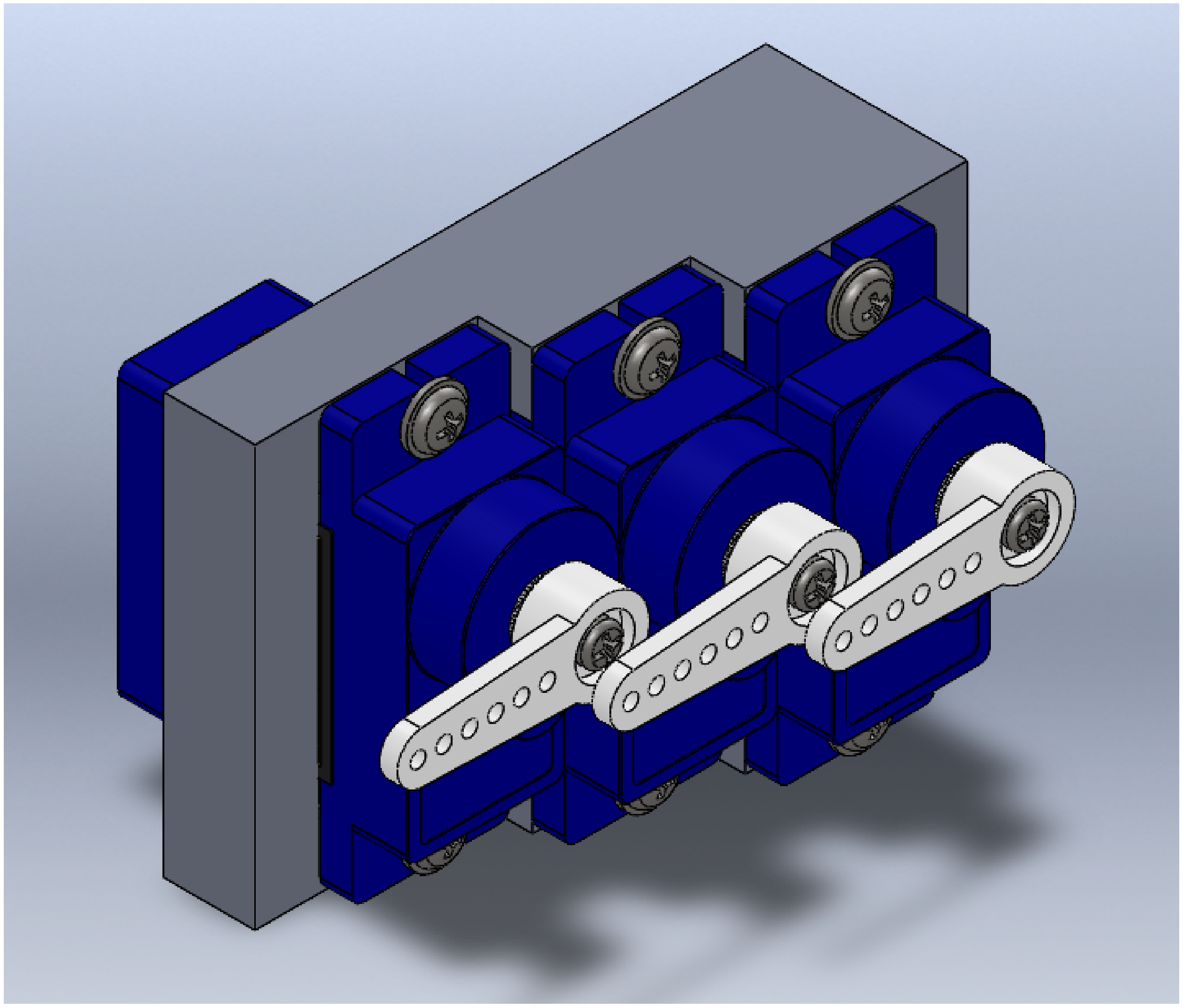


Figure 49: 3D Printed Servo Mount Prototype

The 3D-printed part is a simple rectangle, and the servos can be screwed into it. Next, we need to duplicate this for the other 3 servos, and make sure there is also a side-to-side stagger so that the 6 motors all hit different strings. The figure below demonstrates this.

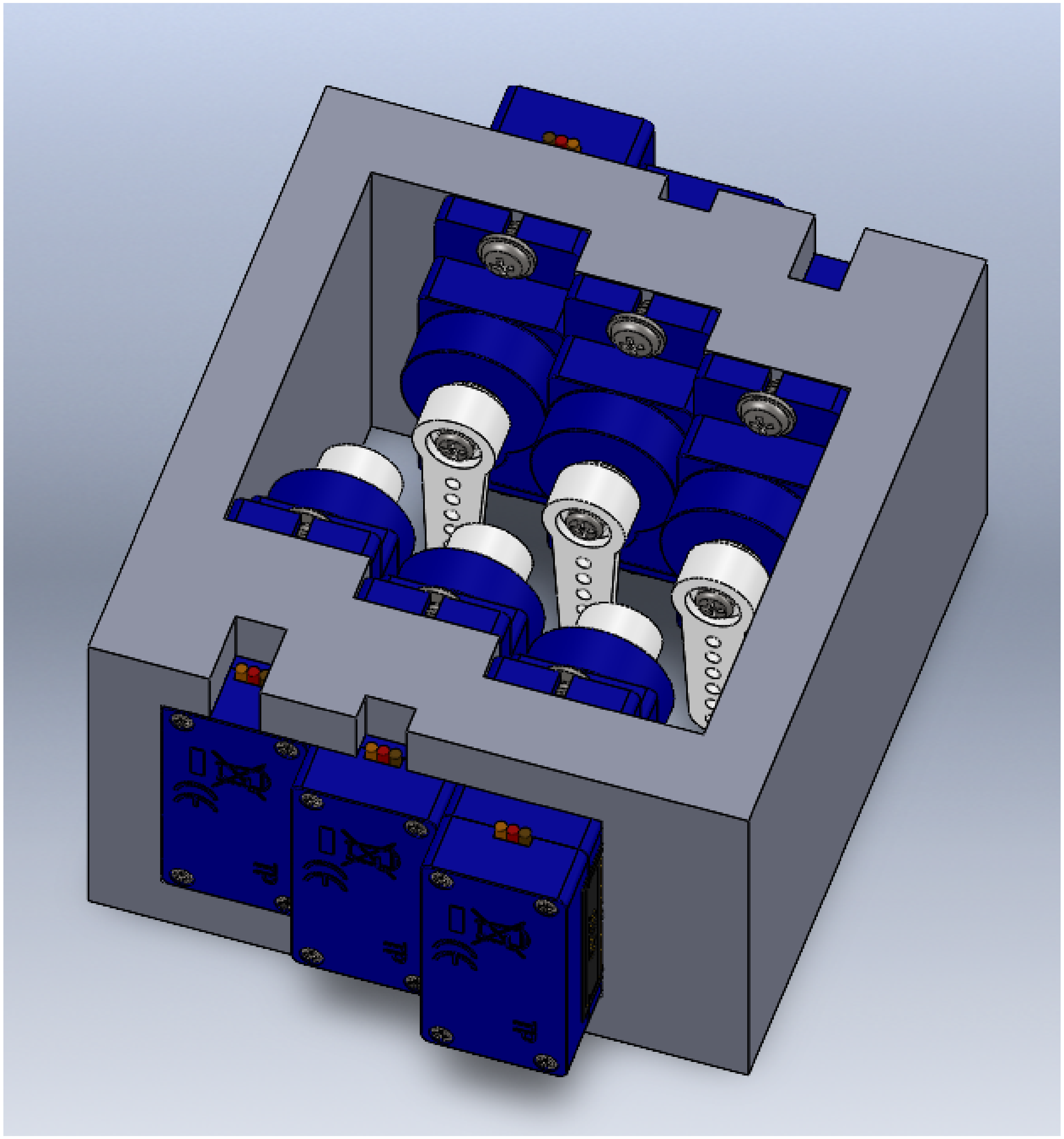
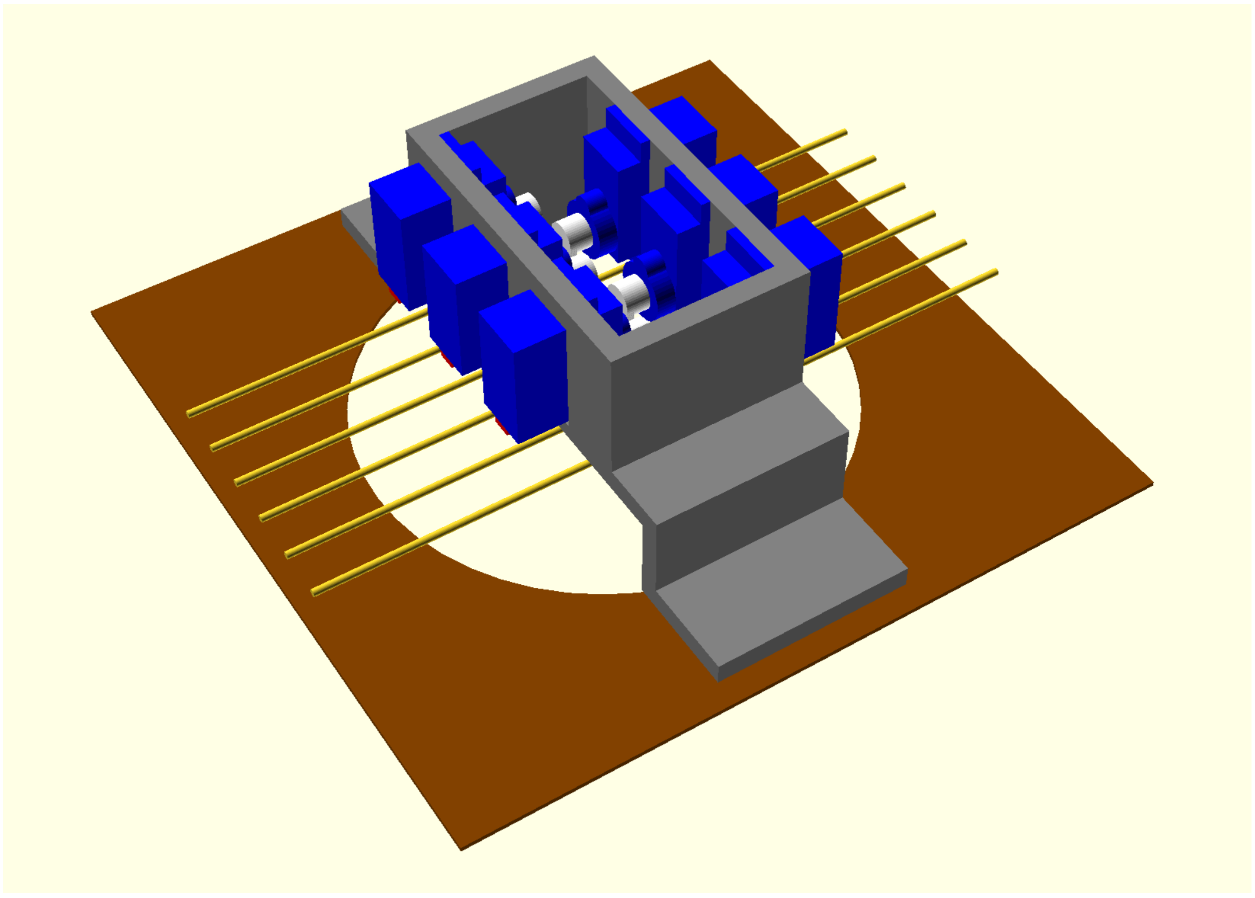


Figure 50: Strumming Servo Mount Prototype

Here we can see all 6 servos organized in a strumming formation. There also need to be cutouts for the power and signal wires that come from the servos.



Strumming Servo Mount Final Design

The final design for the strumming mount contains all six servos. For easier printing and assembly, this design is cut in two halves that are printed separately.

The following figure is the fretting assembly.

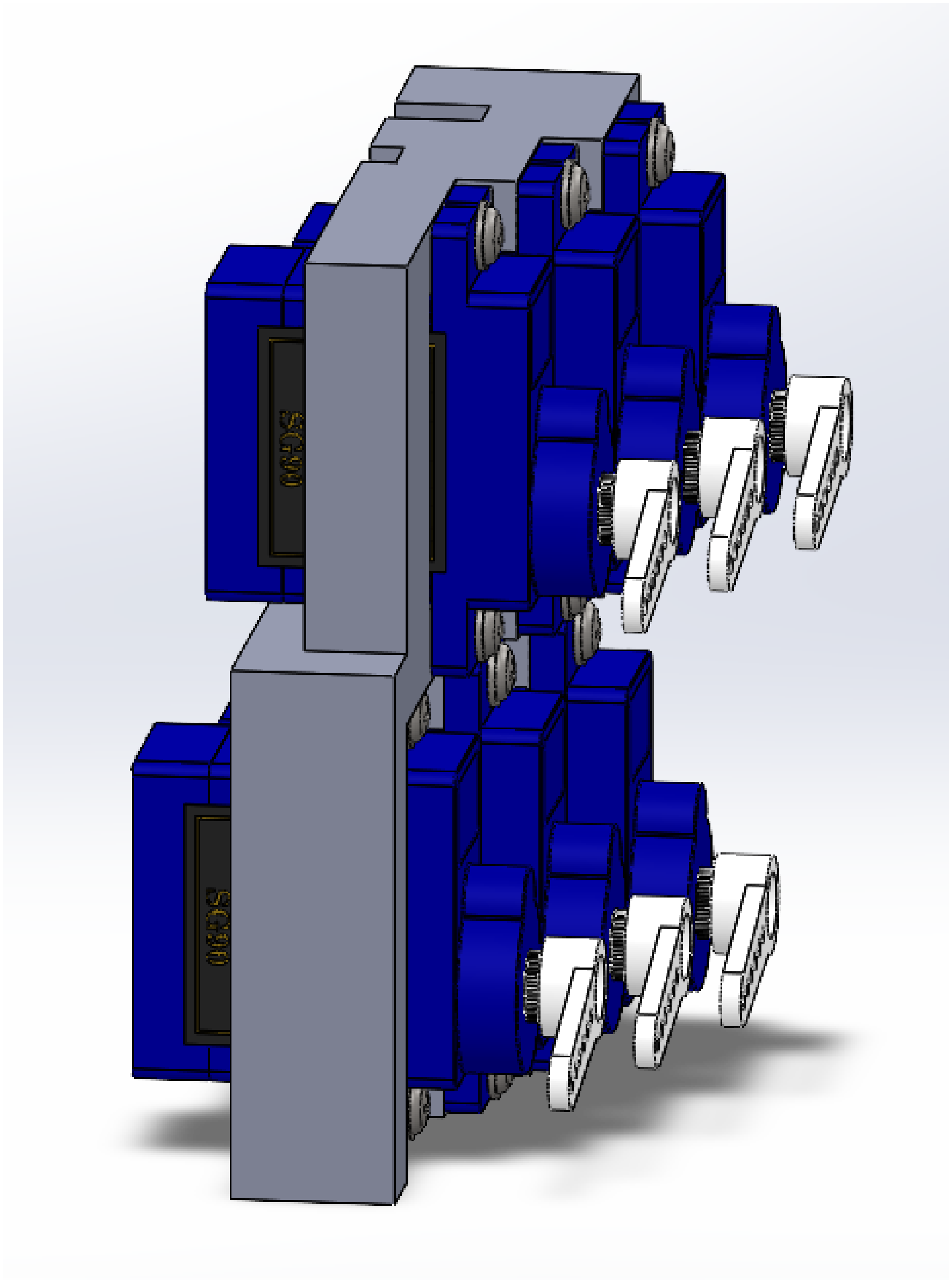
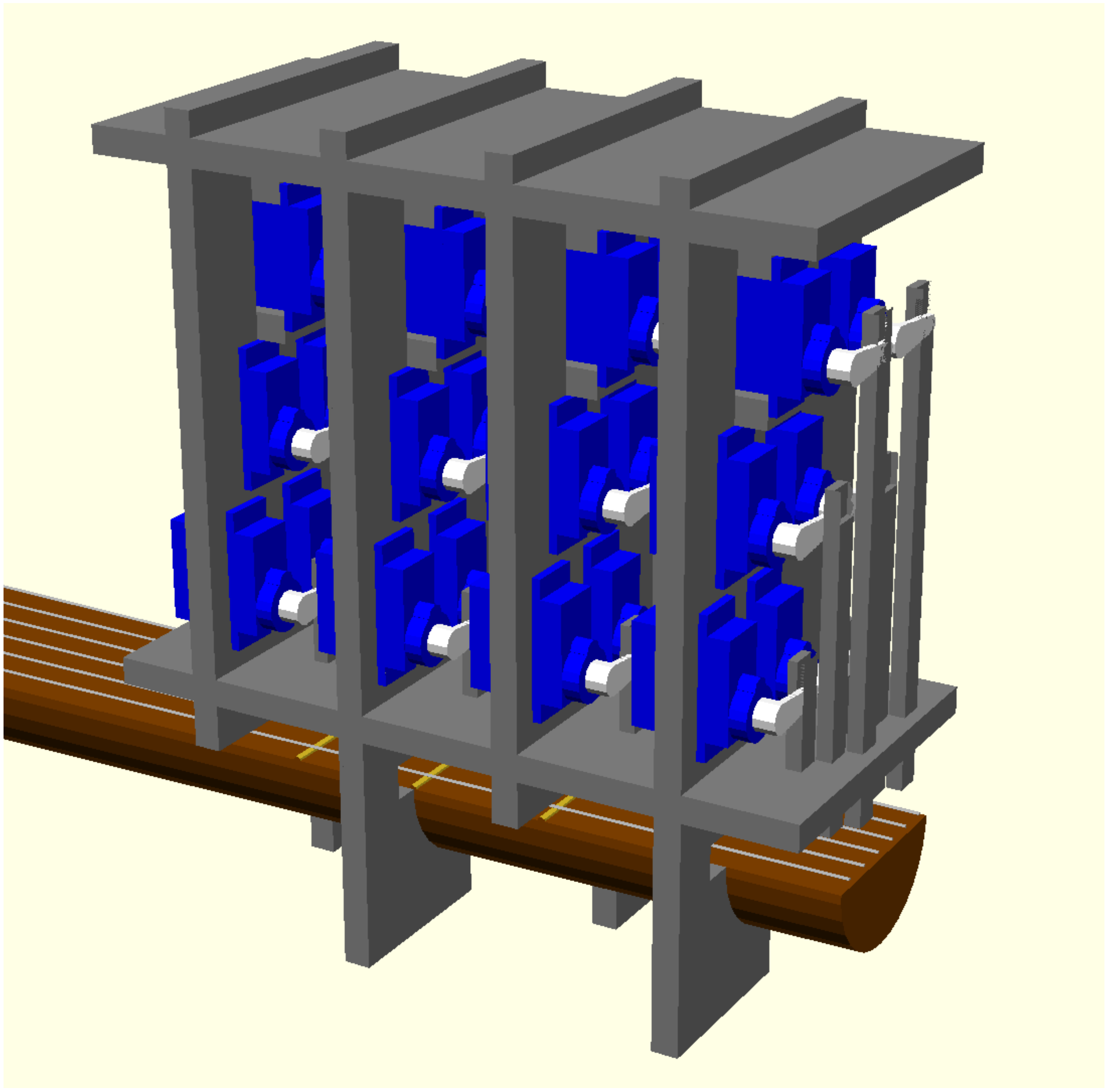


Figure 51: Fretting Servo Mount Prototype

It is two copies of figure 49 stacked on top of each other and staggered so that actuation arms may be placed vertically extending from the servo arms.



Fretting Servo Mount Final Design

The final fretting mount has all 24 servos, and uses slotted fitting to connect all the panels together. The stick designs and the guide holes have all been finalized. The design looks very different from the initial draft due to the fact that we pivoted to a laser cut design. This means the design has to be able to be translated into a 2D figure, and built out of flat panels. There are 4 vertical panels for the servos, one horizontal panel for the guide holes, and one horizontal top panel for added stability. There are also some standoff pieces to offset the top layer of servos by the needed amount so that none of the sticks interfere with the servo arms on the lower layers.

5.2. Software Design

The figure below demonstrates a visualization of how a MIDI file is interpreted by software. Each note is played in sequence as the sheet scrolls to the right.

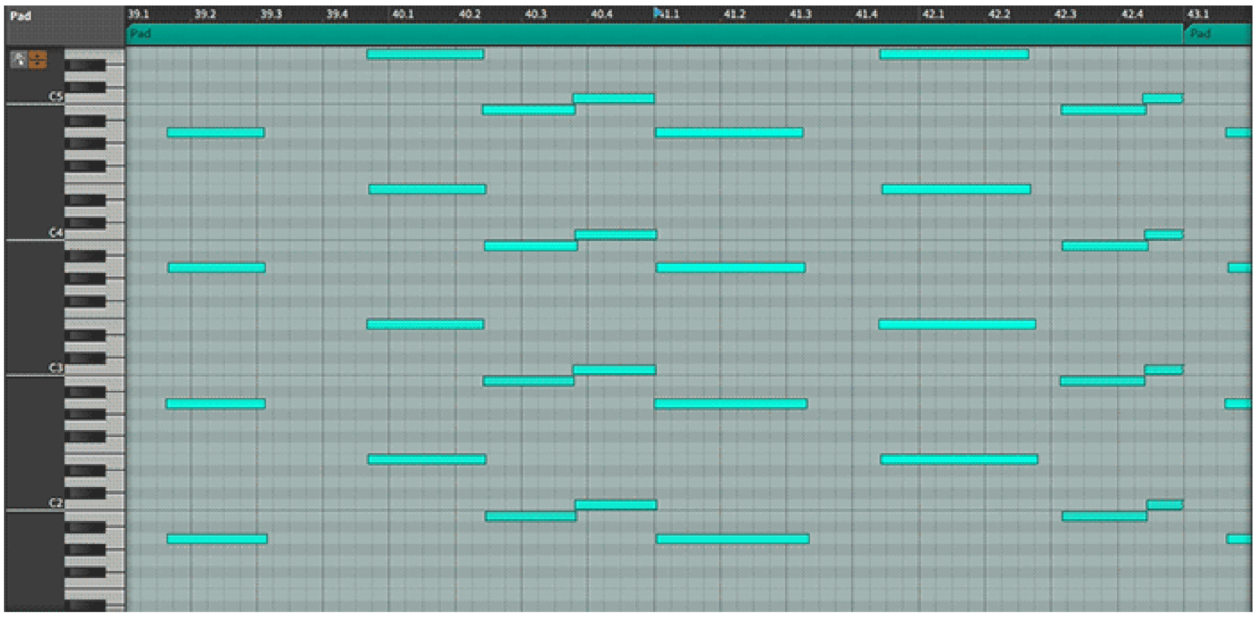


Figure 52: MIDI Representation Visualized
https://blog.landr.com/what-is-midi/

There will be three major components of the software side of this project. They will be as follows. First, a general purpose algorithm which will read and process MIDI file data (which is laid out like the timestamped piano notes shown above) and convert them into a playable stream of notes on a guitar's strings and fretboard. Second, there will be a user interface for uploading the file to the microcontroller via bluetooth as well as playing/pausing/stopping playback of the song on the actual guitar. Third, as a stretch goal, we would like to implement the UI on a cross-platform mobile application for even more portability and flexibility. **For the final project, we ended up using a web server so that the UI for controlling the guitar could be accessed via any device connected to the same wi-fi network (desktop OR mobile) from one single UI.**

5.2.1. Algorithm

We would like to be able to process *any* given MIDI file that satisfies our constraints (under 50 KB). In general, most freely available MIDI files online in databases contain a wide range of notes potentially covering the entire western scale (or a full piano's worth of notes) and potentially having several overlapping notes as well. As per our requirement specifications, our playable note range on the guitar is only from E2 to G#4 (approximately 2.33 octaves), which of course does not cover *every* possible note on the western music scale.

5.2.1.1. Range Compression

In order to resolve this, we will have the first pass of our algorithm first attempt to transpose (or shift) all of the notes in our song by some integer, x, representing the number of half-steps up (positive) or down (negative) to shift by which will **minimize** the number of notes in our song that are outside of our playable range. This step should come at little to no reduction in sound quality, especially to the untrained ear as most humans do not have perfect pitch (that is, as long as the relative differences between all the notes in the song remain constant, it should still sound virtually the same just at a lower pitch). If necessary, this step could technically be optimized by using a ternary search for the shift amount, however due to the small number of shifts, our current plan is to use a simple linear search for the shift amount, for ease of implementation without a severe effect on performance. So generally, the first pass of the algorithm will try to ensure that most notes outside of our playable range have been converted *into* our playable range.

Now, in the case that the particular song simply does not fit in our playable range, we will need a second pass of the algorithm to completely ensure that ***all*** notes outside of our playable range have been converted *into* our playable range by simply finding the closest matching note in an interval we can actually play, using the formula: $2^n * f$, where $f = frequency$ and $n \in Z$). For example, if after the first pass, we still have note C2 as in the figure below, that would be converted to C3 by doubling the pitch.

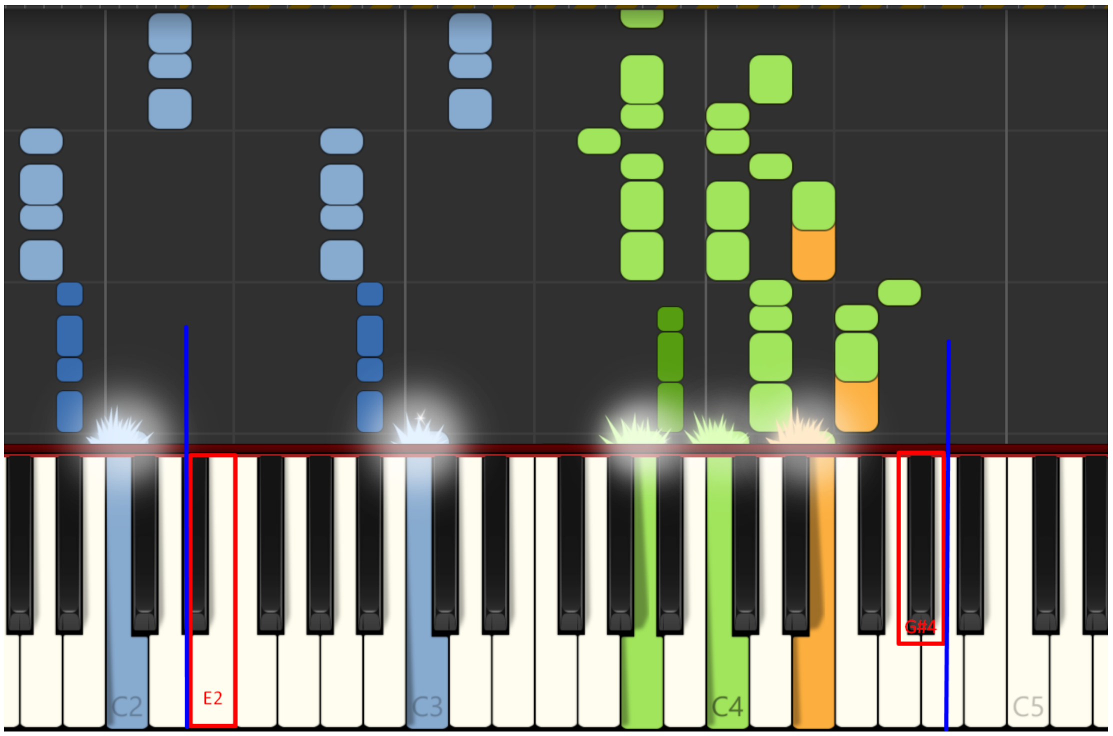


Figure 53: Playable Note Range Visualized on Piano https://synthesiagame.com/

Converting the notes this way may result in a minor difference in audio quality, however it is still a great approach based on physics and music theory to “fake” notes outside our range, keeping in mind that it would be impossible to achieve fully lossless compression behavior without expanding the hardware scope of our project.

5.2.1.2. Handling Self-Overlapping Notes

Then, a third pass can go through and handle all overlapping notes by setting the length of any note whose “end” time overlaps with another of the same note’s “start” times *so* that it ends before the next note starts. For example, the upcoming yellow notes on E4 and F4 in the figure above would be shortened so as to not overlap with the green notes right after. We predict no noticeable difference in sound quality. This is, again, simply a limitation of the hardware scope of our project using a physical acoustic guitar with a limited number of motors as opposed to digital/electronic pianos and keyboards which can overlap notes digitally or via clever use of “sustain” or other pedals.

5.2.1.3. MIDI to 6-string and fretting conversion

This will be the meat and potatoes of our algorithm. The procedures will be as follows:

1. Define function for playing a single note via motor outputs for some amount of time on the correct servos given the string, corresponding fret location, and timing for that note.
2. Define function for finding out what string and what fret location a given note can be found on.
    a. This can be done by looking it up in a precomputed Python dictionary (for speed reasons, since we will be calling this function many times per second).
3. We can accept a user input for the filename in console (or automatically grab the first MIDI file we find in our current directory if there is one), and parse its bytes using Mido https://mido.readthedocs.io/en/latest/parsing.html giving us the full message stream
4. Handle range compression and overlapping notes as in 5.2.1.1. And 5.2.1.2.
5. Iterate through the message stream sending signals to play the notes with added necessary delay to make sure they play at the right times

5.2.2. User Interface or UI

On a laptop computer, there will be a web app with a clickable “Play” button. There was a stretch goal to have a option to upload a MIDI file to the Microcontroller via the UI. We did not meet this stretch goal but it is technically possible if it is connected to a micropython program called webREPL but this cannot be integrated into our UI so we do not consider this stretch goal met. The UI also has a drop down menu that dynamically adds songs to the list as new midi files are added. It can do this by automatically regenerating the HTML file every time the front end is restarted.

The user also has to connect the ESP32 to a PC through the micro USB port to edit the boot file to add their SSID and WiFi password.

5.2.3. Cross-Platform Mobile Application

We had planned to create a cross-platform mobile application for the controller using Flutter. It would be functionally equivalent to the Web UI, while offering us the extra portability and flexibility of being able to control the guitar from any smartphone. However, we ultimately decided on prioritizing other aspects and simply used the browser on a mobile phone to access the UI's web server which worked flawlessly.

5.2.4. Development Environment

When looking at the code for the microcontroller and all the software that will be on the computer we will need tools to write the code, have version control between updates, and have a way to test the transmission from the computer to the microcontroller. The things that are important to using development environments are,

- An environment that will allow for compiling code that can then be flashed onto the microcontroller.
- An application that will allow for sending the information from the PC to the microcontroller.
- A method of version control that will allow every member of the team to have an up to date version of the project.
- A method for the team to share links to sources for the project.
- An application that can help us make diagrams

5.2.4.1. uPyCraft IDE

The code for the motor control and the serial communication will be written in MicroPython which is Python with some functionality removed. To write the code for the microcontroller in this version of Python we will need to use the uPyCraft IDE. It will allow us to write this and allow us to connect the ESP32 to the computer to flash the MicroPython code to the microcontroller.

5.2.4.2. PuTTy

Since we will need to transmit files from the PC to the Autonomous Guitar it is important that we have a way that the PC can transmit the files, we can use PuTTy to transmit to the microcontroller since PuTTy supports FTP and port forwarding. We can use this to test sending the MIDI file to the microcontroller. If we know the Buad rate and Buad rates are standardized so the Baud rate that we will be using is 9600 which is what is usually used by UART, and the port we can then connect to the microcontroller and then upload the file. We can also test the bluetooth connection with PuTTy.

5.2.4.3. GitHub

One thing that is important for the development of the software is version control and making sure that all the team members have the files up to date and we can revert to previous versions of the software in case something goes wrong for this we will use GitHub to keep all changes consistent. The code for the microcontroller, the UI, and the algorithm will be available on GitHub, along with the schematics, the PCB layout and the 3D image files for the 3D printer.

5.2.4.4. Discord

For the team to share information on the project we are using discord to communicate with each other, discord allows for creating a server where information can be shared and a way for the team to meet when we cannot meet in person to discuss the project, it is also another way for the team to share files like github.

5.2.4.5. Diagrams.net

The service that we are using to make all the diagrams that we need is diagrams.net which is a website that lets you make diagrams by clicking on the desired object to place inside the diagram and write text boxes inside the diagram to represent code or hardware components this is very helpful in the development process since it gives the team a reference for what certain code or architectures are supposed to do before the team begins to prototype and write the code or put together the hardware.

5.2.4.6. Thonny

The main IDE used for MicroPython development on the ESP32 microcontroller during the final project was Thonny, due to its simple interface, ease of accessibility, microcontroller file browser, firmware upload capabilities, and prior group member's experience of working with it.

5.2.5. Microcontroller Code

When approaching the code for the microcontroller we consider the code for the algorithm to be separate and just a call within the microcontroller code. The microcontroller code will account for the serial communication, the LCD functionality and the motor control operations, which will include proper timing with timers. In the Figure below is the microcontroller block diagram.

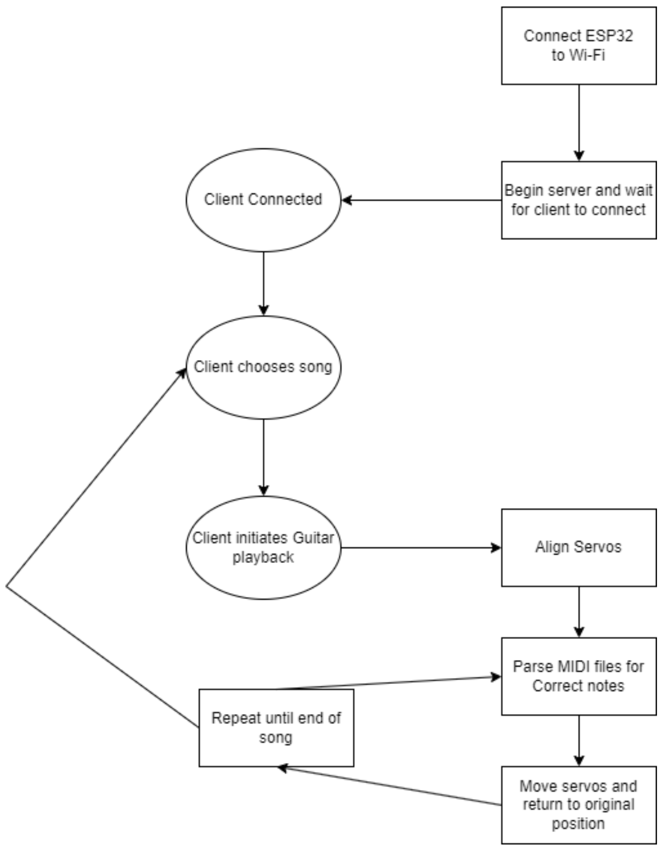

Connect ESP32 to Wi-Fi
Begin server and wait for client to connect
Client Connected
Client chooses song
Client initiates Guitar playback
Align Servos
Parse MIDI files for Correct notes
Repeat until end of song
Move servos and return to original position

Figure 54: Microcontroller Code Block Diagram

Not included in the diagram are the interrupts that would move the servos since there would be too many of them to include in the diagram. In the main function the microcontroller should start the clock to be ready for interrupts to be triggered, it should also turn the LCD screen on and get the values from the function calls to send them to other modules in the autonomous guitar.

In the Communication function the microcontroller will use the bluetooth functionality to receive the file if the microcontroller is paired with an external computer once it receives the file it will store the file on its own storage, if the storage is full it will display that to the user on the LCD screen. When the bluetooth transmission happens it will send the title of the MIDI file as well; this is so the title can be displayed on the LCD screen once the song is selected to be played. The Communication function will also control taking controls from the external device this will let the user change their song choice depending on what songs are loaded onto the device, this will mean that when the user starts the autonomous guitar after turning it off, the microcontroller will have to transmit the title of the MIDI files it has on it back to the computer so the computer will know what songs are available to be played without uploading a new file. The MIDI files can be stored in arrays along with the titles and deleted from the array if the user decides to delete the file.

The LCD will have a series of messages that can be displayed depending on what state the guitar is currently in; these messages are shown in the table below.

State	Message
Waiting for bluetooth connection	“Ready Pair”
Connection established (No Song)	“Waiting Input”
Song Selected	“<Song title>” + timer
Storage low	“Low Storage”
Tuning Mode	“Tuning Mode” + string no.

Table 14: LCD Screen States

These are the possible messages that will show in the LCD screen when nothing is connected to the autonomous guitar it will display that it is ready to pair with a computer, when this connection is successful it will display that it is waiting for input if no songs are currently on the microcontroller, if the songs are on the microcontroller the song titles will be transmitted back to the computer to be displayed on the user interface on the computer and will display the first song in the list on the LCD screen. When there is a song selected it will have a timer to show how long the current song has been playing for if the song has not begun to play and it is waiting for the user to play the song the timer will display 0:00, it will be assumed that the user will not play a song longer than 9 minutes and 59 seconds, since our requirement is only 3 minutes, but assuming a user does play a song that is 10 minutes or longer the timer will revert to 0:00, and if the user pauses the guitar the timer should pause as well. We do not expect the storage space being low to be a common problem since the MIDI files are so small but it can potentially happen so it would be better to account for it to possibly happen. Although the LCD screen did not make it into the final design this will be left in the document as it is still a stretch goal and something that could be added to extend the project.

When the MIDI file gets sent to the algorithm it will be output as a string containing the note that will be sent to the servo control. Here it will make decisions for which servos to activate to play the note it will then move the correct servos to play that note the servos will move in a 20 degree angle across the string over the soundhole to play the note and press down on the fret. We will control the servos with timer interrupts when we know from the midi file how long should be between each note played for the song, once the decision is made for the servo it will go to the interrupt for that servo.

5.2.5.1. Micropython and C integration

One Potential issue with writing everything in micropython is that python is very slow, one potential solution to this problem is to integrate C into the code for the microcontroller this will allow for serial communication and motor control to be done in C while the code for the algorithm will be done with Python and this will make the program run faster.

There is a way to make an external C module run inside of a Micropython program that is on the microcontroller. The ESP32 does support micropython development but C and C++ are the recommended programming languages generally for microcontroller applications due to the speed benefit that having to manage the memory gives you. For the algorithm though this speed benefit will not be necessary since in the worst case the autonomous guitar can just run the entire algorithm on the MIDI file before ever starting the motor control. The benefit of creating the algorithm in micropython instead of in C with all the other microcontroller functionality is that it will be more readable and less complex to make the conversion algorithm in micropython so using C and micropython together on the microcontroller will let us get the added benefit of speed from programming the serial communication and motor control in C where it will be needed and the reduced complexity for the algorithm from python.

According to documentation on the micropython website the C program must be called inside the micropython program for it to work, and recommends calling any libraries used through python rather than in the C code but if using a library that is not available in python it can be called in C. On the computer in which the development for the microcontroller code is taking place the C module needs to be compiled separately before the code is sent to the microcontroller. The following figure demonstrates how micropython will be utilized for our code.

In the final project, we ended up not having to port or integrate the code with C/C++ or Arduino since with algorithm and hardware optimizations, MicroPython proved to have all the capabilities that we required.

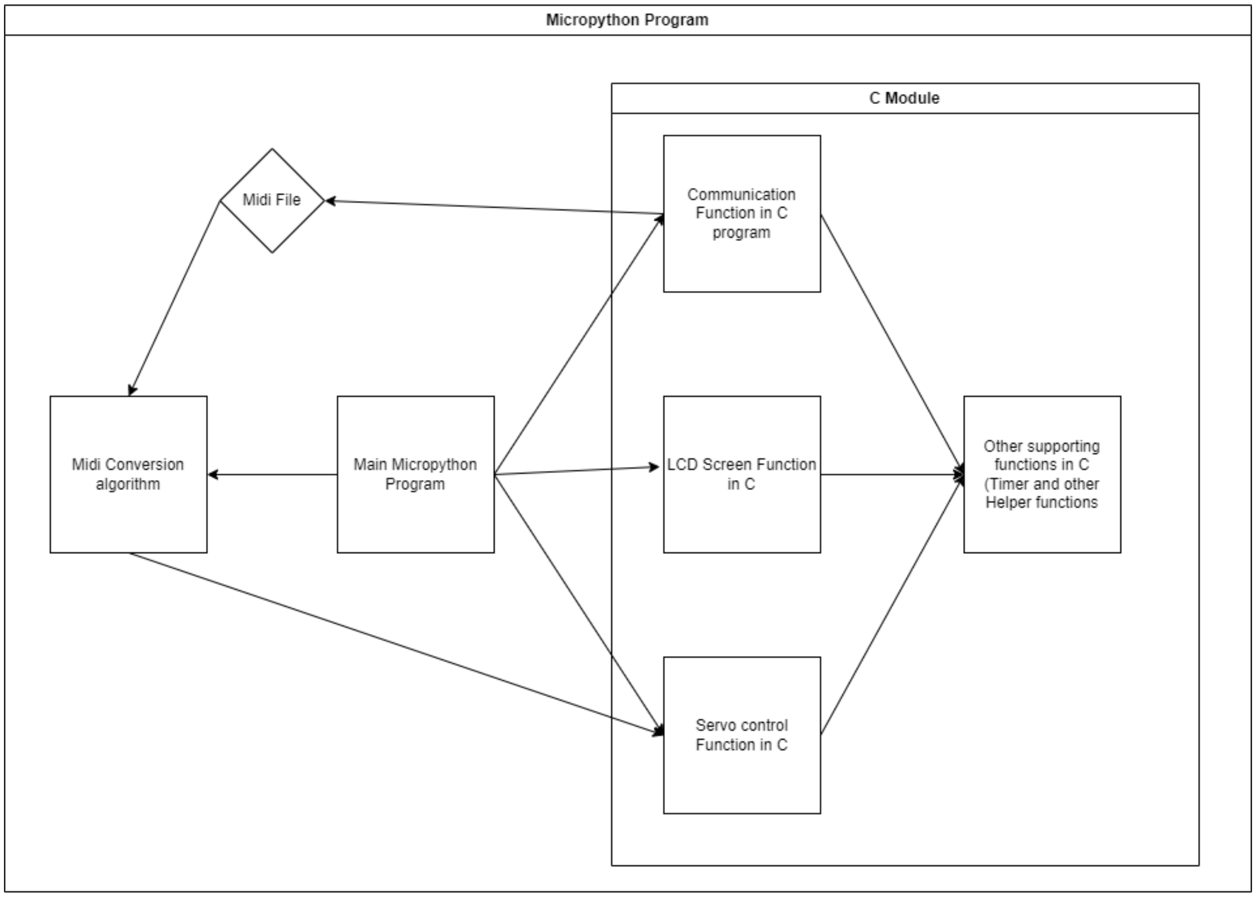

Micropython Program
C Module
Midi File
Communication Function in C program
Midi Conversion algorithm
Main Micropython Program
LCD Screen Function in C
Other supporting functions in C (Timer and other Helper functions
Servo control Function in C

Figure 54.5: C integration into micropython

The figure shows the general flow of how micropython and C will be combined in the autonomous guitar project, everything will be inside the micropython program but each main C function will be called individually even though each C function can call other C functions as needed.

The inputs and outputs of the C module are as follows,

- Micropython calls C function to get the midi file from the PC, The C Function outputs the MIDI file to the micropython algorithm
- Once converted the conversion algorithm sends its output to the servo control inside the C module.

We ended up not using this approach because it was not necessary to make the guitar functional and would have added a lot of unnecessary complexity to the project.

5.2.5.2. Microcontroller libraries

When writing code for the microcontroller we will need libraries to simplify writing the code for the autonomous guitar project we will need libraries for the ESP32 and libraries to operate the servo motors and bluetooth module. The Standard ESP32 library will have pin outs for easily activating pins on the microcontroller to be used, The module for

Servos, Bluetooth and the LCD will contain functions that can will make it readable once the functions are implemented instead of having to manually manipulate the bits.

The autonomous guitar project will be using both C and python libraries since we need the speed advantage given to us by using C. It can be faster and many of the same libraries for the ESP32 will exist in both since the ESP32 has micropython support. The micropython documentation states that unless necessary due to not being available as a python library that all libraries should be implemented in the python side if wrapping a C module inside of the program. If we use Arduino libraries the ESP32 will require libraries that are not in use by regular arduino boards since the ESP32 is not an official arduino so the libraries would have to be ESP32 specific.

Since the main part of the microcontroller program will be in micropython the first attempt at the programming will use micropython libraries as much as possible and as little C libraries as possible outside of standard libraries. The main libraries micropython libraries and modules we plan on using for the autonomous guitar project are the following.

- From the machine module the Pin class will be imported this will make it easy to control the GPIO pins and add names and functions used for controlling the pins
- The esp32 module will be used to control things that are related to the clock on the esp32 and it can be used to set low power modes
- The servo functionality is contained in the pyb module so we will need to implement this module. With this library we can control the angle and speed at which the servo turns, it can also be used to set up the turn angle with PWM.
- Bluetooth low energy does not have a specific module made for micropython but open source apis can be found to implement the functionality.
- For the LCD there is also no official micropython library and either a C header must be used inside of the C module or using an open source library for a 1602 LCD screen.

The libraries for esp32 give us access to the low power modes which is a good way to have the guitar wait for input until it gets input from the PC or the smartphone application. It also gives us access to the functions to control the non-volatile storage which is something that we will need to hold multiple songs on the microcontroller at once.

The pyb library has the controls for the servos for micropython, the class for servos inside the pyb library, that allow for the angle and speed of the servo rotation to be altered, it is also possible to control PWM using the timer function with this class.

5.2.5.2.1. UMidiParser Library https://github.com/bixb922/umidiparser

We ended up using the “UMidiParser” library for the actual timing/playing MicroPython logic of processing MIDI files on the ESP32 microcontroller in the final project. This was necessary as the previous library “Mido” which we used to edit the MIDI files programmatically for preprocessing was only available in Python (running on a computer, not a microcontroller). Our initial attempts to implement ported versions of Mido on the ESP32 were unsuccessful so we found this Github library instead which met all of our requirements.

5.2.5.2.2. PCA9685 Library https://github.com/adafruit/micropython-adafruit-pca9685

We used Adafruit’s PCA9685 interface library to control the servo shields in the final project.

5.2.5.2.3. Servo Library https://github.com/adafruit/micropython-adafruit-pca9685

This is an intermediary library we needed to import from Adafruit’s PCA9685 interface library, which handles the PWM (pulse-width modulation) of servo motors.

6. Prototype Construction & Coding

6.1. Hardware

With prototyping our project there are several steps that need to be taken. Ideally all individual components will be constructed individually and from there we can assemble and create tests in order to ensure all pieces are working adequately, in constructing the prototype there are a couple routes we need to explore.

6.1.1. Linear servo motor

The servo motor is good for strumming and rotational movements for our purposes we need to take advantage of linear movements, there are options out there of parts that can be 3D printed to use linear motion with a servo motor with that being said is important that are pieces fit so it is unlikely we will have a design for this component until we decide which servos to get and then from there we can work on designing the linear component. As of now it is unclear if this is a file we will be able to find available for free use or something we will be required to design. If we must design it will be useful to create several versions in order to determine which design works and does not.

6.1.2. 3D Designing the Prototype

A good way to test without having it built yet is to use 3D models. We can create a 3D model of what layout we want to use for everything by modeling the guitar and pieces in some CAD software. Once we have all the pieces we will be using we will be able to measure precisely and determine where everything fits together. This is a simple way to try and create the conditions we will be using physically using the guitar and motors. That way we can have an ideal solution without risking any damage to any of the electrical components.

6.1.3. Physical Prototyping

As using 3D models may be a good way to go about prototyping our model it may have some limitations which can only be accessed by physically moving the pieces to see what we are actually working with. With the amount of precision we may need in this example it could lead to us having some sort of accuracy issues with the 3D model which will lead to potential inaccuracies in the real life prototype. As we may try and 3D model the project before assembling there may be issues which can only be solved by manually altering pieces without CAD.

6.1.3.1. Mounting System

With our project working on playing a guitar totally independently it is important we have control over the individual notes that we press down, one of the biggest problems we have with this is the physical space that we have to work with. Our goal is to work with micro servo motors, these were the smallest version of the servo motors we were able to find. One issue we have is that in order to place the servos on each fret they will need to be close together but at that point they do not fit next to each other the higher up the neck of the guitar they go. One solution we have is to create some sort of mounting system that the servos can connect to, we would have to stagger them in order for them to fit and this would require us to 3D print longer levers so we can stack them higher. We will be prototyping different size arms for this to see if there is a loss of force or response time doing it in this way. Another possible solution would be to see if there were alternate locations for the same note so we could space them out slightly more. However even in this situation it would require a mounting bracket that would have to be above the neck of the guitar. While prototyping we need to ensure that we can make this as lightweight and small as possible without any interference on the guitar, it is possible that the mounting system will take multiple designs and it is unclear what type of material it will be made of.

6.2. Electronics

Prototyping the electronics will involve wiring up all the components on breadboards and running the code to see if everything functions as expected. This will require that the code is at least somewhat functioning, but the hardware mounting of the motors need not be complete. We can run the code on a couple sample songs and record the movement of the motors, which should be arranged similarly to how they will be placed on the guitar. Then, we can record the movement of the motors and slow down the video, visually confirming that the right notes are being played at the right time.

6.2.1. Power Supply

When planning our schematic we know the voltage we need for the circuit but the current is where we have doubts on our power supply. In order to make sure our power supply is supplying enough current we can adjust the current that is outputted on the power supply in the lab. If the power supply we have selected is not pushing enough current we can design a current gain circuit that can amplify the current and then we can power the servo motors with adequate current. Below is a picture of an example power supply that is commonly used for testing. We can expect that this power supply will be similar if not of the same make as those in the lab. Given that we have considerable experience with operating in labs due to our class schedules, the device will be familiar to us.

KEITHLEY
2231A-30-3 TRIPLE CHANNEL DC POWER SUPPLY
30.00V 12.00V 3.29V
CV 0.485A CV 1.426A CV 2.897A

Figure 55: Example Power Supply Used for Testing

6.2.2. Circuit Prototyping

In order to prototype our project, we needed to have our circuitry up and running before the final design takes shape. However, the monetary cost of PCB production combined with the time investment in getting the board means that creating PCBs is no small investment and impractical for our prototype. Rather than having the PCBs made, we instead elected to create the circuits using breadboards. Breadboarding allowed us to rapidly prototype our circuits as well as changing them as needed, saving us a considerable amount of time. On top of this, because of the availability of breadboards in both the senior design lab and other labs across UCF, the cost of prototyping becomes essentially free. This, however, does not account for the cost of the parts needed to create the circuit.

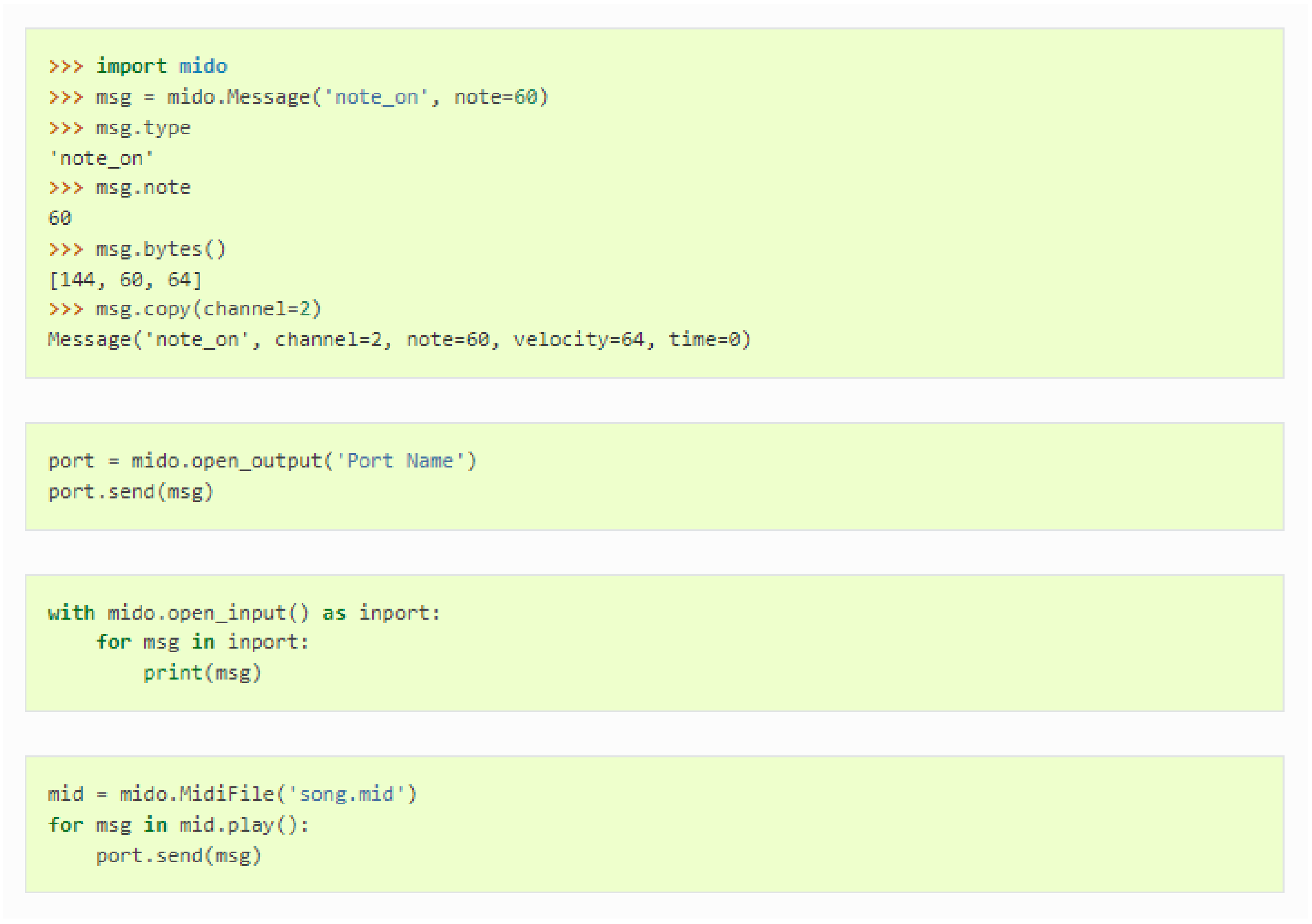
Because the parts we have considered are surface mount, replicating the circuitry will be difficult on a breadboard. Specifically, the regulator chips are surface mount with very small form factors which could lead to considerable difficulty in wiring up the parts to a breadboard. Various methods can be employed to circumvent this, such as using header pins and soldering them to the pins so that they can be mounted to the breadboard. Ultimately however, we decided to order the through-hole regulator package for the LM2576 for our middle-term demo. Later, for our circuit implementation, we ordered the SMT packages.

6.3. Software

This section goes over the software that we will use to realize our project.

6.3.1. Algorithm Prototype and Construction

The following figure is a prototype for the algorithm that we're going to use to convert our midi files. It utilizes the MIDO library for Python which is a popular library for interpreting midi files using Python.



```
>>> import mido
>>> msg = mido.Message('note_on', note=60)
>>> msg.type
'note_on'
>>> msg.note
60
>>> msg.bytes()
[144, 60, 64]
>>> msg.copy(channel=2)
Message('note_on', channel=2, note=60, velocity=64, time=0)
```

```
port = mido.open_output('Port Name')
port.send(msg)
```

```
with mido.open_input() as inport:
    for msg in inport:
        print(msg)
```

```
mid = mido.MidiFile('song.mid')
for msg in mid.play():
    port.send(msg)
```

Figure 56: MIDI file processing in Python using MIDO library

6.3.2. User Interface Prototype and Construction

The following two figures are an example of a UI interface we will utilize for prototyping the project.

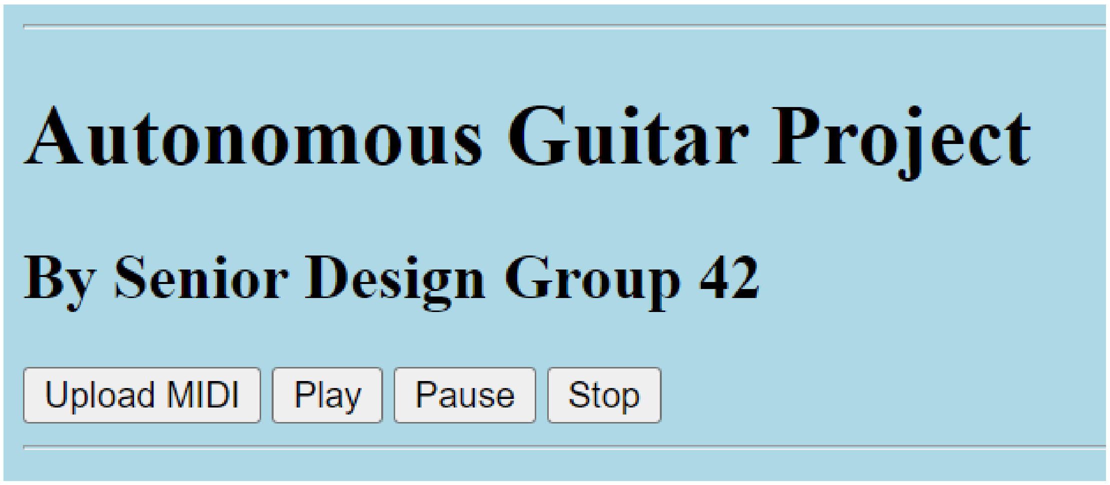


Figure 57: Simple Initial UI Prototype for Controller

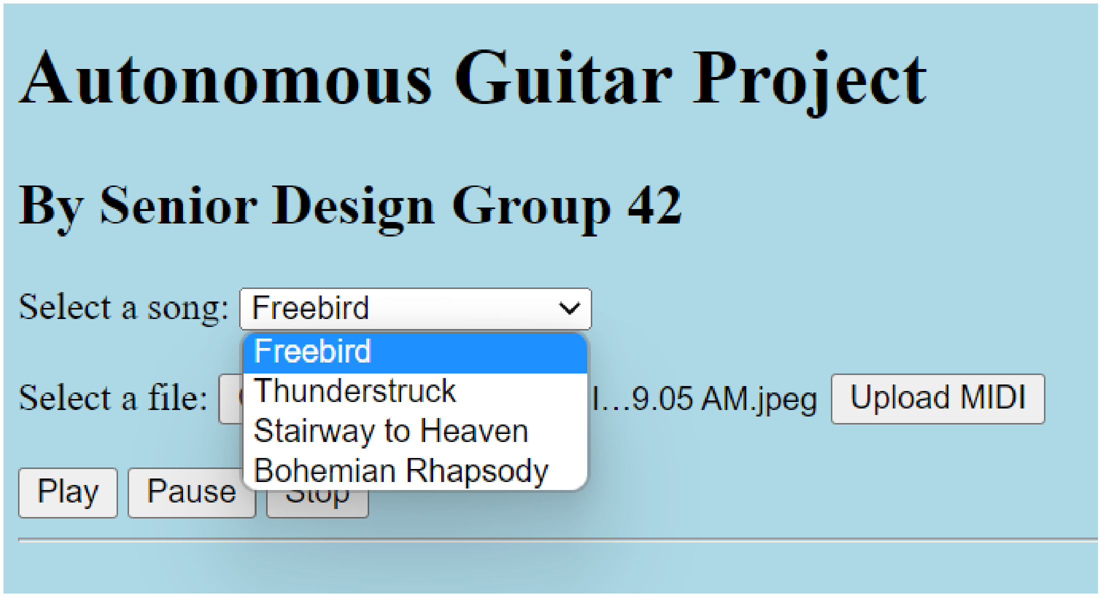


Figure 58: Simple Song Selector Prototype for Controller

Figures 57 and 58 show our example UI made in html with the provided song drop-down selector, file selector, MIDI upload button, as well as play, pause, and stop buttons.

6.3.3. Mobile App Prototype and Construction

The figure below provides a possible alternate configuration for our prototype build, utilizing a mobile app instead of a desktop.

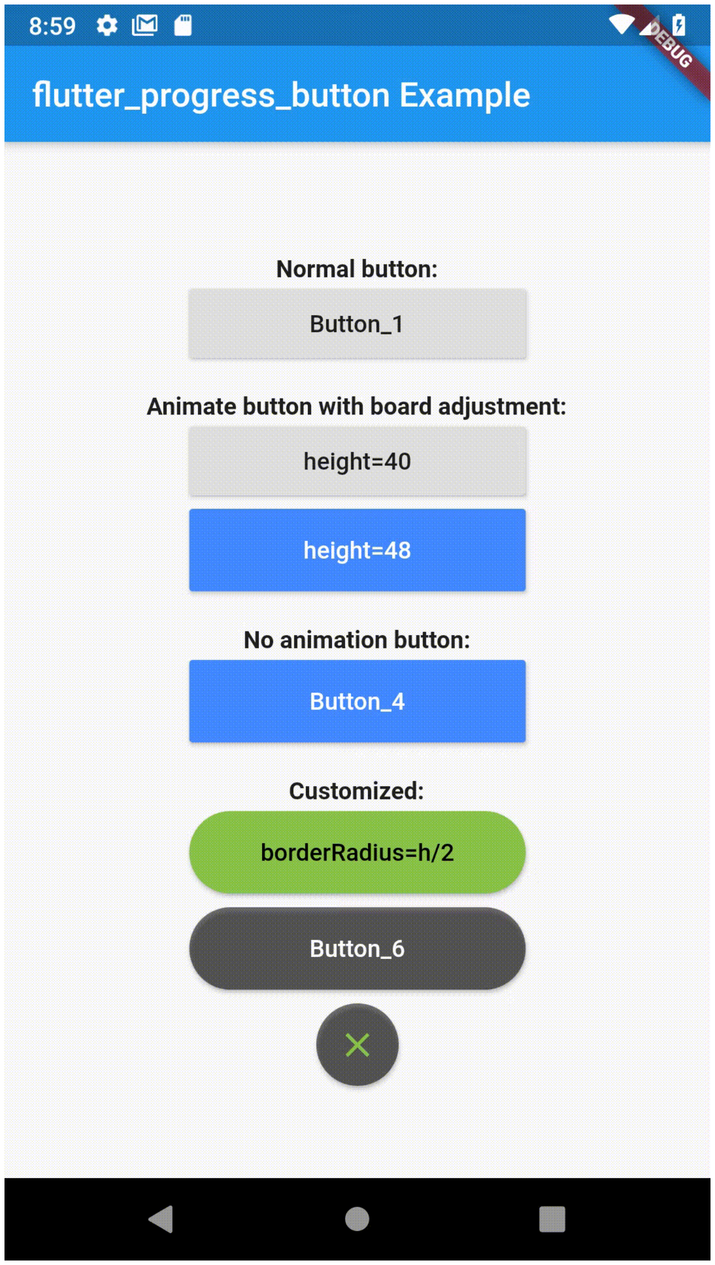


Figure 59: Cross-Platform Mobile App Example using Flutter
https://flutterawesome.com/a-easy-to-use-and-customizable-material-flutter-button/

The app did not make it into the final design but the Server is usable from any mobile device with a web browser since it is mostly just plain html.

6.4. Integrated Prototyping

Once a few individual systems are confirmed to be working, we can construct larger prototypes that span multiple software and hardware components. These prototypes serve the purpose of confirming that the interaction between stages can function correctly. Issues we could run into include low data transmission rate or quality, or lack of compatibility. A few of these prototypes can be built during Senior Design 1, as they are relatively simple and require only a few common parts to get working. More complicated prototypes will have to wait until Senior Design 2, when we've ordered more parts for the project and have more time to focus on building the project instead of writing documentation.

6.4.1. Controlling Servos with Micropython, ESP32, Shift Register, and Servo Shield

This prototype is our first step towards integrating our hardware and software technologies into a single cohesive system. The idea is to get the microcontroller operational with the programming language that we have chosen for this project. Additionally, we will interface it with one of the shift register ICs and a few of the servo motors that we have on hand from Arduino kits we've purchased in the past. The most challenging part of the project for the computer engineers is getting this interaction to work. We should be able to individually control an arbitrary number of servo motors with only 3 pins from the microcontroller. This prototype will confirm that this is feasible.

The shift register design is functional but it has issues which made it not able to be used in the final design. The issue was that since all the shift registers come off the same pin and all the servos connect to that they are all using the same PWM pin which means that they will all have the same turning angle. This is not optimal since each servo will need to rotate a different distance to reach the neck of the guitar and might try to rotate past the point that it can rotate which can lead to the servos stalling which can mess up the song make the playback slower and cause the current draw to rise while it is stalling.

The solution to this was to abandon the shift register design and switch to using the Servo shield, since these support 16 servos per shield and each have PWM pins this makes it a much better option since we can control the angle of each individual servo. We ended up using 4 of the servo shields so that we can distribute the power better.

The shift register design could be made to work if time was not a constraint, the code would have needed to be completely rewritten in C++ to make this design work and we would need to have a much more complex control algorithm.

6.4.1.1. Micropython Code

There exist libraries for shift registers and servos, but the specific combination we are implementing is complex enough that it's worth writing our own code. We want the shift register to output 8 customizable PWM signals to individually control 8 servos. This is achieved in code by splitting the PWM into 3 stages: 1. All signals are high, 2. Only active servo signals are high, 3. All signals are low. At the beginning of each stage we write the proper values to the shift register one at a time until they're all loaded, then forward those signals to the shift register output. Between each stage we use a tuned time.sleep command to get the proper delays. Ideally, we could calculate these time.sleep delays based on the desired input signals to the servo motors and the angles we want them to move to. However, The python code itself has a significant delay, so the time.sleep delays need to be shorter than would be expected in order to get the proper operation.

6.4.1.2. ESP32 Programming

The ESP32 uses a unique UART communication protocol that requires the CP210x USB to UART Bridge VCP Driver to be installed on a personal computer. Additionally, we need to flash the Micropython runtime onto the microcontroller so that it can interpret the code we upload to it. This is done using the uPyCraft IDE. Once all this is done, our Micropython code can operate the ESP32 digital output pins. This can be easily confirmed by writing values to pin 2, which is linked to the on-board LED.

6.4.1.3. Breadboard Wiring

The ESP32 can be plugged into a standard breadboard alongside the shift register IC. We connect the Vcc, Ground, SER, RCLK, and SRCLK pins from the ESP32 to the shift register. Each servo is connected to Vcc, Ground, and one of the outputs of the shift register. This wiring scheme, as shown below, will be similar to the final one we use for the project, just with fewer shift registers and motors.

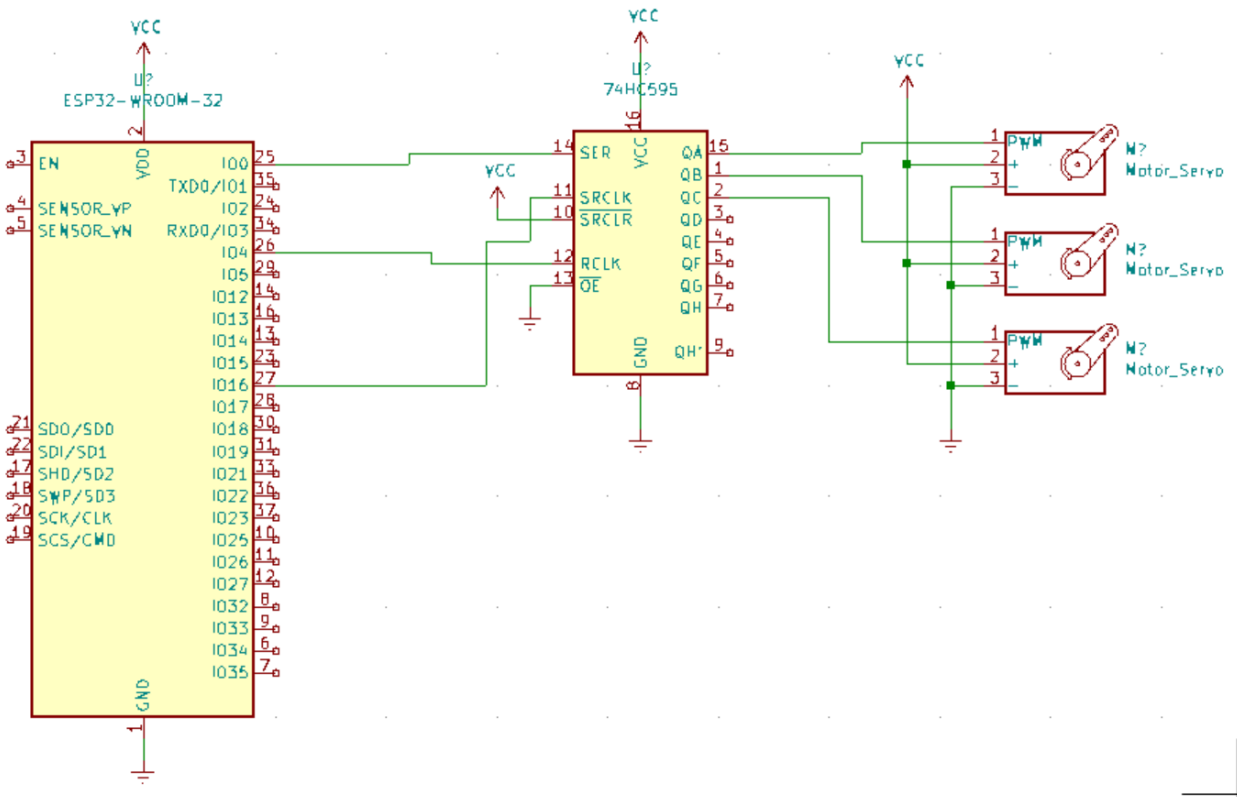


Figure 60: Breadboard Wiring Diagram

6.4.1.4. Prototype Takeaways

From completing this prototype, we were able to verify the ability of the ESP32 to control up to 8 servos simultaneously using a shift register and Micropython code. This confirms many of the assumptions we made about our initial proposed project architecture. The only problem we ran into was the slowness of the Micropython runtime. For example, optimizing the code to do one less modulus operation was enough to speed it up by 100 µs per operation, which suggests it can only do 10,000 of these operations every second, which is much less than the 500,000 we assumed in our initial time complexity analysis. Future benchmarking should be done to exactly quantify the issue.

We also found that the shift registers were not the optimal solution to controlling the servos, since we cannot give each individual servo its own turning angle since they do not have their own PWM signal, however using the servo shields fixed this issue.

6.4.2. Controlling Servos with Arduino C, ESP32, and Shift Register

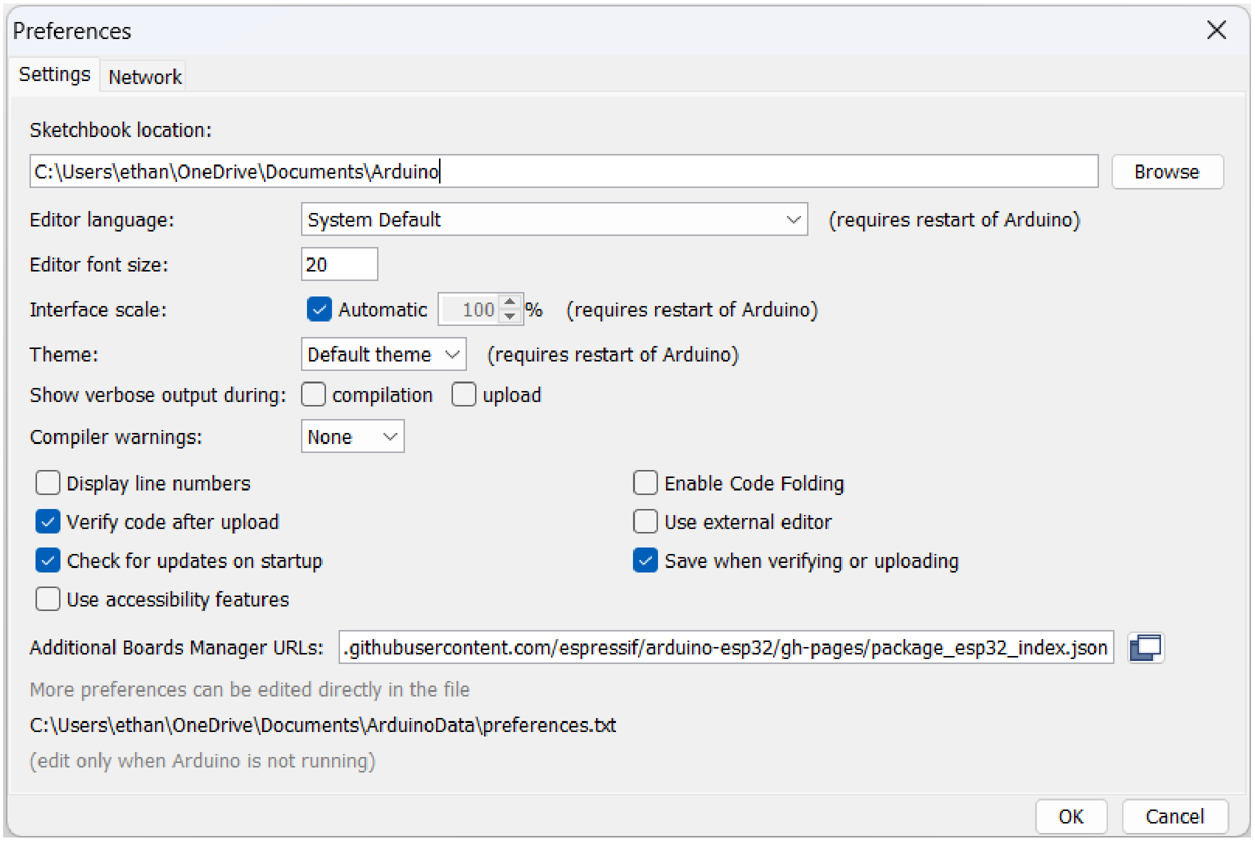
This prototype is similar to the previous one, with a focus on rewriting the code in C to investigate the effect on speed. The prototype above was only tested with 3 servo motors, as that is all we have on hand. When controlling 30 servo motors at once, the added slowdown could possibly cause the MicroPython code to no longer function. As such, this prototype exists as a backup in case we are unable to get enough speed from the MicroPython code.

6.4.2.1. Arduino C Code

This prototype is coded in Arduino C in the Arduino IDE. The code is exactly the same as the code for the previous prototype, using the equivalent function calls wherever possible. This allows us to easily compare the performance.

6.4.2.2. ESP32 Programming

The Arduino IDE has libraries available to download to allow for ESP32 programming. First, we need to add the board manager URL for the ESP32 line of microcontrollers. This is done by pasting https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json into the additional board manager urls text box inside the preferences menu of the Arduino IDE, shown below.


Preferences

Settings | Network

Sketchbook location:

C:\Users\ethan\OneDrive\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 20

Interface scale: [✓] Automatic 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: [ ] compilation [ ] upload

Compiler warnings: None

[ ] Display line numbers
[✓] Verify code after upload
[✓] Check for updates on startup
[ ] Use accessibility features
[ ] Enable Code Folding
[ ] Use external editor
[✓] Save when verifying or uploading

Additional Boards Manager URLs: .githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

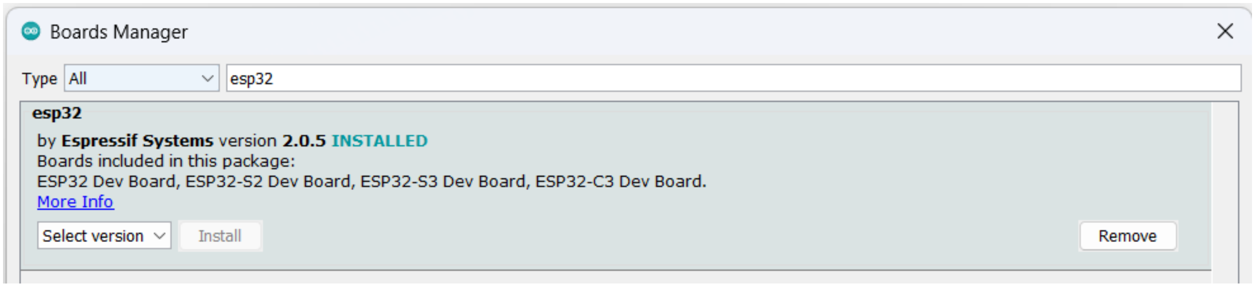
More preferences can be edited directly in the file

C:\Users\ethan\OneDrive\Documents\ArduinoData\preferences.txt

(edit only when Arduino is not running)

Figure 61: Arduino IDE Preferences Menu

Next, in the boards manager just above the board selection menu, we are able to search for the ESP32 package and click install as demonstrated here.


Boards Manager

Type All esp32

esp32
by **Espressif Systems** version **2.0.5** **INSTALLED**
Boards included in this package:
ESP32 Dev Board, ESP32-S2 Dev Board, ESP32-S3 Dev Board, ESP32-C3 Dev Board.
More Info

Figure 62: ESP32 Package in Boards Manager

Now we can select the ESP32 dev module and program it with the Arduino C code as in Figure 63.

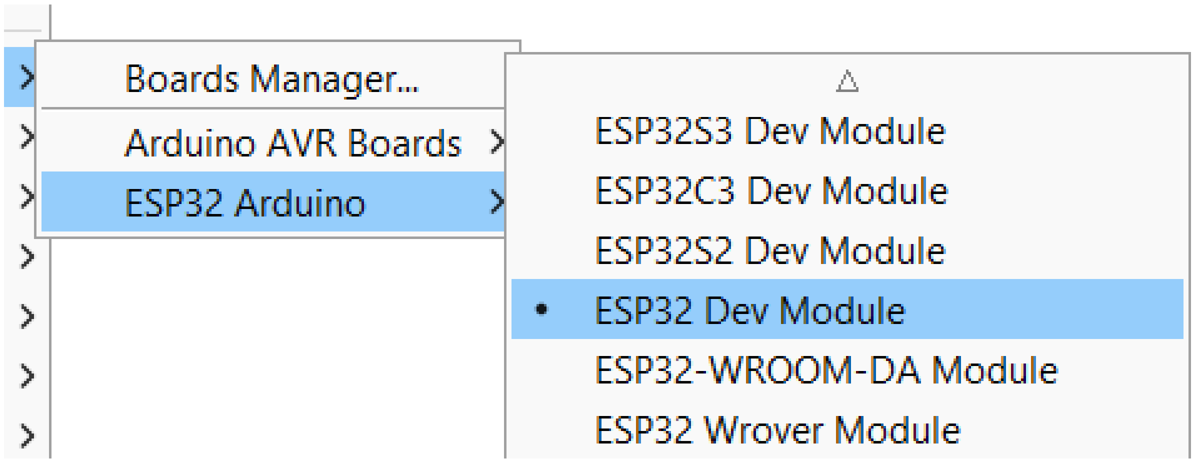


Figure 63: ESP32 Dev Module in Boards Manager

6.4.2.3. Prototype Takeaways

The most intensive part of the code was timed to benchmark the performance of C vs Micropython running on the ESP32. Each loop ran in 270 µs in Micropython, and 1.5 µs in C. This tells us that the C code ran 180x faster than the Micropython. We will continue to use Micropython, but this result lets us know that we can always fall back on programming the microcontroller in C if we need the extra speed to run the MIDI processing algorithm in real time.

7. Prototype Testing & Evaluation

7.1 Hardware Testing

Testing is going to be key in this project to ensure things are working precisely. Each piece will need to be tested differently and adequate through testing we will discover certain things that were previously unknown issues that we will encounter and have to deal with. For testing the different components we can test them independently in order to make sure they work before implementing them into the entirety of the system.

7.1.1. Initial Motor Testing

Initially the testing for the servos will be fairly basic with the motor we will have them wired to a power source and some control signal. The easiest way to ensure they work when the specific signal is triggered is to attach it to a breadboard and a microcontroller and give it power and a signal. Ideally what this will test for is to make sure that the motor can get enough voltage and that it displays the correct characteristics when the control signal is applied. In order to test the servo motors we had our regulating circuits connected to our power supply and then we would send a pwm signal from our code to the servo motors itself. Typically this was done when we were having a problem with the guitar sounding off. Most of the time it was due to a servo shield not being connected correctly, to find this we ran through our all notes test which ran through the different configuration of notes we had and we were able to see which ones were not being used and we were able to immediately test where the issue was one reconnect the power is what fixed it 99% of the time

Since we are buying so many servos, it is important to confirm that they all function as expected. Each servo may also respond slightly differently to the PWM signals, so we need to find a range of duty cycles that works for all of them. Once we confirm that the shift registers are operating properly, we can string 4 of them together to get 32 independently controllable outputs and hook them up to the 30 servos. Ideally, we would cycle through all servo activation combinations in binary to confirm that they all work, but with 30 motors that is way too many combinations. Instead it will likely be sufficient to program a simpler sequence of activations. For example, we could turn all of the servos on, then all of them off, then half on and half off, then the 1st and 3rd quarters on, and the 2nd and 4th quarters off, etc. This should test a decent amount of output interactions in a reasonable amount of time.

Part of our testing we can focus on is figuring out the amount of current that the servo motors are using a digital multimeter in order to test our currents its important we test our current in series vs parallel and we need to test our circuit from the power supply and not the control wire of the servo motors.

7.1.1.1 Strumming Assembly Testing

To test the strumming assembly we need to make sure that all six servos here are capable of plucking strings and they are plucking the correct string for the note they are trying to make. Also we need to ensure that the servo on the fret presses down before the strumming servo. The strumming servo should not break after plucking the strings and the strings should not snap as a result of the pick hitting it. We can test this by running a simple program that just moves the servos to pluck the strings. If it does not work we can decrease the force with the servo or print a shorter guitar pick for the servos. The servos in the strumming assembly should also revert back to a certain position after playback is finished with all picks facing down so that the user is aware that the playback is over and to ensure that it does not interfere with the playback of the next song.

7.1.1.2. Fretting Assembly Testing

To test the Fretting Assembly we need to ensure that the servos press down on the frets with enough force to make the sound of the note without breaking the assembly that will press on the fret or break the servo out of the assembly holding it to the guitar. Also the frets need to be pressed down before the servos at the soundhole pluck the string and hold down the string until the note is played then the servo should lift up from the fret. The fretting assembly should also have its servos revert back to a default position after playback for the same reasons as the strumming assembly and the pieces that hold down the fret should be in an upward position.

7.1.2. Power Supply Testing

In order to ensure our power supply is giving us our adequate voltage and current readings we can come up with our circuits and then recreate them on a breadboard and use a 12 V power supply to simulate a battery. From there we can read our output voltages to make sure we're getting readings that match our needs. Additionally before bringing the circuit to life we can create the circuit digitally and test it to make sure our circuit design is working according to our needs.

7.1.3. Integrated Circuit Testing

To ensure all of our ICs are fully functioning, it helps to make a testing board into which each IC can be inserted and automatically evaluated. Such a board can be fashioned from arduino kits that we already have.

7.1.3.1. Shift Register Testing

The shift register has a simple interface which can be connected to 3 pins from an arduino. On a breadboard, we can place 8 LEDs with current-limiting resistors to indicate which outputs are on or off at any given time. A good sketch for testing the proper operation of the output would be a binary counter, as it systematically covers all possible output combinations. It would also be a good idea to test the daisy-chaining functionality of the shift registers. We could place 16 LEDs on the breadboard and test 2 shift registers at a time.

7.1.3.2. Motor Driver Testing

Instead of LEDs, we will need to test these drivers on actual motors. The arduino kits we have come with some simple DC motors, but we only have 3. Therefore, we may have to wait to test these components until we have purchased and received all the motors we are going to use for the final project. Once we have those, the testing will be very similar to the shift register testing; we can go through every possible combination of motors being on and off by going through them all in binary. This may require the use of the shift registers, which is not ideal since we would like to test each component individually. Still, we can just test the shift registers first and then only test the motors once we've confirmed that they work.

For now, we plan on using servos, which won't need this IC to function. However, this testing plan is a good backup in case we have to switch to a different type of motor later in the project.

7.1.3.3. WiFi/Server Testing

To test the WiFi we will enter the SSID and the password of the selected WiFi into the boot file if the WiFi connection is established we will receive a local IP address back from the ESP32, The server will be connected to the standard HTTP port (port 80) and we can test this by building a simple HTML file to show that the web browser was able to connect to the server.

After this we will need to test the two functionalities of the server. The first functionality to test is whether or not the url will change to include the song name after an option from the drop down menu is selected. After that has been confirmed to work the play button will need to be tested to see if "play" is added to the GET request if the guitar starts to play after both the song and "play" are in the GET request then the test was successful.

7.1.3.4. LCD Testing

To test the LCD all that needs to happen is it needs to be able to display every possible state that was addressed above in the document, it also needs to display the song title and display a timer during the playback. It will also need to switch between song titles when the user changes the song on the user interface. It also needs to pause the timer on the LCD if the user pauses the song and revert back to 0:00 when the user changes the song. The LCD screen did not make it to the project but like other sections it will be left here as it was a stretch goal.

7.1.3.5. PCB Testing

In order to test our PCB we will need to make sure all our hardware components will be in the correct location and the correct orientation. We should have the orientation they need to be in already determined due to previous planning. After we solder all our components into the space they should be going to, this should be the last step on setting up the PCB. Next we will have to test the PCB to make sure everything works. If something ends up not working we will have to go back to make sure all the electrical components are in the correct orientation which means we will probably have to use the oven to remove pieces and reposition them. If that doesnt fix anything we have to go back to the drawing books and redesign the PCB and ensure that the circuit works on a breadboard. Ideally we get our PCB right the first time because if we have to order a new PCB and design in it that will severely eat into the time as we have to wait for it to ship before we can even begin testing anything.

Once we order and receive our custom PCB we will need to test it in various ways in order to ensure all our functions are operating as intended. We will need to test to make sure that the PCB can handle the amount of voltage and current that will be pushed into it. In order to do that we can connect our PCB to a power supply and crank the voltage up to 5-6 volts and about 8.1 amps which will be essentially our max voltage and current ratings of our project so if our PCB can handle those without having something smoke or burn would be a big win. Also we will then look to connect it to the ESP 32 and servo motors. If we can turn it on with the power supply and have nothing to smoke immediately that will be a good sign to continue. Next we will be looking to send signals to the servo motors and if we can send signals to the servo motors that will be fantastic and it will basically be our final test.

7.1.5. ESP32 Benchmarking

From earlier prototyping, we saw the need to benchmark the performance of the ESP32 processor operating under Micropython code versus Arduino C code. We wrote some simple code that does 1 million multiplications and additions and measures the time taken per operation. For the Micropython code, it took 19,373 nanoseconds per operation, which is roughly 50,000 operations per second. For the Arduino C code, it took 8.4 nanoseconds per operation, which is roughly 120 million operations per second. The Arduino C speed more closely matches the expected speed of the processor, and is 2,300 times faster than Micropython. There was some difficulty in getting the Arduino C result, since C compilers like to make optimizations to simple tests that pretty much make them instant.

7.1.6. Note Pitch/Frequency Testing

Guitar note pitch testing will be performed as follows:

1. Manual tuning of the guitar is performed by a human using the Pano Tuner guitar tuning smartphone app and microphone to measure frequencies of open strings

beginning with low E string all the way up to high E, while adjusting tuning knobs for optimal tension of the strings for standard tuning

2. Strumming motors are applied at exact forces decided for the operation of our autonomous guitar and frequency is checked again using Pano Tuner

3. Now we move on to checking every possible fret position we can press down on as per our requirements and checking the frequency is correct as intended

4. Now move on to sending entire chord combinations and checking the desired frequency range of notes is played and able to be recognized by a human and by Pano Tuner app

See evaluation section 7.2.1. for how these checks will be evaluated. We manually created several hardcoded MIDI files to test all of the notes in our range with various constraints such as: different speeds, overlapping notes, simultaneous notes (first input to the preprocessing algorithm, then played on the physical guitar)

7.2. Hardware Evaluation

There will be several checkpoints in which must be checked in order for us to determining our project a success:

- Our servo motors will not exceed the target area by not over rotating on the servo motor

- The current supplied to each servo motor is around 200-250 mA without any notice of heat

- Voltage at each of the points to ensure 5V can be read at all of the necessary points

- We will ensure that our servo motors will not be exceeding 10 mA while idle to ensure no false readings

- The motor mounting frame is stable and can hold the servo motors

- The wires are not interrupting with any fret pressing our interfering with the strings

7.2.1. Note Pitch/Frequency Evaluation

Below is a figure demonstrating the app will use to evaluate what we will use to evaluate our notes' proper tuning.

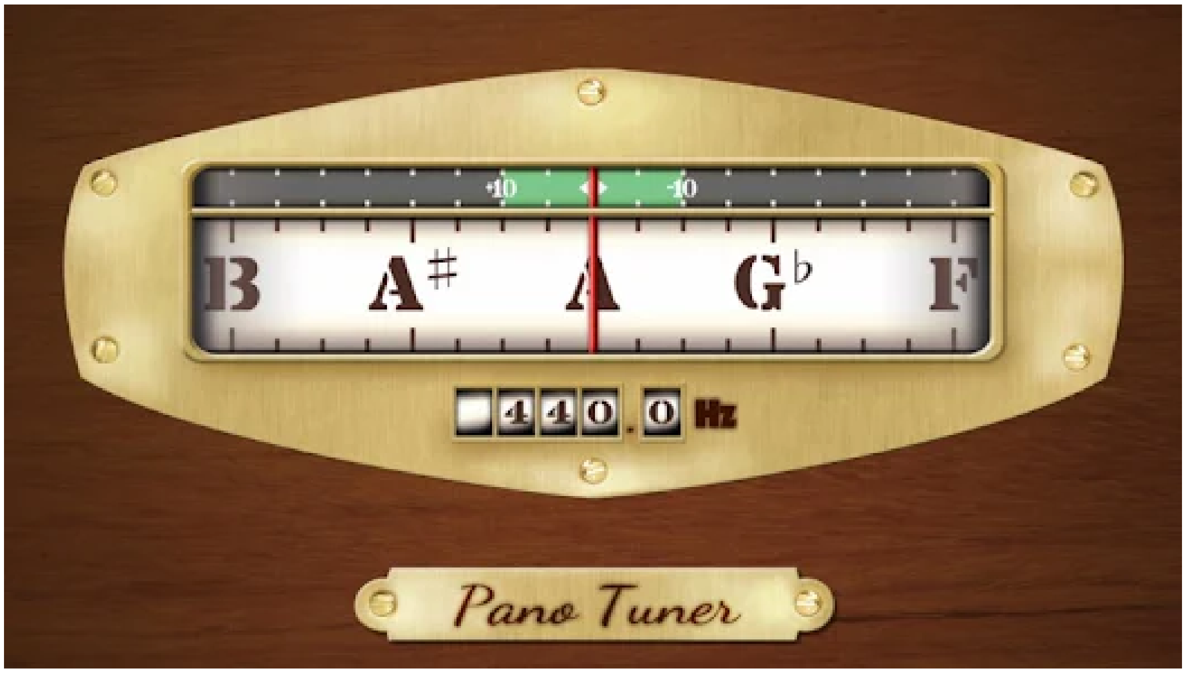


Figure 64: Pano Tuner App from Google Play App Store
https://play.google.com/store/apps/details?id=com.soundlim.panotuner&hl=en_US&gl=US&pli=1

Guitar note pitch evaluation will be performed as follows:

1. Given a single note either on an open string or played on a fretted position, or a full chord as outlined in our testing section 7.1.5.

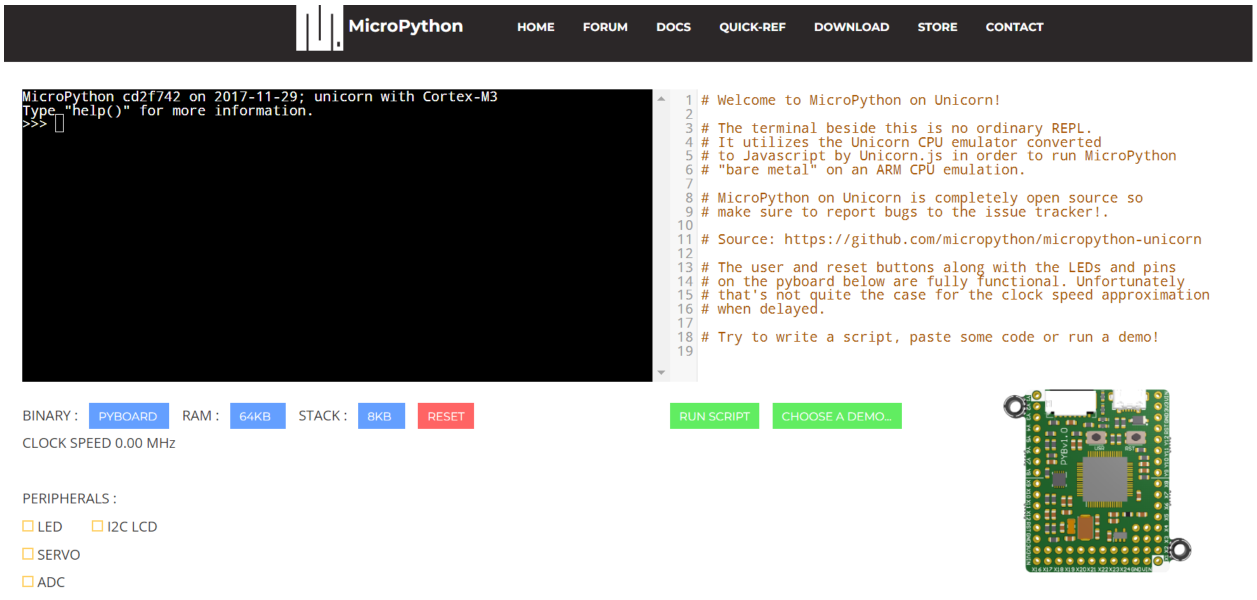
2. We check that the Pano Tuner smartphone app is able to register the note in within +/- 10 Hz of the desired frequency. For example, as in the picture above for the “A” (440 Hz) note we would check that our allowed frequencies are in the “Green” zone of 430 - 450 Hz.

3. We will allow a +/- 15 Hz error for recognizing full chords due to the sensitive nature of playing multiple notes at once on the instrument

The reasons for this possible range of error can be due to several factors including but not limited to: temperature, air pressure, string material, overall build quality of the guitar, echo, resonance of the room, quality of the smartphone microphone, and ambient noise; all of which we have identified as negligible factors to the overall success of our project. We believe it is reasonable to flag a failure only if notes are played in the invalid range of over +/- 10 Hz from the true values as then it is much more likely that this was due to an error on our parts and we will revisit our software and hardware components to try and find, isolate, and fix the cause of the failure.

7.3 Software Testing

We can easily simulate real MicroPython code on the Unicorn CPU emulator provided by https://micropython.org/unicorn/ for starting software development and testing before the parts have actually arrived, reducing one of our main anticipated blockers on progression on the software side of things. This emulator is shown in the figure below.



```
MicroPython cd2f742 on 2017-11-29; unicorn with Cortex-M3
Type "help()" for more information.
>>> 
```

```
# Welcome to MicroPython on Unicorn!

# The terminal beside this is no ordinary REPL.
# It utilizes the Unicorn CPU emulator converted
# to Javascript by Unicorn.js in order to run MicroPython
# "bare metal" on an ARM CPU emulation.

# MicroPython on Unicorn is completely open source so
# make sure to report bugs to the issue tracker!.

# Source: https://github.com/micropython/micropython-unicorn

# The user and reset buttons along with the LEDs and pins
# on the pyboard below are fully functional. Unfortunately
# that's not quite the case for the clock speed approximation
# when delayed.

# Try to write a script, paste some code or run a demo!
```

BINARY : PYBOARD RAM : 64KB STACK : 8KB

CLOCK SPEED 0.00 MHz

Figure 65: Unicorn CPU Emulator for MicroPython https://micropython.org/unicorn/

Our testing plan will comprehensively test:

- Sending servo motor control signals according to our algorithm
- Playing multiple notes/strings at once
- A simple song such as “Twinkle Twinkle, Little Star”

7.4 Software Evaluation

- We will check that each note’s sustain is within a 5% margin of error of what it should be as per our requirements

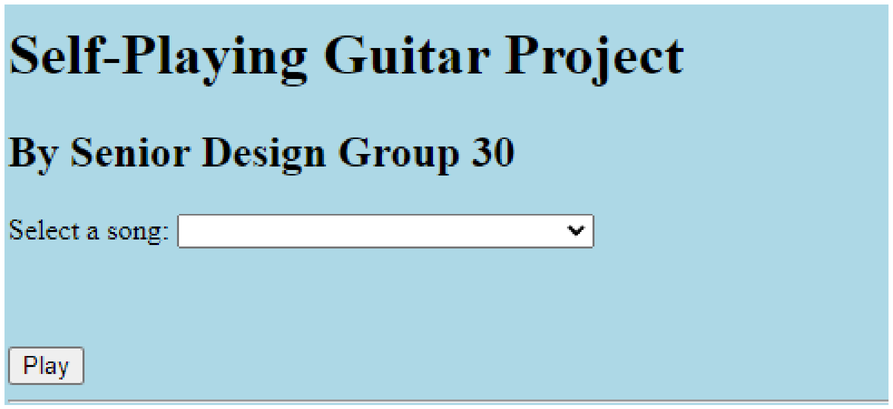
- We will check the timing between notes, or delay, within 5% margin of error as well, as per requirements

- We will check that we are able to hit our design requirements of playing at least 2 notes per second per string by sending generated MIDI data of 2 fully fretted bar chords each second

- We will check that we are fully able to upload and store 10 songs on the microcontroller device

8. Project Operation

The autonomous guitar's full project operation is as follows:


Self-Playing Guitar Project

By Senior Design Group 30

Select a song:

Figure 66: File Upload in Project Operation

1. The UI is accessible through any web browser on a computer or on a smartphone the user must be connected to the same WiFi that the guitar is connected to. The upload pause and stop buttons have been removed from the project due to constraints with the platform and time constraints, the stop functionality is still included in the project but it is now controlled with a button interrupt.

2. The software on the microcontroller will display the server whenever it is requested by a client, the server will read the HTTP GET request and wait for the user to initiate the playback when they choose a song and press the play button. When the user chooses a song from the dropdown menu each item on the list will have a hyperlink that will add the song title to the URL and then the play button is also added to the URL when it is pressed. When the server reads that the song was selected and the play button was pressed it will begin if just the play button was pressed, it will not do anything unless both the song title and the play command are in the url.

3. The stop functionality is available through a hardware interrupt in case the user was not happy with the sound of the song. It was done this way because micropython does not officially support multithreading which would be needed to stop the code from playing mid song, but the button interrupt will have the same functionality as the stop button in the UI would do but this was much quicker for us to implement.

8.1. Operating Modes

The following two sections refer to the LCD screen stretch goal that did not make it into the project. The Autonomous Guitar will have 2 operating modes one is a regular mode where the MCU will wait for the user to upload a song onto it and then wait for the the user to push play on the device the microcontroller should be able to hold multiple songs on it until it's memory is full. The user should be able to select the song they want to play; they can cycle through songs on the user interface and then the song title will show up on the LCD screen on the guitar. The song will then play to completion or whenever the user wants.

The autonomous guitar will also have a tuning mode, since it will be difficult for the user to pluck the string when the assembly is attached and we do not want the user to remove the assembly unless necessary, the autonomous guitar will have a mode where the guitar will pluck a string repeatedly until the user finds the correct tune for the string and then move to the next string and repeat the process until the user tunes all six strings. The LCD screen will show “TUNING MODE” on the screen and will show which string number is currently being tuned. In the event that a string on the guitar does break due to operator error or just from being played extensively the assembly on the soundhole and the neck will need to be removed to replace the string.

8.2. Error Correction

There are potential issues that the autonomous guitar can have that will be of potential concern, one such issue is that it will be possible for the microcontroller storage to fill up and no more songs can be placed on it this is the issue that we believe will happen the most often and it is detectable by our software, so we can inform the user that it is a problem so that they can delete some songs in the microcontroller. The LCD screen on the guitar will display “LOW MEM” even though we do not perceive this happening often due to the fact that MIDI song files are around 30 KB and the microcontroller can contain 4 MB of data.

The concern that we can see will happen more frequently is the bluetooth connection failing or cutting off, if this does happen the autonomous guitar should keep playing the song it is playing if it is currently playing a song since this function does not require the bluetooth connection, otherwise the no new songs can be uploaded to the microcontroller until the connection is restored. While waiting for the connection to re-established the LCD screen will show the message “PAIR READY” until a connection is made. This message will show anytime the guitar does not have a bluetooth connection.

There are also things that the autonomous guitar will not be able to know about such as a string or a pick breaking in this case no error message can be shown but an observant user will notice one of strings or picks breaking since they will not be completely covered and the picks will be replaceable without removing the assembly or the servo.

9. Administrative Content

9.1. Division of Labor

Pedro Contipelli

- Reading MIDI data in Python
- Preprocessing algorithm
- Full song-playing algorithm
    - Processing Notes
    - Range Compression
    - Handling Overlapping notes/tracks
    - Converting notes to fret/strumming motor output positions
    - Timing logic
- Buying and inspecting guitar
- Project manager / meeting organization
- SCRUM master / labor oversight and team check-ins
- Project budget and expense distribution

Blake Cannoe

- Choose microprocessor
- Port algorithm to microprocessor
- Write code for motor control pins
- Breadboard prototype for shift register circuits
- Web app UI and functionality
    - Web socket / HTTP GET
    - Design architecture choice of bluetooth low-energy or wi-fi
- Cross-platform mobile app UI and functionality
- Final PCB to PCA9685 controller wiring

Ethan Partidas

- Mechanical design of fretting motor mount assembly
- Mechanical design of strumming motor mount assembly
- Servo controller algorithm testing and debugging
- Multimeter and oscilloscope breadboard prototype debugging
- Breadboard prototype for PCA9685 circuits

Jonathan Catala & Kyle Walker

- Battery, electronics, wiring, hardware, etc
- Planning schematic and board layout for PCB
- Ensuring DC to DC converter has correct power output
- Ensuring power is supplied to motors

9.2 Project Milestones

Below are the project milestones for both Senior Design 1 and 2 that we will use to gauge our progress. These milestones are tentative and will be changed based on the evaluation of our schedule.

9.2.1. [Fall] Senior Design I	
Date	**Milestones**
10/7/22	(CS) TA Check-In & (CS) Assignment #3 & (ECE) D&C 2.0 Submission
10/14/22	Design Plans / 15-pages written
10/21/22	Dr. Leinecker Status Meeting Check-In / 15 pages written
10/28/22	50 pages of design document
11/4/22	(ECE) 75-page report submission / (CS) Assignment #4 Due 11/6
11/11/22	100 pages design document written
11/18/22	(ECE) 125-page report submission
11/25/22	Begin Design Implementation / 135 pages
12/2/22	Schematic complete and Documentation revision / 150 pages
12/5/22	**(CS) Final Design Document Due**
12/6/22	All parts ordered

Table 15: Project Milestones for Senior Design I (Fall 2022)

9.2.2. [Spring] Senior Design II	
Date	**Milestones**
1/13/23	Receive ordered parts - (Team) Implement MIDI File Processing in Python - (Pedro) Begin PCB design schematics - (Kyle & Jon) 3D model linear actuators and fretting motor assembly mount - (Ethan) Implement LCD Screen functionality using shift registers - (Blake)
1/20/23	Implement basic guitar-note outputting algorithm using simple test MIDI files as inputs - (Pedro) Consolidate schematics into one PCB and place order for it - (Kyle & Jon) Finish 3D fretting mount assembly model and order prints - (Ethan) Microprocessor code for motor outputs - (Blake)
1/27/23	Finish core algorithm and digitally test on real MIDI files - (Pedro) Wire up one row of servo motors to fretting mount module and test fretting with manually strumming notes - (Team) Fretting mount iteration and begin modeling strumming mount - (Ethan) Regulator implementation - (Kyle & Jon) Bluetooth connection to microprocessor code - (Blake)
2/3/23	Finalization of algorithm and begin testing software-hardware interface with real motor outputs - (Pedro) Last design iteration of mounts. Place final print orders - (Ethan) Strumming and fret motor assemblies connection to microprocessor - (Blake) Battery-Regulator-PCB connection - (Kyle & Jon)
2/10/23	Begin work on desktop user-interface - (Pedro) PCB-Microcontroller-Code connection - (ECE Team)
2/17/23	Finish desktop user-interface - (Pedro) Begin wiring up all motor assembly modules and load-testing - (ECE team) Project Prototype & Requirements Revision - (Team)
2/22/23	**CDR** (Critical Design Review) presentation - (Team)
2/28/23	Begin mobile app user-interface - (Pedro) Finish wiring up all motor assembly modules and load-testing - (ECE team)
3/3/23	Finish mobile app user-interface - (Pedro)

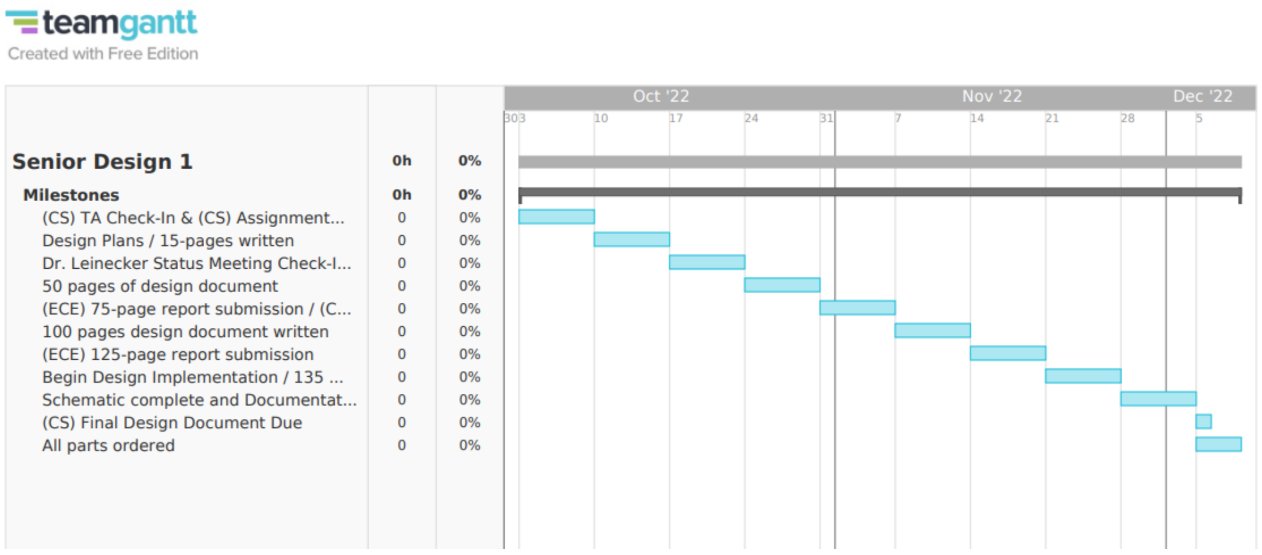
9.2.2. [Spring] Senior Design II	
Date	**Milestones**
3/3/23	Full software-hardware interface connection - (Team) Minimum Viable Product Finished - (Team)
3/10/23	Testing / Integration / Bugfixes - (Team)
3/17/23	**Spring Break**
3/21/23	**Leinecker Demos**
3/31/23	Implement any demo feedback - (Team) Revise design documentation - (Team)
4/7/23	Finalization of working project - (Team) Stretch Goals / Buffer Time - (Team)
4/13/23	**Final Committees**
4/19/23	**Final Committees**

Table 16: Project Milestones for Senior Design II (Spring 2023)

9.3 Gantt Charts

The following sections are Gantt Charts that we will use to dole out our time appropriately for each milestone. Like the milestones, these times are again tentative.

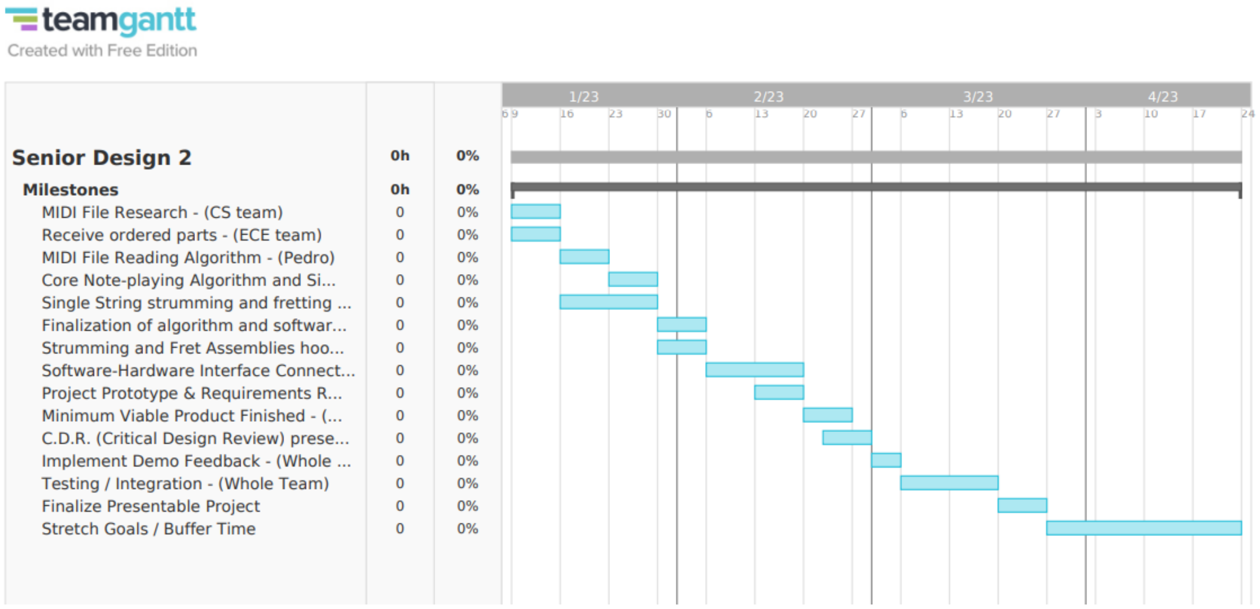
9.3.1. Senior Design I [Fall 2022]



Senior Design 1	**0h**	**0%**
Milestones	**0h**	**0%**
(CS) TA Check-In & (CS) Assignment...	0	0%
Design Plans / 15-pages written	0	0%
Dr. Leinecker Status Meeting Check-I...	0	0%
50 pages of design document	0	0%
(ECE) 75-page report submission / (C...	0	0%
100 pages design document written	0	0%
(ECE) 125-page report submission	0	0%
Begin Design Implementation / 135 ...	0	0%
Schematic complete and Documentat...	0	0%
(CS) Final Design Document Due	0	0%
All parts ordered	0	0%

Figure 67: Senior Design I (Fall 2022) Tasks Gantt Chart

9.3.2. Senior Design II [Spring 2023]



Senior Design 2	**0h**	**0%**
Milestones	**0h**	**0%**
MIDI File Research - (CS team)	0	0%
Receive ordered parts - (ECE team)	0	0%
MIDI File Reading Algorithm - (Pedro)	0	0%
Core Note-playing Algorithm and Si...	0	0%
Single String strumming and fretting ...	0	0%
Finalization of algorithm and softwar...	0	0%
Strumming and Fret Assemblies hoo...	0	0%
Software-Hardware Interface Connect...	0	0%
Project Prototype & Requirements R...	0	0%
Minimum Viable Product Finished - (...	0	0%
C.D.R. (Critical Design Review) prese...	0	0%
Implement Demo Feedback - (Whole ...	0	0%
Testing / Integration - (Whole Team)	0	0%
Finalize Presentable Project	0	0%
Stretch Goals / Buffer Time	0	0%

Figure 68: Senior Design II (Spring 2023) Tasks Gantt Chart

9.4 Budget/Financing

The following were rough estimated costs of some components of the project which we looked into at the time of starting our report. The cost for individual passive components are included in the Misc. Electrical Components item.

Used Acoustic Guitar: $30
ESP32 Microprocessor: $20
Misc. Electrical components: $10
30 Servo Motors for frets and strumming: $60
Filament for 3D Printer: $5

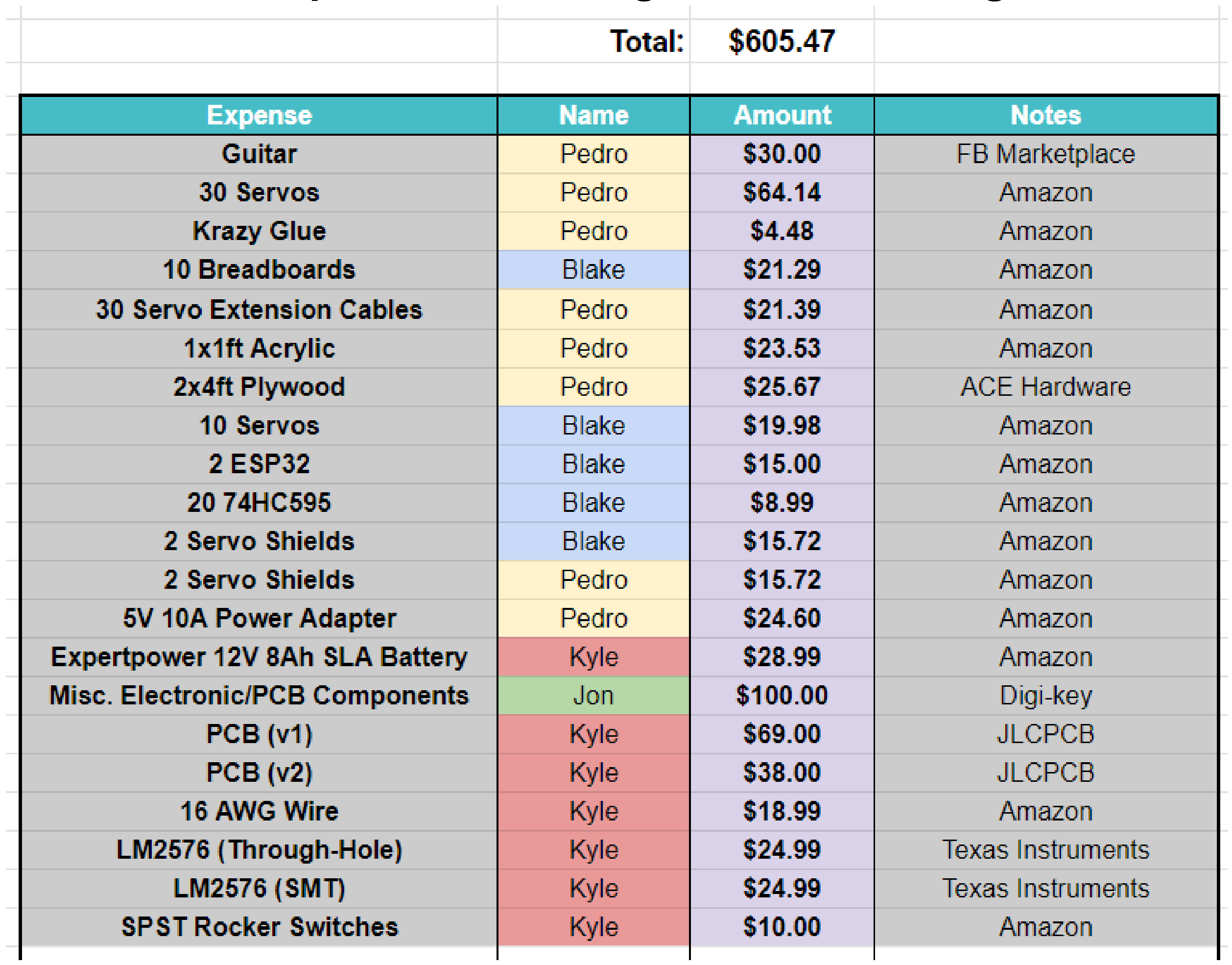
9.4.1. Original Prototype Bill of Materials

In this section is our original prototype's bill of materials. It is updated to account for the cost of various components that we have established that we will use for our prototyping phase.

Item	**Quantity**	**Price per unit**	**Price**
Used Acoustic Guitar	1	$30.00	$30.00
Expertpower SLA Battery 12V 8Ah	1	$28.99	$28.99
Microprocessor (ESP32)	1	$7.00	$7.00
Misc. Electrical components	1	$30.00	$30.00
Micro-servo Motors	30	$2.00	$60.00
74HC595N shift register	4	$1.00	$4.00
LM2576HV	5	$3.66	$18.3
1602 LCD	1	$7.00	$7.00
2x4 ft Plywood	1	$25.67	$25.67
1x1 ft Acrylic (2 pk)	1	$23.53	$23.53
Servo Extension Cables (30 pk)	1	$21.39	$21.39
DIODE SCHOTTKY	1	$0.51	$0.51
ESP 32 Chip	1	$3.50	$3.50
Terminal Block	2	$2.34	$4.68
Micro USB connector	1	$1.36	$1.36
USB TO UART BRIDGE	1	$6.80	$6.80
PCB (from jlcpcb / per 1 mm^2)	2500	$0.01	$25.00
Total			**$297.73**

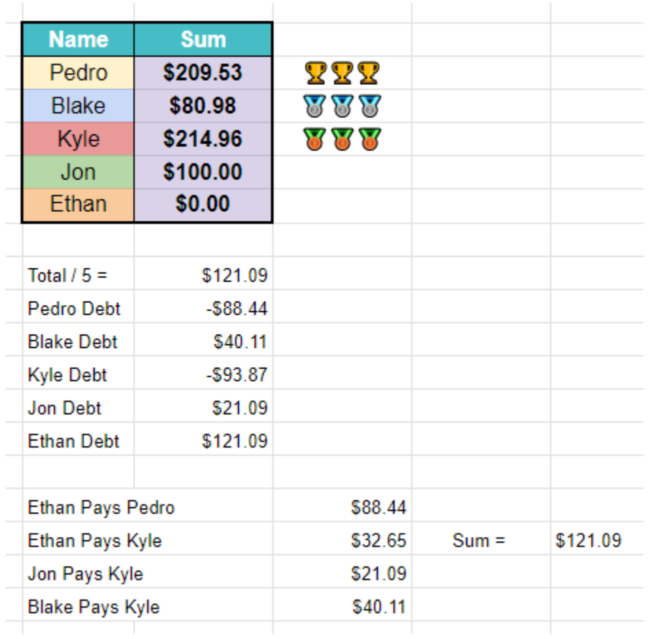
Table 17: Prototype's Bill of Materials

9.4.2. Final Expenses and Budget Distribution Agreements



	Total:	**$605.47**	

Expense	Name	Amount	Notes
Guitar	Pedro	**$30.00**	FB Marketplace
30 Servos	Pedro	**$64.14**	Amazon
Krazy Glue	Pedro	**$4.48**	Amazon
10 Breadboards	Blake	**$21.29**	Amazon
30 Servo Extension Cables	Pedro	**$21.39**	Amazon
1x1ft Acrylic	Pedro	**$23.53**	Amazon
2x4ft Plywood	Pedro	**$25.67**	ACE Hardware
10 Servos	Blake	**$19.98**	Amazon
2 ESP32	Blake	**$15.00**	Amazon
20 74HC595	Blake	**$8.99**	Amazon
2 Servo Shields	Blake	**$15.72**	Amazon
2 Servo Shields	Pedro	**$15.72**	Amazon
5V 10A Power Adapter	Pedro	**$24.60**	Amazon
Expertpower 12V 8Ah SLA Battery	Kyle	**$28.99**	Amazon
Misc. Electronic/PCB Components	Jon	**$100.00**	Digi-key
PCB (v1)	Kyle	**$69.00**	JLCPCB
PCB (v2)	Kyle	**$38.00**	JLCPCB
16 AWG Wire	Kyle	**$18.99**	Amazon
LM2576 (Through-Hole)	Kyle	**$24.99**	Texas Instruments
LM2576 (SMT)	Kyle	**$24.99**	Texas Instruments
SPST Rocker Switches	Kyle	**$10.00**	Amazon



Name	Sum	
Pedro	**$209.53**	🏆🏆🏆
Blake	**$80.98**	🥈🥈🥈
Kyle	**$214.96**	🥉🥉🥉
Jon	**$100.00**	
Ethan	**$0.00**	

Total / 5 =	$121.09			
Pedro Debt	-$88.44			
Blake Debt	$40.11			
Kyle Debt	-$93.87			
Jon Debt	$21.09			
Ethan Debt	$121.09			
Ethan Pays Pedro		$88.44		
Ethan Pays Kyle		$32.65	Sum =	$121.09
Jon Pays Kyle		$21.09		
Blake Pays Kyle		$40.11		

Ultimately, as this was an entirely student self-sponsored project, we decided on a budget agreement where regardless of who paid for what expenses for sake of convenience throughout the design and construction of the project, at the end, we would fully split the costs of the entire budget for the project, which after even distribution came out to the 5 of us students each paying **$121.09** for a total project budget of

5 students × $121.09 = **$605.47** final expenses

All receipts for project-related purchases were shared and saved for bookkeeping purposes, as well as expenses and redistribution calculation performed on a shared Google Sheets document accessible and verifiable by any team member.

10. Project Summary & Conclusions

10.1. Project Summary

The goal of our project is to create an autonomous self-playing guitar that is able to produce its own music. We plan to buy the actual guitar and electronic components such as motors and microprocessors, but everything else must be pretty much built from scratch. We have analyzed and thought about design concepts utilized by other/previous similar projects which have inspired us, and plan on improving upon them to create a project that is fully our own.

The system would be able to take in any MIDI audio file within our design constraints (other file types can always be converted to MIDI beforehand) and play the notes on the guitar using a microcontroller with separate mechanisms for strumming and pressing the right strings against the right frets at the right time. It would be lightweight and maintain the general form factor of the guitar (i.e, fits closely to the body). The design should ideally be portable, and thus it would be powered by portable batteries. It should be responsive enough to accurately replicate the provided MIDI file compositions, comparable to - if not exceeding - the abilities of the average learnt guitar player. Not only should this design be lightweight and portable, an issue with similar concepts is the price and size. They are typically not an attachment for a guitar and are more commonly an entire unit within the guitar. They are also extremely expensive with some models going for up to $1,100. Our goal for this project was to bring this idea to reality for significantly cheaper.

10.2. Design Summary

Our Self-Playing Guitar project setup consists of an acoustic guitar outfitted with servo motors that will strum and fret the guitar to play a song that has been uploaded to the system as a MIDI audio file via Bluetooth. The MIDI file will be interpreted by software written to an ESP32 Microcontroller using MicroPython, taking in the file and using a custom algorithm which preprocesses the song's notes data, compresses it to fit the playable range of notes on our guitar, and then operates the servos in conjunction with that data in realtime to play the song that was selected.

The project's controller will be on a separate device (accessible via web or mobile app) which will handle song selection, file uploading, playing, pausing, and stopping via Bluetooth communication to the microcontroller.

The project's mechanical assembly consists of a strumming assembly with 6 servos placed over the guitar hole and 24 servos for the fretting assembly placed along the neck. The strumming assembly servos will swivel in order to pluck each string independently from one another, allowing for any combination of strings to be played at once. The fretting assembly servos will rotate linear actuators, acting as pistons, to press down each string, with the 24 servos spanning across 3.5 octaves of the range of the guitar.

The project's main electronic assembly consists of one PCB containing voltage regulators for the power supply of the motor assembly and the ESP32 connected using its through-hole devboard. The PCB takes input power from the SLA battery and outputs 5 power/ground connections alongside I2C datalines.

See ***Figure 2:*** *Functional Class Diagram* in ***Section 2.7*** for a high-level overview of our complete design.

10.3. Planning / Engineering Design Conclusions

As Winston Churchill once said, "He who fails to plan is planning to fail." One definitive conclusion we've taken from writing this design document is how important it can be to plan, design, write things down, prototype, and think things through before jumping into the fire. It's already helped us avoid plenty of failures and missteps which could have led us down a wrong path that might not have worked out. Another conclusion that closely ties into this, is about how important it is to have good, solid, robust requirements. It really wasn't until we sat down and started listing all of our project's requirements and constraints that we were able to really get a good idea of all the things that would have to be implemented and what kind of scope / big picture problems we'd be having to address.

We have also come to learn a great deal about teamwork and coming together to achieve a common goal. We succeeded in meeting our goals/requirements and leveraged all of our team members' strengths to create something that none of us could have done individually. Everyone contributed something, and we can confidently say that this wouldn't have been possible without each other's help. Working together also helped to mitigate our own personal biases and weaknesses, helping each other notice and correct flaws, giving good feedback to each other when it was most necessary.

10.4. Philosophical Conclusions

Philosophical conclusions can be drawn from this project about the very nature of what constitutes science, technology, engineering, or mathematics and what constitutes art and if they can, at times, both be one and the same. Certainly a conclusion that our group has drawn from this project is that there really is a deep connection between the two (we have built this project to bring to life a part of that bridge/interface). We've learned not just that we are creating art with technology, but that properly designing and implementing technology *is* an art itself. And we have come to a greater appreciation for well-implemented and thought-out engineering specifications, requirements, and design. As well as come to a newfound appreciation for music and musicians, more specifically the immense amount of practice, talent, dexterity, timing, and coordination it takes for a human being to be able to play this instrument using their own brain and hands.

The limitations and achievements of what we could accomplish with this design naturally brought us to think about machine intelligence and many of the analogous connections between computer processors and the human brain itself. About how they can both achieve many of the same goals and operationally solve similar problems, yet are and have to be implemented entirely differently in most if not all areas of design. Our linear actuators powered by servo motors and the mount we're using to hold them in place are analogous to the muscles, joints, and ligaments in the human arm, hands, and fingers in that they work together to accomplish the same overall goal, but each individual part can serve drastically different functions not to mention the way that they interoperate across themselves.

And yet, even in and amongst all of the nuance and complexity in the differences between the two, a human without understanding even a single part of the way the technology is implemented, can see, hear, and feel all of that difference in the music. It is extremely easy to tell the difference between an autonomous guitar playing vs. when a human is playing it because autonomous guitars result in giving us this sort of staccato 8-bit/16-bit NES sound that is unlike what it sounds when a human is playing. For example, it can be heard in Demin Vladimir's Guitar Robot playing the main theme from Pirates of the Caribbean: https://youtu.be/n_6JTLh5P6E.

Even if the autonomous guitar can be better than a human at doing certain tasks, such as when having perfectly precise timing down to the millisecond without ever making a mistake or playing a wrong note, and always having a consistent measurable way to press down on each fret with the same amount of force each time, playing each and every single note perfectly and consistently the same way every time. It fails at one thing: which is sounding *like* a human. Because humans are imperfect and imprecise and we play, hear, and feel music in a way that is different and can be individually meaningful for different reasons to ourselves and the others around us listening. We play instruments for so much more, because it is in our nature to think and be creative and express ourselves. Whereas a machine can play music for no more than the simple fact that it was programmed to do so.

Still, the genre and range of music that autonomous guitars *can* produce, sounds beautiful to us group members in its own right and category. We've learned it is not about which of man or machine is necessarily better or worse than the other, but more about widening the range of what we can accomplish symbiotically when working together and understanding and appreciating the best of both worlds.

11. Broader Impacts

The main impacts of this project are entertainment for public engagement in STEM areas, whether it be for a live audience or a global audience via recording/uploading on YouTube. We are pushing against the barrier to entry for making live music on an instrument, by putting our engineering brains together and distributing the work. We're giving us and our audience the experience of listening to live music without needing the skill of a master who has practiced playing that instrument for dozens of years, which can be hard to come across.

We are hoping that by showing people just a taste of what is possible when an engineering team works together to accomplish a task, we can make the public as a whole more interested and educated in STEM (Science, Technology, Engineering, and Mathematics) and its extremely wide range of applications, even in a field as "far-fetched" as music. We hope to possibly inspire the next generation of engineers to dream big and pursue their own passions by showcasing that STEM can be interdisciplinary and interact with so many different subjects even as "different" as music or art. With this project, we hope to accomplish creating a spark of interest and wonder in the eyes of the future generations just as others before us have inspired us with their own projects.

12. Legal, Ethical, and Privacy Issues

12.1. Legal Issues

We have anticipated and considered the potential legal issue of needing to secure a mechanical license for any copyrighted songs which we plan on using for demonstration purposes, whether that be for live demonstrations or recordings uploaded on YouTube. As per our current research, the potential legal ramifications we have identified would be in the case of posting our “cover” on YouTube, which may lead to our video getting removed or a deal being negotiated with the copyright owner to obtain revenue from ads displayed on our video, or in the worst case scenario, a strike being placed on the YouTube channel from which it was uploaded. As per live non-recorded demonstrations, there is no legal precedent specifically pertaining to autonomous guitars, however the current interpretation is that it would essentially fall under the same category or be the legal equivalent of a human playing a song on the guitar in public, which is perfectly fine.

“*Some copyright owners don’t mind YouTube covers—they increase a song’s exposure and may introduce a new audience to the songwriters’ or original performer’s music. If songs are posted by fans, a band isn’t likely to risk alienating them by taking down their videos. Other copyright owners object to unlicensed use of their work. A few years ago, Prince famously had YouTube remove a video that showed a toddler dancing to one of his songs.*

If a copyright owner objects, YouTube may remove your video or it may negotiate a deal for the copyright owner to obtain revenue from ads that appear on YouTube. If YouTube removes the video for copyright issues, it will also place a strike against your YouTube channel. After multiple strikes, YouTube will delete your channel, along with the videos, subscribers, likes, views and comments. If you’ve worked hard to cultivate your channel, this can be devastating.”
Source:https://www.legalzoom.com/articles/posting-cover-songs-on-youtube-what-you-need-to-know

12.2. Ethical Issues

We will be referencing and complying with the A.C.M. Code of Ethics and Professional Conduct (https://www.acm.org/code-of-ethics) throughout the process of planning, designing, prototyping, building, testing, and evaluating our project.

12.3. Privacy Issues

At the moment, we do not anticipate encountering any privacy issues as our project does not perform user data collection of any kind.

13. Facilities and Equipment

Through completing this project we have gained several experiences through various labs throughout the campus. While completing the courses we have taken in exploring our majority has given us several skills we will be implementing in our senior design project. Previous labs have given us several skills that range from circuit design to programming.

In some of the electrical engineering labs it was more focused on circuit design and troubleshooting the circuits. We have gained experience using breadboards which will apply to us designing and testing our schematic. We will hook up our circuit looking for various voltage and current readings. This is a vital skill and important in all senior design projects.

In some other computer engineering classes we got to see how programming can be used to complete tasks. This involves programming hardware and implementing different code languages and it will help us a lot when we will need to be programming our microcontroller. This is what controls the servo motors and anything else we need control signals for. As far a programing hardware it is has been extremely important in terms of our education and growth as engineers.

13.1. Facilities

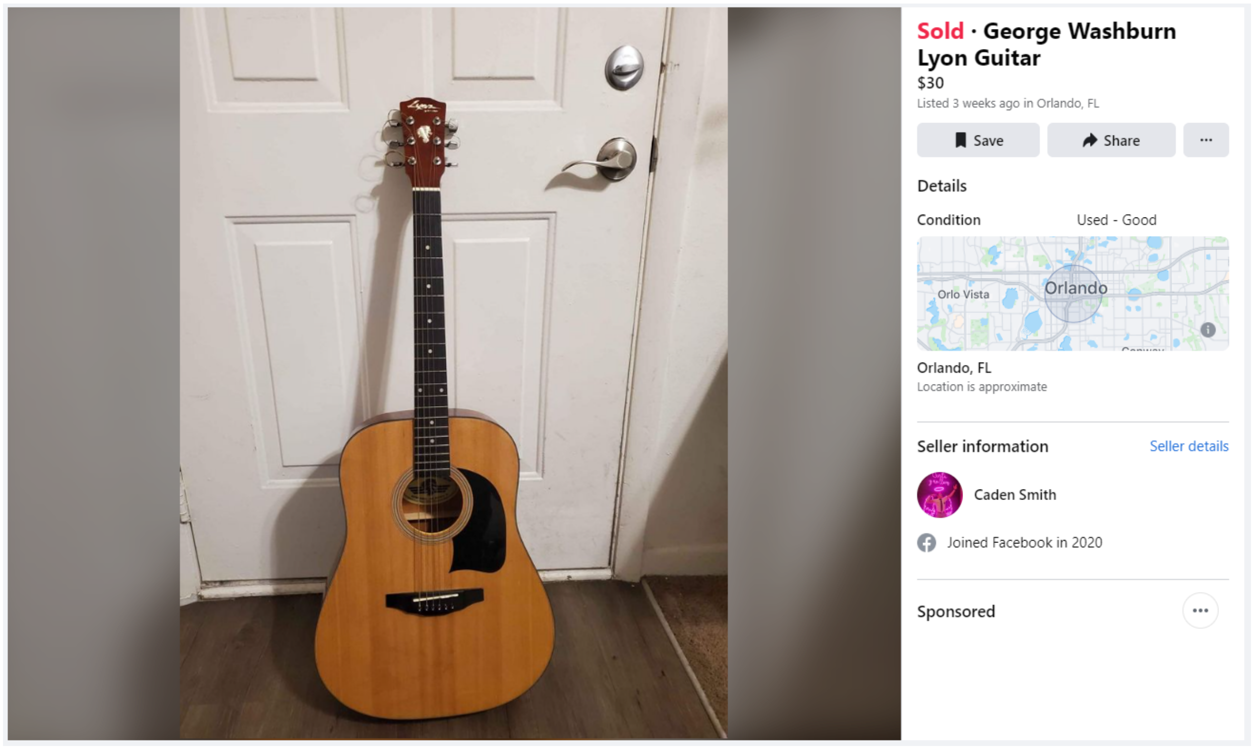
The facilities we will be using include:

- HEC 102 (Computer Science Senior Design Lab)
- ENG1 456 (Electrical and Computer Engineering Senior Design Lab)
- UCF Texas Instruments Innovation Lab (https://www.cecs.ucf.edu/innovationlab/)
- Pedro Contipelli's current home residence in Oviedo, FL

13.2. Equipment and Materials

13.2.1. George Washburn Lyon Acoustic Guitar

The figure below shows the guitar in which the final project is completed as well on which all tests will be run.


Sold · George Washburn Lyon Guitar
$30
Listed 3 weeks ago in Orlando, FL

Details

Condition Used - Good

Orlando, FL
Location is approximate

Seller information Seller details

Caden Smith

Joined Facebook in 2020

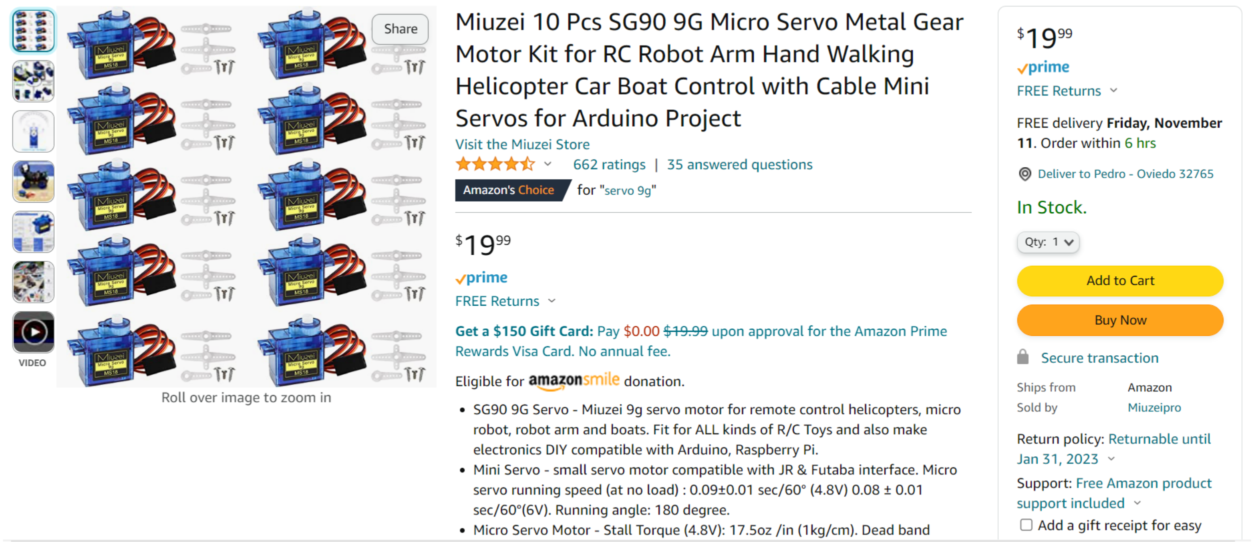
Sponsored

Figure 69: George Washburn Lyon Acoustic Guitar

We chose and bought this guitar as it was available and within our budget of $30. We will implement this project with the exact specifications necessary to make it work only on this guitar. While there is the possibility of generalizing our design to fit other guitars in the future, many of our project-specific parameters would have to be modified.

13.2.2. Miuzei Micro-Servo Motors

The below figure shows which brand of micro servo motor we will use throughout the project, specifically how we will go about obtaining the motors in bulk over Amazon.


Miuzei 10 Pcs SG90 9G Micro Servo Metal Gear Motor Kit for RC Robot Arm Hand Walking Helicopter Car Boat Control with Cable Mini Servos for Arduino Project

Visit the Miuzei Store

662 ratings | 35 answered questions

Amazon's Choice for "servo 9g"

$19.99

prime

FREE Returns

Get a $150 Gift Card: Pay $0.00 ~~$19.99~~ upon approval for the Amazon Prime Rewards Visa Card. No annual fee.

Eligible for amazonsmile donation.

- SG90 9G Servo - Miuzei 9g servo motor for remote control helicopters, micro robot, robot arm and boats. Fit for ALL kinds of R/C Toys and also make electronics DIY compatible with Arduino, Raspberry Pi.
- Mini Servo - small servo motor compatible with JR & Futaba interface. Micro servo running speed (at no load) : 0.09±0.01 sec/60° (4.8V) 0.08 ± 0.01 sec/60°(6V). Running angle: 180 degree.
- Micro Servo Motor - Stall Torque (4.8V): 17.5oz /in (1kg/cm). Dead band

$19.99

prime

FREE Returns

FREE delivery **Friday, November 11**. Order within 6 hrs

Deliver to Pedro - Oviedo 32765

In Stock.

Qty: 1

Add to Cart

Buy Now

Secure transaction

Ships from	Amazon
Sold by	Miuzeipro

Return policy: Returnable until Jan 31, 2023

Support: Free Amazon product support included

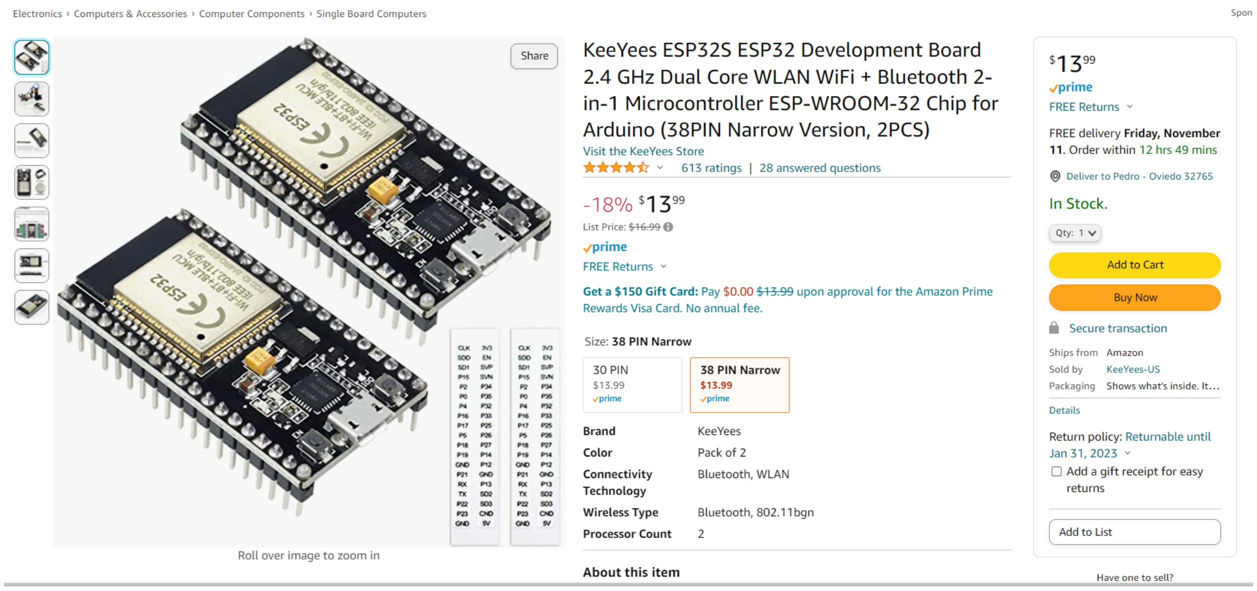
Add a gift receipt for easy

Figure 70: Micro-Servo Motors Purchased

We will buy 3 packs of 10pc. Miuzei Micro Servo Motors Kits. The micro-servos will be what a bulk of our project is reliant on. We will be using them in various ways from strumming to pressing down on frets. Depending on what function they need to do, they will be placed at different locations along the guitar. We will have 6 dedicated to strumming the 6 strings and 24 used for pressing frets.

13.2.3. ESP32 2.4 GHz Dual Core WLAN WiFi + Bluetooth Microcontroller 38PIN Narrow Version

The following figure is the microcontroller we will use for the project.


Electronics › Computers & Accessories › Computer Components › Single Board Computers

KeeYees ESP32S ESP32 Development Board 2.4 GHz Dual Core WLAN WiFi + Bluetooth 2-in-1 Microcontroller ESP-WROOM-32 Chip for Arduino (38PIN Narrow Version, 2PCS)

Visit the KeeYees Store

613 ratings | 28 answered questions

-18% $13.99

List Price: ~~$16.99~~

prime

FREE Returns

Get a $150 Gift Card: Pay $0.00 ~~$13.99~~ upon approval for the Amazon Prime Rewards Visa Card. No annual fee.

Size: **38 PIN Narrow**

30 PIN $13.99 prime

38 PIN Narrow $13.99 prime

Brand	KeeYees
Color	Pack of 2
Connectivity Technology	Bluetooth, WLAN
Wireless Type	Bluetooth, 802.11bgn
Processor Count	2

About this item

$13.99

prime

FREE Returns

FREE delivery **Friday, November 11**. Order within 12 hrs 49 mins

Deliver to Pedro - Oviedo 32765

In Stock.

Qty: 1

Add to Cart

Buy Now

Secure transaction

Ships from	Amazon
Sold by	KeeYees-US
Packaging	Shows what's inside. It...

Details

Return policy: Returnable until Jan 31, 2023

Add a gift receipt for easy returns

Add to List

Have one to sell?

Figure 71: ESP32 Purchased

The ESP32 is the microcontroller we have decided to use for our project. This board is going to allow us to control the servo motors as well as interpret our data, the ESP32 is the nucleus of our project. Initially we were going to only be using the chip for this project when implementing it onto the PCB however through some trial and error and time as a major constraint we decided to use the ESP32 dev board for ease of use

13.2.4. Breadboard

The breadboard for our project will primarily function as our main test circuit we will be able to test our circuits on there by wiring up everything and ensuring that our circuits work before we bring them over to the PCB. One issue we actually ended up having with breadboards is the amount of current it could actually handle was a lot less than what our project was using, this led to the breadboard melting after some components got too hot.

13.2.5. Custom-Designed PCB

Our PCB will be constructed once we have our circuits finalized and working with the entirety of the projected and thoroughly tested on the breadboard. Once the PCB is designed it will be able to handle all our circuit needs and provide the power to all the correct loads.

13.2.6. Shift Registers (74HC595N)

Shift registers are what we use to flip our control signal from one position to the other through a series of flip flops. The component is provided below.

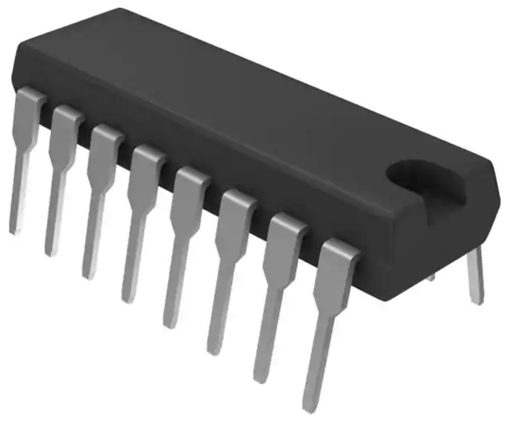


Figure 73: Example Shift Register Render

13.2.7. Power Supply

Our power supply will be used to power our entire project. We need to make sure our power supply is able to supply the proper current to all the servos needed. It will be tested using the breadboard and then be implemented to the final PCB design.

13.2.8. Oscilloscope

The oscilloscope will allow us to test different voltage points throughout the circuit. This will be one of the ways we will be able to troubleshoot our project as far as the voltage is concerned. A common brand of Oscilloscope is provided below. We have already used this device in labs before, so utilizing it again will not take much time.

Figure 74: Example Oscilloscope used for Debugging

13.2.9. Digital Multimeter

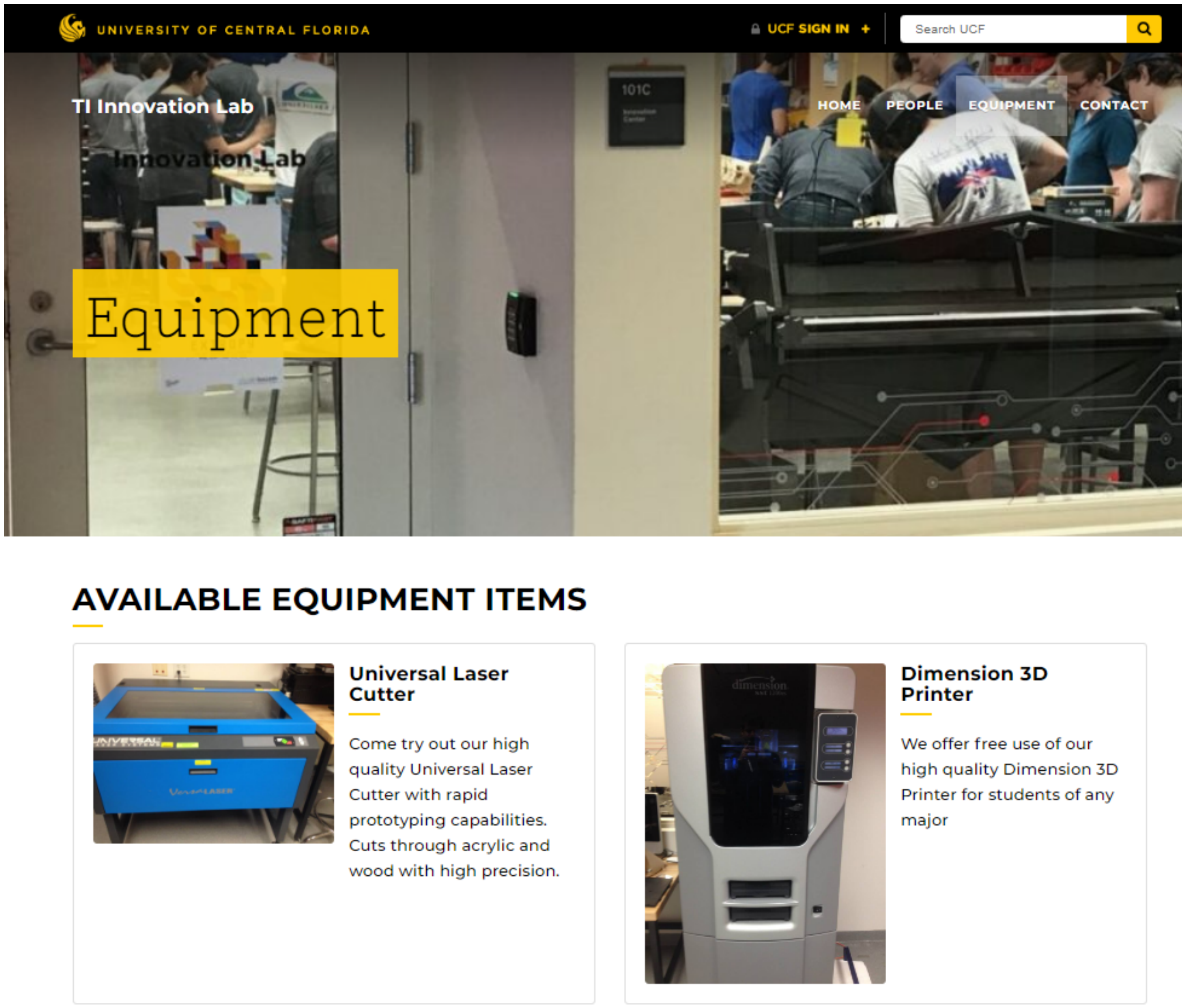
We will be able to use the multimeter in order to test voltages and currents easily. This will be our best way to test our currents to make sure that our currents are what we expect them to be. After working on our project we also found the use of the continuity feature extremely useful for being able to detect shorts or making sure wires were connected correctly.

13.2.10. Other Miscellaneous Electrical Components

We will be using typical electrical components in our project such as resistors, capacitors, and inductors in order to complete our circuit. These items can either be purchased or found in the senior design lab depending on the values able to be found.

13.2.11. Universal Laser Cutter & Dimension 3D Printer

As demonstrated in the figure below, the TI Innovation Lab at UCF includes a Universal Laser Cutter & Dimension 3D Printer. We will be using these devices for our prototyping and finalization phases.



AVAILABLE EQUIPMENT ITEMS

Universal Laser Cutter

Come try out our high quality Universal Laser Cutter with rapid prototyping capabilities. Cuts through acrylic and wood with high precision.

Dimension 3D Printer

We offer free use of our high quality Dimension 3D Printer for students of any major

Figure 75: TI Innovation Lab Equipment
https://www.cecs.ucf.edu/innovationlab/equipment/

14. Consultants, Subcontractors, and Suppliers

At the moment, we do not anticipate needing to connect with any 3rd party consultants, subcontractors, or suppliers outside of resources that are already here and provided for us at UCF. We will be getting help from the TI Innovation lab and Manufacturing lab TAs/technicians for assistance using 3D printer and laser-cutting equipment.

Figure 76: TI Innovation Lab in Engineering Building at UCF
https://www.facebook.com/photo/?fbid=1556858027919411&set=pb.100068901999309.-2207520000

Appendices

Appendix A: References

1. Table 5: Comparison between WiFi, Bluetooth, and BLE
   https://www.bluetooth.com/learn-about-bluetooth/tech-overview/
   https://www.intel.com/content/www/us/en/support/articles/000005725/wireless/legacy-intel-wireless-products.html

2. Figure 1: Playable Note Range for Requirement Specifications
   https://yousician.com/blog/guitar-fretboard-learning-guide

3. Figure 13: Fret Override
   https://nationalguitaracademy.com/how-to-play-bar-chords/

4. Figure 18: Assembly for conversion of rotational servo motion to linear actuation
   https://www.youtube.com/watch?v=MeILaIGI1es

5. Figure 52: MIDI Representation Visualized
   https://blog.landr.com/what-is-midi/

6. Figure 53: Playable Note Range Visualized on Piano (Synthesia)
   https://synthesiagame.com/

7. Figure 59: Cross-Platform Mobile App Example using Flutter
   https://flutterawesome.com/a-easy-to-use-and-customizable-material-flutter-button/

8. Figure 64: Pano Tuner App from Google Play App Store
   https://play.google.com/store/apps/details?id=com.soundlim.panotuner&hl=en_US&gl=US&pli=1

9. Figure 65: Unicorn CPU Emulator Testing for MicroPython
   https://micropython.org/unicorn/

10. Figure 75: TI Innovation Lab Equipment
    https://www.cecs.ucf.edu/innovationlab/equipment/

11. Figure 76: TI Innovation Lab in Engineering Building at UCF
    https://www.facebook.com/photo/?fbid=1556858027919411&set=pb.100068901999309.-2207520000
    https://www.cecs.ucf.edu/manufacturing-lab/

12. Section 3.1.1. Denim Vladimir's Guitar Robot
https://youtu.be/n_6JTLh5P6E

13. Section 3.1.2. TECHNICally Possible's Lego Mindstorms Guitar
https://youtu.be/cXgB3lIvPHI

14. Section 4.2.1.8.1. PCB Standards - Common-Emitter Circuit
https://www.protoexpress.com/blog/ipc-2221-circuit-board-design/

15. Section 4.2.1.1. MIDI Audio Storage Standard
https://www.midi.org/specifications/midi-2-0-specifications/midi2-core

16. Section 5.2.5.1. Micropython and C integration
https://docs.micropython.org/en/v1.19.1/develop/cmodules.html

Appendix B: Copyright Permissions

For the purposes of our project, we have not needed to secure any copyright permissions, as discussed in section **12.1. Legal Issues**.

Appendix C: Purchase Links

10 Servos for $20:
https://www.amazon.com/Dorhea-Helicopter-Airplane-Walking-Compatible/dp/B08FJ27Q1H/ref=sr_1_6?keywords=micro+servo&qid=1667483173&qu=eyJxc2MiOiI1LjU3IiwicXNhIjoiNS4yNCIsInFzcCI6IjQuNzYifQ%3D%3D&s=toys-and-games&sr=1-6

ESP32 Development Board:
https://www.amazon.com/KeeYees-Internet-Development-Wireless-Compatible/dp/B07HF44GBT

Appendix D: Datasheets

Microcontroller
https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

74HC595N Shift Register
https://www.ti.com/lit/ds/symlink/sn74hc595.pdf?ts=1667432577048&ref_url=https%253A%252F%252Fwww.google.com%252F

SNx4HC595 8-Bit Shift Registers With 3-State Output Registers

1 Features

- 8-bit serial-in, parallel-out shift
- Wide operating voltage range of 2 V to 6 V
- High-current 3-state outputs can drive up to 15 LSTTL loads
- Low power consumption: 80-μA (maximum) I_{CC}
- t_{pd} = 13 ns (typical)
- ±6-mA output drive at 5 V
- Low input current: 1 μA (maximum)
- Shift register has direct clear
- On products compliant to MIL-PRF-38535, all parameters are tested unless otherwise noted. On all other products, production processing does not necessarily include testing of all parameters.

2 Applications

- Network switches
- Power infrastructure
- LED displays
- Servers

3 Description

The SNx4HC595 devices contain an 8-bit, serial-in, parallel-out shift register that feeds an 8-bit D-type storage register. The storage register has parallel 3-state outputs. Separate clocks are provided for both the shift and storage register. The shift register has a direct overriding clear ($\overline{\text{SRCLR}}$) input, serial (SER) input, and serial outputs for cascading. When the output-enable ($\overline{\text{OE}}$) input is high, the outputs are in the high-impedance state.

Device Information

PART NUMBER	PACKAGE[1]	BODY SIZE (NOM)
SN54HC595FK	LCCC (20)	8.89 mm × 8.89 mm
SN54HC595J	CDIP (16)	21.34 mm × 6.92 mm
SN74HC595N	PDIP (16)	19.31 mm × 6.35 mm
SN74HC595D	SOIC (16)	9.90 mm × 3.90 mm
SN74HC595DW	SOIC (16)	10.30 mm × 7.50 mm
SN74HC595DB	SSOP (16)	6.20 mm × 5.30 mm
SN74HC595PW	TSSOP (16)	5.00 mm × 4.40 mm

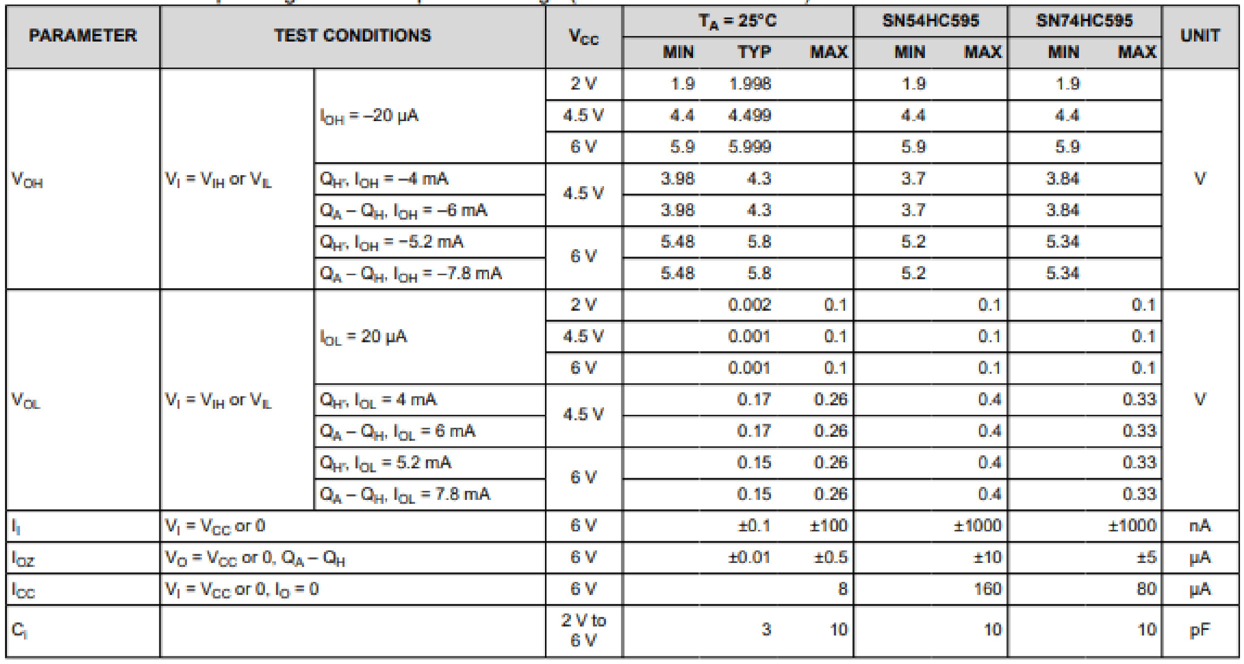
(1) For all available packages, see the orderable addendum at the end of the data sheet.

6.3 Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)[1]

			SN54HC595			SN74HC595			UNIT
			MIN	NOM	MAX	MIN	NOM	MAX	
V_{CC}	Supply voltage		2	5	6	2	5	6	V
V_{IH}	High-level input voltage	V_{CC} = 2 V	1.5			1.5			V
		V_{CC} = 4.5 V	3.15			3.15			
		V_{CC} = 6 V	4.2			4.2			
V_{IL}	Low-level input voltage	V_{CC} = 2 V			0.5			0.5	V
		V_{CC} = 4.5 V			1.35			1.35	
		V_{CC} = 6 V			1.8			1.8	
V_I	Input voltage		0		V_{CC}	0		V_{CC}	V
V_O	Output voltage		0		V_{CC}	0		V_{CC}	V
Δt/Δv	Input transition rise or fall time[2]	V_{CC} = 2 V			1000			1000	ns
		V_{CC} = 4.5 V			500			500	
		V_{CC} = 6 V			400			400	
T_A	Operating free-air temperature		–55		125	–40		85	°C

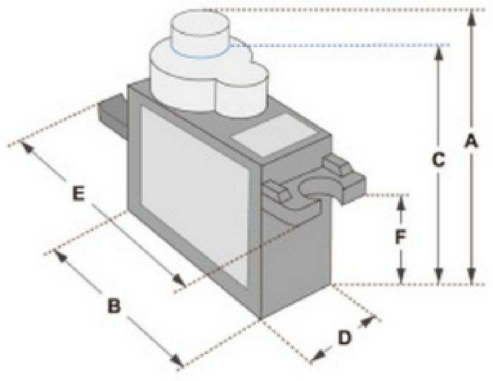
6.5 Electrical Characteristics

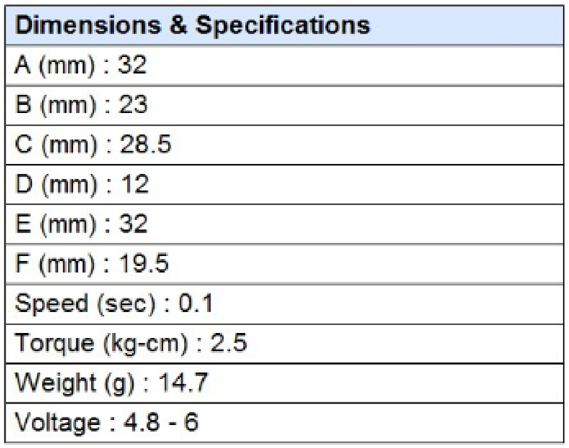


over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS		V_{CC}	$T_A = 25°C$			SN54HC595		SN74HC595		UNIT
				MIN	TYP	MAX	MIN	MAX	MIN	MAX	
V_{OH}	$V_I = V_{IH}$ or V_{IL}	$I_{OH} = -20\ \mu A$	2 V	1.9	1.998		1.9		1.9		V
			4.5 V	4.4	4.499		4.4		4.4		
			6 V	5.9	5.999		5.9		5.9		
		$Q_{H'}$, $I_{OH} = -4$ mA	4.5 V	3.98	4.3		3.7		3.84		
		$Q_A - Q_H$, $I_{OH} = -6$ mA		3.98	4.3		3.7		3.84		
		$Q_{H'}$, $I_{OH} = -5.2$ mA	6 V	5.48	5.8		5.2		5.34		
		$Q_A - Q_H$, $I_{OH} = -7.8$ mA		5.48	5.8		5.2		5.34		
V_{OL}	$V_I = V_{IH}$ or V_{IL}	$I_{OL} = 20\ \mu A$	2 V		0.002	0.1		0.1		0.1	V
			4.5 V		0.001	0.1		0.1		0.1	
			6 V		0.001	0.1		0.1		0.1	
		$Q_{H'}$, $I_{OL} = 4$ mA	4.5 V		0.17	0.26		0.4		0.33	
		$Q_A - Q_H$, $I_{OL} = 6$ mA			0.17	0.26		0.4		0.33	
		$Q_{H'}$, $I_{OL} = 5.2$ mA	6 V		0.15	0.26		0.4		0.33	
		$Q_A - Q_H$, $I_{OL} = 7.8$ mA			0.15	0.26		0.4		0.33	
I_I	$V_I = V_{CC}$ or 0		6 V		±0.1	±100		±1000		±1000	nA
I_{OZ}	$V_O = V_{CC}$ or 0, $Q_A - Q_H$		6 V		±0.01	±0.5		±10		±5	μA
I_{CC}	$V_I = V_{CC}$ or 0, $I_O = 0$		6 V			8		160		80	μA
C_i			2 V to 6 V		3	10		10		10	pF

Servo





Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

From SG90 micro-servo datasheet