MQP Final Paper

Piano-Playing Robotic Arm

2018-2019

Project Members

Thomas Maloney

Anthony Topper

Project Advisors

Prof. Scott Barton

Prof. Xiangnan Kong

Abstract

This project explores the intersection of robotics and technology with music. The study looks specifically at the expressive aspects of a human performer and how to translate and represent that in a robotic system. By analyzing live performances of multiple performers and talking with professional performers, the team built an understanding of human gesture in performance. This was demonstrated by the creation of a robotic piano-playing system, including an industrial arm and custom-built hand. To fully understand the interpretation and performance of music, the team implemented a set of machine learning techniques, which included training a recurrent neural network (RNN) to analyze audio signals and reproduce musical input.

Table of Contents

1. Introduction

To what extent can robotics mirror human motion and expression? How “human” can robotic behavior get? We can address these issues by studying musical performance, and applying robotics to emulate human music production. Musical performance often demands a level of depth in its emotional expressivity. Producing a performance that matches a human level of sophistication will help us understand how far robotics can go to emulate true human behavior. Such a challenge covers three domains: 1) the physical construction of a robotic arm that is anthropomorphic enough to mimic human gestures, and 2) the software is complex enough to mimic human behavior in decision-making, and 3) the outputted musical performance is composed of physical gestures that add to the expressivity of the performanCe. The human arm playing a piano keyboard is a good model to study, for it gives us a good amount of freedom in terms of expressivity. A human arm playing the violin, for example, would be more difficult to study due to the extremely subtle nuances in positioning and motion in the fingers. The piano is a relatively simple interface for creating music. The only action required to produce a note is to depress a key. However, we still have access to a full range of human expressivity, as we can manipulate a note’s volume, articulation, etc. Therefore, the piano offers us a good interface that is not too complex for a robot to realistically use, but still requires specific human technique in terms of motion and expression. We are looking specifically to explore the physical gesture used to add emotion to a performance while playing the piano.

We constructed an anthropomorphic robotic hand that is capable of playing the piano in a human-like manner while coupled with an industrial robotic arm. We demonstrated the arms ability to play piano through a variety of pieces. We started with nursery songs like twinkle twinkle little star and progressed to passages from Mozart and Chopin. In addition, we verified sonic timing by playing a duet of Heart and Soul with the arm. We looked at the decisions that humans make when interpreting a piece of music, and built an algorithm that can mirror the emotionally expressive nature of human musical performance as realistically as possible.

In order to validate the success of our project, we planned on hosting a concert in which the robotic arm performs a number of pieces for an audience. We would then have polled the audience on multiple parameters, based on how realistic, expressive, and entertaining the performance is. These questions were intended to not only help us evaluate the efficacy of our solution, but it would guide possible further work on the project. Not all of these categories need to be fully satisfied, but we are aiming to at least give a realistic humanoid performance. Ultimately, if it is entertaining, then we have succeeded in building a robot arm that can please a crowd like a real performer.

2. Background

Here we introduce our existing hardware and software, along with an analysis of prior art. We also will be using various methods of data collection, including motion capture, to help us with our design.

2.1 The Robot

Our focus lies in the anthropomorphic aspects of the robotic hand. In order to delve deeper into this we decided to use an existing arm. The first one we gained access to was an industrial-grade FANUC arm in the WPI Washburn Shops as a starting point in our construction. This arm will support a smaller unit (the “hand”) with the means to play notes on a piano. The robot has an approximate linear operating space, inside its total operating arc, of 1m. The American standard for piano key sizes, for all instruments made in the 20th century, is an octave span of 165 mm. This means white keys are about 23.5 mm wide at the base and black keys are around 13.7 mm. An 88 key keyboard, therefore, is approximately 1.47 m. This means that without additional reach from the actuator, the arm cannot reach the full width of an 88 key keyboard. With the aforementioned 165 mm octaves, the robot can effectively reach 6.06 octaves or 48.5 keys without an actuator attached as seen in the operating arcs in Figure 2.1.1. With the smallest standard digital keyboard, outside a two octave midi board, being 61 keys, the arm would have to focus on a smaller range of playing. This is acceptable in a musical aspect as most melodic lines in a piece stay within three to five octaves of playing space.

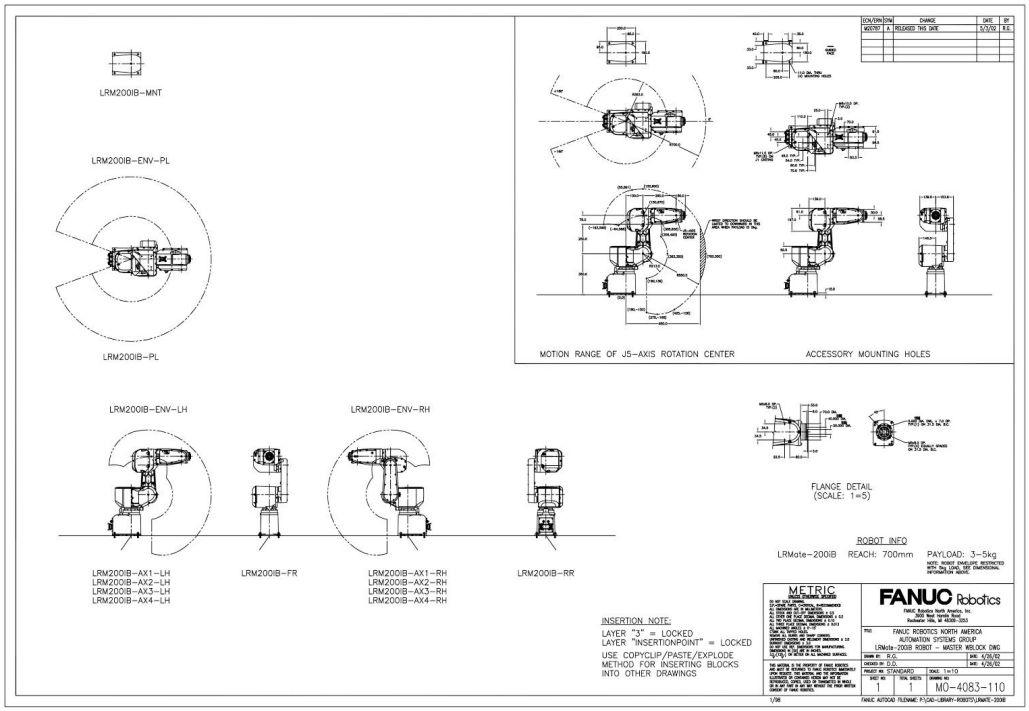


Fig. 2.1.1 - FANUC LR Mate 200iB robotic arm schematic

Unfortunately, due to multiple factors, we ended up having to change arms twice. We moved from the small Fanuc arm to the large Fanuc arm once it was found that the interfacing protocols were outdated and overly complicated. We moved from the large Fanuc arm to the ABB IRB 1600-6/1.45. This arm was an intermediate size between the two Fanuc arms and had more than enough reach to play the piano keyboard that we acquired. The ABB arm has a position repeatability of 0.02mm and a path repeatability of 0.13 mm, allowing for consistent and precise inter-note motions and finger placement. ABB's robust robot studio software also allows us to run and test our code in a virtual environment before running it on the real arm. WPI's extensive work and full class with this arm greatly helped our efforts and programming process.

The arm is a ABB IRB 1600-6/1.45. Each joint has a different maximum movement angle and angular velocity, based off the specific motor and physical limitations. The rotations of each Axis, as seen in Table 2.1.1. The arm has a horizontal reach, the distance from base to wrist at full extension, of 1450 mm (57.08 in) as seen in Figure 2.1.2. The arm, as stated above, has a very precise repeatability factor with a 6.00 kg (13.2277 lbs) payload meaning it is highly accurate and precise, well within the desired needs for the project.[1] In addition, the link speeds allow the robot to move quickly from position to position. When running in manual mode the robot has a speed limit of 250 mm/s which is safe to be near and play alongside. However, in automatic mode the robot moves at up to 5000 mm/s which requires operators to be outside the safe operating zone guarded by a laser sensor gate. When moving in manual mode there may be some sonic delays or discrepancies. However, when in automatic mode the arm can move more than fast enough to be in place by the required times.

Table 2.1.1

Axis	**Axis 1 (Rotation)**	**Axis 2 (Arm)**	**Axis 3 (Arm)**	**Axis 4 (Rotation)**	**Axis 5 (Bend)**	**Axis 6 (Turn)**
Range (°)	+180° to -180°	+120° to -90°	+65° to -245°	+200° to -200°	+115° to -115°	+400° to -400°
Speed (°/s)	180°/s	180°/s	185°/s	385°/s	400°/s	460°/s

[1]Technical data for the IRB 1600 industrial robot [cited April 28, 2019]. Available from https://new.abb.com/products/robotics/industrial-robots/irb-1600/irb-1600-data (accessed April 28, 2019).

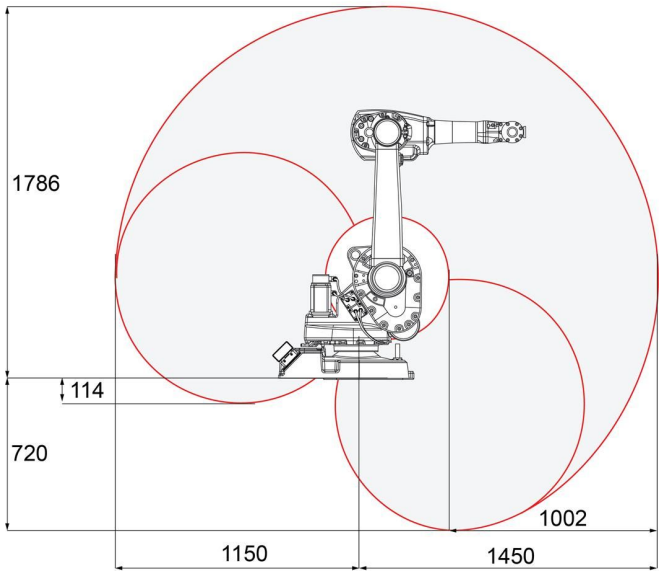


Figure 2.1.2: Reach Ranges of the ABB IRB 1600/ 1.45

The range of motion of the ABB arm is more than necessary for the keyboard we acquired. We did not get a full 88 key keyboard but instead got a 49 key midi controller. This was more than enough to represent the capabilities of the arm as a 49 key keyboard spans four octaves, allowing us to show large drastic jumps in contrast to smaller intricate motions.

2.2 Human Gesture and Musical Performance

We also must explore the nature of human music performance and human gesture. Much research exists on the nature of human movement and decision-making in music. This work will be critical in how to design a system that accurately mimics human actions.

Hadjakos discusses the various aspects of motion in human piano playing, particularly the effect of using weight from the elbow. Researchers took two separate samples of pianists, one playing with elbow motion and one playing without such motion. They used a set of goniometers to assess the angle of the pianist's elbow. It was found that in the time before striking a key, the angle of the elbow changes at a rather consistent rate[2]. As seen here in Figure 2.3.1, the angle decreases at a steady rate. Given this motion, it can be possible for us to generate a model for playing a note with our robot.

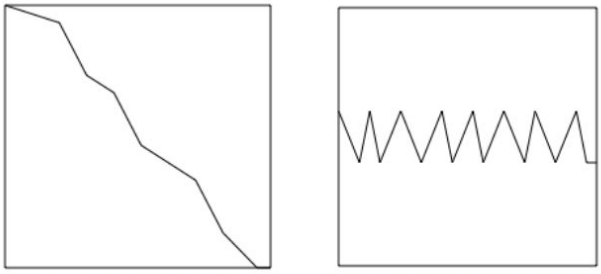


Fig 2.2.1: Visualization of Elbow Touch vs. Non-Elbow Touch

The complexity of motion will rely on joints in both the elbow and wrist, where the wrist must be flexible with at least 3 degrees of freedom to allow for the elbow to move freely before a note is actually struck. Human motion heavily relies on the supple motion in the wrists to apply weight to a piano keyboard from the elbows, which is what contributes to the force applied to the keys. The fingers themselves do not actually apply much force at all. The finger joints are involved in decide which keys will receive the force, so they act more as a filter than a means of pressing the keys. This kind of motion is very different than simply attaching actuators to a set of rods that resembles a hand — the fingers are simply transferring force.

Thompson delves into the relations between expressive body movement in piano performance and the desired expressive intentions. The desired amount of expressivity shows up in a linear relation to how much the performer moves. By using motion capture systems they were able to analyze three different pianists in their expressive movements while playing the same few measures of music. Expressivity is relatable in a quantitative way, and certain gestures are more

[2] Hadjakos, Aristotelis, Erwin Aitenbichler, and Max Mühlhäuser. "The Elbow Piano: Sonification of Piano Playing Movements." In *NIME*, pp. 285-288. 2008.

expressive than others. The gesture was analyzed both qualitatively and quantitatively through visual analysis and by comparing variations in the motion tracking results[3].

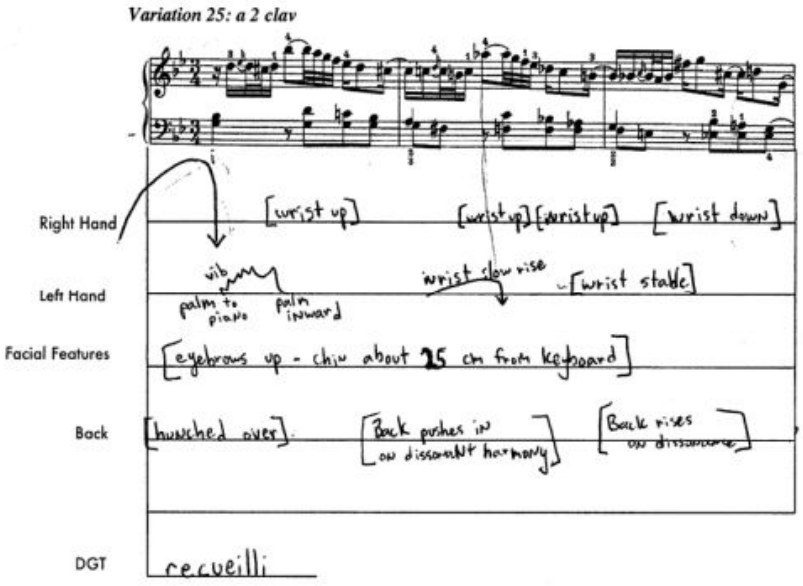


Figure 2.2.2: Gestural Analysis for Glenn Gould's performance of Goldberg Variations

When watching Martha Argerich, widely considered one of the greatest living pianists, play piano, not only does expression come from her arms but it also comes greatly from her torso and face. While motion of the arm and hand can be linear along movements such as a glissando, the torso and head moves down and back in a circular pattern. This begs the question, does the motion of arms actual portray expression if not moved extraneously outside the path of motion? In what ways does variations of linear or simplistic motion of the arm convey expresion? If working as a full system the human body portrays expressive emotion, how will isolating one system affect the perceived effects? While working with the ABB arm it must be determined if the motion planning is just for mimicking an arm or for mimicking a whole torso and arm system. If mimicking a full system, how would that limit or affect range of motion?

[3] Thompson, Marc R., (June 2007) University of Jyväskylä. EXPRESSIVE GESTURES IN PIANO PERFORMANCE. Master's Thesis, Music, Mind & Technology https://jyx.jyu.fi/bitstream/handle/123456789/13593/URN_NBN_fi_jyu-2007468.pdf?sequence=1

2.3 Existing Robotic Systems

We begin by looking at prior art in the realm of robotic prosthetic appendages, which will give us insight into how to construct an anthropomorphic arm and hand. The human hand is a complex system. Its dexterity and nuances present a number of challenges when replicating it with robotics. Many designs are possible, but in our case, we are trying to make this hand as human as possible, so there are many constraints involved. Here are a few existing designs that best achieve human-like qualities:

A previous MQP developed a biomechanically accurate prosthetic hand[4] as seen in figure 2.3.1. This hand is capable of three distinct grip styles, pinch, hook, and point. This arm was controlled through an EEG reader to form the different grips. The hand was designed around a tendon system to mimic the human hand. It used tendons on the front side of the finger, to pull it into a curve, and springs on the back to return it to a straight configuration. This system allows for a uniform curve and force across the entire finger instead of isolated forces and bending. While being able to give a clean and humanlike reaction of motion, the lack of isolation in the joints is detrimental for playing piano. If a single line tendon system was used to actuate the final joint of the finger, the base of the finger would have to be held rigid and actuated separately, instead of just relying on a spring. This tendon method was also able to produce 3.5 lbs of articulation force; more than enough to actuate the keys of a piano. We can certainly mimic designs in this research.

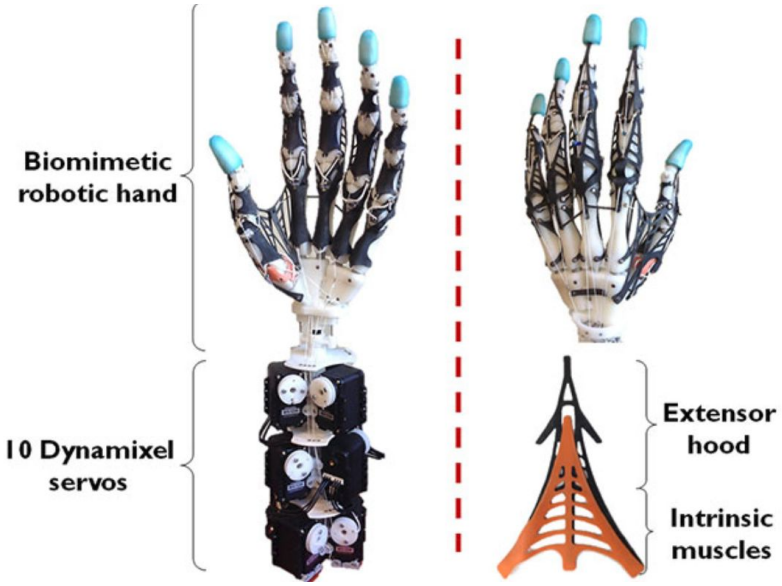
Fig 2.3.1: MQP for an Accurate Prosthetic Hand

Fig 2.3.2: IRIS Hand -- Smart Robotic Prosthesis

[4] Mervyn A. Larrier, Jr., Elina Saint-Elme, Casey Kracinovich, Dylan Renshaw, Major Qualifying Project Accurate Prosthetic Hand

Another MQP developed a robotic hand to grasp objects and determine their shape based on finger positioning[5]. The fingers they designed had the same range of motion as an average human american male and the same degrees of freedom. Their design for a compound thumb also allowed a full range of opposability and rotation of the digit. The finger actuation control for this hand used a four-bar linkage system. This gave them more rigidity in the finger but only gave a full curling arc, no partial extension of certain joints. This allowed a higher rigidity of the finger and a better assumption of the tip's location after the motion. The thumb's rotation was able to be isolated from the bending but it meant the rotation also had to carry the system responsible for actuating the bend. This lead to a bulky base joint similar to a human thumb so it was inconsequential to design aesthetics as seen in figure 2.3.2 above.

Other research has explored replicating human motion of the fingers and palm as precisely as possible. One such project by Zhe Xu and Emanuel Todorov utilized an exterior rubber hood and a series of 10 internal servos to achieve the numerous degrees of freedom in a human hand. They examined each joint inside a human hand and replicated the DOF for each location. Some had one, bending only in one direction, while others had two, allowing for more complex motions. The combination of these carefully planned joints, 10 servos, and the external rubber housing allowed the hand to bend and replicate complex human motions.[6]



[5] Ozgoren, Deniz Berk, Jardim, Adam Sebastian, Choopojcharoen, Thanacha, Casley, Sean Vincent, Önal, Çağdaş Denizel, and Padir, Taskin. IRIS Hand -- Smart Robotic Prosthesis. Worcester, MA: Worcester Polytechnic Institute, 2014. , https://web.wpi.edu/Pubs/E-project/Available/E-project-043014-213851/unrestricted/IRIS_HAND_MQP_REPORT.pdf.

[6] Zhe Xu, and Emanuel Todorov. May 2016. Design of a highly biomimetic anthropomorphic robotic hand towards artificial limb regeneration. , https://ieeexplore.ieee.org/document/7487528.

Figure 2.2.3: Rubber Hooded Hand

While this hand focused on the complex motions of the human hand, others strove to replicate the durability and strength of a human hand. A group of researchers from Germany built a hand that can endure multiple collisions and strikes from hard objects without breaking. They utilize a counter stressed tendon system. Instead of just a single cable for each finger or joint they used multiple cables per joint, as seen in Figure 2.3.4, with a total of 38 tendons. This allowed the hand to absorb violent shocks such as those from a baseball bat. The team wanted to build a hand that could perform as a human hand could in terms of dexterity and resilience. The hand had a total of 19 DOF, one short of a real hand, with the ability to exert a force of up to 30 newtons at the tip. The hand was also designed with a unique spring system connected to each tendon. This allows the hand to rotate joints at 2000 degrees a second by tensioning and releasing the springs, giving it the capability to do things like snapping.[7]

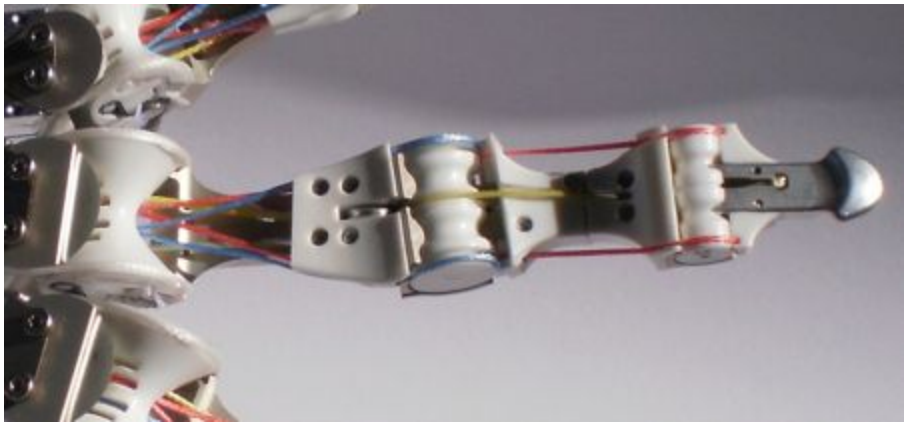


Figure 2.3.4: The Multiple Cables for Each Joint

[7] Guizzo, Erico. 2011. Building a super robust robot hand. IEEE Spectrum (January 25,), https://spectrum.ieee.org/automaton/robotics/humanoids/dlr-super-robust-robot-hand.

2.4 Musical Robotic Systems

Now we delve further into robotic architecture that is specifically designed to produce music. We analyze various designs of the physical construction as well as the software behind it. We must consider advanced motion planning, control over motor force, and decision-making. Many kinds of music-playing robots exist, but not many are aimed at replicating an anthropomorphic model. We must examine prior work in the realms of force control and how motion planning ties into playing music. Musical interpretation is affected by so many small factors, all which must come together in the right manner to be effective.

Much attention must be paid to the nature of robotic fingers. The tips must be able to respond to various amounts of pressure, so our design perhaps should consider a tip that can undergo deformation. Jen-Chang's work focused on the design of a humanoid robotic hand for playing the piano, including this kind of deforming finger. Their work describes the nature of the finger design and the placement of motors in the assembly to most effectively control the fingers. But most importantly, this research focuses on the aspects of touch sensitivity. It describes the nature of the rated torque control. They describe two different theories for the construction — one is with motors and one is with a pneumatic cylinder. Both feature steel wires that run to the joint in the finger and control the amount of force at the end point that contacts the key. The finger is made of an aluminum alloy, which is extremely light— thus the impact of additional weight from the finger itself is not as much a factor in the amount of force applied to a key. The fingers are incredibly thin aluminum, and undergo slight deformation in playing. The stress analysis on the finger is seen in fig . As a key is depressed, the finger slightly deforms at the tip, which allows it to give way to some pressure as a key is pressed[8].

Fig. 2.4.1: Deforming Fingertip, Stress and Strain Analysis

Li's first paper, *Intelligent algorithm for music playing robot*, contains many aspects of producing music, but in particular addresses the issue of motion planning and placement of an arm-like appendage along a piano keyboard. Some issues these researchers faced included how to decide what notes to play, when the algorithm is given too many notes to physically play at

[8] Jen-Chang, Lin, Hsin-Cheng Li, Kuo-Cheng Huang, and Shu-Wei Lin. "Design of Piano-playing Robotic Hand." *IAES International Journal of Robotics and Automation* 3, no. 2 (2014): 118.

once. This robot could still produce polyphony, but had to understand the physical limitations of the arm. This is a perfect source to help us understand what is needed in making real-time decisions on arm placement, and how to omit notes that are not possible to play at a given time. They minimized movement for the hands and fingers to most optimize and "anthropomorphize" the robot. We will definitely need to address these exact same issues, and will be using a very similar algorithm. However, we will be modifying it slightly, as we are not necessarily looking to minimize motion all the time. We still want to limit unnecessary movements across the keyboard, but we want to incorporate human-like expressive gestures in our arm motion[9].

In Li's second work, *Force control for the fingers of the piano playing robot*, researchers focused on the precise manipulation of force to strike piano keys using a robotic hand. Using a gain switched controller that drives a cylindrical linear motor, they were able to produce notes of different volumes. Most importantly, however, they were able to ensure that these notes were played at the right time. When using greater force, the motor would move more quickly, and thus a software-controlled variable timing-offset was needed to ensure that the musical rhythm was still respected. Such a level of precision is exactly what we need when considering the nuances in how to produce musical dynamics. So many factors come into play, especially with regards to timing, that a simple change in force and speed can cause other aspects of the music to fall apart. For both the physical construction and the software architecture these factors will need to be considered from the start[10].

Human choices in music are quite complex, and we should understand why humans choose to make certain gestures and not others. Hoffman discusses a robotic marimba playing robot that can improvise melodic lines alongside a human performer in real-time. The robot creates a melodic in a call-and-response manner, responding to lines in a human jazz solo. Several factors are taken into account in the algorithm here, the most of important of which is the musical synchronization. A musician naturally has a steady internal clock which keeps the tempo, and that ultimately dictates how they play. When listening to lines of a jazz solo, however, the meter can be hard to determine. They started with being able to read a simple known chord progression to listen for. They incorporate a simultaneous "sequence spotter" and "beat estimator" algorithm given the known chord information, which can in turn give the robot a BPM of the passage played by the human. Then they generate an "opportunistic overlay improvisation" which consists of a method of generating a melody based on tones in the detected chord. It also generates a set of note durations and note intensities it has heard from the human playing as a

---

[9] Li, Yen-Fang, and Chi-Yi Lai. "Intelligent algorithm for music playing robot—Applied to the anthropomorphic piano robot control." In 2014 *IEEE 23rd International Symposium on Industrial Electronics (ISIE)*, pp. 1538-1543. IEEE, 2014.

[10] Li, Yen-Fang, and Chun-Wei Huang. "Force control for the fingers of the piano playing robot—A gain switched approach." In 2015 *IEEE 11th International Conference on Power Electronics and Drive Systems*, pp. 265-270. IEEE, 2015.

base set from which to choose random melodies, each with a probabilistic weight attached. Also, much of the sophistication herein lies in the real-time DSP involved in detecting what harmonies are present at a given time and being able to discern a melodic line from the underlying harmonies. The human ear can pick up on this easily, but given the complexity of overtones in an instrument's timbre, this can sometimes be very difficult. They look for the strongest frequencies and assume these are fundamental for all the actual tones present[11].

Our goal is specifically to mimic the gesture of human piano playing with robotics, which is the biggest existing gap in prior art. Research has covered human gesture, and robotic human-like musical performance. Both of these fields will help us construct a robot that can help us understand the nuances in human expression, both visually and musically.

[11] Hoffman, Guy, and Gil Weinberg. "Interactive improvisation with a robotic marimba player." *Autonomous Robots* 31, no. 2-3 (2011): 133-153.

2.5. Software Systems

2.5.1. Recurrent Neural Networks

A recurrent neural network (RNN) is a machine learning method that preserves data in its hidden layers as it is trained. In modern applications, it often uses a technique known as long short term memory (LSTM), in which information is saved each time a new set of data is entered into the network. The network is organized into a series of cells, each having its own cell state. This cell state is kept whenever new data is entered, and is slightly modified using linear transformations. This is known as the hidden state. With these hidden states, the network can build on all previously accumulated information. Such a model is superior to other methods, such as hidden Markov models (HMM), for the processing of continuous musical auditory input.

Each cell in the LSTM is comprised of three gates: an input, output, and forget gate. The input regulates data coming into the cell, and likewise the output does that for data out of the cell. The forget gate is what regulates to what extent the current value remains in the cell. These gates use an activation function, commonly a logistic function, to make decisions in the network[12].

Figure 2.5.1 shows the structure of the network, with inputs $X^{(T)}$ contributing to the output layers $z^{(T)}$, and hidden states $h^{(T)}$ in between. The dashed lines represent the predictions being passed to the output layer, while the dotted lines represent temporal smoothing being applied to further hidden layers. An RNN alone without this kind of smoothing is not as effective in a larger-sale spatio-temporal setting. In the case of audio recognition, this feature is not as particularly important, for the dimensionality is not as high as in other domains[13].

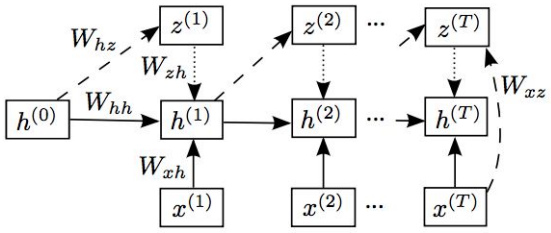


Fig 2.5.1: A Generic RNN Structure

[12] Boulanger-Lewandowski, Nicolas, Yoshua Bengio, and Pascal Vincent. "Audio Chord Recognition with Recurrent Neural Networks." In *ISMIR*, pp. 335-340. 2013.

[13] Jain, Ashesh, Amir R. Zamir, Silvio Savarese, and Ashutosh Saxena. "Structural-RNN: Deep learning on spatio-temporal graphs." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5308-5317. 2016.

The LSTM structure reduces the likelihood of a "vanishing gradient" in which the network prematurely reaches a point where it can no longer be trained. The value weights in the network can no longer be altered once this issue has occurred during a training session. The error gradient, coming from the partial derivatives of the error function, can be vanishingly small on certain iterations. Many small values are multiplied, leading the final result to become nearly zero. With LSTM, however, the hidden state is retained throughout a session and thus creates a connection between the forget gate activations and computing the gradient. Thus the weights can be updated on each iteration, even given low values in the partial derivatives from the error function[14].

2.5.2. Preventing Overfitting with Dropout

In supervised machine learning, a prediction model is fitted to a training data set, which includes input and output data. In some cases, the model can become "overfitted" to the training set. It is too accustomed to the features and nuances of the training set. When predicting based on completely new input data, it may not properly extrapolate information and relies too heavily on exactly what appeared in the training set. Artifacts such as noise may skew results in an unnecessarily complex and irrelevant manner. This completely defeats the purpose of machine learning.

In the case of a deep neural network, there are multiple nonlinear hidden layers, which are connected in complex ways. These connections can be easily prone to noise. Researchers at the University of Toronto proposed a solution known as "dropout." It randomly selects units, both in the hidden and visible layers, and completely drops them from the network. All ingoing and outgoing connections for these units thus disappear. The "thinned" network is then less prone to being thrown off by artifacts in the data[15].

---

[14] Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6, no. 02 (1998): 107-116.

[15] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research 15, no. 1* (2014): 1929-1958.

3. Methodology

3.1. Objectives

The overall objective of this project is to create a cohesive robotic system that can recreate human gesture while playing piano, for the purpose of better understanding how music is connected with gesture. This includes the creation of many subcomponents, such as a method for interpreting musical input and various mechanical means to produce musical output. For the creation of the robotic system, we define our objectives in a tiered fashion, starting with the simple ability to play notes on the piano.

3.1.1. Objective I

Our robotic system can physically play piano keys in a software-defined temporal and melodic manner

3.1.2. Objective II

Our robotic system can produce motions that are connected to the music and add to the emotional expressivity of the performance.

3.1.3. Objective III

Our robotic system can reproduce a musical line based purely on auditory input, and interpret expressive factors, such that it can give a compelling and entertaining performance of the given line.

3.2. Design Specifications and Requirements

Our robotic design must meet a set of specifications, including simply playing the piano, and replicating human motion while doing so. The basic requirements to play a piano already involves a set of design challenges. Thus, we are focusing on the most intricate part of the whole model of a pianist's arm, the hand. We already have an existing industrial arm that can move with many degrees of freedom and with great speed. We are focusing on the motion of the wrist and fingers, which is where most of the sophistication in human piano-playing comes in.

3.2.1. Requirements for Objective I

a. Given an existing industrial robotic arm, we will construct an appendage that resembles a human hand. This hand ought to be able to play keys on a standard piano keyboard, being able to span 61-keys (given the industrial arm we are using, this is the maximum span we can achieve), such that it can play high melodic lines as well as low bass lines.
b. Our robotic arm and hand can play notes on a piano keyboard based on a given musical data as input (e.g. MIDI data). This means that it must be able to reproduce rhythms and notes accurately, and adequately plan on how to move along the keyboard when needed. Motion planning work in a similar manner is described by Li & Lai, taking into account the timing offsets needed when shifting the arm to a new position[16].
c. Our robot should be able to play up to 5 notes at the same time, as the human hand has 5 fingers and can play 5 notes in that manner. Therefore, it need not be able to play every single note on the keyboard at the same time. Similar to a human arm, it is limited in the number of notes it can play simultaneously. Thus, we will be able to effectively filter out notes that we cannot play at a given time, and decide which notes to prioritize.
d. Our robot should be able to play passages that are fast in nature, meaning it can play notes in rapid succession. Humans can play certain passages with great speed and precision, so we should be able to at least match the dexterity of human motion. Franz Liszt's La Campanella Etude is known for its incredibly fast and animated character. This piece features 64th-note runs at an Allegretto tempo. If we consider this tempo to be roughly 120 beats-per-minute, there are 1,920 fast notes per minute, which is 32 per second. It is reasonable to assume that a concert pianist would not be asked to play faster than this. Such repertoire is known for being at the edge of human capability. Thus it is reasonable for us to say that our robot should be able to play at a rate of 32 notes per second, given that it remains in one position. Of course jumping to other areas of the keyboard will yield a slight delay, so this requirement need not hold in that case.

[16] Li, Yen-Fang, and Chi-Yi Lai. "Intelligent algorithm for music playing robot—Applied to the anthropomorphic piano robot control." In 2014 *IEEE 23rd International Symposium on Industrial Electronics (ISIE), pp. 1538-1543. IEEE*, 2014.

e. We need to be able to control the exact placement of the finger, such that it plays the correct key, and travels the correct vertical depth into the key. We also need control over the velocity at which this occurs, so that we can alter the volume of the notes we play.

3.2.2. Requirements for Objective II

a. The robotic arm can replicate motion of a human pianist using the robotic arm in a manner that is expressive and accurate with respect to the gestures of a pianist. If performing alongside a human, there should be a rough correspondence in what kinds of physical gestures are produced. Even though there would be differences between any two human interpretations, there are a number of common patterns that human pianists will tend to follow when playing a passage, as found by Hadjakos et al[17].
b. The robot's physical gesture should correspond with its musical expressivity. The robot can vary this motion based on the velocity/duration/articulation of notes, along with anticipating motions it should make to play upcoming notes. This means that if we ask the robot to play a certain note with a certain volume, it would adjust its physical approach to playing that note differently than if we asked for a different volume. This includes the motion before and after that note is played.
c. The robot can make decisions regarding its physical motions, and the parameters of each note (such as volume, duration, etc.) even if it is given very limited information on what to play. If we limit the amount of information we provide, such as omitting the specific information on how loud each note should be, the robot can actively make decisions on how loud each note should be. This will be based on musical convention, but also contain a certain level of freedom and unpredictability that would come from a human performance. Specifically, we are looking at the volume/velocity at which we play each notes, and the duration of each note/articulation of the music.

3.2.2. Requirements for Objective III

a. The robotic system can respond to auditory musical input with a reliable pipeline. The master controller software for the arm and hand system can be called from another software component. The musical input can therefore cause the robotic system to make changes in its motion planning and gesture.
b. The processing software can extract musically expressive information from an incoming audio signal by analyzing changes in the frequency spectrum, which include dynamics and vowel shapes
c. The software can translate extracted expressive information into meaningful gestures in the robotic system.

[17] Hadjakos, Aristotelis, Erwin Aitenbichler, and Max Mühlhäuser. "The Elbow Piano: Sonification of Piano Playing Movements." In NIME, pp. 285-288. 2008.

3.3. Analysis from a Humanities Perspective

When we started to examine physical characteristics of human performers in order to bring their expressions into the robotic system, as stated in objective II, we first analyzed video recordings of performers in concert. We took scores of the music and compared them side by side with the videos. By looking at specific passages we started to analyze how the performer related their physical gesture to the sonic production. As we went through these recordings there were many moments that were cut or panned away from the performer. This left out important information about idle posture, or gesture, when the performer was not the focus of the piece. In order to help combat this we started attending and analyzing live performances. This allowed us to see the performers gestures throughout the entire piece.

At these performances we started to also look at other types of instrumentalists and how their gesture differed or related to that of a pianist. For example, looking how a cellist reacts to an ascending line and if that is different than how a pianist does. We took this information and drew parallels between different gestural styles and examined how everyday human body language played into performance.

During this process we also conducted interviews with a couple of performers to examine their views on gesture in performance. We talked formally to an organist and harpsichordist and a cellist, and informally to multiple pianists. We started with a series of guiding questions and then let the conversation go wherever they guided it. Each person has a slightly different view on it and a different focus, so by letting them guide the conversation we could delve deeper into areas they thought more about.

Through all of this research, we continued to pair and examine how physical gesture related to sonic production; how people moved with certain phrases and passages and how that was impacted by and how it impacted the sound. This drove our focus on auditory analysis with software, looking at how a sung phrase could be interpreted and converted into forms of physical gesture.

3.4. Software System Design

Design objective III focused on the processing of audio input. Various factors of the sound were meant to influence the motion of the robot. To start off simply, our first goal was to determine the musical pitches that were being produced in an audio signal. We then added other features on top of that, including a more complex analysis of overtones to understand the nuances of the musical performance.

Our design approach focused on machine learning that would be able to understand the subtle details the audio. Given a multitude of methods, we decided to use a time-dependent recurrent neural network (RNN) to perform supervised machine learning. For this kind of application, other techniques such as convolutional neural networks or simple feedforward neural networks do not properly handle the time dimension.

How good is this method at analyzing musical input? How can it determine changes in a performer's musical tone over time? How can it recognize the contour of a musical phrase? This is certainly not the first time an RNN has been applied to audio, but here we are looking to determine how well it works for subtleties in musical audio input.

We began with a model for feeding data to the neural network. The "*X*" data, which came from the audio input source, contained a series of frequency vectors for each moment in time. Each frequency vector was computed using an FFT, given a certain window size, using a sliding window across the audio signal. For example, given an FFT window size of *NFFT*, we would start a $t = 0$ and compute the FFT from $t = 0$ to $t = NFFT$. Then, for the next vector, we would compute the FFT for $t = 1$ to $t = 1 + NFFT$.

Each window was smoothed with a Hanning function. This allows for a more accurate view of the signal at a given time, and prevents spectral leakage with small window sizes. When taking an FFT of a small segment of a signal, it is not guaranteed that either end of the segment is 0. Thus, the interpretation of the signal can be altered to include higher frequencies when seeing the clipped end of the segment. See figure 3.4.1 for an example of a clipped signal that would cause this issue. A smoothing window function, such as the Hanning function, can be applied to smooth the ends of the signal and prevents the introduction of leaked frequencies. We used this technique to produce the most accurate series of frequency vectors.

Figure 3.4.1: Signal segment without windowing

Next, we collected corresponding "*Y*" data, which was our ground truth information for performing the supervised learning. This was gathered from a MIDI keyboard, which would be played alongside the inputted musical performance. We collected basic information, such as note onsets and releases, but also the velocities of each note, changes of the mod wheel, and after-touch on the keys. All of this MIDI information was consolidated into a sequence of time-stamped changes in state. Then, using these changes in state, a state vector was constructed for each moment in time, $t = 0, 1, 2, etc.$. This complete set of time-dependent vectors was then our complete "*Y*" vector.
All the MIDI was gathered using PyGame's MIDI API, as the library is easy to use and can collect all kinds of MIDI state changes, including those from the mod wheel and after-touch.

For the construction of the network itself, there were many possible configurations that were tried. We used the Keras API from TensorFlow, which includes a variety of tools to easily construct a machine learning architecture. We started with a simple "Sequential" model, meaning that the network had a simple sequence of layers, each of which would feed data into the next. We began with a simple LSTM cell, which function as our RNN. To train the network, we used the Adam optimization algorithm, which worked best in terms of performance.

We also wanted to experiment with more layers, in order for the network to gain a deeper understanding of the data. These layers included more LSTM cells, but also we experimented with other kinds of layers as well. We used some "dense" layers, which are standard fully-connected feedforward network layers. These layers did not retain information across samples, and thus was not "recurrent" in nature. Instead, these layers were meant to prevent longer-term dependencies from overfitting the network to the training data. Consider the simple example of a three-note musical phrase. If most of the training data consists of ascending notes, or a certain musical contour that gets louder as the notes ascend, then the model would become used to seeing notes and volumes that increase. For longer phrases, it is common to have a melody that starts low, and then rises in pitch and volume, and then comes back down. However, what if we are looking to analyze the contour of a phrase that only descends? What if there are notes that are atypical for a musical phrase? The time-dependent nature of an RNN can cause the information at the beginning of the sample to influence how the network learns about the end of

the sample. Of course, this is the entire goal of using an RNN, but it can be too good at fitting this time-dependent data and can easily overfit. Especially in the case of music, two different melodies can start the same way and end completely differently. Thus, we wanted to have the option to introduce some layers that do not feature LSTM and can learn more "local" information about what is going on at a single instant in time.

However, these dense layers can also become overfitted in their own way. This is not a time-dependent overfitting, but overfitting at each independent moment in time. The network can get too used to a certain series of overtones, which may correspond to different notes with different kinds of tone quality. Thus, we used a method known as "dropout" to randomly remove a set of nodes from the dense network. This method has been known to help reduce overfitting in a fully-connected network (source?).

For the final output layer, we trained based on multiple components in the "*Y*" vectors. This included both classification and regression. A subset of *Y* contained information about what note was being played at a given time. This subset was a simple one-hot vector, where each value in the vector corresponded to a possible pitch. The active note at a given time would be represented with a 1, and all other positions would be filled with 0. As all our melodies were monophonic, we only dealt with one note at a time, and therefore only one slot being equal to 1 at a time. For this kind of classification, an activation function is applied to the output of the neural network in order to produce a single result. The one-hot *Y* vector is essentially a probabilistic model — in the given ground truth data, an active note has a probability of 1, and all others have a probability of 0. A "Softmax" activation function was best at dealing with this output. It produced a probability for each "category," or each possible pitch at a given time.

For the regression outputs, the *Y* vector also contained some plain numerical values, each of which was a different expressive factor in the inputted MIDI data. For these outputs, no activation function was used, and the network produced a single numerical result, with an output vector size of 1. We set up a series of different configurations that could then be used for training and testing.

3.5. Robot Controller Design

The robot controller software consisted of multiple independent components that all were coordinated by a master controller. There were two primary interfaces: the hand controller and

arm controller. The hand controller communicated over serial to the Arduino, and the arm controller sent commands over a network socket to the ABB arm. Figure 3.5.1 shows how data flowed to these components.

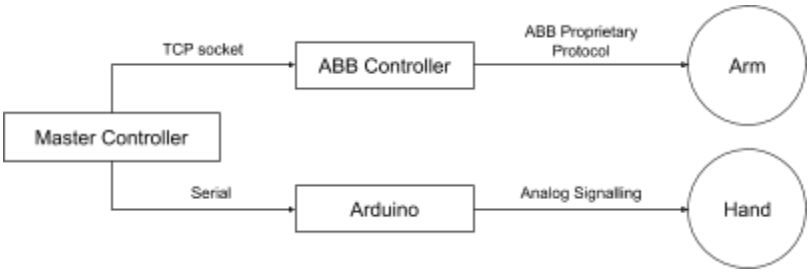


Fig 3.5.1: Controller Data Flow Architecture

3.5.1. Master Controller

The master controller issues all the high-level instructions to the arm and hand controllers, and coordinates all movement between this two disjoint components. It must incorporate complicated timing information in order to accurately play notes at the correct time with the correct rhythm. It operates using a main thread, which continuously loops and listens for new instructions. It uses an abstraction we call a `NoteStream` to process a sequence of `Note` objects, each of which is a musical note with corresponding metadata and a timestamp. When new notes are appended to the `NoteStream`, we then construct a series of `RobotAction` objects to appropriately handle the new data. Each `RobotAction` consists of command and timing information. The command itself has a command type and arguments, information of which can be seen in table 3.5.1. These could consist of information for simple movement of the arm, or to depress a finger on the hand. The timing information can then be defined in many possible ways. It could be a simple delay given in milliseconds, or it could be a defined using a function that returns when the command is ready for execution. The latter is useful when waiting for commands to finish executing in the arm controller.

Upon receiving a new `Note` in the `NoteStream`, the master controller evaluates whether or not the arm must be shifted into a new position to reach the note. A `NoteRange` object defines a certain range which the hand can reach given the current position of the arm. If the `Note` at a certain time lies outside of this range, the arm must be shifted. This shift takes a certain amount of time.

Table 3.5.1: RobotAction Commands for the Master Controller

Command Type	**Description**	**Arguments**

`noop`	No operation: controller will do nothing	*None*
`move_arm_j`	Moves the arm in a joint-wise manner to a given *(X, Y, Z)* coordinate	3 float values: *X, Y, Z*
`move_arm_c`	Moves the arm in a circular path to a given (X_1, Y_1, Z_1) coordinate given an intermediate point (X_2, Y_2, Z_2)	6 float values: $X_1, Y_1, Z_1, X_2, Y_2, Z_2$
`finger_down`	Depresses a finger at index *i*	1 integer: *i*
`finger_up`	Releases a finger at index *i*	1 integer: *i*
`finger_up_all`	Releases all fingers simultaneously	*None*

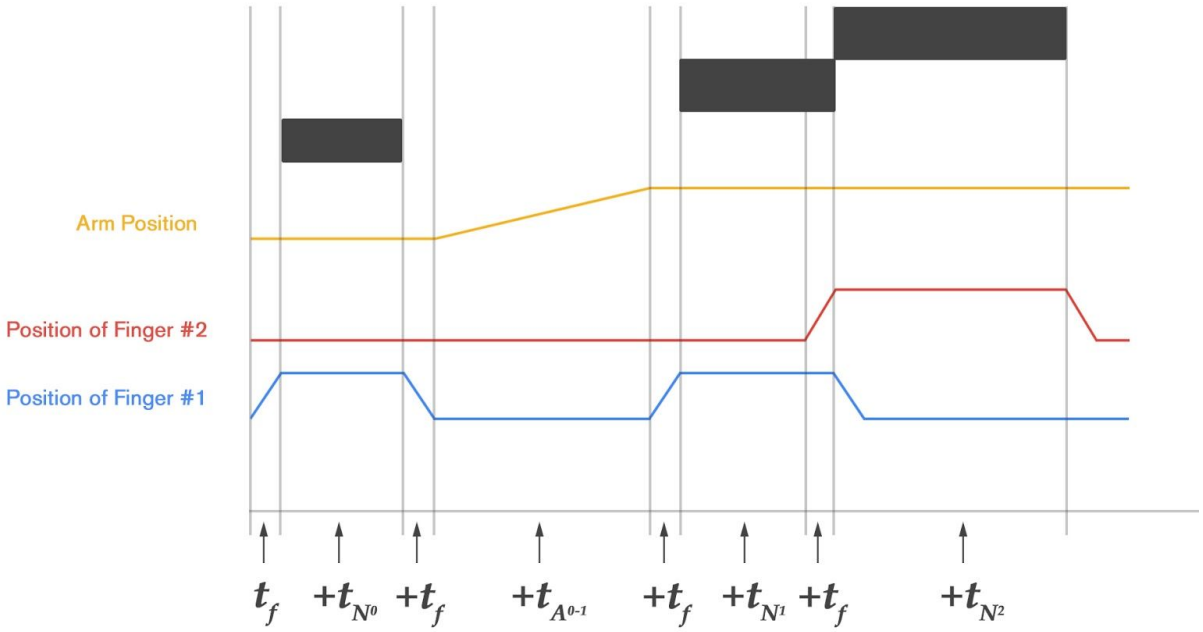


Fig 3.5.2: Timing Graph for Motion Planning in Master Controller

Each action takes a certain amount of time to complete. In order to ensure rhythmic accuracy in the output, preemptive planning of motion must be applied. See figure 3.5.2 for an example of

three notes that must be played in sequence, with a given rhythm. t_{N0}, t_{N1}, and t_{N2} are the durations of the three notes. Multiple factors must be taken into account — first is the t_f, time it takes for the finger to be depressed onto the key. We must also consider the time it takes the arm to move to a new position. In figure 3.5.2, the second note is outside the range of where the hand can reach, and thus the arm must shift. This shift is $t_{A0\text{-}A1}$ for arbitrary arm positions 0 to 1. However, before the arm can begin to move, the finger must be raised, otherwise it will get caught in between keys and cause damage. Thus, the entire system must begin transitioning to the next note for a duration of $t_f + t_{A0\text{-}A1} + t_f$ before the next note is played, allowing for the finger to be raised, arm to move, and for the finger to be depressed again for the next note. For the third note, it is only necessary to start considering motion t_f ahead of time, as the note is within the range that the hand can reach. Thus it is only necessary to depress a new finger.

When the robot is not moving, it is possible to use a "noop" command, or "no-operation" command, to simply wait for further instruction. This occurs in figure 3.5.2 during the time a note is held and nothing is needed to be done, such as during t_{N0}. This is important in ensuring rhythmic accuracy in the musical performance.

3.5.2. Hand Controller

In order to control the servos we ran a function in arduino called *pwm.setPWM* which was from the reference library of the controller. This function took in three values, the pin, the low to high trigger, and the high to low trigger. This would determine the pin that the command was set to and then the PWM percentages. We used a low to high trigger of 0 always in order to simplify the code. Then, all that had to be changed was the high to low trigger in order to adjust the pulse length. In order to ensure the servos were not overdriven we did a sweep with them individually to find the max and mins. From this, we took the last value before any of the servos were overdriven. This resulted in a servomin value of 250 and a servomax of 550. By storing these values, we could adjust all of the servos together if something changed. We also then had to just call the servomin and servomax values when we wanted a key pressed.

3.5.3. Arm Controller

The controller for the arm uses a simple high-level ASCII-based communication protocol to talk with the ABB arm. The arm controller itself acts as a server, and binds to a port on its internal network. Any other machine on its network can thus send commands to it. Each command is sent as a sequence of numerical ASCII values separated by spaces, terminated with a pound ('#') character. Because the protocol is simple and text-based, it is easy to parse on the robot's end. The sequence of values in each command begins with a command type, then is followed by a sequence number, and then a list of arguments. The command types and their arguments can be seen in table 3.5.1. The server responds with a status character and a corresponding sequence number.

The sequence number is used for acknowledgements of command completion. For example, if a command of type = 1 with seq = 5 is issued, the robot will move using a joint-based translation to a certain location in a certain amount of time. The server running on the controller will send back a response indicating the status of the command. If it returns a status of 0 for seq = 5, then the command with sequence number 5 will be acknowledged as successfully completed. This helps coordinate motion in the arm. Certain motions are hard to precisely estimate, due to small network delays or small motor errors. Also, this allows us to see if there was an error in executing a certain command, in which case a nonzero status is returned.

3.6. Hand design

When designing the hand the design of the finger itself went through a number of revisions, updates, and modifications throughout the course of this project. The initial finger, seen in Figure 3.6.1, used top mounted cable housing and a push system to push the finger downwards. When testing, it was determined that a pushing motion would allow the cable to bunch and fold over on itself when outside of the sleeving both on the finger end and the motor end. This would not allow efficient or accurate motion and cause the maximum force to be determined by the flexibility of the steel internal cabling.

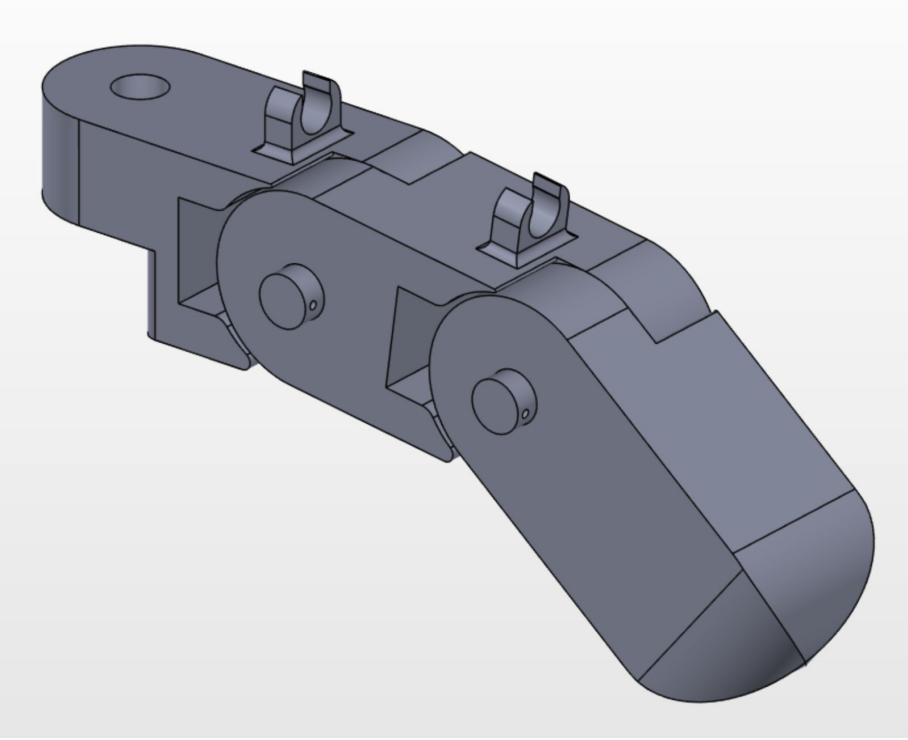


Figure 3.6.1: Version 1 of the Finger

Because of these discovered issues, the second version had the cable clips on the underside of the finger. Placing the cable clips on the underside fixed push cable issue but introduced a new issue. With the cabling hanging off the bottom of the finger, there is interference with the keys when moving around. We attempted a solution by trying to hold the cables up from the keys by mounting them to the bottom of the hand. This, unfortunately, did not allow enough flexibility in the finger joints as the cable sleeving was too stiff to bend over a short distance. In addition, as seen in Figure 3.6.2, the clips are lined up very close to each other. When the cables were attached here they had issues clearing each other. The force application in this edition, using a pulling motion, was much improved from the previous pushing methods.

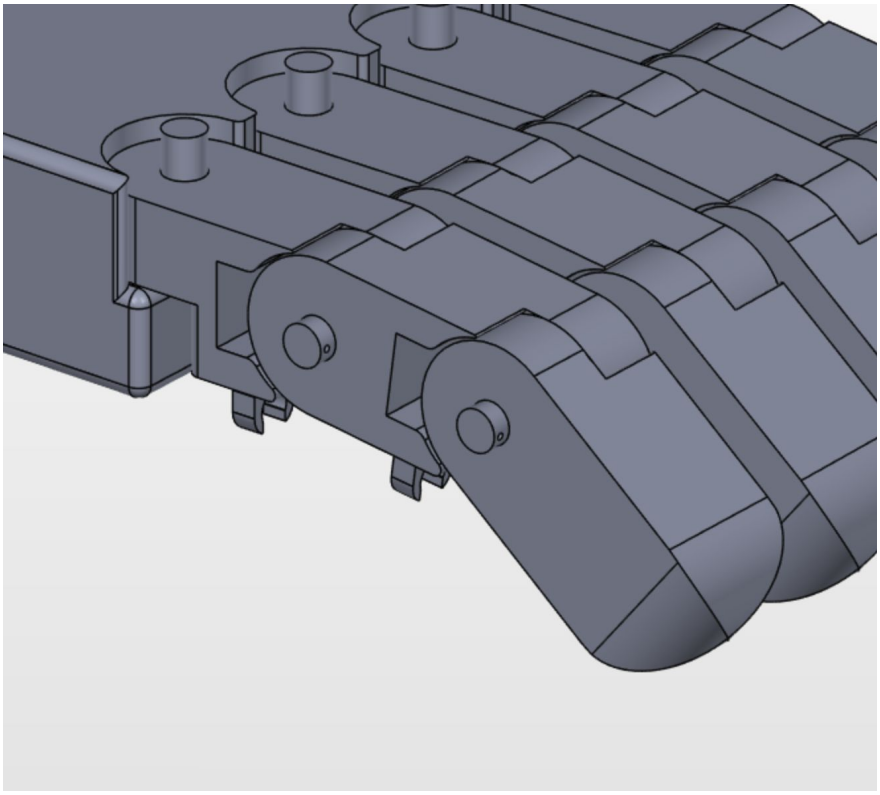


Figure 3.6.2: Version 2 of the Finger

Finally, the third version of the finger used a passthrough method. The cable entered through the top of the finger, through the holes seen in Figure 3.6.3, and then passed out through the bottom where it connected to the next finger to actuate it. This allowed the cables to be up and out of the way while ensuring a pull stroke for the force application.

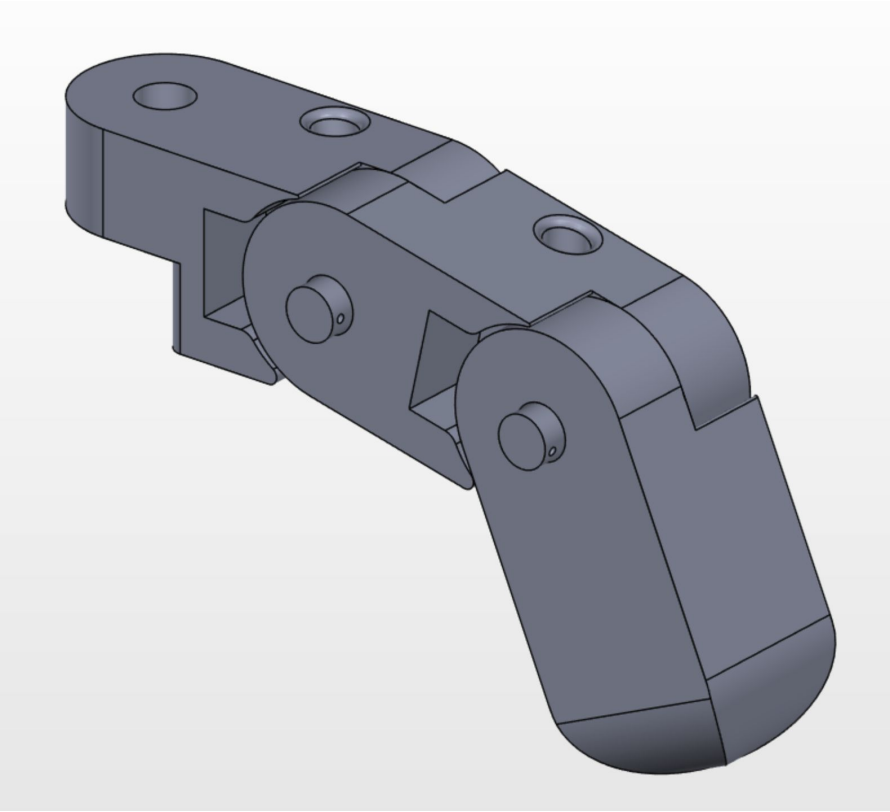


Figure 3.6.3: Version 3 of the Finger

This method, while better, is still not perfect as the cable housing for the second joint has to move with the entire finger when the first joint moves. This adds an effective spring resisting keystrokes and return motion. The servos could not overcome this if the mounting plate was too far away. We needed to mount the motor bar closer to the arm, inducing a large arc in the cabling. This arc, as seen in Figure 3.6.4, pushed down slightly on the final link and allowed it to bend easily and fully.

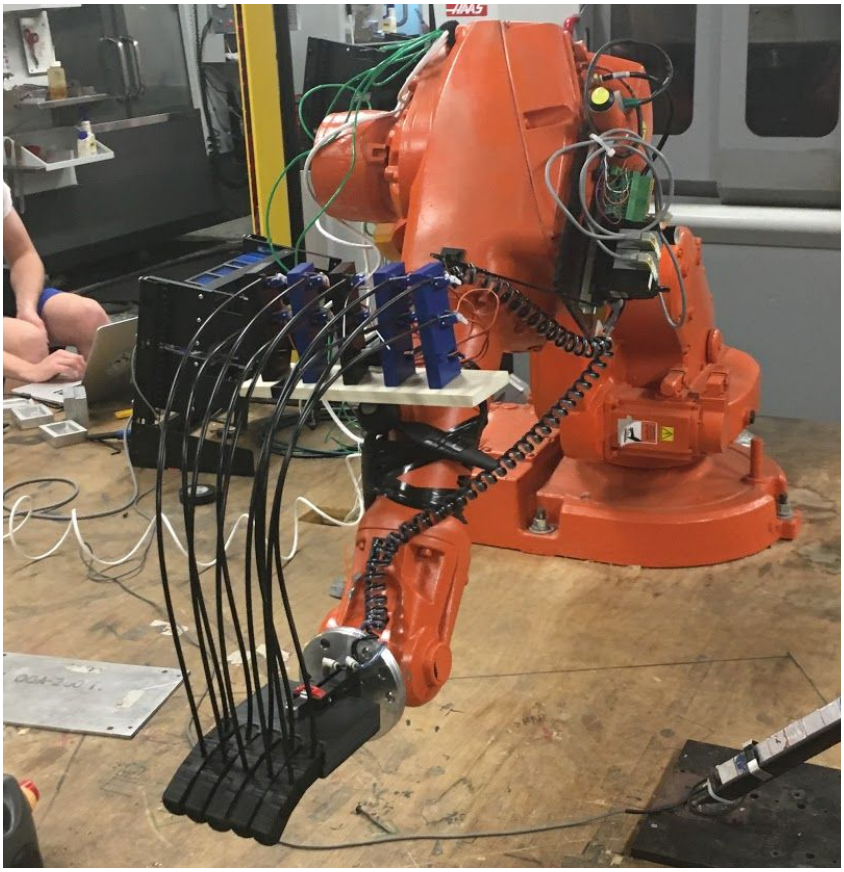


Figure 3.6.4: The Hand Mounted on the Arm

The palm also followed a number of design phases. The original plan was to use a traditional five finger approach, four forward facing fingers and a thumb. The second idea was to use a six finger system, with a thumb on either side, allowing the hand to act as both a left and right hand while playing. However, both of these ideas were scrapped before we even moved on to the modeling phase. It was decided that the complications of the thumb, such as a ball joint or cross under motions, were too great for the scope of this project. While we lost some aspects of piano technique by removing the thumb, we were able to focus more on other aspects to improve them. The first modeled design of the hand used a four finger system with cable pass throughs for controlling left and right rotational motion of the fingers to line them up with the keys. These are the holes seen just below the row of posts in Figure 3.6.5.

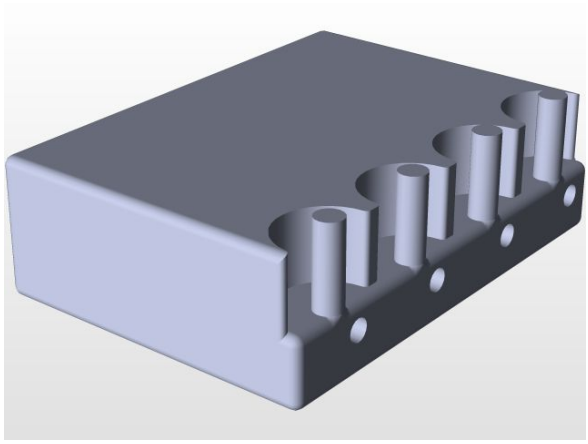


Figure 3.6.5: Version 1 of the palm

However, when placing the palm into the pneumatic gripper on the arm it rotated left and right too much even from just the force of the bowden cables themselves rotating. This lead to our second design iteration. For this one we added another finger, to have five forward facing fingers, and a slot to mount around the gripper base. We kept the cable pass throughs for all but the middle finger. If the middle finger remained straight then we could index the arms location off of it. After mounting it on the arm and attaching the servos and bowden cables we realized it was far more practical to just lock all of the fingers in place since they all settled naturally over the series of keys. This allowed us to focus on the motion planning in the arm in order to cover the distances instead of rotating the finger to move small ranges. The hand itself was printed before we made this decision so the holes for the cables in the palm are still present, just left empty as seen in Figure 3.6.6. If reprinted we would have removed the holes entirely. However, there was no structural issues with the hand and a reprint was not necessary.

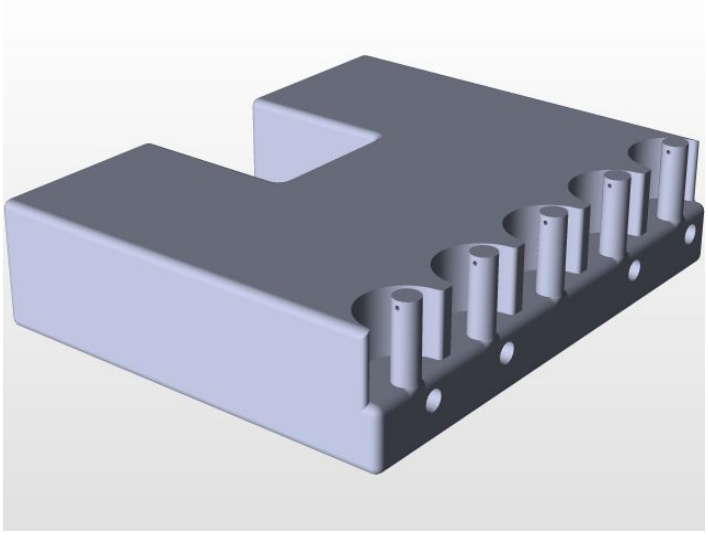


Figure 3.6.6: Version 2 of the Palm

The servos were mounted in their own holder grouped by finger. Each finger had a servo board with three slots in it and a corresponding clip for cable termination, as illustrated in Figure 3.6.7. This allowed the whole board to pivot as needed by each finger. Each board was then mounted onto a piece of wood to hold them all equidistant from the hand. The board then had a 1/4-20 threaded bolt mount pressed into the bottom of it. This allowed us to attach a small tripod to the board. The tripod was then attached to the last link of the arm with electrical tape in order to minimize slipping. Other attachment methods were tested but did not provide a stable enough grip when the arm was in motion.

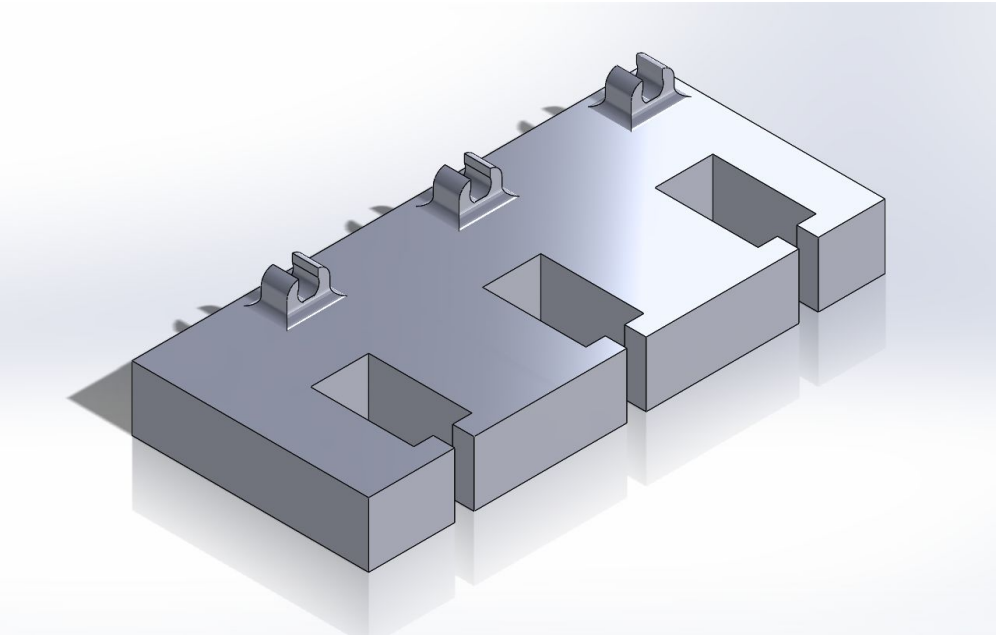


Figure 3.6.7: Servo Board

3.7. System Wiring

The fingers were controlled by bowden cables attached to servos. These servos were then all wired into an Adafruit PCA9685 servo controller board. This board is a 16-channel 12-bit PWM servo driver. This board also has its own built in clock, allowing a signal to be sent once and it will continue with the pwm signal on its own. This means it doesn't have to be continuously sent and is never blocking. This allows the command for multiple fingers to move to be sent simultaneously without worry about continuous signalling. The board itself only required four connections to our arduino in order to run. It took 5V VCC pin, to set the logic level to use, GND, for grounding, SCL, I2C clock pin, and SDA, an I2C data pin. The first three pins handled assigning certain constants to the board itself. The SDA pin then handled all of the signalling and control of all the servos. This was connected to the controller board by means of a length of cat 5 cabling. This cable was attached to the arduino next to the laptop and run through cable loops up the side of the arm. In addition to this, we had to supply the board with additional external power, as the arduino 5V port could not pull enough current to move all the servos. This was accomplished with a 120V AC to 5V DC 20 amp power supply. This was attached to the controller through a long cable threaded up the side of the arm alongside the data cable.

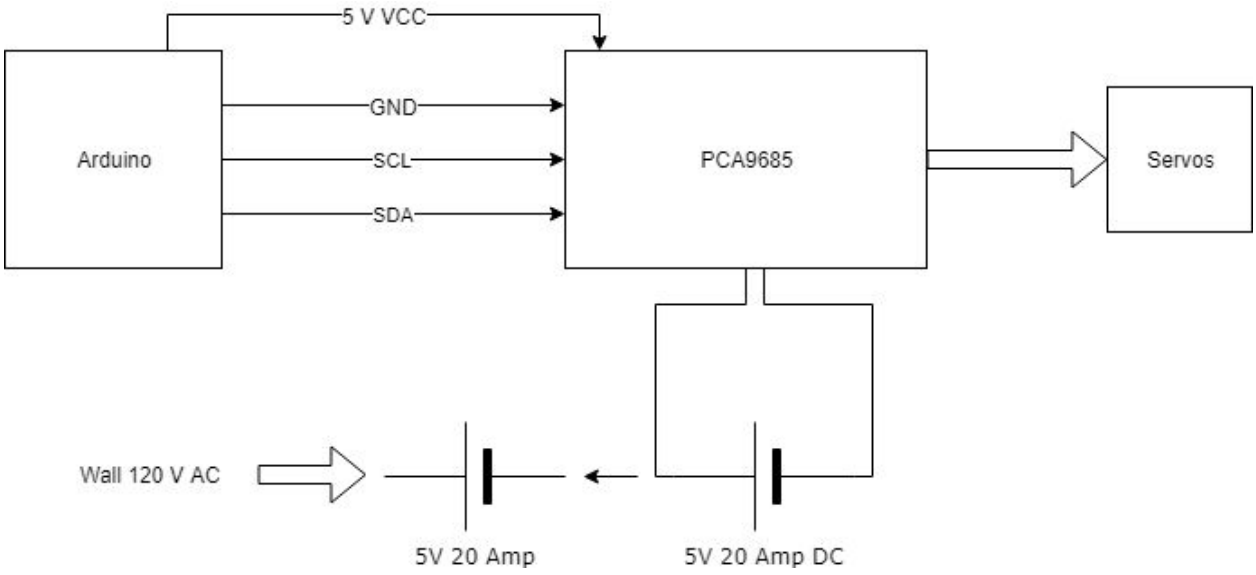


Figure 3.7.1: Wiring for the Arduino and Servos

4. Results

4.1. Musical Performance of the Hand

A video sample of a performance of the hand can be seen at:
http://static.anthonytopper.com/piano-arm-mqp/demo-video.mp4

The arm can move and press keys to create a sonic performance. We had the keyboard connected to Ableton Live to record as a midi instrument. Through this we could view the midi data from the notes. Figure 4.1.1 shows the resultant midi data from playing a duet with the arm[18]. The highlighted notes are from the arm and the others are from the human performer.

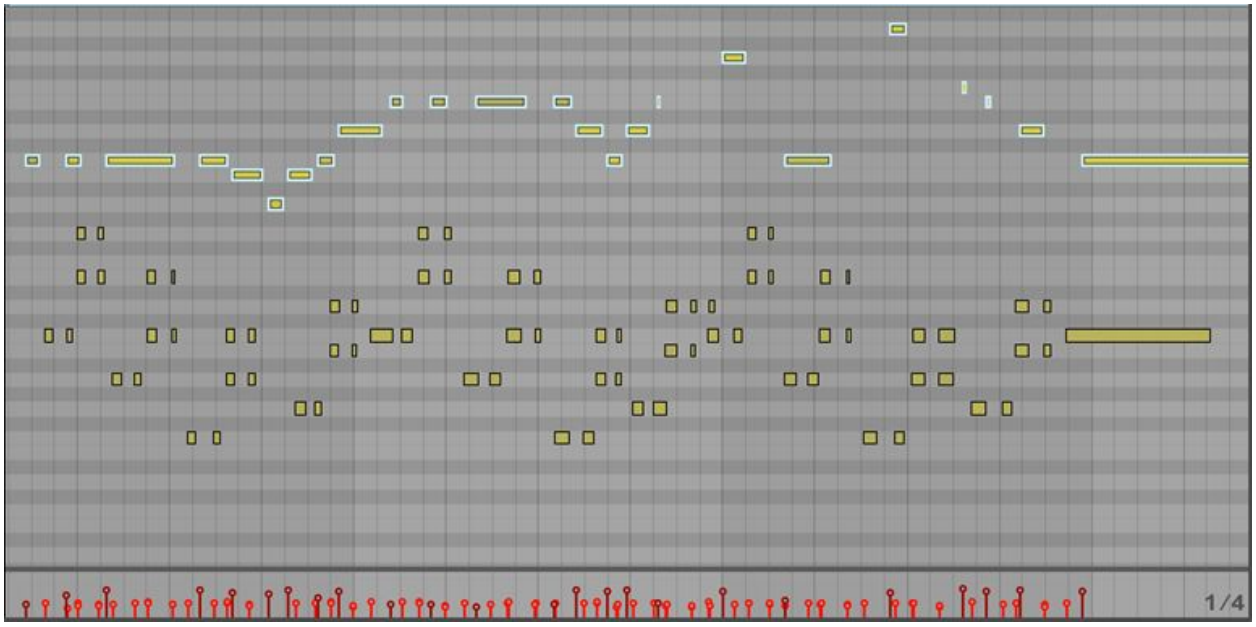


Figure 4.1.1: Midi Output from a Duet

When analysing the resultant midi data we saw variations in the velocities and timings of the notes. Some of these results were accidental but we found it to be reminiscent of a human performance. There are certain fingers on the human hand that press harder or are stronger than others naturally. This creates variations in velocity when trying to keep the same note velocity in a passage. This could also then be modified manually through the arm. If we intended for a certain not to have a greater velocity we would lower the arm a few millimeters while the note was being pressed, assisting the finger and emulating the motion of a pianists elbow as they lean

[18] http://static.anthonytopper.com/piano-arm-mqp/demo-video.mp4

into the keys. This allowed us to emphasize certain notes or moments in the music such as the highest note and the subsequent downward run. It can be seen in the velocity data at the bottom that the highest was a little softer with the run down getting louder and then softer as it went.

When moving from place to place the arm followed an arched path, this is not reminiscent of a human player as they cannot follow perfectly round paths, just close imitations. These paths, however, could then be varied in height to create differences in the gesture. These variations came not only from the distance between the keys but also the importance of phrasing or moment in a passage. These gestural plans were mostly hard-coded into the passages when we gave it the midi data to play. These pieces and passages also could then be modified by the arm's approach and departure by changing the approach vector and starting location and the return home position at the end. How it started the passage and placed itself on the keys could indicate a difference in intent. If it placed itself gently on the keys it could show a softer passage, where if it landed firmly in preparation it indicated passion, anger, or more aggressive passages. Finally, in its departure it could leave itself on the keys, letting the not linger, or it could lift off and move away, as a dramatic final lift. All of these factors combined lead to a more engaging performance than a simple pressing of the keys. However, it is not yet sophisticated enough to emulate or evoke human performance and emotions.

4.2. Human Motion in Performance

4.2.1 Overview

When looking at expression many people may assume that the more expression means more motion. However, it seems like it is often the exact opposite. Lack of general motion and smaller specific motion allow for clearer transmission of intention and emotion. When watching professional musicians or performers their motions are subdued aside from their specific gesture. If they are moving freely and randomly during their general performance, then when they try and indicate something specific, it is lost on the observers.

When talking to Simon Halsey in Berlin during a master class on conducting he targeted areas of extraneous motion such as the knees, second arm, and even the mouth. He said that if you continuously move these areas then when you intentionally want to show something by doing something it will have less importance. For example, when always mouthing the lyrics with a choir, when you indicate a breath to them it appears to be just another part of the words instead of an indication of an entrance. Same with something like your knees. If you bounce to the beat as you conduct, when you want to indicate a drop in dynamics, by lowering your stance and dropping your posture, it can come across as just another beat in the piece.

4.2.2 Analysis of Live Performances

Through my attendance to a number of classical music performances in person I observed different instrumentalists and performers playing in a professional setting. When playing piano, in order to indicate certain sections the performer leaned into chord progressions of importance. Leaning toward the piano showed intent of intensity when flowing into and digging into a chord or a complex section. When showing ease or comfort, he leans back and away from the keys with the body and lets the head fall back. In addition to adding physical gesture without modification of the auditory output there are times where the two are paired together to make a more dramatic effect. Near the end of the piece, in order to show emphasis he not only leaned in but also slowed down the run of notes, pairing physical gesture with modification of auditory gesture to create a more powerful experience. Throw shoulders forward and into certain lines of music.

When playing cello, as the hands need to be precisely placed and any extraneous motion can cause detrimental effects on the performance, nearly all of gesture is done with the head, neck, and upper torso. For violins, violas, and other smaller string instruments however, their size allows motion of the full body and instrument, forming gesture with the scroll of the instrument

in the air as movement of the neck would dislodge the chin rest from its proper positioning. However, with proper placement on the shoulder, some violinists can motion with their head independently of the scroll, allowing for extra indication of gesture. I saw this primarily in the concert-master of the ensemble as she was able to separate scroll gesture intended for the ensemble and head or body gesture intended for the audience.

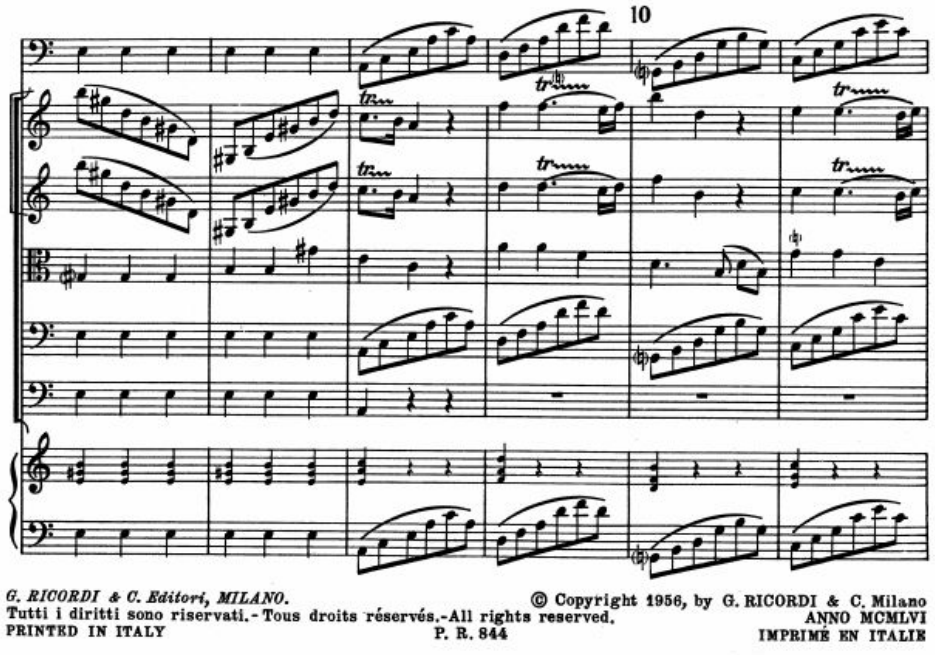


Figure 4.2.1: Passage of Vivaldi's Cello Concerto 418

Through the above passage of Vivaldi's cello concerto 418, in the fifth line down, Joshua Rohde indicated every measure based run with a movement of the head. Starting out away from the neck of the instrument at the beginning of the passage, moving towards the neck, and then a little away at the end. This followed the melodic flow of each measure.

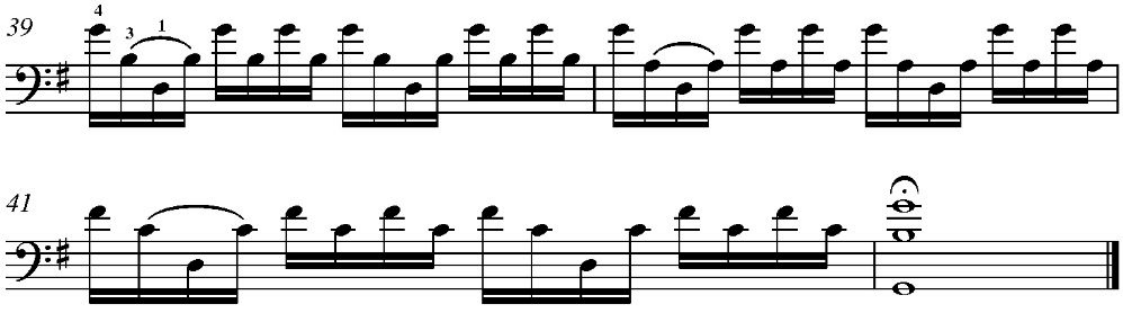


Figure 4.2.2: Passage of Prelude of Bach's Cello Suite No 1

When coming to the end of the passage for the Prelude of Bach's Cello Suite No 1. Joshua Rohde started with his head down, leaning over the cello as he played the series of sixteen note arpeggios and jumps. He leaned back slightly during each of the slurred sections indicating smoothness rather than the almost aggressive jumps from before. Finally, when he pulled the bow for the last chord he leaned out away from the instrument and then back in a circular motion leaving his body spaced away from the body of the cello for the final ringing note in the cathedral.

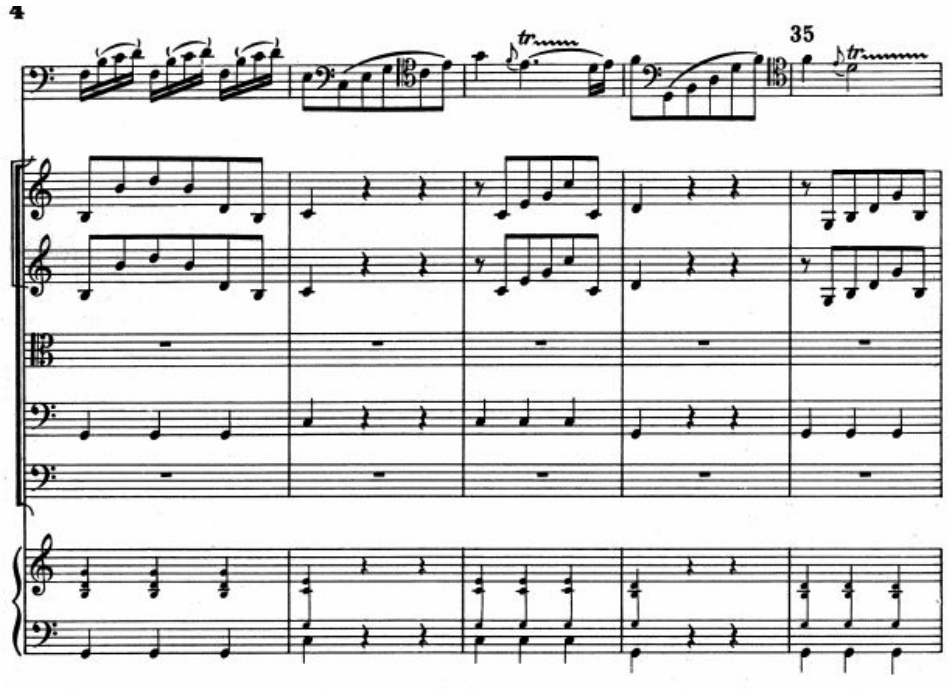


Figure 4.2.3: Later Passage of Vivaldi's Cello Concerto 418

In this later passage from the Vivaldi, Rohde punctuated each quarter note with a small movement of the head. As he was not the primary cellist (on the top line) he did not want to draw too much attention away. Then, during the rests he froze and kept his body very still. The contrast between these periods of silence and punctuated quarters draw contrast in the background as the primary cellist has a flowing and fast passage over the top.

When observing Wesley Hall play piano in a style that is looser and more sporadic than his formal classical performance, such as a ragtime beat, his hands rise high off the keys in order to accentuate the jaunty nature of the piece. In this situation his body stayed still while his hands moved more. Isolation of gesture seems to be important to indicate the expression clearly. If it is too spread across the body it can become confusing to what is intended. By isolating the motion to one location, either arms or torso, it is easier to show specific areas of significance and draw the viewer into one style of cueing gestures.

4.2.3 Analysis of Recorded Performances

When watching Martha Argerich play Ravel's Piano Concerto in G Major,[19] much of her expression throughout this piece is shown in her head and torso. Her hands are very methodical and precise, rarely making large gestures in between phrases close together. Her only larger gestures are large arcs when she finishes a section of the piece. Her hands remain close to the keyboard, as this piece has many fast passages, and move between them efficiently. Her accents and energy comes mainly from her head, moving with the beat and marking certain areas as unique.

In her first solo section, starting at 0:54 in the video and matching up with rehearsal number 4 in the music as seen below in Figure 4.2.4, she starts each passage with her hands close to the keys and then lifts off as she reaches peaks of phrases. Specifically her left hand takes a little arc into the air after the downstroke final note of the left hand phrases. It can be seen in the second line of her solo below the pedaled phrases with the rising phrase with a drop. She moves in an arc from the top to the bottom note and then adds an additional flourish to the end of the phrase each time. As soon as her solo starts you see her lean in a little bit towards the piano. Adopting a more engaged body position as opposed to her more relaxed position at the beginning of the piece with the full orchestra.

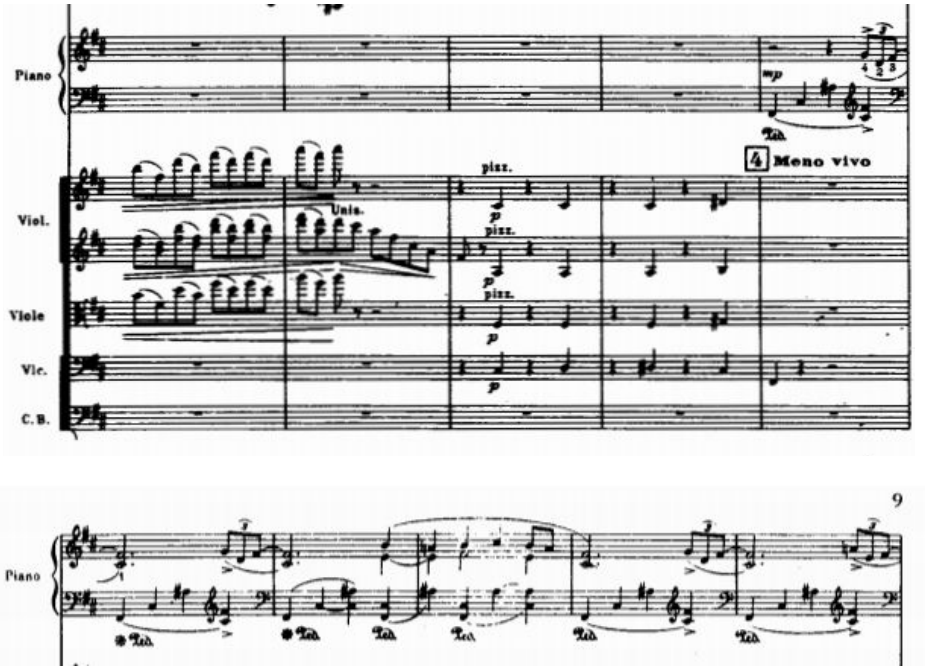


Figure 4.2.4: Rehearsal 4 of Ravel's Piano Concerto in G Major

[19] https://www.youtube.com/watch?v=cJOW5mlhH_Y&ab_channel=EuroArtsChannel

While her arms maintain a clean and normal position for the most of her performance, she still exudes expression in the endings and beginnings of phrases with her arms, and the entire time with her torso. Creating the difference between and neutral upright positioning and a leaned in "engaged" posture can indicate certain sections importance or flow. The position analysis, forward kinematics, and path planning for the robot could then be based off the initial link's angled position. This would effectively remove one degree of freedom when planning how the arm moves as that position would be "determined."

While this lean is one part of Martha's movements, the other main part is her swaying. She exemplifies this trait even more in another one of her performances of Shumann's Piano Concerto[20]. This back and forth sway may be difficult to implement into the robot as it would require a large amount of additional kinematic planning to keep the fingers steady while the arm moves independently.

When looking at Yeol Eum Son play piano she followed a similar gestural pattern. In her performance of Motzart's Piano Concerto No. 21, K.467[21] she uses primarily her head to gesture the flow of the passage, keeping the hands and fingers basically set on the keys and not lifting off at all. Throughout much of her performance the extraneous motions and gestures are avoided for efficiency and expression is added dynamically, melodically, and with small motions of her body.

Evgeny Kissin is a classical pianist from Russia, widely known for his virtuosic interpretations of Romantic piano repertoire. In this recording[22], starting at 2:30, he plays Brahms's Intermezzo in A, Op. 118 No. 2, regarded as one of Brahms's most beautiful piano works. To analyze his gestures throughout this performance, it is important to understand why each of the gestures is made. There are, in general, two categories of these movements: 1) for utility to help make certain articulations/tone/etc. more effective, and 2) for emotionally expressing the music, whether it be to help the performer themselves feel the music more strongly, or to help convey such feelings to the audience.

Measure 1, video 2:03

He begins in a relatively "neutral" position, with a straight back and hands comfortable resting on the keyboard. The distance from his chest to the keys is roughly the length of his forearm, so his elbows are bent at around at 90-degree angle. This is a common stance for pianists to assume when adjusting themselves to the piano bench and preparing to perform.

---

[20] https://www.youtube.com/watch?v=8Z3LHX38y78&ab_channel=CristianRadu
[21] https://www.youtube.com/watch?v=fNU-XAZjhzA&ab_channel=taky_classic
[22] https://www.youtube.com/watch?v=8Y3MypevjcA

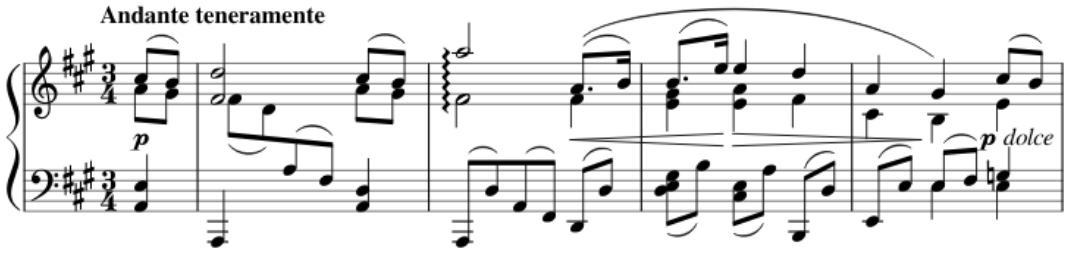


Fig 4.2.5. Brahms Op.118 Intermezzo No. 2, measures 1-4

Measure 3, video 2:10
Kissin brings out the broad sweep in the crescendo leading into the third measure, moving his torso out and extending his right arm. His motion is directly proportional the volume, as he reclaims his "balance" when leaning back in during the subsequent diminuendo.

Measure 8, video 2:30
Kissin lifts his right hand slightly to accommodate the new phrase beginning on the third beat of measure 8. While physically subtle, it brings out a profound change in the sound produced — there is a clear break as the main theme is introduced a second time. There is a "breath," just as a vocalist or one playing a wind-instrument would need to breathe before beginning a legato phrase. This gesture thus is made not only for an abstract expressive purpose, but to aid in achieving the sound Kissin is looking for.

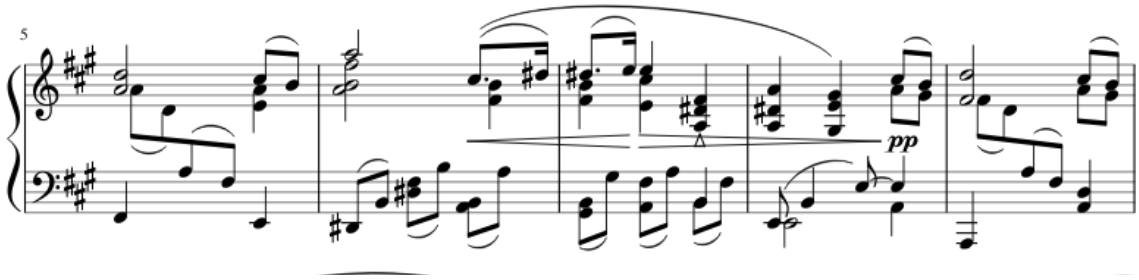


Fig 4.2.6. Brahms Op.118 Intermezzo No. 2, measures 5-9

Measure 19, video 3:08
Kissin begins circular motions with his torso, as Brahms introduces a continuous 4-3 suspension. This leads to other melodic lines, such as those during measure 19, where Kissin increases the radius of his circular motions to match the crescendo, bringing the movement to a peak in measure 23, alongside the peak of the melody in terms of pitch and volume.

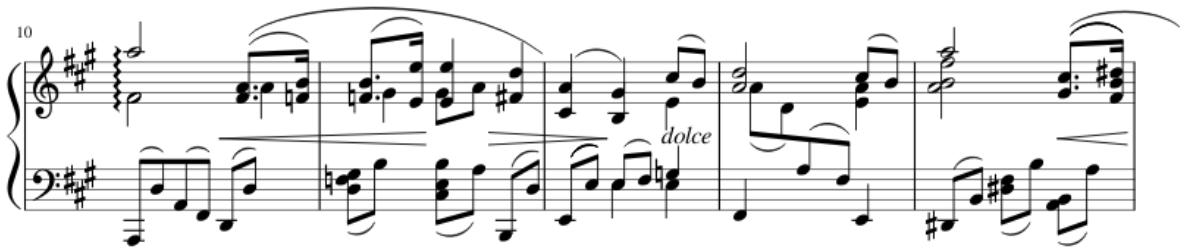


Fig 4.2.7. Brahms Op.118 Intermezzo No. 2, measures 10-14

It becomes clear how the movements are affecting the performance — some affect sound directly, others are purely emotional outlets and seem subconscious.

Stefan Müller and Guerino Mazzola explore the nature of expressive gesture in a more discrete and mathematical sense, taking events during a performance to generate a “performance vector field” to represent various expressive notions. However, as they acknowledge in their analysis, techniques such as the “Chopin Rubato” incorporate a wide variety of factors, including general tempo changes, such as an improvised rallentando, along with agogic delays and other expressive interpretations, that it becomes too difficult to tie a single expressive feature to a single justification.

4.2.4 Physical Gesture as Discussed with Professor Joshua Rohde

According to Joshua Rohde, music itself is a physical activity that manifests an outward expression of inward emotions when playing. It is only natural that these inward emotions or feelings in the music itself manifest themselves in an exterior manner. Often these gestures go beyond the physical motions required mechanically to create the music, either through an instrument or the performers body. Sometimes the physical gesture can relate directly to the musical expression such as a sharp bow stroke representing a punch or a strike, allowing it to add to the musical story of the piece. Sometimes, however, these extraneous gestures can get in the way of the music as it can force bad technique and methods of playing. The choice to make these extraneous gestures, however, are more than just what is on the page. It relates to spirit of the music, showing the spirit and story of a piece when it may not be fully expressed otherwise.

When choosing to emphasize certain passages or moments in the music a number of factors are taken into account. Both the musical background of the piece, based off the performers understanding and musical analysis of the piece, and certain tonal and musical aspects. Some key factors to consider are those like pitch and height of notes, with higher notes and peaks of phrases carrying importance, duration, dynamics, rhythm, and accents in the music itself. Also, certain addition gestures can be attached to notes that are outliers to the expected tonality set, giving a focus or a drive to these additional moments of color or contrast.

When making these gestures paired with the music, much of the knowledge and understanding come through a developed sense of what works and what doesn't over time while you have played. Each instrument and even instrumentalist on the same instrument can express themselves in a different way. However, general body language cues are easily read and can be translated across many instruments such as a slouched or slumped positioning or a smile. Utilizing these attributes allow for easy expression on a universal level and can be built off of to express different things through the music.

While gesture is often viewed as a key piece of musical performance and expression it is possible to have an expressive performance without any physical gesture outside of required mechanical motions. Someone who is blind, has a performer hidden from sight, or is listening to a recording can still be moved by music. Therefore, gesture is not needed to complete the experience of a piece of music but it can greatly impact it. However, since music is a personification of emotions and art its is beneficial for the performer to use whatever capacities, including movement, that they have in order to express the music and emotions. While not seeing gesture can still allow someone to experience the music fully, lack of gesture, in deadpan or no motion, can negatively impact a viewer's experience. There is a balance and a proper level of gesture that benefits the performance, with no gesture degrading from the viewing experience and too much possible

messing with the mechanical requirements and perception of the performance as it can be overly distracting.

While it is easy for a human to use body language to express themselves, it is difficult to replicate these motions in a robotic system. When a robot it moving, sometimes people can place similarity to certain body parts onto the robot, humanizing it and anthropomorphizing it in order to better understand the system in their mind. When programming motions into a system the goal is to move in such a way that similar ideas and associations are formed in multiple viewers, allowing replication and repetition of certain emotional gestures. This is the hardest part of making a robotic system expressive as you are working with a single factor, the motion of a single limb. When a performer functions they are using their multiple limbs, body, face, or other extraneous factors such as clothing in order to augment their expression. According to Professor Rohde it is hard to replicate or describe definitively as it is an innate or observable known ingrained in humans through social interactions.

When making gesture, while humans can utilize their entire body to add to their performance, it is actually beneficial to isolate these gestures to specific locations allowing for a clearer transfer of messages and what the performer is trying to portray. Isolating this motion cuts out extra movement that may not necessarily mean anything and draws the focus to the specific location that is giving or guiding the expression. However, if a performer plans it out and coordinates it, they can use multiple body parts to indicate gesture, using the combination to their advantage, However, it must be deliberate and used together instead of random or unique movement in different locations. Another benefit a human performer has over a robotic arm is that they can vary the location of their expression throughout a performance, using their face for one thing during one passage and their torso in another, while a robot can only use the motion of its limb in order to indicate these gestures and expressions.

4.2.5 Physical Gesture as Discussed with Organist Wesley Hall

Wesley Hall holds a very different view of expression in performance than some other instrumentalists. To him, music is sound that is organized by a human and intended to be music. This stems heavily from his training as an organist. When playing the organ, every extraneous motion is wasteful, detracting from the technique and process required to create the music. When moving across a piano there are motions of sometimes multiple feet to cover and if there is extra motion in these situations your accuracy and precision suffers. These gestural aspects are just how people perceive the required mechanics of playing the instrument. When playing the piano it is additional mechanical requirements that often drive the differences in gestures. You must use the weight of your arm, the positioning and motion of your wrists, and the fluidity of your hands in order to create and modify the sound of the hammers on the strings. While an audience may perceive this as additional and added gesture, it is in fact just the motion required to create the correct sonic signature. To him, since there are ways to input variables digitally drawn into midi that will output in the exact same way as a quantized performance, physical performance itself is unimportant to the creation of music. He still believes that it is possible for these motions to augment a performance but only if it is worked up to so that it doesn't take away from the sonic performance.

When looking at how an organist moves during their performance, a lateral movement vs an arced one can be determined by the type of organ. These motions are efficient and only move in the direction they need to in order to cover the distance. These motions are all introduced purely in order to create the music itself and not to add expression to the piece. It is possible that this motion can show the audience more of what is happening than if they had no visual, like the experience of listening to a recording. In addition, Wesley is a trained Harpsichord player. When playing harpsichord you must be light and still, as adding any additional arm weight can have a negative effect on the resulting sound.

4.3. Software Accuracy

A major component of the entire musical analysis pipeline is the audio processing engine. It is also a potential source of error and can skew the results of our robotic system. The engine itself is trained using audio and MIDI data, consisting of changes in pitch, volume, and tone. In order for the project to fulfill objective III, this system must be able to properly recognize features in the audio and tell the robot how to interpret them.

We collected training data in 30-second chunks, each of which consisted of an independent musical melody. These were recorded using a Blue Snowball condenser microphone and corresponding MIDI data was recorded on an M-Audio Keystation-49. The audio and MIDI information were gathered simultaneously.

The audio input was all produced by a vocalist, who sang while simultaneously playing the same melody on the MIDI keyboard. Singing is not as discrete and strict as other means of producing music, and thus was used for our audio samples. The singer could change their vowel shape or volume very easily to reflect expressive changes in the music. And these changes were clear such that the layman could understand them, for example changing from an "*AH*" sound to an "*OOH*" sound.

Changes in pitch are the most obvious and easiest to interpret in an audio signal. See fig 4.3.1 for a spectrogram, in which there are 5 distinct pitches played one at a time. The entire series of frequencies shifts up for higher pitches and down for lower pitches at each moment in time. On the other hand, some changes are more subtle. See fig 4.3.2 for a spectrogram of changing vowel shapes. The pitch does not change, but the structure of the overtones does. This kind of analysis becomes more complicated and is where the RNN comes into play.

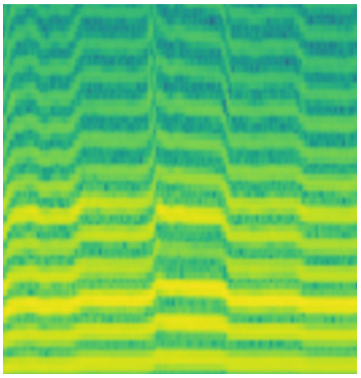


Fig 4.3.1: Spectrogram for Changes in Pitch

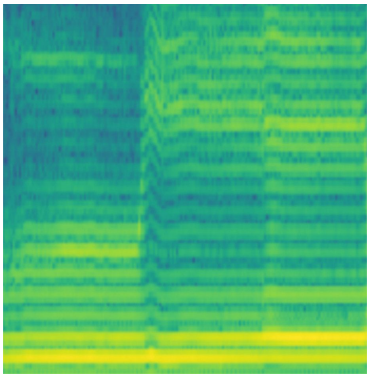


Fig 4.3.2: Spectrogram for Changes in Vowel

We ran multiple trials with varying configurations and parameter values for the RNN. First, we began by modifying the number of epochs the fitting algorithm would use for training the network. This number represents how many times the entire dataset is iterated when training. See table 4.3.1 for a list of trials and accuracies for various numbers of epochs.

Table 4.3.1: Accuracies for Number of Epochs

Number of Epochs	**Model Accuracy**
epochs = 3	Training: **0.3792** Testing: **0.5816**
epochs = 4	Training: **0.4307** Testing: **0.5707**
epochs = 5	Training: **0.4587** Testing: **0.6534 (Best)**
epochs = 6	Training: **0.4607** Testing: **0.5991**
epochs = 10	Training: **0.4738** Testing: **0.5818**
epochs = 20	Training: **0.5598** Testing: **0.4185**
epochs = 30	Training: **0.6070** Testing: **0.4024**

We also experimented with various architectures to process the audio data. Some of these included adding extra layers or filters — each emphasized a different kind of data processing. For example, adding more dense layers would make the network better at understanding the nature of overtones in the frequencies at a given instant in time, but may introduce some overfitting. For all code samples below, the following environment in Python was used:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, TimeDistributed

model = Sequential()
```

First, we considered the simple classifier for determining pitch. We first tried a simple model for the network, using a single LSTM layer and a single dense layer.

```
model.add(LSTM(255, input_shape=input_shape, return_sequences=True))
model.add(TimeDistributed(Dense(30, activation='softmax')))
```

Training accuracy: **0.5315**
Testing accuracy: **0.5817**

Then we start over, but add more dense layers to the configuration:

```
model.add(LSTM(255, input_shape=input_shape, return_sequences=True))
model.add(TimeDistributed(Dense(30)))
model.add(TimeDistributed(Dense(30)))
model.add(TimeDistributed(Dense(30, activation='softmax')))
```

Training accuracy: **0.5956**
Testing accuracy: **0.5247**

With more dense layers, there is a slight amount of overfitting, as the training accuracy is a bit higher than the testing accuracy.

With more dimensions between the dense cells, there is even more overfitting:

```
model.add(LSTM(255, input_shape=input_shape, return_sequences=True))
model.add(TimeDistributed(Dense(255)))
model.add(TimeDistributed(Dense(100)))
model.add(TimeDistributed(Dense(30, activation='softmax')))
```

Training accuracy: **0.5495**
Testing accuracy: **0.4533**

We then tried adding more LSTM layers rather than dense layers

```
model.add(LSTM(255, input_shape=input_shape, return_sequences=True))
```

```
model.add(LSTM(30, input_shape=input_shape, return_sequences=True))
model.add(TimeDistributed(Dense(30, activation='softmax')))
```

Training accuracy: **0.5315**
Testing accuracy: **0.5817**
With a very simple model, with only two LSTM layers and one dense layer, the results are the most accurate. Due to the time-dependent nature of the samples, the LSTM layers are able to pick up on features that the dense layers cannot.

With the same model used for the regression analysis, the results were similar. Multiple regressions were used: one for volume, which was trained using the velocities of the MIDI notes played. Another was trained using the data collected from the MIDI mod wheel, and was associated with the vowel shape of the audio input. This vowel shape was performed as a smooth gradient between the vowel “OOH” and “AAH” with a clear and consistent shape to the mouth and lips for each. These values were both bounded to 0-127, with 0 being mapped to a very closed “OOH” sound and 127 being mapped to a very open and tall “AAH” sound. Values produced by the regression that fell outside of these bounds were clipped (i.e. anything less than zero would become zero) to ensure the range would remain consistent. The regression for volume was less accurate, with an overall testing accuracy of 46% for the primary dataset, using one minus the normalized root-mean-square deviation to calculate accuracy. The same data yielded 68% accuracy for the vowel shape detection regression. The measurement of volume was harder to determine from the spectral information alone, as the only factor that changed was the amplitude. The actual nature of the overtones did not change much. On the other hand, the vowel shape led to clear changes in the overtone structure.

5. Conclusion & Future Recommendations

Music, while it is considered by some to be a human creation, can be replicated and presented by autonomous systems. Humanistic and musical characteristics can be analyzed and transferred to robotic systems in multiple ways, both sonicly and physically. There are many small nuances that can affect a performance in various ways. We successfully built the software and hardware for the robot to play piano. Despite certain limitations of its mechanisms, the system was able to produce an expressive musical performance based on audio input.

For further work, it would be great to explore a means to make the finger actuation quicker. Some pieces of piano repertoire, which would be fascinating to study, demand a rather quick tempo and complicated changes in hand and finger placement. Many works by romantic composers, such as Liszt or Chopin, are quite expressive but also demand this caliber of precision and timing. This would allow for more quick and complex melodic lines to be played. Additionally, it would be great to collect more data for training the audio processing neural network. More factors of the music, such as tempo, timbre, etc. would be possible to explore given a larger dataset, with the inclusion of more samples and more features.

Future research in the areas of human robotic interaction can explore not only the gesture of a robotic performer but how a robotic performer can interpret and react to a human co-performer or conductor’s motions. Looking deeper into how humans guide and indicate their desired gestural output and how they vary from person to person. Given more time, and deeper analysis through motion capture technology, these gestures can not only be further analyzed but quantitatively analyzed.

Appendix A: Robot Controller Source Code

Source for controller.py

```
from .arm_controller import ArmController
from .hand_controller import HandController

from cmd import Cmd

import time
import json
import threading

NOTES_BLACK =
[126,None,123,121,None,118,116,114,None,111,109,None,106,104,102,None,99,97,None,94,92,90,None,87,85,Non
e,82,80,78,None,75,73,None,70,68,66,None,63,61,None,58,56,54,None,51,49,None,46,44,42,None,39,37,None,34
,32,30,None,27,25,None,22]

def get_white_notes():
        result = []
        for i in range(127):
                if i not in NOTES_BLACK:
                        result.append(i)

        return result

NOTES_WHITE = get_white_notes()


finger_motion_full_delay = .300 # milliseconds
min_note_duration = .100

class MasterController():
        def __init__(self,hand_serial,hand_baud,arm_ip,arm_port):

                self.hand = HandController(hand_serial,hand_baud)
                # self.hand.listen(pt)

                self.arm = ArmController(arm_ip,arm_port)
                self.arm.connect()

                self.thread = threading.Thread(target=self.loop)
                self.action_queue = []
                self.finger_states = [False,False,False,False,False]

        def play_note_simple(self,note):
                finger = 0
                state = True
                self.hand.cmd_raw(finger,state)
                self.arm.move_j([200,0,100])
```

```
	def play_stream(self,stream):
		self.stream = stream
		self.start_thread()


	def start_thread(self):
		self.thread.start()

	def execute_action(self,action):


		print('executing',action)



		if action.type == 'noop':
			pass

		if action.type == 'move':
			action.is_done = self.arm.move_c_auto(action.args[0:3]).is_complete

		if action.type == 'finger_down':
			self.finger_set(action.args[0],True)

		if action.type == 'finger_up_all':
			self.fingers_reset()


	def execute_action_queue(self,time=0):
		while len(self.action_queue) > 0 and self.action_queue[0].is_ready(time):
			print('Executing at t='+str(time))
			self.execute_action(self.action_queue.pop(0))


	def add_action(self,action):
		self.action_queue.append(action)
		if action.delay == 'wait_prev':
			action.delay = 'wait_cond'
			action.ready_condition = self.action_queue[-2]

print('wait_cond',self.action_queue,self.action_queue[-2],self.action_queue[-2].is_done)

	def arm_pos_for_range(self,r):
		if r.type == HandRange.TYPE_BLACK:
			index = NOTES_BLACK.index(r.start.pitch) - 38
			return [10 + index * 23.5, 40, -70]

		index = NOTES_WHITE.index(r.start.pitch) - 44
		return [index * 23.5,0,-85]

	def loop(self):

		window_size = 5

		predelay = 0
```

```
                current_range = HandRange.range_from_notes(self.stream.notes)
                print('range start',current_range.start)

                self.arm.move_j(self.arm_pos_for_range(current_range))
                time.sleep(1)

                prev_note = None

                start = time.time()


                while True:
                        now = time.time() - start - predelay
                        focus = now + window_size

                        self.execute_action_queue(now)


                        current_stream = self.stream.notes_in_window(now,focus)
                        next_note = current_stream.next_unprocessed()




                        if not next_note:
                                if len(self.stream.notes_after(now).notes) > 0:
                                        continue
                                break

                        time_until_note = next_note.time - now

                        next_note.processed = True


                        if next_note in current_range:
                                # Move the right finger at the right predelay

                                inter_delay = 0
                                if prev_note and next_note.pitch == prev_note.pitch:
                                        inter_delay = 0.5



self.add_action(RobotAction('noop',time=now,delay=time_until_note-inter_delay))
                                self.add_action(RobotAction('finger_up_all'))

self.add_action(RobotAction('noop',time=next_note.time-inter_delay,delay=inter_delay))

self.add_action(RobotAction('finger_down',args=[current_range.get_index(next_note)]))

self.add_action(RobotAction('noop',time=next_note.time,delay=finger_motion_full_delay))

                                print('action simple',time_until_note)

                        else:
```

```
                                new_range =
HandRange.range_from_notes(current_stream.notes_starting_at(next_note).notes)

                                if next_note not in new_range:
                                        print('not in new range')
                                        print(new_range.start.pitch,[n.pitch for n in
current_stream.notes],now,next_note.pitch)
                                        # We are looking too far ahead
                                        continue


                                estimated_delay = 0

                                # Finger must go up from current position, and down at new position
                                estimated_delay += finger_motion_full_delay
                                estimated_delay += finger_motion_full_delay

                                new_pos = self.arm_pos_for_range(new_range) # BIG TODO WTF

                                arm_delay = self.arm.time_move_j(new_pos,250)

                                # How long will the arm take to move?
                                estimated_delay += arm_delay




                                # It takes too long to move to the next note - we should delay
everything
                                if estimated_delay > time_until_note:
                                        predelay += estimated_delay - time_until_note +
min_note_duration
                                        # self.stream.add_delay(estimated_delay - time_until_note +
min_note_duration)

self.add_action(RobotAction('noop',time=next_note.time-estimated_delay-min_note_duration,delay=min_note_
duration))
                                else:

self.add_action(RobotAction('noop',time=now,delay=time_until_note-estimated_delay))

                                self.add_action(RobotAction('finger_up_all'))

self.add_action(RobotAction('noop',time=next_note.time-estimated_delay,delay=finger_motion_full_delay))

self.add_action(RobotAction('move',args=new_pos))#+[next_note.velocity/127.0]))

self.add_action(RobotAction('noop',time=next_note.time-estimated_delay+finger_motion_full_delay,delay='w
ait_prev'))#delay=arm_delay))

self.add_action(RobotAction('finger_down',args=[new_range.get_index(next_note)]))

self.add_action(RobotAction('noop',time=next_note.time-finger_motion_full_delay,delay=finger_motion_full
_delay))

                                print('action big',estimated_delay,new_pos)
```

```
                                print('gonna play finger',new_range.get_index(next_note))


                                current_range = new_range
                                print('range start',current_range.start)

                        prev_note = next_note

        def arm_move_j(self,pos):
                self.arm.move_j(pos)

        def finger_set(self,finger,state):
                self.finger_states[finger] = state
                self.hand.cmd_raw(finger,state)
                # self.arm.move_j([200,0,100])
        def fingers_reset(self):
                for i in range(len(self.finger_states)):
                        if self.finger_states[i]:
                                self.finger_set(i,False)

class RobotAction(object):


        def __init__(self, type, **kwargs):
                self.type = type
                self.time = kwargs.get('time',None)
                self.args = kwargs.get('args',None)
                self.delay = kwargs.get('delay',None)
                self.is_done = kwargs.get('is_done',None)
                self.prev_action = kwargs.get('prev_action',None)
                self.ready_condition = kwargs.get('ready_condition',None)
                self.delay_start = 1e12
                self.arm_cmd = None

        def __str__(self):
                s = 'RobotAction '+self.type+' '
                if self.args:
                        if list(self.args):
                                s += ' '.join([str(x) for x in list(self.args)]) +' '

                if hasattr(self,'time'):
                        s += 'time='+str(self.time) + ' '

                if hasattr(self,'delay'):
                        s += 'delay='+str(self.delay) + ' '
                return s


        def is_ready(self,time):

                ready = True

                if self.type == 'noop':
                        if self.delay == 'wait_cond':
                                if callable(self.ready_condition):
                                        return self.ready_condition()
```

```
                        if isinstance(self.ready_condition,RobotAction):
                                return self.ready_condition.is_done()

                        return False

                return time > self.time + self.delay

        if self.type == 'waitfor':
                return self.ready_condition()

        return ready



        if hasattr(self,'ready_condition') and self.ready_condition != None:
                ready = self.ready_condition()
                if not ready:
                        return False
                self.delay_start = time
                self.ready_condition = None

        if hasattr(self,'prev_action') and self.prev_action != None:
                ready = self.prev_action.is_ready()
                if not ready:
                        return False
                self.delay_start = time
                self.prev_action = None

        # DELAY after above conditions met
        if hasattr(self,'delay') and self.delay != None:
                ready = time - self.delay_start > self.delay

        # FIXED time
        if hasattr(self,'time') and self.time != None:
                ready = ready or self.time > time




        return ready


PITCH_MAX = 90
PITCH_MIN = 30

class NoteStream():

        @classmethod
        def load_file(self,filename):
                with open(filename,'r') as f:
                        notes = []
                        line = f.readline()
                        while line != '':
```

```
                        data = json.loads(line)
                        notes.append(Note(data['note'],data['time']))
                        line = f.readline()

                return NoteStream(notes)

def __init__(self, notes=[]):
        self.notes = []
        for n in notes:
                self.add_note(n)

def add_note(self, note):
        self.notes.append(note)
        self.notes.sort(key=lambda n: n.time)

def notes_in_window(self, start, end):
        result = []
        for n in self.notes:
                if n.time > start and n.time < end:
                        result.append(n)

        return NoteStream(result)

def notes_after(self, time):
        result = []
        for n in self.notes:
                if n.time > time:
                        result.append(n)

        return NoteStream(result)

def notes_starting_at(self, note):
        result = []
        for n in self.notes:
                if n.time >= note.time:
                        result.append(n)

        return NoteStream(result)


def next_unprocessed(self):
        notes = [n for n in self.notes if not n.processed]
        if len(notes) == 0:
                return None
        return notes[0]

def add_delay(self,delay):
        for n in self.notes:
                n.time += delay

def multiply_time(self,mul):
        for n in self.notes:
                n.time *= mul

def min_pitch(self):
        val = PITCH_MAX
```

```
                for n in self.notes:
                        val = min(val,n.pitch)
                return val

        def max_pitch(self):
                val = PITCH_MIN
                for n in self.notes:
                        val = max(val,n.pitch)
                return val



class HandRange():
        TYPE_BLACK = "black"
        TYPE_WHITE = "white"

        def __init__(self, start):
                self.start = start
                self.type = self.note_type(start)

        @classmethod
        def range_from_notes(cls,notes):
                if len(notes) == 1:
                        return HandRange(notes[0])

                subarr = notes[:-1]
                last = notes[-1]
                r = cls.range_from_notes(subarr)
                r2 = HandRange(last)

                if subarr in r2:
                        return r2

                return r


        def __contains__(self,note):
                if isinstance(note,list):
                        return self.are_inside(note)

                return self.is_inside(note)

        def get_index(self,note):
                if note not in self: return None

                try:
                        result = self.get_span().index(note.pitch)
                        return result
                except:
                        return None

        def is_inside(self,note):
                if self.type != self.note_type(note):
                        return False

                return note.pitch in self.get_span()
```

```
        def are_inside(self,notes):
                for n  in notes:
                        if not self.is_inside(n):
                                return False
                return True

        def get_span(self,size=5):
                space = self.get_space()
                start_index = space.index(self.start.pitch)

                return space[start_index:start_index + size]


        def get_space(self):

                if self.type == self.TYPE_BLACK:
                        return NOTES_BLACK
                return NOTES_WHITE

        def note_type(self,note):
                if note.pitch in NOTES_BLACK:
                        return self.TYPE_BLACK
                return self.TYPE_WHITE




class Note():
        def __init__(self, pitch, time, velocity=100):
                self.pitch = pitch
                self.time = time
                self.processed = False
                self.velocity = velocity



def pt(data):
        print(str(data) + " -- " + ", ".join([str(a) for a in data]))


class ControllerPrompt(Cmd):
        prompt = 'ROBOT> '


        def set_controller(self,controller):
                self.controller = controller

        def do_exit(self, inp):
                print('Exiting')
                return True

        def do_armmove(self, inp):
                tok = inp.split(' ')
                if len(tok) == 4:
                        print('Need exactly 3 arguments for position X, Y, and Z')
```

```
                        return False

                pos = [float(n) for n in tok]
                self.controller.arm_move_j(pos)

        def do_armmarc(self, inp):
                tok = inp.split(' ')
                if len(tok) == 7:
                        print('Need exactly 6 arguments for 2 positions, each X, Y, and Z')
                        return False

                pos = [float(n) for n in tok]
                self.controller.arm_move_c(pos[1:4],pos[5:7])


        def do_armzero(self, inp):
                self.controller.arm.move_zero()

        def do_fingerdown(self, inp):
                finger = int(inp)
                self.controller.finger_set(finger,True)

        def do_fingerup(self, inp):
                finger = int(inp)
                self.controller.finger_set(finger,False)

        do_EOF = do_exit



cmd1 = 0
cmd2 = 0
def loop():
        global cmd1
        global cmd2
        c = 1
        d = 1e9
        e = 1e9
        start = time.time()
        while True:
                now = time.time() - start
                if c == 1:
                        cmd1 = controller.arm.move_j([0,0,0])
                        c = 2

                if cmd1 != 0 and cmd1.complete and c == 2:
                        controller.finger_set(0,True)
                        c = 3

                if c == 3:
                        time.sleep(0.5)
                        controller.finger_set(0,False)
                        c = 4
                        time.sleep(0.1) # finger motion full delay
```

```
        if c == 4:
                d = now
                cmd2 = controller.arm.move_j([400,0,0])
                e = time.time()-start
                c = 5

        if cmd2 != 0 and cmd2.complete and c == 5:
                controller.finger_set(4,True)
                time.sleep(0.5)
                controller.finger_set(4,False)
                c = 6

        if now - d >= 1.6:
                d = 1e9

        if now - e >= 1.6:
                e = 1e9
```

Source for arm_controller.py

```
import socket
import time
import threading
import fcntl, os
import errno
import math

SPEED_DEFAULT = 5000
DELAY_NETWORK = 0.1

class ArmController():

        def __init__(self, host='localhost', port=3000):
                self.host = host
                self.port = port
                self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                self.sock.settimeout(20)

                fcntl.fcntl(self.sock, fcntl.F_SETFL, os.O_NONBLOCK)

                self.seq_num = 1
                self.commands = []

                self.current_pos = None

        def connect(self):
                server_address = (self.host, self.port)
                print ('Connecting on %s port %s' % server_address)

                self.sock.connect(server_address)

                print('Connected!')
```

```
        print('')

        threading.Thread(target=self.listen).start()

def listen(self):
        while True:

                try:
                        msg = self.sock.recv(4)
                except socket.error as e:
                        err = e.args[0]
                        if err == errno.EAGAIN or err == errno.EWOULDBLOCK:
                                time.sleep(1)
                                print('No data available')
                                continue
                        else:
                                # a "real" error occurred
                                print(e)
                                return
                else:
                        # got a message, do something :)
                        print("GOT",msg)
                        seq_num = int(msg[0]) - 48
                        print("GOOTTA",seq_num)

                        for c in self.commands:
                                if c.seq_num == seq_num:
                                        c.complete = True
                                        self.current_pos = c.args[0:3]


def move_default():
        pass

def move_zero(self):
        print('CMD: Reset zeros')
        return self.cmd_raw(2)

def move_j(self,pos,speed=-1):
        print('CMD: MoveJ')
        if speed < 0: speed = SPEED_DEFAULT

        self.bounds_check(pos)

        return self.cmd_raw(1,pos+[speed])

def move_c(self,pos_c,pos_d,speed=-1):
        print('CMD: MoveC')
        if speed < 0: speed = SPEED_DEFAULT

        self.bounds_check(pos_c)
        self.bounds_check(pos_d)

        return self.cmd_raw(4,pos_c+pos_d+[speed])

def move_c_auto(self,pos,speed=-1):
```

```
                if not self.current_pos:
                        raise Exception("Cannot move_c_auto without knowing current position")

                if dist(pos,self.current_pos) < 15:
                        return self.move_j(pos,speed)

                pos_c = [
                        (self.current_pos[0] + pos[0])/2,
                        (self.current_pos[1] + pos[1])/2,
                        pos[2]+abs(self.current_pos[0] - pos[0])/2
                ]
                print('auto circle',self.current_pos,pos_c,pos)
                return self.move_c(pos_c,pos,speed)

        def move_c_h(self,pos,height,speed=-1):
                if not self.current_pos:
                        raise Exception("Cannot move_c_auto without knowing current position")


                pos_c = [
                        (self.current_pos[0] + pos[0])/2,
                        (self.current_pos[1] + pos[1])/2,
                        pos[2]+ abs(self.current_pos[0] - pos[0]) * height/2
                ]
                return self.move_c(pos_c,pos,speed)

        def time_move_j(self,pos,speed):
                if not self.current_pos:
                        return None

                cp = self.current_pos
                np = pos
                dx = cp[0]-np[0]
                dy = cp[1]-np[1]
                dz = cp[2]-np[2]

                return math.sqrt(dx*dx + dy*dy + dz*dz) / speed + DELAY_NETWORK  # + 1 for network
delay


        def bounds_check(self,pos):
                if pos[0] < -200 or pos[0] > 600:
                        raise Exception('Safety warning: Position out of bounds')
                if pos[1] < -200 or pos[1] > 100:
                        raise Exception('Safety warning: Position out of bounds')
                if pos[2] < -100 or pos[2] > 100:
                        raise Exception('Safety warning: Position out of bounds')

        def cmd_raw(self,cmd,args=[]):
                payload = " ".join([str(cmd),str(self.seq_num)] + [str(a) for a in args] + ["#"])


                self.sock.send(bytes(payload,"utf-8"))

                print('Sending command type = '+str(cmd)+' with payload = '+payload)
```

```
                command = ArmCommand(self.seq_num,args)
                self.commands.append(command)

                self.seq_num = (self.seq_num + 1) % 10

                return command

        def disconnect(self):
                self.sock.close()

def dist(p1,p2):
        d0 = p1[0] - p2[0]
        d1 = p1[1] - p2[1]
        d2 = p1[2] - p2[2]

        return math.sqrt(d0*d0 + d1*d1 + d2*d2)

class ArmCommand():
        def __init__(self,seq_num,args):
                self.complete = False
                self.seq_num = seq_num
                self.args = args

        def is_complete(self):
                return self.complete
```

Source for hand_controller.py

```
import serial
import time
import threading

class HandController():
    def __init__(self, port, baud=9600):
        self.port = port
        self.baud = baud
        self.arduino = serial.Serial(port,9600,timeout=5)

        time.sleep(2)  # Must wait for setup to complete


    def cmd_raw(self,finger,state):

        if finger < 0 or finger > 4:
            raise Exception("Invalid finger ID: "+str(finger))

        finger_byte = finger
        state_byte = 1 if state else 0

        cmd_string = chr(finger_byte+1) + chr(state_byte+1)

        print(", ".join([str(a) for a in cmd_string]))

        self.arduino.write(cmd_string.encode())
```

```
    print(cmd_string.encode())

    time.sleep(0.01)

def force_relax_all(self):
    self.hand.cmd_raw(0,False)
    self.hand.cmd_raw(1,False)
    self.hand.cmd_raw(2,False)
    self.hand.cmd_raw(3,False)
    self.hand.cmd_raw(4,False)


def listen(self,callback):
    self.listen_thread = threading.Thread(target=self.read, args=(callback,))
    self.listen_thread.start()

def read(self,callback,n=-1):
    c = 0
    while True:
        line = self.arduino.readline()
        callback(line)
        time.sleep(0.01)
        c = c + 1
        if c >= n and n > 0:
            break
```

Appendix B: Source Code for Auditory Analysis

Source for load_sample.py

```
import matplotlib
matplotlib.use('Agg')
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft
from scipy.io import wavfile
import json
import tensorflow as tf
import math


BLOCK_SIZE = 400

midi_min = 60
midi_max = 90
midi_span = midi_max - midi_min

def midi_sample_make(note,vel):
    note = int(note)
    arr = np.zeros(midi_span)
    if note - midi_min < 0:
        return arr

    arr[note - midi_min] = vel
    return arr

def load(sample_id):

    fs, data = wavfile.read(sample_id+'.wav') # load the data
    b = [(e/2**8.)*2-1 for e in data] # normalized on [-1,1)



    freq_boxes = []

    sample_step = 10
    sample_rate = 8000
    NFFT = 512
    duration = len(b) / sample_rate # seconds
    window = np.hanning(NFFT)

    for i in range(0,len(b)-NFFT,sample_step):
        block = b[i:i+NFFT]
        block = np.multiply(block,window)
        freq = fft(block)
        freq = freq[:(math.floor(len(freq)/2))]
        freq = np.real(freq)
```

```
            freq_boxes.append(freq)

        # Save the spectrogram for confirmation
        plt.specgram(b,NFFT=512, Fs=2, Fc=0)
        plt.savefig(sample_id+'-specgram.png')

        midi_samples = []
        midi_mod = []
        try:
            with open(sample_id+'.log','r') as f:
                line = f.readline()
                time = 0
                while line != '':

                    data = json.loads(line)
                    if data['velocity'] != 0:
                        while time < data['time']:
                            time += 1./sample_rate * sample_step
                            midi_samples.append(midi_sample_make(data['note'],data['velocity']))
                            midi_mod.append(data.get('mod',0))

                    line = f.readline()

        except FileNotFoundError:
            result = {
                'x':freq_boxes
            }
            result['blocks'] = blockize(result,BLOCK_SIZE)
            return result

        n_input_total = len(freq_boxes)
        n_output_total = len(midi_samples)

        n_total = min(n_input_total,n_output_total)

        # n_input_train = math.floor(n_total*0.7)
        # n_output_train = math.floor(n_total*0.7)



        mod_max = np.max(midi_mod[0:n_total])

        result = {
            'x':freq_boxes[0:n_total],
            'y':[tf.keras.utils.to_categorical(np.argmax(m),30) for m in midi_samples[0:n_total]],
            'y2':[[x/mod_max] for x in midi_mod[0:n_total]]
        }
        result['blocks'] = blockize(result,BLOCK_SIZE)
        return result


    def blockize(obj,blocksize):
        result = {}

        for i in obj:
            arr = np.array(obj[i])
```

```
        result[i] = np.array_split(arr,len(arr)/blocksize+1)

    return result
```

Source for train.py

```
#!/usr/bin/env python
# coding: utf-8

SAMPLES_TRAIN = ['r110','r112','r113','r114','r115','r116','r117','r118','r119','r120'];
SAMPLES_TEST = ['r111'];
MODEL_ID = 'm005';


import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, CuDNNLSTM, TimeDistributed, Activation
from tensorflow.keras import backend as K
from tempfile import TemporaryFile

import load_sample



def zeropad(input,size):
    x = input['x']
    y = input['y']
    result = {}
    result['x'] = np.pad(x,((0,size-len(x)),(0,0)),'constant')
    result['y'] = np.pad(y,((0,size-len(y)),(0,0)),'constant')
    return result

samples_train = [load_sample.load(s) for s in SAMPLES_TRAIN]
samples_test = [load_sample.load(s) for s in SAMPLES_TEST]

# Maximum size for all samples
max_size = max([len(s['x']) for s in (samples_train+samples_test)])

samples_train = [zeropad(s,max_size) for s in samples_train]
samples_test = [zeropad(s,max_size) for s in samples_test]


tf.reset_default_graph()
K.clear_session()

#np.array([[next(([i] for i, x in enumerate(m) if x), None) for m in midi_samples[20000:23657]]])

# [samples, time steps, features]

x_train = np.array([s['x'] for s in samples_train])
y_train = np.array([s['y'] for s in samples_train])
```

```
x_test = np.array([s['x'] for s in samples_test])
y_test = np.array([s['y'] for s in samples_test])



print('pt',x_train.shape,x_test.shape,y_train.shape,y_test.shape)
input_shape = (None,x_train.shape[2])

model = Sequential()
model.add(CuDNNLSTM(255, input_shape=input_shape, return_sequences=True))
model.add(CuDNNLSTM(30, input_shape=input_shape, return_sequences=True))
model.add(TimeDistributed(Dense(30, activation='softmax')))


opt = tf.keras.optimizers.Adam(lr=0.001, decay=1e-6)
print(model.summary())
model.compile(
    loss='categorical_crossentropy',
    optimizer=opt,
    metrics=['categorical_accuracy'],
)


print('done compiling')
print(model.fit(x_train,
          y_train,
          verbose=2,
          epochs=5))

print('done fitting')
print(model.evaluate(x_test,y_test))
print('done evaluating')

model.save(MODEL_ID+'.h5')
```