Keynetic

Sun A Cho, Katherine Dettmer, Lance Yarlott

Department of Electrical and Computer Engineering,
Carnegie Mellon University

***Abstract*—A system capable of** allowing users to play piano in a new, fun, and simple way. Using computer vision (CV), it captures the location of the user's hand and translates that into notes that will be played on a keyboard. From there, the system will analyze notes played by the user and generate a matching chord progression. This pattern matching can be adapted to long-form playing, or simply to short periods. Additionally, due to the physical nature of the system and extensible software framework, it can be extended to work on any size keyboard given enough time and resources.

***Index Terms*—**Beat, Chord, Diatonic Note, Feature Matching, Musical Key, Solenoid, Subdivision, Time Signatures

I. Introduction

This project builds a system that provides a method to play the piano through movement, without the need to press physical keys. It is a mechanically actuated keyboard that is managed by a microcontroller, using computer vision to produce notes based on user movement.

Musical instruments and music are a large part of human life, and almost all children, should they not learn to play an instrument, will interact with music in some shape or form. Music is a method of emotional expression, creativity, and is often embedded in culture. In fact, learning to play an instrument has been shown to improve cognitive development, memory, and concentration. These benefits extend far later into life, with pianists being known to have improved memory function later in life. However, not everyone has an equal opportunity to play an instrument- and in our system's case, the piano.

There are many different reasons why someone may not be able to play the piano the traditional way, as it requires dextrous control of one's fingers to press the keys and play moving melodies. Our system aims to bridge the gap for those who wish to play piano but cannot for one reason or another. The user will utilize color placement to signify notes, which even allows amputees or those with limb differences to learn how to read music and play it on the piano. By using color and computer vision, most people capable of movement will find themselves able to play the piano. From that moment forward, it will simply be a mental task to read music.

In the current market, there are no devices like ours. In society, it is a common notion that only those who "can" play piano should be able to play it. These are justified largely by the lack of performable repertoire, as well as the immense difficulty in creating a system that would allow those who are unable to even put their hands on the keys to be able to play. In this sense, our system will excel in allowing players of all kinds to do just that: play.

We strive to create a system that is both fun and intuitive to use while allowing those who cannot traditionally play music to engage with it and learn about it in a hands-on manner.

II. Use-Case Requirements

Our system has several strict but straightforward requirements:

On the musical end, we require that any generated notes stay within their home key (in our case, C major) 100% of the time. Given that we are only playing the white keys, this can be relatively simple. However, when considering the fact that we generate chord progressions, there's a small chance that we can slip into a different "mode" of another key (for example, A minor contains all the notes of C major, but starts on A). Generated chords should also avoid large numbers of dissonances with user-played notes when at all possible. Musically speaking, a dissonance is defined as the 2nd, 4th, and 7th intervals (there are more, but we are unable to play them given our designed hardware constraints). If given a sample chord progression, the software should be able to generate a melody with chord tones that fall on the beat 100% of the time.

We also require that all inputs received from the camera be parsed and passed to the controller within 1 second. This requirement is somewhat lax, and our real goal is to send signals as fast as possible. As such, our minimum requirement is 1 second, while our goal requirement is as close to zero as the hardware and software are capable of.

Additionally, we require that hands (more specifically the colors on them) should be detected at least 90% of the time. This is to ensure a smooth user experience. It also includes any trails the hand makes, and lost data points should be interpolated without fail.We also require that the user is farther away than 4 ft and closer than 7ft to use the project.

Our hardware should be able to safely support at least 14 solenoids, with at least 4 activated simultaneously, and support additional quick actuation of a fourth solenoid. This allows us to play simple triad chords along with a melody line. On top of the hardware systems supporting 14 actuators safely and soundly, it should also be an easy installation for the users since it is one of our crucial differences from similar products in the market (self-play piano), which will be identified in Section 9. Related Work. Making the hardware system easy to install (with a mounting system) will maximize the accessibility of Keynetic.

Lastly, our integrated system has to have the least lag between the feature/ motion detection and the actual playing of the keyboard. In order to maximize the user experience of playing an instrument and making music, we believe that the less latency there is between the motion detection and the actuators playing the keyboard is the crucial part of making the experience more enjoyable for the users. Currently, we aim to achieve 500 ms latency but hope to decrease the latency as close to 0 ms as possible.

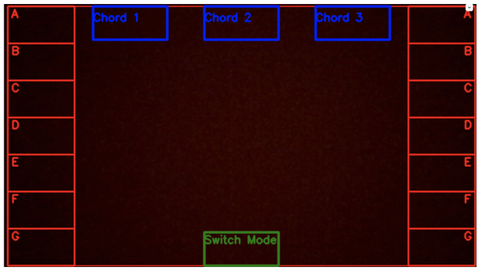
We strive to provide an enjoyable experience to our targeted audience and provide them with the opportunity to create and play music even if they do not have the traditional physical ability to do so.

III. Architecture and/or Principle of Operation

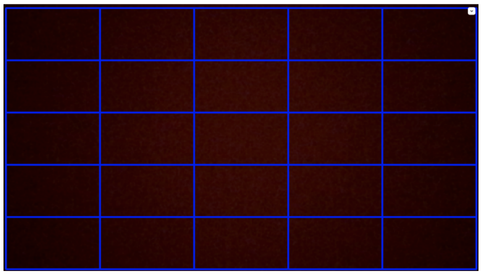
Our system can be broken down into three major subsystems. These are vision software, music software, and hardware as shown in the block diagram in Figure 3 (d).

The software system will use computer vision to process a live camera feed of someone with certain color markings on their body. There is a camera feed displayed to the user, with two different graphical interfaces for the user to interact with. The first one is a simple note generation mode - the screen has boxes with displayed notes on each side, with chords on the top. On the bottom is a box that says 'Switch View.' The user triggers these notes/actions by putting the designated color into one of the boxes for a short amount of time.

The Switch View button will lead the user to another interface, this one is just the video feed broken into a 5x5 grid. The display is what we call the 'generative' mode. The generative mode will allow the user to make movements, and it records the sequence. The sequence of grid boxes the colors go through is then matched to predetermined sequences that we have mapped. If one of them matches, a few musical notes are played based on our mapping.



(a) note-playing mode



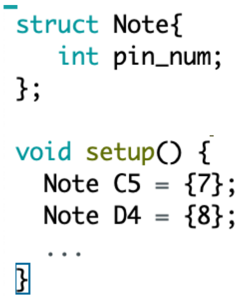
(b) generative mode

In terms of musical software, our main focus is on sending notes to our keys in a timely manner. This is done using an Arduino. The general software loop for this is incredibly straightforward. We simply need to take input passed from our camera parser and then send that to the Arduino, which will actuate the keyboard's keys. There are also two more components that will be used when writing our music software.

The first is chord generation. This involves recording the user's past note inputs, dividing them by measure, and then taking a statistical measurement of the average note played for every measure in a predefined phrase (this is generally 4 or 8 measures, though can vary in length). This measurement can be done using Bayesian Updating. In C major, there are 7 possible chords, and we can assign each a probability that they are being played over at any given time. From our Bayesian updating algorithm, we can choose the chord that has the highest probability of being correct, and send that to the Arduino to be played. We can also tweak these probabilities based on the current chord. For example, in jazz, if one is playing a D minor chord, they would tend to move to a G major chord, then finally back down to C major. This is known as an ii-V-I progression, named for the intervals that the chords start on (C being the first interval). You can even "nest" these progressions, so an IV-vii-iii-vi-ii-V-I progression is just a series of three modified ii-V-I progressions (F-B-E, E-A-D, D-G-C). These can be readily utilized in our software and implemented to make the piece move and back up the player. Not every progression must make sense, but it certainly helps the player when they do.

There is also the issue of timing. The Arduino will, in most cases, directly manage all timing-related issues. This involves separating time into measures, beats, and subdivisions. This generally relies on the internal clock of the Arduino. We can precompute note lengths and then directly use these to determine when and how long to play for. The calculation itself is simple and just involves a conversion between beats per minute and seconds per note (an inversion and time conversion).

The hardware system will consist of the following: a row of solenoids – responsible for playing the actual keyboard – and a proto-board that will support the solenoids. As of now, we are planning to play up to 2 octaves and at most 4 keys concurrently at each time. Given that the hardware system sits below everything else – it will receive all of the required data from the above systems to control the solenoids. For instance, the vision software system will detect features, which will then transfer that information to the music software system. The music software system will generate a pattern of notes (using MIDI keys) and send it to Arduino, as mentioned above. Then, using this data, the Arduino can either turn on or off the corresponding solenoid(s). Each Arduino digital pin will be assigned to a solenoid and will be contained in an instance of a Note struct in the Arduino program – making it more object-oriented as shown in Figure 3 (c). Having an object-oriented Arduino program will allow us to have a simpler (fewer lines as well) code to control 14 solenoids. Furthermore, the music generation data from the music software system will also dictate the duration of every note, which will then control the solenoids accordingly. However, before building a full, two-octave system, we have already built a test system with one solenoid and now will scale up to building a 7-solenoids-system. Once we succeed in building a 7-solenoids-system, we will, then, build a complete 14-solenoids-system. This scaling schedule is there because we believe that debugging on a smaller scale is faster than starting with a full 14-solenoids system. However, while building the 7-octaves system, we will start integrating all three systems to at least play 7 notes.

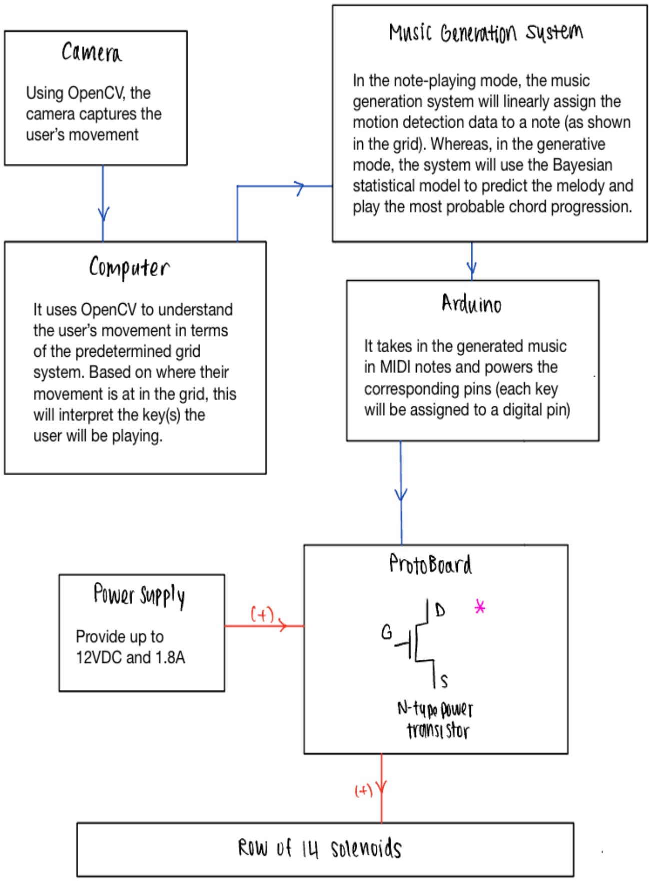


```
struct Note{
   int pin_num;
};

void setup() {
   Note C5 = {7};
   Note D4 = {8};
   ...
}
```

(c)

Fig. 3. Example of such system:. (a) vision software – note-playing mode. (b) vision software – generative mode (c) hardware – Arduino system's struct



Camera

Using OpenCV, the camera captures the user's movement

Computer

It uses OpenCV to understand the user's movement in terms of the predetermined grid system. Based on where their movement is at in the grid, this will interpret the key(s) the user will be playing.

Music Generation System

In the note-playing mode, the music generation system will linearly assign the motion detection data to a note (as shown in the grid). Whereas, in the generative mode, the system will use the Bayesian statistical model to predict the melody and play the most probable chord progression.

Arduino

It takes in the generated music in MIDI notes and powers the corresponding pins (each key will be assigned to a digital pin)

ProtoBoard

Power Supply

Provide up to 12VDC and 1.8A

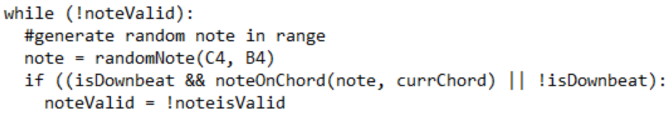
Fig. 3. (d) The full block diagram that represents our Keynetic system

IV. Design Requirements

As previously discussed, we require that our pipeline takes less than 1 second to traverse, with a goal of taking the time as close to 0 seconds as possible (our MVP goal is 500ms). This means that, from the moment an image frame is pulled, data is sent from the computer to the Arduino, and a command signal is sent from the Arduino to the solenoids within 1 second. This requirement should require no further explanation.

Another requirement for the software is to have the person standing farther than 4 ft away from the camera, but closer than 7 ft from the camera. This is so the color can be accurately detected. The computer will try to determine how far the user is from the camera and prompt them to move closer or farther away.

To further explain the requirement regarding chord tones, a scale consists of 7 distinct notes. For C major, they are [C, D, E, F, G, A, B]. Our chords will be made up of 3 notes and will be in a standard triad format. C major is [C, E, G], D minor is [D, F, A], and so on. Whether a chord is major or minor is not relevant to the software. At the start of a new phrase, it is generally best to return to the home chord (for us this is the C major triad), to not confuse players. This is not a hard rule in music, but we will make it one for simplicity's sake. So, we will require that the start of a phrase returns to C major 100% of the time. For melodies, we want to make sure that notes fall on chord tones on the downbeat. With a simple 4/4 time signature (4 notes of length ¼, or 4 quarter notes per measure), this means that a chord tone should fall on beats 1, 2, 3, and 4. The chord tones are simply the notes that make up the current chord. This requirement is easy to satisfy as it is a generation constraint. A small bit of pseudocode is written below to illustrate this:



```
while (!noteValid):
   #generate random note in range
   note = randomNote(C4, B4)
   if ((isDownbeat && noteOnChord(note, currChord) || !isDownbeat):
      noteValid = !noteisValid
```

V. Design Trade Studies

To meet the use case requirements, we had to make some trade-offs as such:

A. Design Specifications for the Hardware System

The solenoids that we chose for this project are from Adafruit and the manufacturer recommends that we power it at 12W rating. However, given that one of our use case requirements is to minimize the power used in our hardware system, I conducted my research to find out exactly how much power it would take to play a key on the keyboard, I concluded that the minimum requirements of voltage and current are 10V and 0.6A. To come to that conclusion, I conducted my research by varying voltage from 10V to 12V and varying current from 0.5A to 1A as shown in Table I. Pairs of V and I for Solenoids (Hardware System). The lowest pairs that I found were 11V and 0.6A and 10V and 0.6A. Once I found that 10V and 0.6A were enough to power a solenoid, that's when we decided to use 6 Watts to power the solenoid to play the keyboard – the graphical distribution of power rating can be found in Figure 5 (a).

B. Design Specifications for Music Generation

In Figures 5 (b) – (g), we can see an example of what one would expect when playing notes. We generate probability measures for what chord the user is playing over, and continuously update that as new data comes in. This data will collect over time and become more accurate with every new data point. This specific updating algorithm helps us avoid dissonances when playing, as per our requirement regarding them. Using a C major chord, a dissonance is considered as a D, F, and B. It will be seen that two dissonances are played, but these are unavoidable and a natural part of music. There is no reason for dissonances to be avoided completely, it is just that sticking on them can sound unpleasant to players and listeners alike.

Table I. Pairs of V and I for Solenoids (Hardware System)

V (Voltage)	I (Amps)	Playing	Power Rating (V * I)
12	1	1	12
12	0.9	1	10.8
12	0.8	1	9.6
12	0.7	1	8.4
12	0.6	1	7.2
12	0.5	1	6
11	1	1	11
11	0.9	1	9.9
11	0.8	1	8.8
11	0.7	1	7.7
11	0.6	1	6.6
11	0.5	0	0
10	1	1	10
10	0.9	1	9
10	0.8	1	8
10	0.7	1	7
10	0.6	1	6
10	0.5	0	0

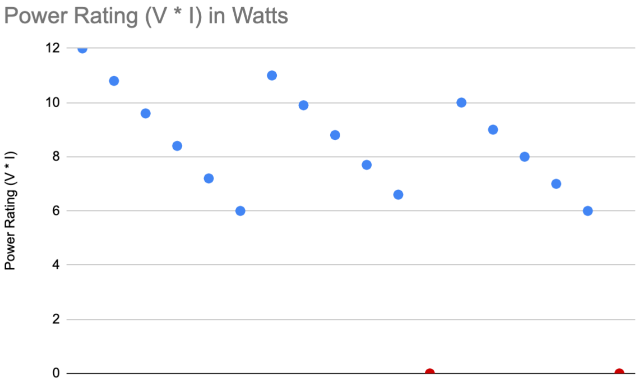


Fig. 5. (a) Graphical figure of the different power rating and red dots represent 0W as in the solenoids do not have enough power to draw from

Figure 5 (b) – (g). Bayesian Updating Example Charts (Music System)

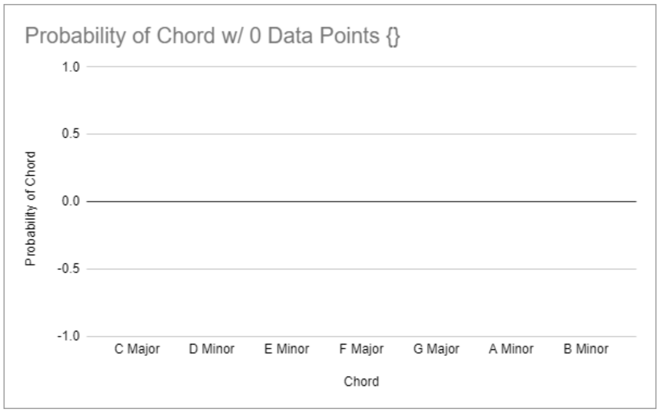


Fig. 5. (b)

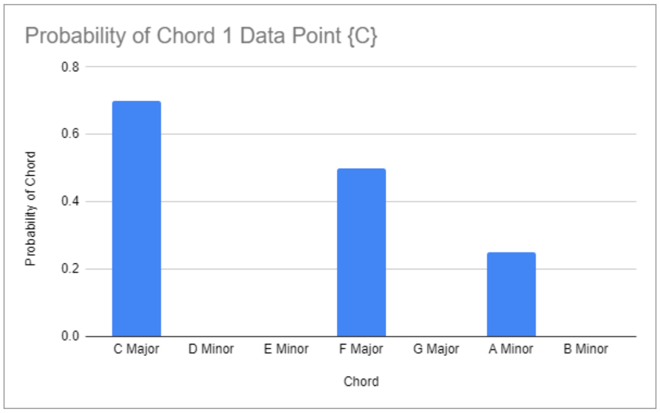


Fig. 5. (c)

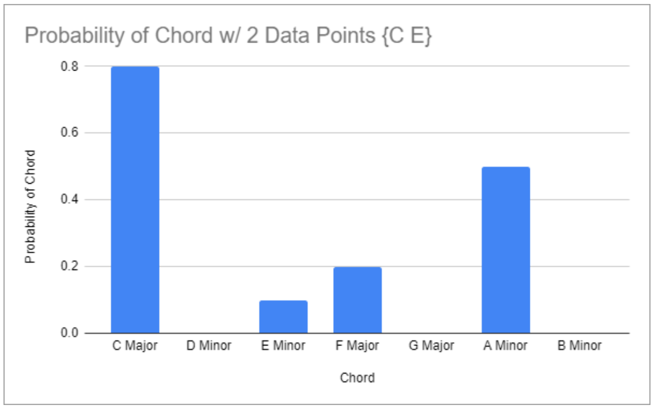


Fig. 5. (d)

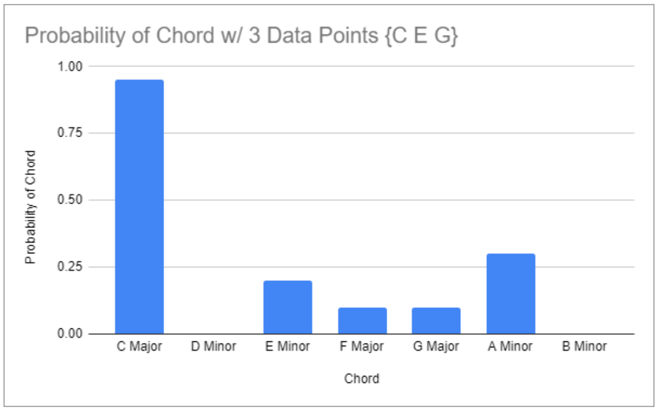


Fig. 5. (e)

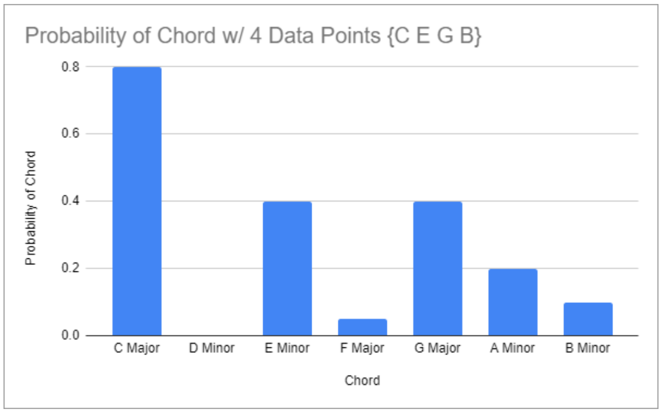


Fig. 5. (f)

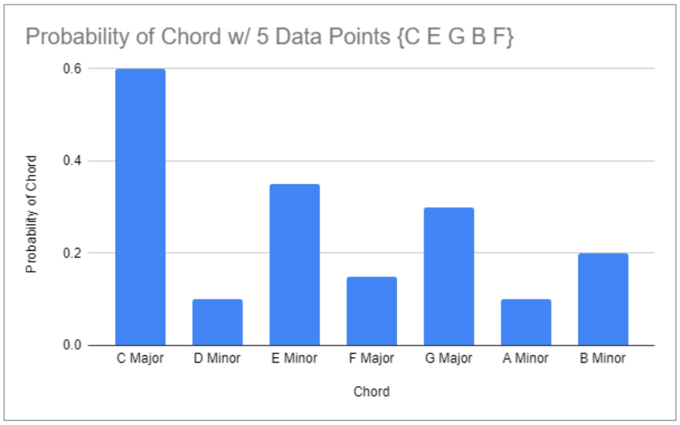


Fig. 5. (g)

Fig. 5. (b) – (g). Using example values, these charts show the relationship between notes played and the predicted chord based on them. These values are handpicked to show that convergence on a single chord can happen quickly. The first note played might be the root note, so the probability of the chord being based on that is high. However, other chords use said note, so their probability will not be 0. This continues over time and predictions will only grow stronger. If we carry data between phrases, these predictions will either grow stronger or weaker, but will always match the player's actions.

VI. System Implementation

A. *Software System*

The software system consists of a camera that takes in video input and feeds it back to the computer. OpenCV is then used to continuously analyze the images received by the video camera. The user will use colors to play notes (for example, a red glove on one hand and a blue glove on the other), and the computer will continuously record the location of the designated color.

The video is displayed for the user, with an interface overlaying it. The interface for the note-generation mode has boxes for each note in the two octaves and for 3 chords and a 'Switch Mode' box. If the computer determines that one of the designated colors has been in the boxes, it will consider that as 'clicking' the box.

The software interface will consist of getLocation() for each color, and getContours() which will isolate the colors from the image. It also needs translateToNote() to translate between box positions and piano notes. Finally, matchPattern() will be needed for the generative mode, in order to match the user's movements to specific note sequences.

By 'clicking' on the Switch Mode box, the interface overlaid on the video feed switches to the gridded 'generative mode' interface. We will have several patterns mapped into a data structure with corresponding note patterns. As the user moves the colors around the grid, the grid boxes that they pass through will be recorded in a different data structure, and if they create any of the mapped patterns, the corresponding note sequence will be played.

B. Music and Note Generation Subsystem

The music system will be split between the computer and the Arduino.

On the computer side, it will receive a value (or values) from the image interpreter, and convert that input into a note pitch and duration. This will then be simplified for transfer speed and passed to the Arduino.

The Arduino will take simplified data passed from the computer, and reconvert it to the specified pitch and note duration. It will then be sent to a simple sequencer algorithm that will place it in time with the rest of the music. This algorithm is relatively simple.

Given a starting time, it will calculate the length of measures, and then base note starting times off of that. If the program starts at time 0, and a measure is 4 seconds long, then it will know that if a half note is received and is supposed to be played on beat 2, it will start at 1 second and continue until 3 seconds. The pitch selection itself is a matter of selecting a digital out pin to flip the value of on the Arduino. There is also a chance that notes can cross between measure boundaries, but as this happens in normal music, it is of no concern as long as the Arduino can keep time throughout.

The equation to determine if a note should be playing is as follows:

$$t(s,\, d) = (s - T) + d > 0 \wedge (s - T) + d < d$$

With s being the start time of the note, d being the duration, and T being the current global time.

To generate starting times we will use the following equation:

$$t = B * L + measureFloor(T)$$

With B being the representation of the current beat in the program, and L being the length of said beat's subdivision. *measureFloor(T)* is a function that returns the start time of the measure. So, if we are at 60 bpm, on beat 2, at time 6, we can generate a new quarter note on beat three by using this equation. This algorithm can be generalized to take in b as a parameter:

$$t(b) = (b + B) * L + measureFloor(T)$$

Where b represents the beat offset from the current beat. We shall not generate new notes on a beat that is already passing us by, so the equation will be constrained to always generate times at $t(B+1)$.

The Bayesian Updating algorithm will be carried out on the computer and uses the following equation:

$$P(H \mid d_1) = \frac{P(H)P(d_1 \mid H)}{P(d_1)}$$

P(H) referencing the probability of a given chord being correct prior to any new information. $P(H|D_1)$ refers to the likelihood of a chord being correct given the addition of new data (in our case, notes). The value will be updated on a per-measure basis, every time a new note is passed from the camera to the computer.

C. Hardware System

While our goal is to build a 2-octaves system, we are currently in the process of building a smaller-scale system including only one solenoid. In Figure 6 (a), you can find the circuit diagram of the test system that we have built and successfully tested. In Figure 6 (b), you can find the actual circuit we built on a breadboard for testing purposes. During the test, we assessed the power of the solenoid at each power rating (ranging from 12W to 6W) and determined the ideal power rating for the solenoid to successfully play a key on the keyboard. As a result, we have decided to use 10V and 0.6A to control each solenoid – and since we are only allowing 4 solenoids to turn on at a time, the total power of the hardware system would be 10V and 2.4A or 24W. Given that this system is quite "heavy" for a regular breadboard, we also decided to create our own proto-board to support the system. Eventually, we will be scaling our current test design to design a proto-board with 14 solenoids, 14 diodes (model:1N4001), 14 transistors (model: irfb7440pbf), 14 10K resistor, and an Arduino (Uno). To explain the components, the transistor will be used as a switch to control the solenoids based on the voltage output from the Arduino (digital pin assigned to each solenoid). There will be a resistor between the Arduino's output and the gate of the transistor to manage the unnecessary current flow (which will be varying from 5V to 0V). Similarly, I decided to add a diode in between the power supply (to power the solenoids) and the transistor to reduce the risk of ruining the solenoid from a potential voltage spike. Given that we are switching from 10V to 0V (and whatever the current may be needed to control the hardware systems at the time), there's a possibility of voltage spikes. Adding a diode also manages a potential current "mis-flow" in the circuit, since this is a DC system and diodes restrict the current to flow in one direction, and that's a good measure to have when building a pretty heavy system with more than 20 Watts.

Also, this design will eventually require a mounting system – which will be built using MDF sheets with the help of a laser cutter in Tech Spark. We will be lining up the solenoids horizontally – next to one another – right above the key, in order to minimize the power needed to push the keys as shown in Figure 6 (c).

But, for now, in the next two weeks, I will be focusing on scaling the system to be half of the final design – only 7 solenoids to support an octave.

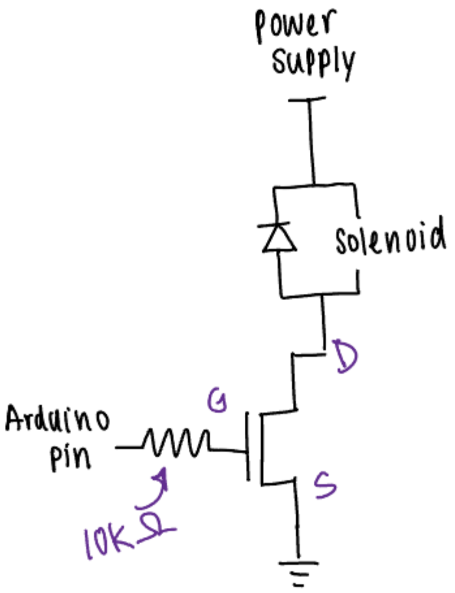


Fig. 6 (a) Test Circuit Diagram

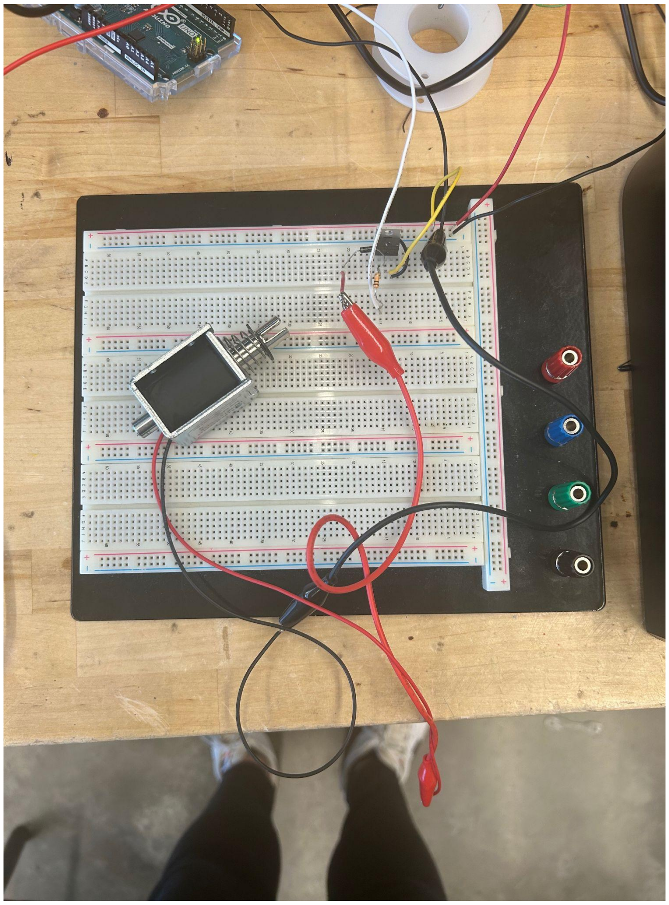


Fig. 6 (b) Test Circuit

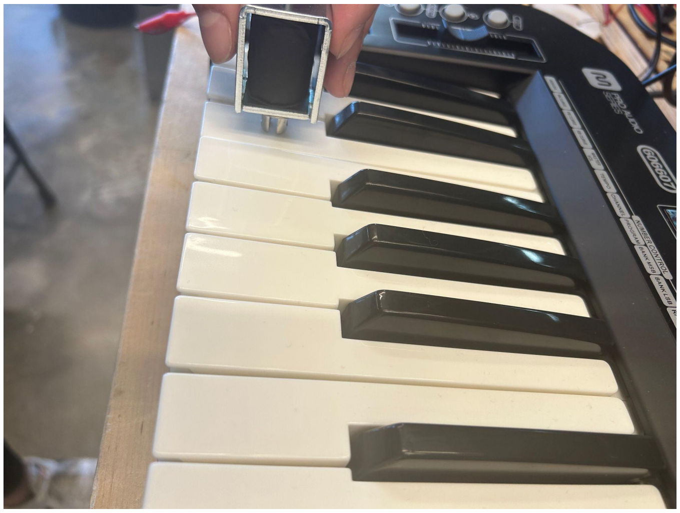


Fig. 6 (c) The location of the solenoid in relation to the keyboard

VII. Test, Verification and Validation

The entire system can be tested both by subsystem and holistically as an integrated unit. The subsystem tests are a prerequisite full-system test and should be completed first. Once they are complete, we will begin integrating systems and verify that they work together as expected. This will be used as a jumping-off point to validate our requirements.

A. *Tests for Software Requirements*

To test the requirement of having less than 1-second latency between signaling a note and the note playing, we can simply start a timer when the person gestures and stop it when the physical note plays.

To test how accurately hands are detected, we can continuously attempt to signal certain notes and record the percentage of times the computer is accurate in translating the user's desired note.

To test the enforcement of the person standing within 4-7ft of the camera, we will have someone move around that scale and see how accurate the computer is in detecting how far they are.

B. *Tests for Music Requirements*

A simple but effective test is to run a metronome on the Arduino alongside a real metronome. If the Arduino strays from the metronome, something is going wrong regarding floating point error or rounding issues.

For Bayesian Updating, we can simply pass in an array of predetermined notes and confirm that the output matches our expected output.

Note sending will be tested by sending power signals from the Arduino to the solenoids, and this will be a joint task between hardware and software.

C. Tests for Hardware Requirements

In order to test if the hardware system is working properly or not, we will be playing a test piece such as "Twinkle, Twinkle Little Star." This music will be manually inputted into the Arduino system for testing purposes. Once the system successfully plays the music, we will know that the hardware system, at least individually, works correctly.

VIII. Project Management

A. Schedule

We attached the most-updated Gantt chart in Figure 8.

B. Team Member Responsibilities

Katherine is responsible for the software part of the project. This includes all of the OpenCV work, detecting movements and gestures, and designing the user interface for the video feed overlay.

Lance is responsible for the bulk of any music-related code, from timing to note generation. He is also partly responsible for the verification of the system's hardware.

Sun A is responsible for building the hardware system of actuators to play the piano. She will also build a mounting system for the actuators to sit above the keyboard.

C. Bill of Materials and Budget

You can refer to Table II. Bill of Materials at the end of the report. You will find a list of items purchased and used with the quantity, price, and manufacturers.

D. Risk Mitigation Plans

One of the risks is being able to connect the entire system and have the solenoids actually be able to reliably press the notes hard enough to play them. Our mitigation plan for this is having a backup music generation plan using just the software. We can generate the music using the computer, rather than the solenoids, so the product is still usable.

Another risk is getting OpenCV working in a way that benefits the project. This is a risk because no one in the team has worked extensively with OpenCV before, so there is a significant learning curve. However, there are a lot of different ways to turn detected image features into notes, so our mitigation plan for this is to just change how we detect someone.

This subsection should identify the critical risk factors in your design and plans for how you will manage that risk. This is where you will list the known unknowns. For example, no one in the team has ever previously designed an UART for this FPGA, so our primary risk is getting sensor data to the FPGA to take advantage of its rapid computing capability.

No team members have had the opportunity to design cyclic, or in our case, rhythmic timing algorithms. Music is based on rhythm, and as such is subject to strict timing requirements. There is a risk of our rhythm slowly slipping over time and keeping playing from being consistent. However, to curb the risk presented by this dilemma, we plan on ensuring that our algorithms minimize floating point error where possible.

There is also the possibility that we will have to deal with the clocks of the Arduino and the computer not being synced. This could cause issues regarding timing, which have already been touched upon above. One possible solution is to send notes with a generic timestamp attached to them. For example, we could receive a note 2ms after beat 2 in a measure, and mark that for beat 3 based on our current subdivision.

For the hardware system, we already had to not use the full, recommended power rating to control the solenoids – we were supposed to use 12 W but we are now using 6 W. However, if there's any issues with the amount of power we are using for the hardware system, we will have to scale it down to only play one octave.

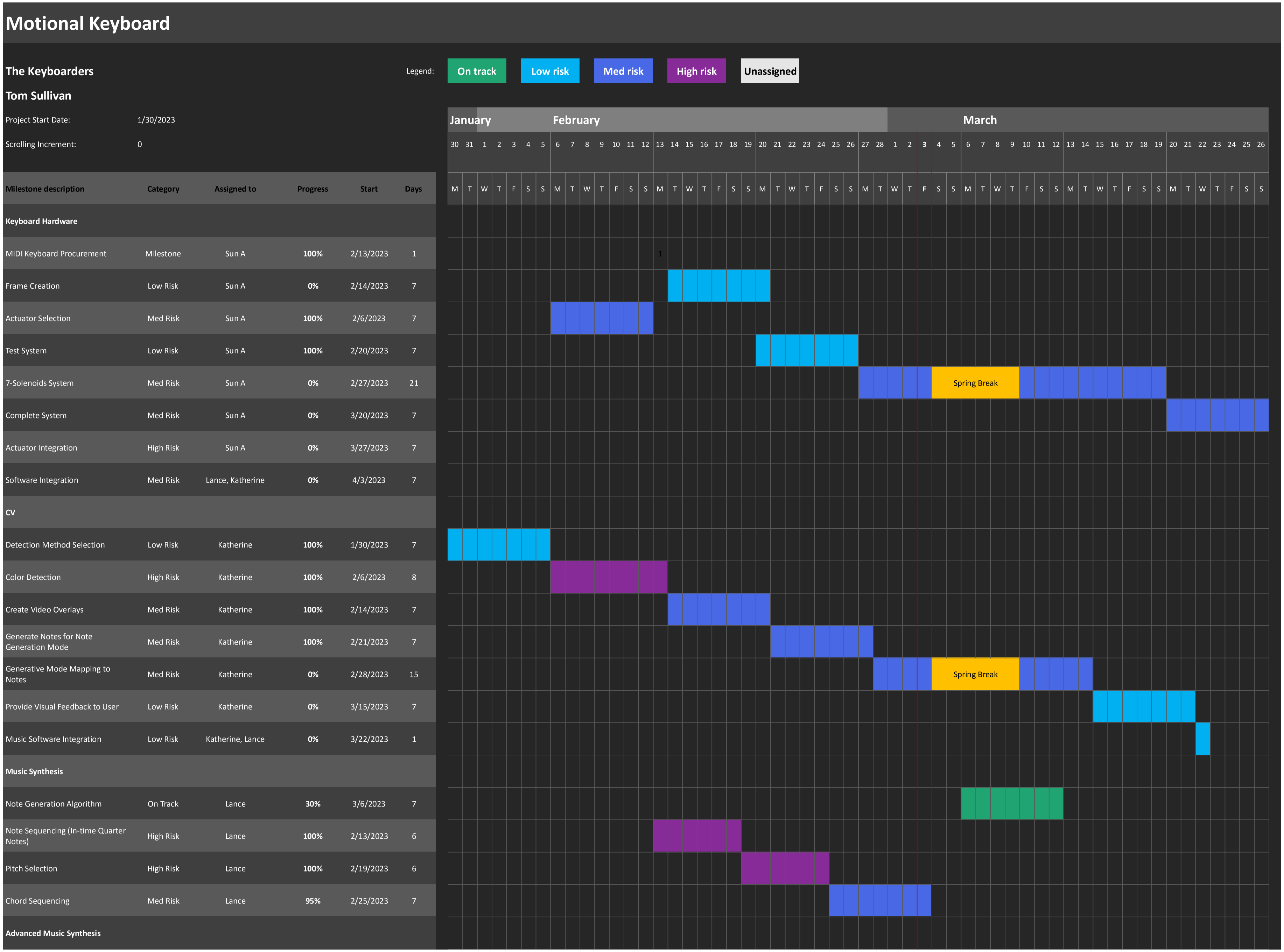
IX. Related Work

There are similar products already in the market such as self-play piano, which has to be installed internally by a professional. However, our hardware system does not require an internal setup and only requires that it has to be mounted on top of the piano. The self-play piano also is not accessible to people who do not have the traditional means of playing an instrument – i.e. music enthusiasts with physical disabilities or impairments. This is where our project is different from the self-play piano where we use feature detection to create music. Similarly, there are other groups in 18500 who are creating a similar hardware system but have drastically different use case requirements.

X. Summary

Our system will open a new world of piano playing to those that may have been unable to do so in the past. With the use of modern technologies, we will be able to allow these users to play on a keyboard with minimal interference. Although it won't perfectly capture the feel of playing piano, it should still feel like a real instrument, rather than a toy or gimmick.

As for our implementation challenges regarding design requirements, we do have to be careful regarding the timing of the notes. Humans are surprisingly adept at detecting small variations in rhythm (though this only accounts for steady beats, which happens to be what most beginners play), and we need to account for that to ensure that the user experience is as close to perfect as possible.



Motional Keyboard

The Keyboarders

Tom Sullivan

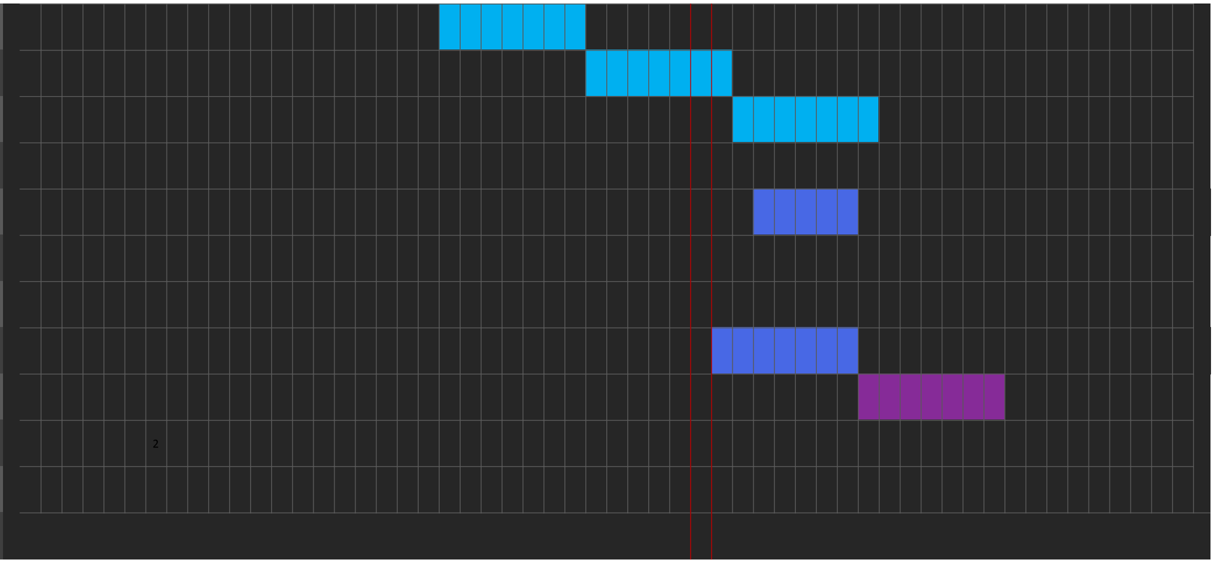
Project Start Date: 1/30/2023

Scrolling Increment: 0

Milestone description	Category	Assigned to	Progress	Start	Days
Keyboard Hardware					
MIDI Keyboard Procurement	Milestone	Sun A	**100%**	2/13/2023	1
Frame Creation	Low Risk	Sun A	**0%**	2/14/2023	7
Actuator Selection	Med Risk	Sun A	**100%**	2/6/2023	7
Test System	Low Risk	Sun A	**100%**	2/20/2023	7
7-Solenoids System	Med Risk	Sun A	**0%**	2/27/2023	21
Complete System	Med Risk	Sun A	**0%**	3/20/2023	7
Actuator Integration	High Risk	Sun A	**0%**	3/27/2023	7
Software Integration	Med Risk	Lance, Katherine	**0%**	4/3/2023	7
CV					
Detection Method Selection	Low Risk	Katherine	**100%**	1/30/2023	7
Color Detection	High Risk	Katherine	**100%**	2/6/2023	8
Create Video Overlays	Med Risk	Katherine	**100%**	2/14/2023	7
Generate Notes for Note Generation Mode	Med Risk	Katherine	**100%**	2/21/2023	7
Generative Mode Mapping to Notes	Med Risk	Katherine	**0%**	2/28/2023	15
Provide Visual Feedback to User	Low Risk	Katherine	**0%**	3/15/2023	7
Music Software Integration	Low Risk	Katherine, Lance	**0%**	3/22/2023	1
Music Synthesis					
Note Generation Algorithm	On Track	Lance	**30%**	3/6/2023	7
Note Sequencing (In-time Quarter Notes)	High Risk	Lance	**100%**	2/13/2023	6
Pitch Selection	High Risk	Lance	**100%**	2/19/2023	6
Chord Sequencing	Med Risk	Lance	**95%**	2/25/2023	7
Advanced Music Synthesis					

Fig. 8. Schedule example with milestones and team responsibilities

Subdivions	Low Risk	Lance	**100%**	2/19/2023	7
Syncopation	Low Risk	Lance	**100%**	2/26/2023	7
Chords (Non-Triads)	Low Risk	Lance	**20%**	3/5/2023	7
Verification and Validation					
Unit Testing	Med Risk	All	**40%**	3/6/2023	5
Hardware Testing	Low Risk	Sun A	**45%**	4/3/2023	5
Software Testing	Low Risk	Katherine, Lance	**30%**	4/10/2023	5
Music Synthesis Testing	Med Risk	Lance	**50%**	3/4/2023	7
Integration Testing	High Risk	All	**0%**	3/11/2023	7
Slack Creation	Goal	All	**100%**	2/5/2023	1
To add more data, Insert new rows ABOVE this one					



Glossary of Acronyms

MDF – Medium Density Fiberboard
MQTT – Message Queuing Telemetry Transport
OBD – On-Board Diagnostics
RPi – Raspberry Pi

References

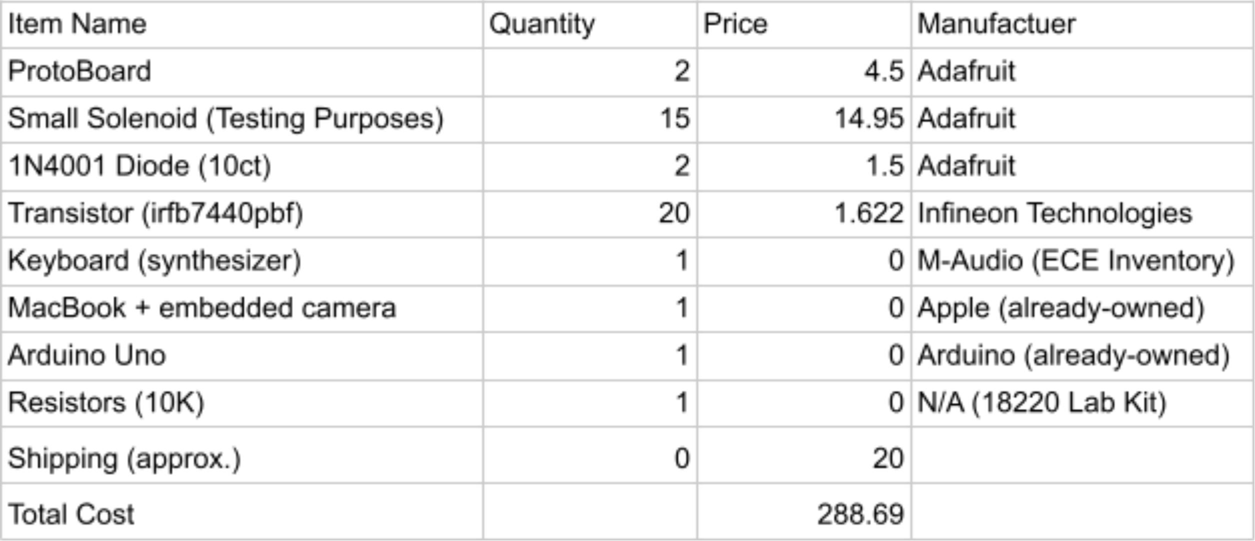
[1] Industries, Adafruit. n.d. “Large Push-Pull Solenoid.” Www.adafruit.com. Accessed March 4, 2023. https://www.adafruit.com/product/413#description.

[2] Howell, Egor, “Bayesian Updating Simply Explained,” Medium. Accessed March 4, 2023. https://towardsdatascience.com/bayesian-updating-simply-explained-c2ed3e563588

[3] GeeksforGeeks. (2023, January 3). Multiple color detection in real-time using python-opencv. GeeksforGeeks. Retrieved March 3, 2023, from https://www.geeksforgeeks.org/multiple-color-detection-in-real-time-using-python-opencv/

[4] Saha, A. (2021, May 5). Read, write and display a video using opencv |. LearnOpenCV. Retrieved March 3, 2023, from https://learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/

Table II. Bill of Materials



Item Name	Quantity	Price	Manufactuer
ProtoBoard	2	4.5	Adafruit
Small Solenoid (Testing Purposes)	15	14.95	Adafruit
1N4001 Diode (10ct)	2	1.5	Adafruit
Transistor (irfb7440pbf)	20	1.622	Infineon Technologies
Keyboard (synthesizer)	1	0	M-Audio (ECE Inventory)
MacBook + embedded camera	1	0	Apple (already-owned)
Arduino Uno	1	0	Arduino (already-owned)
Resistors (10K)	1	0	N/A (18220 Lab Kit)
Shipping (approx.)	0	20	
Total Cost		288.69	