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 body part 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 and printed signals 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 an instrument 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, the A minor scale contains all the notes of the C major scale, but starts on A). Mode changes can happen if we stick too long on the A minor chord, which is the 6th interval of the C major scale. 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 most of the time. Rather than 100% of the time, we still want some musical freedom and variance, since the rule regarding chord tones is more of a suggestion than anything.

We also require that all inputs (in this case symbol) detected by the camera be parsed and passed to the actuators within 500 ms. This requirement is somewhat lax, and our real goal is to send signals as fast as possible. As such, our minimum requirement is 500ms second, while our goal requirement is as close to 100ms as the hardware and software are capable of. We are not considering the actual detection latency as part of the critical path of this product because we are currently working on perfecting the detection functionality – this will be explained later in the report.

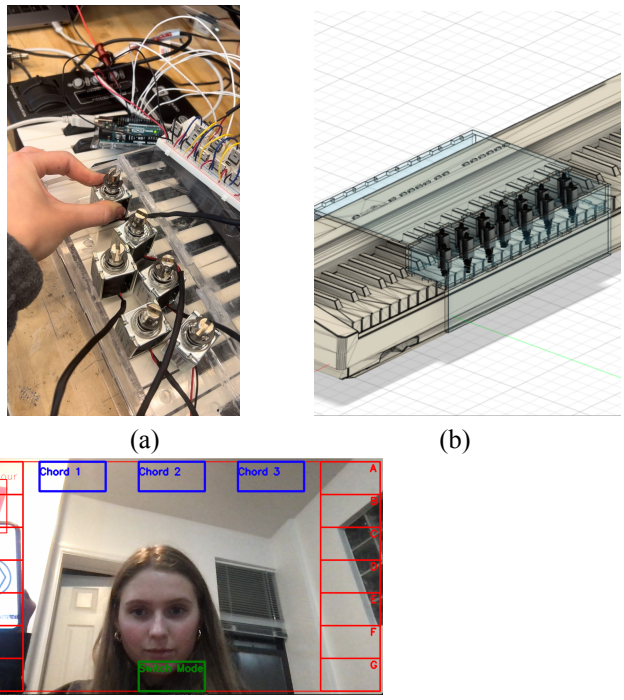
Additionally, the signals should be detected at least 90% of the time. This is to ensure a smooth user experience.We also require that the user is farther away than 4 ft and closer than 7ft to use the project.

Our hardware should be able to safely support at least 14 solenoids, with at least 4 activated simultaneously, and with no seeming latency from the detection to actually playing the keyboard. 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 provide an intuitive

playing experience with the smallest gap between the user using the detection functionality to the actuators playing the piano. In order to maximize the user experience of playing an instrument and making music, we believe that the less latency there is between the symbol detection and the actuators playing the keyboard is the crucial part of making the experience more enjoyable for the users. Currently, we achieved 750 ms latency but hope to decrease the latency as close to 100 ms as possible.

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



(c)

Fig. 1. Overall system. (a) Photo of hardware and mounting system. (b) CAD Rendering of hardware and mourning system. (c) software system/user interface.

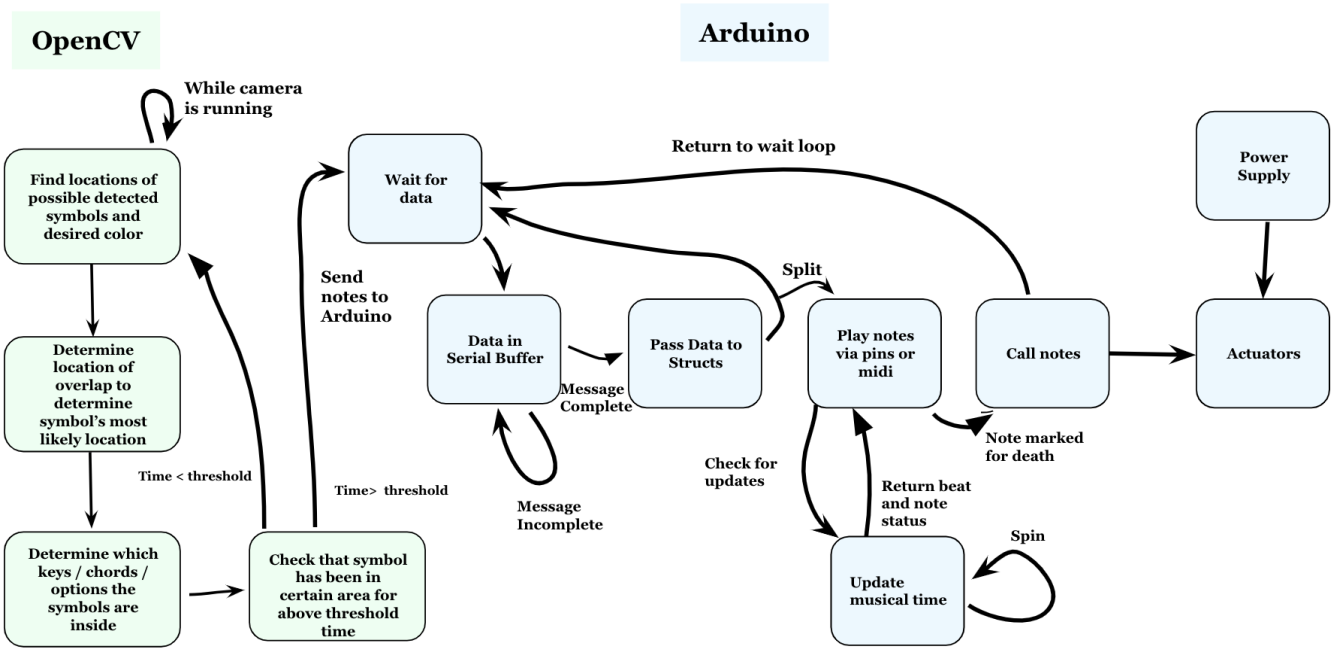


Fig. 2. The full block diagram

III. Design Requirements

As previously discussed, we require that our pipeline takes less than 500 milliseconds to traverse, with a goal of taking the time as close to 100 ms as possible. 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 500 milliseconds. 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 irrelevant 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 add unnecessary complexity to the music. 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. Additionally, this requirement is probabilistically satisfied, as it is more of a "musical suggestion" than a hard rule, as stated previously.

IV. Design Trade Studies

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

A. Design Specifications for the Detection System

At first, we used conventional color detection to detect the user's movement and assign a note based on their movement/placement of a certain color, like a colored glove. However, we quickly realized that this detection algorithm became problematic when people were wearing a certain color or in certain lighting because this changed what the algorithm would pick up. Therefore, we pivoted to building our own symbol detection model. To do, this we created a custom Haar cascade for a symbol as shown in Figure 3. This avoided the issue where someone may be wearing the color we want to detect, or it picks up color on their face, like red in the lips. However, Haar cascades are still not 100% accurate, especially with limited time to train them, so I decided to combine symbol detection with color detection by adding a ring of color around them and then testing whether the detection overlapped. This prevents the issues that arise when using only one of the detection methods.

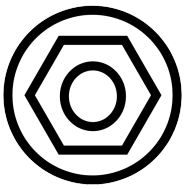


Fig. 3. Symbol that we created for our detection algorithm

B. *Design Specifications for Music Generation*

Music chord progression choice is now dictated by a normal distribution sample. All progressions are arbitrarily rated by their "tone," and the distribution helps determine what progression will be chosen. A very similar approach is taken with melody notes, the main difference being in the fact that melody note generation takes in additional parameters, measure position and subdivision. These distributions are updated on a phrase-by-phrase basis, but could be updated every measure or every half-phrase.

C. *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, we conducted my research to find out exactly how much power it would take to play a key on the keyboard, we concluded that we need 20V and 2A to safely play all of 14 solenoids – maximum of 4 solenoids at a time.

V. System Implementation

VI. *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. There are two symbols that have been trained by creating a custom Haar Cascade, which is then imported into OpenCV and used for object detection. The symbol also has color surrounding it, so we can make the detection more accurate by also incorporating color detection. The user will use these symbols, and the computer will continuously record the location.

The Haar Cascade was trained using a large negative image set and several built-in commands in previous OpenCV versions. It is difficult to get an incredibly accurate Haar Cascade file with reasonable time limits so

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. The getLocation() function utilizes the custom Haar Cascade and color detection by determining where which detected object also has the designated color detected near it. This avoids incorrect detection that happens in both pure color detection and pure object detection. 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.

VII. *Music and Note Generation System*

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 or chord. Depending on the mode, that will be the extent of the data, otherwise, it will be encoded with a location to interpret its place in the Arduino's phrase struct. This data 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.

The music sequencer is constantly running, and notes are placed in the phrase struct. The sequencer marks the current beat and measure. The phrase struct contains measure structs and note structs. Based on the sequencer's output, the phrase struct will return the value of the note that should be played. This is then passed to a helper function that either activates the corresponding pin or sends a MIDI "on" value. In player mode, a separate timer then begins counting the time elapsed since a note was marked as active. If the note has been alive for longer than its designated lifespan, it will be marked for death and culled the next time the helper function runs. The same will happen if a new note comes in. In generative mode, notes will be deactivated once a new note is played. Chord notes will be deactivated once a new measure is reached. This algorithm is greatly simplified from its initial implementation.

Chords will be randomly picked from a set of hand-picked progressions.

VIII. *Hardware System*

As our goal was to build a 2-octaves system, we started by building a smaller-scale system including only one solenoid – and gradually added more solenoids. In Figure 4, you can find the circuit diagram of the test system that we have built and

successfully tested. Figure 1 (a) shows the actual circuit we built on a breadboard for testing purposes. In the end, we designed and built a hardware system with 14 solenoids, 14 diodes (model:1N4001), 14 transistors (model: irfb7440pbf), and an Arduino (Due for MIDI, otherwise 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 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 requires a mounting system – which will be built using 6mm acrylic 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. The 3D version of the fabricated system can be found in Figure 1(b).

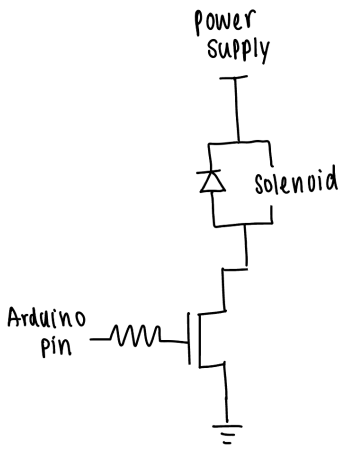


Fig. 4. circuit diagram of an individual actuator

IX. Test, Verification and Validation

Once we have individually tested our subsystems – software system (CV), music generation, and the hardware system – to ensure it was functional, we moved on to better understanding how "well" the systems were working. For instance, we measured the latency and gap of our sub-systems to verify that we are meeting our latency requirement of 500 ms. From there, we worked together to meet the rest of the use case requirements.

A. Results for Software System (CV)

The software system for the computer vision and user interface was tested by measuring two different metrics. The first was the time it took between the user placing the symbol in the note box to when the note was signaled. This was done using a timer and repeatedly testing the time taken. This was measured to be, on average, 0.6 seconds. Our goal was to have this time measurement be less than 1 second, in order for a more natural piano-playing experience. Therefore, we did accomplish the use-case requirement.

Another use case requirement for the software system was that it could detect the symbol with greater than 90% accuracy. This ensures that the system can play the notes that the user wants most of the time. We measured this by repeatedly placing the symbol in one of the note boxes and measuring how often it detected it. The result of this test was detection 92% of the time. Neither color detection nor symbol detection has perfect accuracy, so occasionally it will be incorrect in poor lighting conditions.

We also tested how fast the object detection was with no delay for the note. This is a more important metric since it is playing the note that limits how fast the OpenCV program detects the symbol. This was measured at 68 ms, so the object detection is very fast without the note-playing.

Description	Goal	Measured
Hand Recognition Accuracy (using symbol detection)	> 90%	92%
Latency (from when a hand is placed in the box to when software recognizes)	< 500 milliseconds	68 milliseconds
Transmission Gap (from SW detecting the symbol to when key is played)	< 500 milliseconds	300 milliseconds

Fig. 5. Verification Metrics and Results for the Software System (CV)

B. Results for Music Generation

For testing, a majority of test results were hand-verified. This is due to the nature of music. As long as a piece of music was reasonably inoffensive, it was considered as good. This verification was carried out over numerous musical phrase generations. They were played on piano and judged to be either good or bad.

For serial verification, tests were carried out during implementation. These tests involved sending single bytes to the Arduino, then having the Arduino send the same data back over serial. If the data was received in the correct format, then it could be concluded that it was sent correctly. These tests then evolved into tests with LEDs directly attached to Arduino pins. From these, it was concluded that pulldown resistors needed to be used, as the Arduino sets all pins to INPUT on startup, leading to floating voltage values. This has randomly activated solenoids in the past. Tests with MIDI protocols were also carried out, just by testing that the Arduino can accurately send notes directly to FL Studio. The results of all these tests were not compiled, past the confirmation that systems "work," because there were no requirements for them besides latency. For latency, the LED tests were used as a reference. On visual inspection, the activation was nearly instant for communication from the computer to the Arduino, and the time delay was considered negligible.

C. Results for Hardware System

In order to test and verify that the hardware system was working properly, we used various testing metrics as shown in Figure 5. One of the most important verifications that we did was ensuring that the power rating of the hardware system does not exceed the conventional power supply's limit (we have tested with RIGOL DP832A). After testing with the fabricated mounting system and the keyboard, we decided that the power rating that we will use is 20V/2A = 40W. This makes sure that even at some distance from the keyboard (from being installed in the mounting system), the actuators have enough strength to press on the keys.

Furthermore, we tested for latency in the hardware system – for this particular test, we only considered the critical path from the Arduino sending digital signals to the hardware system. We determined that it took less than 100ms. From there, we also tested the transmission gap between the stage where the CV detects a symbol and the actuator(s) playing the corresponding note(s). On average, this process took 300 ms – which is less than our goal latency of 500 ms. Overall, the transmission gap and latency ensure that the user experience of using Keynetic is as close to the reality of playing an instrument themselves as possible.

Description	Goal	Measured
Power to run the actuator system	Less than 30 V / 3 A	20V / 2A
Transmission Gap (from SW to when key is played)	< 500 milliseconds	300 milliseconds
Latency between digital signal to the key being played	<500 millseconds	<100 millseconds

Fig. 6. Verification Metrics and Results for the Hardware System

X. Project Management

A. Schedule

We attached the most-updated Gantt chart in Figure 7.

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. She is also responsible for the design of the mounting system in CAD.

Lance is responsible for the bulk of any music-related code, from timing to note generation. He is also responsible for all serial code that transfers data from the computer to the Arduino. He is also partly responsible for the verification of the system's hardware, in that the Arduino activates pins based on the output of the serial code functions.

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 I. 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 to have 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 was a significant learning curve. However, there are a lot of different ways to turn detected image features into notes, even though our primary goal is to use the symbol detection mechanism, we have a working color-detection algorithm that we can always resort to.

As for music generation, 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 had concerns about the power rating of turning on up to 4 actuators concurrently. Thankfully, we have not had any issues with having 4 actuators at a time, but if there is a problem during the demo, we plan to switch to 3 solenoids or even less if it came down to it. We also noticed that solenoids have a very fragile and short lifecycle. We have been testing with our solenoids to ensure that it's working as expected; this led to some solenoids "aging out." The regular use of solenoids creates an internal gap between the digital HIGH signal and the solenoid actually turning on and playing the designated note on the keyboard. Based on this, we decided to purchase additional solenoids that we can switch out during the demo in case the solenoids age out and create a visible gap between the digital signal and the solenoid turning on.

E. Potential Ethical Issues

One of the ethical concerns of our product is a potential public safety concern for electrocuting the user/anyone installing the product. And, given that 100mA is lethal, this

product uses up to 2A. Furthermore, even though our use case is to help people with physical disabilities to experience the joy of playing an instrument, the installation is quite difficult for people who have physical disabilities. Because we are assuming that able-bodied people will install the mounting system of our product, if anyone else were to install it, this could put them at a higher risk of facing our worst-case scenario – worst-public health concern. However, in a case where a differently-abled person is the one attempting to install, they will be even more vulnerable to the potential safety concerns of injury. However, if this product were to get commercialized, we may have the ability and the right resources to produce this product with enclosed and already-installed actuators inside the mounting system, so the users can simply connect this to power to make music. However, this extra step may lead to an increased cost of manufacturing – which could also lead to a different ethical concern of creating a financially-inaccessible product.

II. 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.

XI. Summary

One of the biggest lessons that we learned was to make a thorough plan before going in and start doing it. For instance, we noticed that a lack of preparation (which includes testing each part) led the hardware system to burn an unnecessary number of solenoids. If we had planned to test individual actuators before combining them to build two octaves, we could have avoided draining our budget by replacing the burned solenoids. We also noticed that we encountered more bugs (and these bugs took us a long time to debug as well) during the integration process because we were not doing "unit tests" on our individual parts. This led us to taking longer time than we initially expected to integrate our system.

Additionally, deciding to do symbol detection ended up being much more difficult than expected, due to the OpenCV software not keeping functionalities up to date in recent versions. This caused a lot of time being put into a detection that may not have worked. Thankfully it did, but we could have improved by putting a hard deadline on just stopping the symbol detection progress. It is hard to leave behind something that has a lot of work put into it, but sometimes it is the better decision.

Aside from the technical challenges we encountered throughout the semester, we experienced the importance of clear communication. The lack of clear expectations and communication led us to a longer integration process and oftentimes frustration for not knowing what stage others were at. This was a great learning opportunity for everyone and we hope to carry this lesson with us as we enter the professional workforce as engineers.

As for the product itself, if we had a longer timeline and more resources, we would have included octave expansions and further expression for musical phrases, from dynamics to phrasing in general, though dynamics would require a hardware change.

At the end of the day, we believe that 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.

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

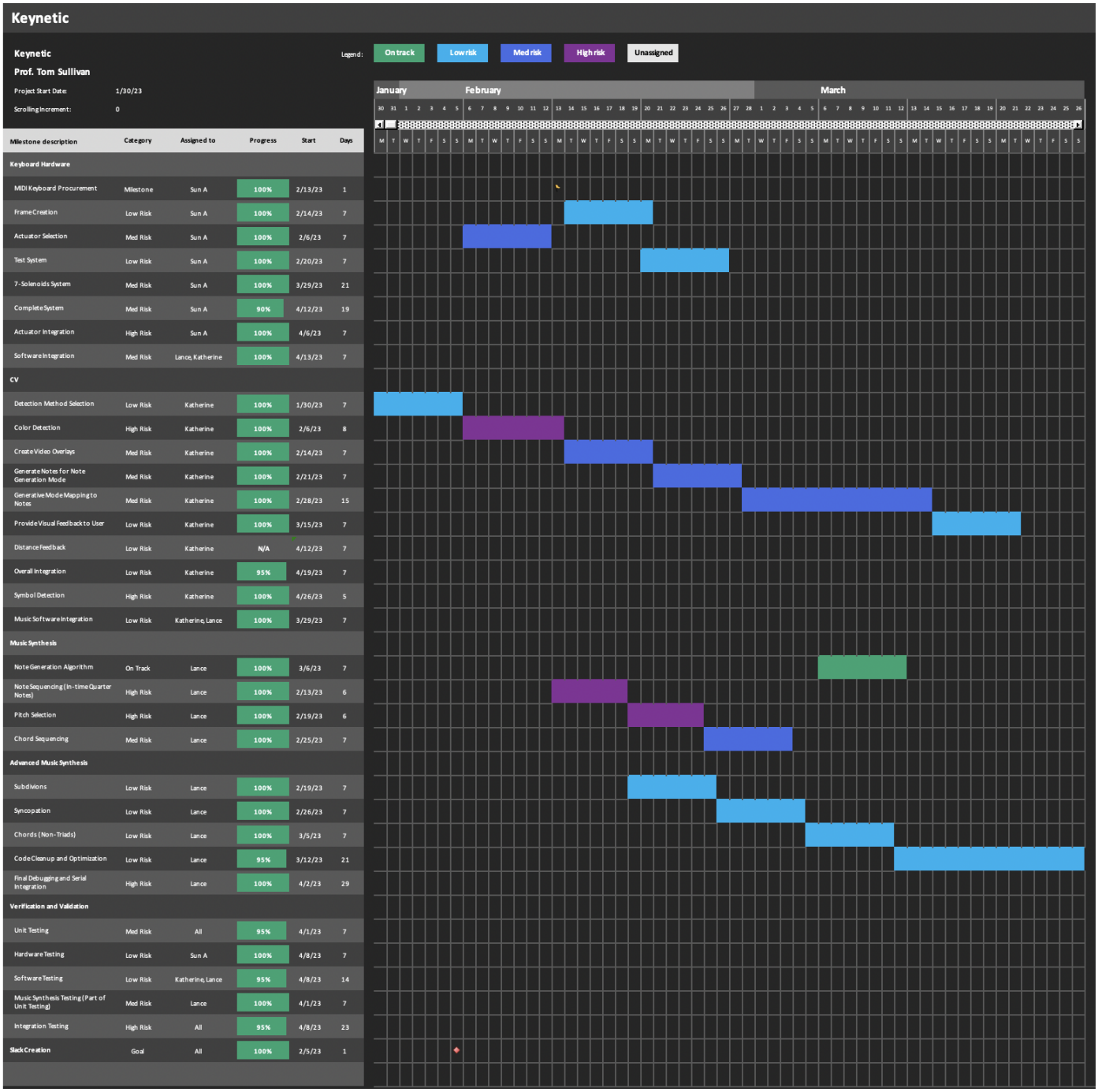
[3] GeeksforGeeks. (2023, January 3). Multiple color detection in real-time using python-opencv. GeeksforGeeks. Retrieved March 3, 2023, from https://www.geeksforgeeks.org/multiple-color-detection-in-real-time-using-python-opencv/

[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 I. BILL OF MATERIALS

Item Name	Quantity	Price	Manufactuer
ProtoBoard	2	4.5	Adafruit
Large Solenoid (Testing Purposes)	30	14.95	Adafruit
1N4001 Diode (10ct)	2	1.5	Adafruit
Transistor (irfb7440pbf)	45	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)
Acrylics (for fabrication)	2	11.25	TechSpark
Glue	1	12	Amazon
Shipping (approx.)	1	40	
Total Cost		634.63	

Fig. 6. Gantt Chart



18-500 Final Project Report: Keynetic 05/05/2023

Keynetic

Keynetic

Prof. Tom Sullivan

Project Start Date: 1/30/23

Scrolling Increment: 56

Legend: On track | Low risk | Med risk | High risk | Unassigned

Milestone description	Category	Assigned to	Progress	Start	Days
Keyboard Hardware					
MIDI Keyboard Procurement	Milestone	Sun A	100%	2/13/23	1
Frame Creation	Low Risk	Sun A	100%	2/14/23	7
Actuator Selection	Med Risk	Sun A	100%	2/6/23	7
Test System	Low Risk	Sun A	100%	2/20/23	7
7-Solenoids System	Med Risk	Sun A	100%	3/29/23	21
Complete System	Med Risk	Sun A	90%	4/12/23	19
Actuator Integration	High Risk	Sun A	100%	4/6/23	7
Software Integration	Med Risk	Lance, Katherine	100%	4/13/23	7
CV					
Detection Method Selection	Low Risk	Katherine	100%	1/30/23	7
Color Detection	High Risk	Katherine	100%	2/6/23	8
Create Video Overlays	Med Risk	Katherine	100%	2/14/23	7
Generate Notes for Note Generation Mode	Med Risk	Katherine	100%	2/21/23	7
Generative Mode Mapping to Notes	Med Risk	Katherine	100%	2/28/23	15
Provide Visual Feedback to User	Low Risk	Katherine	100%	3/15/23	7
Distance Feedback	Low Risk	Katherine	N/A	4/12/23	7
Overall Integration	Low Risk	Katherine	95%	4/19/23	7
Symbol Detection	High Risk	Katherine	100%	4/26/23	5
Music Software Integration	Low Risk	Katherine, Lance	100%	3/29/23	7
Music Synthesis					
Note Generation Algorithm	On Track	Lance	100%	3/6/23	7
Note Sequencing (In-time Quarter Notes)	High Risk	Lance	100%	2/13/23	6
Pitch Selection	High Risk	Lance	100%	2/19/23	6
Chord Sequencing	Med Risk	Lance	100%	2/25/23	7
Advanced Music Synthesis					
Subdivions	Low Risk	Lance	100%	2/19/23	7
Syncopation	Low Risk	Lance	100%	2/26/23	7
Chords (Non-Triads)	Low Risk	Lance	100%	3/5/23	7
Code Cleanup and Optimization	Low Risk	Lance	95%	3/12/23	21
Final Debugging and Serial Integration	High Risk	Lance	100%	4/2/23	29
Verification and Validation					
Unit Testing	Med Risk	All	95%	4/1/23	7
Hardware Testing	Low Risk	Sun A	100%	4/8/23	7
Software Testing	Low Risk	Katherine, Lance	95%	4/8/23	14
Music Synthesis Testing (Part of Unit Testing)	Med Risk	Lance	100%	4/1/23	7
Integration Testing	High Risk	All	95%	4/8/23	23
Slack Creation	Goal	All	100%	2/5/23	1