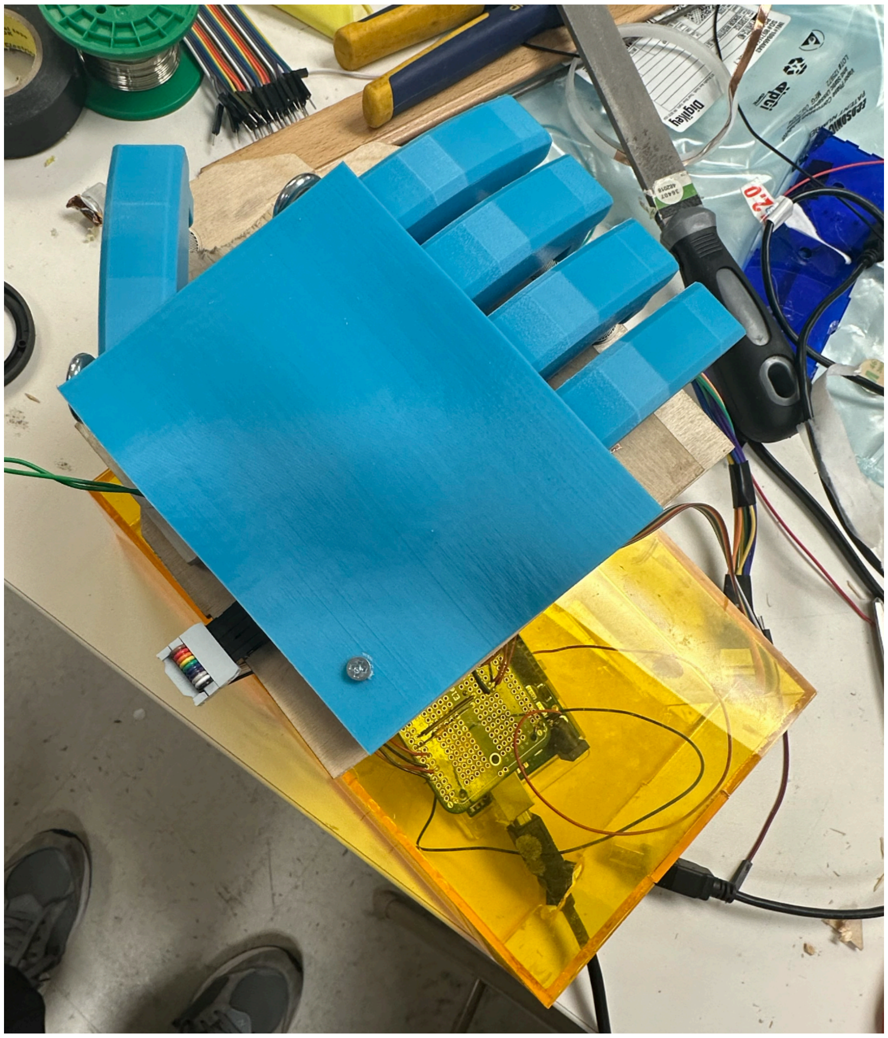
Jacob Galuten Dec 17 2024

# Pitch Hand Instrument



**Instrument Design Overview** (mostly the same as first iteration)

With this project, we aimed to create a very expressive instrument that is simultaneously ergonomic and easy to play. The goal was to allow the player to achieve very musical and creative performance without requiring as much training as a traditional musical instrument. This would be achieved by having only five keys that generate sound, one for each finger. This way, the player would not have to move their fingers to change the sound and would not need as much dexterity or technical ability to play notes quickly.

However, this design creates a new problem: How can this instrument can have the same range of pitches and musical flexibility as an instrument with many keys? To solve this problem, we incorporated a pitch bend feature into the instrument. The playable keys are the fingers of a hand-shaped, 3-D printed hand rest (see image above), and this ergonomic hand piece is mounted on a joystick, so that the hand can move left and right (x-axis) and forward and back (y-axis). We made the x-axis of the joystick control map to pitch bend with a range of three diatonic steps in either direction, so that if each of the five keys represents the first five diatonic steps of a scale, bending the pitch up (moving the joystick to the right) can allow the player to complete the scale and reach the root with their pinky, an octave up from the root which can be played with the thumb in neutral position. And, moving the joystick to the left allows the player to shift down to play the 5th scale degree with their thumb (down an octave from the 5 which can be played by the pinky in neutral position). So, in total, the full range of the instrument is 3 + 3 + 5 = 11 diatonic steps (5 diatonic steps plus 3 steps in either direction), but notice that player does not need to move their hand relative to the instrument in order to play this full range. Instead, the player can achieve the full range of the instrument while keeping full contact and with all keys remaining firmly under the player's fingertips.

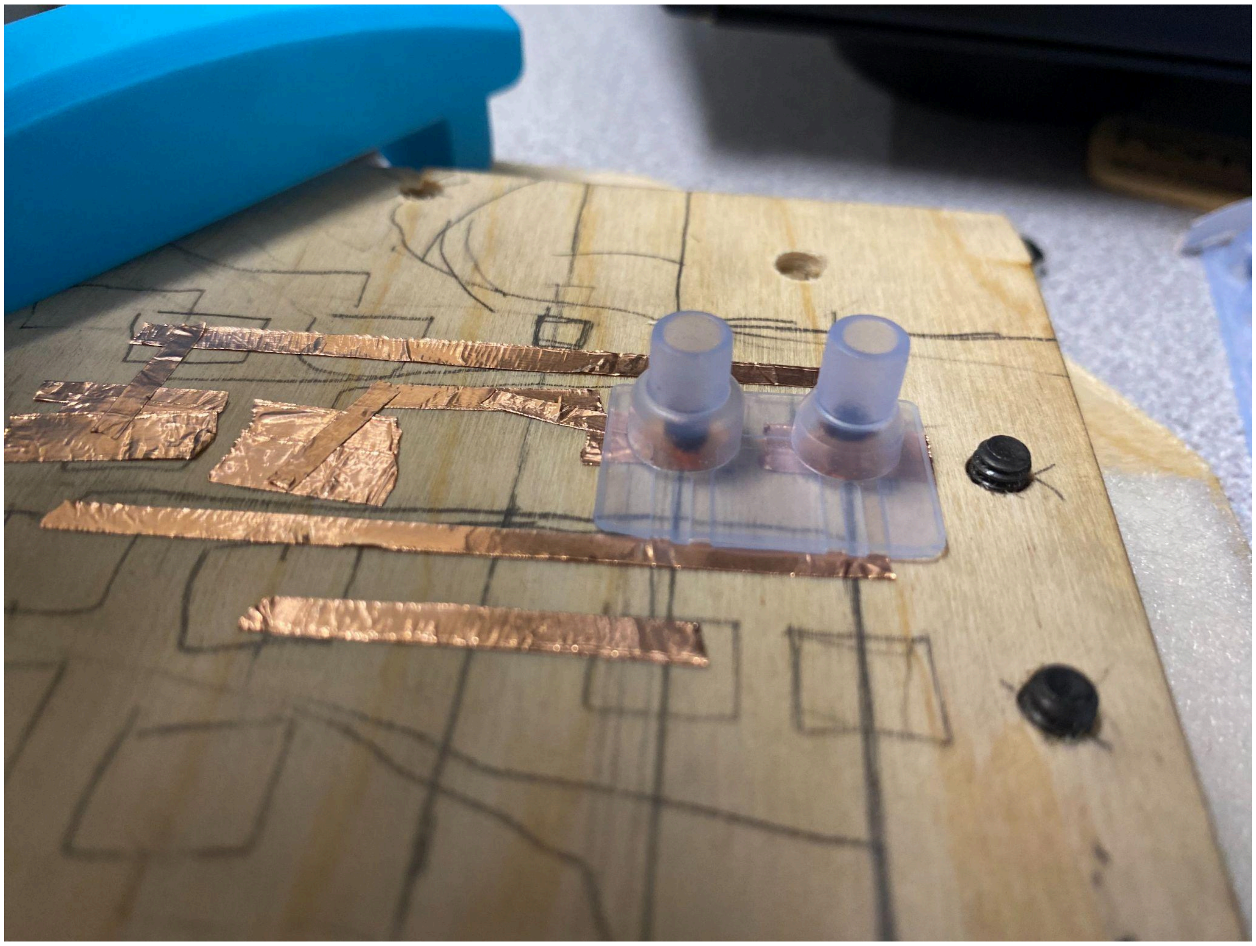
This design worked well to maintain ergonomic and intuitive user experience. Still, we needed to allow a way for the player to choose the diatonic scale they wanted to play from. We might have set it up to include various combined inputs which would map to certain scales or keys, but I did not want to require memorization in order to play the instrument. Remember, one of the primary goals was ease of use. So, instead, we opted to have our instrument be a companion to a MIDI keyboard, where you can select the diatonic scale simply by playing the chord over which you want to play.

**Hardware**

To build this instrument we used:

- 3D printer to generate the keys and palm rest
- Aluminum and other scraps found in NOLOP
- Springs
- Plexiglass rectangular sheets
- Wooden rectangular sheets
- 5 DigiKey FSRs
- 1 AdaFruit Mini Analogue Joystick
- 1 Breadboard
- 7 Button-switch pairs
- 1 Arduino
- Tape
- Copper tape
- Wires

We soldered the FSRs and the joystick outlets to wires and connected them all into the breadboard. This iteration included much better wire management. The wires were connected in bundles, and were housed in a clear box made from the plexiglass sheets. The keys were designed in onshape and 3D printed along with the palm rest. We screwed the keys and palm rest on to a wooden sheet with FSRs attached and simple wiring made of copper tape taped on to the wood.



This copper tape would send a signal when any of the keys is pressed down, as each key would mechanically press 2 of the 14 buttons (7 pairs of 2). Code in arduino was used to determine the velocity of key presses based on the differential between the timing in the pressing of the two buttons in each pair. The FSRs would also send an analog signal proportional to the amount of force applied after the button was pressed down. This hand rest / keys / wood with copper tape and FSRs was all mounted on top of the joystick which was housed in the plexiglass box along with the more complicated circuitry and wiring. The joystick outputs voltage ranges from -500 to 500mv, but the Arduino does not accept negative voltages so, in the breadboard, we created a simple circuit using resistors to add 500 millivolts, converting voltage ranges to all positive values ( -500 to 500 becomes 0 to 1000). From the breadboard, all the data got wired into the Arduino, before sending it to a computer via USB to be managed in Max.

**Software** (similar to first iteration but code is more robust and modular)

The Max patch is rather complicated. Within the instrument's main Max patch, several sub-patches were created to break up the overall functionality into several main pieces.

One sub-patch was needed to convert the notes/chord played on the MIDI keyboard into a full diatonic 7-note scale corresponding to the set of notes given. Within this subpatch, several subpatches were needed—one to guess the root of the key, one to manipulate the notes of the chord until the chord was in root position (this way any chord can be interpreted, even those in various inversions), and another subpatch to generate a plausible scale from the chord once it was in root position. This third sub-sub-patch worked by generating all 7 normal modes as well as two different melodic minor and 2 harmonic minor modes based on the supposed root note. Then, it utilized a cost function to compare each of these modes with the notes/chord given and took the scale which had the most notes in common. Moreover, all of the generated scales were weighted so that if two scales tied, the more commonly used scale would be preferred. This way, even if only one note was played on the MIDI keyboard, it would be able to reliably generate a major scale from this root.

Another sub-patch was needed for the pitch bend mechanic to work properly. Because the instrument was meant to play within the diatonic scale, it could not be that the joystick would pitch bend all of the held notes equally, as playing multiple notes at the same time and pitch shifting would modulate the key (and move out of the current diatonic key). Instead, the x-axis of the joystick would shift the pitch of each note at a different rate, so that if multiple pitches were held down and all were shifted up or down, the pitches would note remain the same relative distance from each other but instead remain within the diatonic scale degree (and the same number of diatonic steps from each other. For example, in natural minor there is a minor third interval between 1 and 3 and a major third interval between 3 and 5. If I held down my thumb and middle finger in neutral joystick position I would be playing 1 and 3, and if I pitch shifted up two diatonic steps, without moving my fingers, I would be playing 3 and 5. But notice that the interval between the two pitches I am playing has changed from a minor third to a major third. This was all accounted for in Max, which took proposed scale from the previous sub-patch as input to generate a custom function which would correctly map the x-axis joystick input to the correct MIDI pitch bend messages. This function also mapped regions of the joystick input to singular MIDI pitch bend messages so that the correct diatonic pitches could be played despite imprecision in human handling ("gravity wells"). In the latest iteration of the project, the under-the-hood design of this custom function was completely reworked, in order to remove bugs, and so that the size of the gravity wells could be changed by the user in real time. Previously, the joystick input space was divided evenly between single pitch values and linear transitions between pitch values. However, now this range of joystick inputs is categorized depending on its distance from 7 possible diatonic steps as well as the inputted gravity well size.

Apart from these sub-patches, a lot of other work needed to be done in the main Max patch in order for all these pieces to come together. The chord played on the MIDI keyboard needed to sound indefinitely until a new chord was played, and the generated scale needed to do the same, with each of the first five notes on the scale being mapped to a different channel. The FSR data needed to control aftertouch on each of these channels, respectively. The joystick x-axis data would need to be sent into the pitch bend sub-patch and would also need to affect each of these channels separately. And, the joystick y-axis data would need to be sent to the mod wheel of all channels.

The sustain mechanism for both the chords and the scale notes has been completely revamped in the latest iteration, with particular focus on robust design, so that no bugs or other glitches might surface during play time. Moreover, the new system no longer registers chords using the thresh object (essentially it recognizes a chord when notes are played within a small time interval), which means two things: first, there is no lag time between playing the chord and the chord sounding (this happened because thresh would not send an output until its specified time interval has passed), and second, the new system no longer requires all the notes of a chord be played within a specified time interval—instead, each note in a chord just needs to be played soon after the previous note of the chord was played. This latter functionality makes it much easier to play more interesting and complex chords, as there is not the same kind of time limit placed on the articulation of a chord.

The contents of the reason patches were not particularly important, apart from three key features: 1) For each of the five channels corresponding to the first five notes of the scale, the aftertouch needed to be mapped completely to volume (with the nob turned all the way). This way, if no aftertouch was present, the volume of these five channels would be 0, which is crucial, because we do not want any of these five notes to sound unless their respective FSRs are being pressed. And 2) for the y-axis of the joystick to have an effect, the mod wheel needed to be mapped to something in each of the reason patches for each of the five FSR channels. And finally 3) the channel with the chord played on the MIDI keyboard needed to not decay naturally and needed to have sustain turned all the way up so that it would not get quieter over time (remember we wanted each chord to sound indefinitely until the next chord is played).

## Future Considerations

After the previous iteration, I decided on very specific goals for future work on this project. I wanted to finish earlier to give myself more time to completely flesh out any bugs or other imperfections in the software functioning. I really wanted my code to be bullet proof. I also decided that the gravity wells should be larger for easier playability. I am happy to say that I took those goals and over-delivered during this iteration of the project. Not only is my code completely bullet-proof, complete with error message statements, warnings, and default behavior if the user does something unexpected, but it is also more modular and easier to read. Instead of making the gravity wells bigger, I made them very easy to change using a slider in the Max interface, so that the user can decide how large or small they want the gravity wells to be. They range all the way from nothing (complete glissando), to maximum (there is no transition between scale tones).

Moreover, apart from software, we had very specific hardware goals for future iterations of the project, one of which was to make the instrument more sturdy / robust, and to make it look and behave cleaner, including cleaning up the wiring and circuitry. My team worked hard and definitely made this happen.

However, there is still a lot of room for improvement. Namely, the reason patch still seems to need some work. It was hard for me to hear over zoom, but the reason patch did not seem to sound as musical as it could be, especially considering all the work I did to generate scale notes within the context of any chosen chord and to preserve the key across pitch bend functionality. Perhaps, in the future, we could create a reason patch that sounds interesting but still allows pitches to be easily recognizable, so that the instrument might be mastered and used in many different contexts and genres of music.

Moreover, the instrument still acts as a companion to a MIDI keyboard, which I think is very intuitive and works well. However, it still could be nice to have the instrument stand alone, or to have another original piece of hardware that accompanies the instrument. Perhaps there could be some kind of chord-building control pad with either keys or knobs or both that allows a musician to choose the name and type of chord they want to play. I think, however, that this is not completely necessary, as monophonic instruments are certainly not obsolete, despite there being very few solo monophonic performances. This instrument, like a saxophone, might make most appearances in a group setting, and that’s fine.