Haptic Keyboard Aid

A Major Qualifying Project Report:

Submitted to the Faculty

Of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

_____

Jose Meneses

_____

Julia Rugo

_____

Date: December 18, 2014

Approved:

_____

Major Advisor: Professor John A. Orr

_____

Co-Advisor: Professor Scott D. Barton

Abstract

This MQP was to develop a prototype for a haptic learning device for piano. The aim of the device is to use haptic feedback to reinforce proper piano playing. The device is a keyboard that has electromagnets embedded in the keys. The user of this device wears gloves that have magnets on the fingertips. The device uses the electromagnets to pull the user's fingers to the correct finger positioning. The keyboard has touch sensors to detect user input and RGB LEDs to provide visual queues while in operation. The device is controlled using an Arudino microcontroller and Max computer software. The final product is a prototype that implements the haptic technology and can demonstrate its operation with some piano exercises.

Acknowledgments

Our team would like to thank Professor John Orr and Professor Scott Barton for advising this project. Their guidance and support was a valuable resource for the development of this project. We would also like to thank Dominic Lopriore for giving us access to his shop and helping us with the design and construction of the piano keyboard. Without their assistance this project would have not been completed successfully.

Table of Contents

Table of Figures

1. Introduction

Learning to play a musical instrument is an activity that takes time and effort. Many beginners struggle with playing skills and have difficulty finding motivation to practice. There are a variety of tools on the market that provide assistance for learning to play an instrument. These tools focus on helping users learn basic playing skills, while also adding a fun component to the process, such as a game or enticing visuals; an example would be lights that correspond to the musical notes being played. This approach can liven a player's instrument experience and provide an incentive to keep practicing. Such products not only increase enthusiasm towards learning an instrument, but are also meant to efficiently speed up the learning process.

A popular instrument is the piano, and there are multitudes of resources available in software and physical form to aid in the piano learning process. Most learning tools available for beginning piano players use technology that focuses on visual interaction. Products, such as PianoMaestro [1], focus on utilizing visual aids with LEDs on a piano or keyboard to teach lessons. Steinway Etude [2] is a software application for iPads or iPhones that simulates a piano keyboard and gives preemptive visual instruction to help the player anticipate which notes should be played. Both of these products provide feedback that helps the user learn. However, these products only use visual signals for learning. The goal of this MQP is to add a second stimulus to improve the learning process. Our team has developed a physical device meant to enhance a beginning piano student's learning experience by guiding the user's hands and fingers to the correct piano playing position on the keyboard. The project implements a tactile stimulus to explore the viability of haptic feedback in piano learning technology.

2. Background

In order to design a haptic learning device it is important to look at previously implemented and successful piano-learning devices and techniques. The concept of this project is to combine both tactile and visual cues into an educational piano keyboard; therefore, it is vital to understand haptic feedback and study some of the applications in this field. An overview of design approaches for haptic systems reveals different implementation methods for haptic technology. Learning devices that implement haptic systems have displayed positive results, such as the Georgia Tech PianoTouch device [3].

To create an effective educational keyboard, piano-learning techniques must be evaluated. Music teachers use a variety of academic approaches to teach the piano. Most include repetitive finger exercises and practicing songs. Repetitive exercises develop coordination between the brain and nerves controlling the action. Repeating piano exercises enhances coordination between a player's fingers and hands and will familiarize the player with proper finger placement upon the keys on the keyboard.

2.1 Piano Pedagogy

Pedagogy is defined as a method or practice of teaching, especially when applied to an academic subject or theoretical concept [4]; piano pedagogy focuses on musical skills that are used to teach individuals studying the instrument. However, there are a variety of pedagogical approaches that used to teach piano. Common pedagogical methods include scales and exercises that help beginners acquire proper finger placement and playing techniques for the piano. While practicing, students are also familiarized with the musical note's sound and its associated key [5].

Developing proper piano-playing technique assures that the time and effort a student dedicates to practicing will be used most efficiently [8]. Internationally-renowned piano teacher Martha Beth Lewis graduated with a Ph.D. in musicology from the University of Florida and has a successful reputation in piano pedagogy for beginners [6] There is a large "Readers' Questions and Answers About Pedagogy" section on Lewis' website that is aimed mostly for piano teachers and parents of piano students [7]. Many of the questions address teaching students with physical or mental disabilities and how to teach, correct, or refine proper piano playing technique to these individuals, as well as tips on how to fix common bad habits that beginning students struggle with. Questions 114, 138, and 156 on Martha Beth Lewis's website refer to teaching hand position, correcting bad fingering, and reversing a student's dropping thumb habit, respectively [7]. Those new to piano will struggle with maintaining good hand position while playing [9]. Lewis says that students must be aware of their wrists and thumbs while practicing [7]. The wrist is meant to hold a player's hand and fingers strong over the keys. A proper hand position is curved as if the student is holding a baseball [Figure 1]. Lewis uses techniques that she calls "piranha", which is when she tickles the bottom side of a student's wrist when they sag. Beginning students also struggle with "dropping the thumbs" while playing, so that they are pointing towards the floor [Q. 156]. Lewis suggests playing "alligator", in which

a piano teacher “bites” the bottom of the students thumb by giving it a soft pinch or tickle. This makes the student aware of their problem so that they are able to focus on correcting it while practicing. Thumbs do not follow the same arched finger form for proper playing. Rather, the thumb rests on its side upon the key it is placed upon [10], as can be seen in Figure 3 below. Correct finger position is also important, and allows a student to play songs and scales easily and efficiently. Using the wrong finger position can lead to developing bad practice habits, which can be detrimental to a beginner’s progression [11].

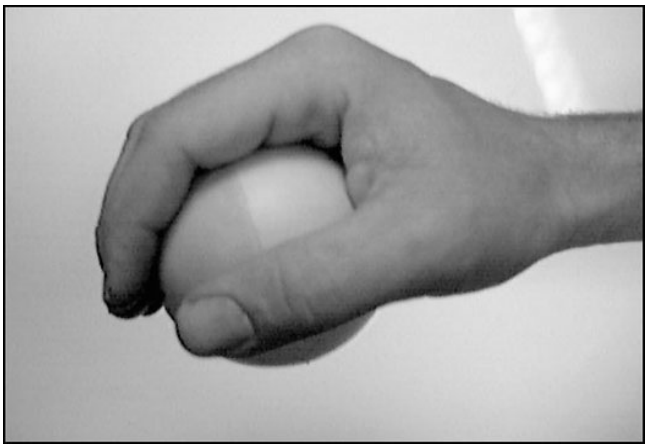


Figure 1: Proper Hand Position, Baseball Curve [10]

Figure 2: Proper Hand Position Piano [8]

Figure 3: Proper Hand Position, Emphasis on Thumb [12]

Acquiring proper technique is an important aspect to be considered when learning to play the piano, and is mostly a process of brain and nerve development [11]. The combination of proper

technique and efficient learning methods decreases the amount of time it takes for a novice player to progress. The physical act of playing the piano exercises one's physical and cognitive coordination between the brain and nerves [13]. Association between which finger corresponds to which musical note on the keyboard must be processed and acted upon instantaneously. Practicing the piano and acquiring technique develops faster nerve connections. Muscle memory is a common term associated with procedural memory, which helps a person become very good at something through repetition. If a beginning piano student practices good technique while playing, the novice will build procedural memory that quickens the brain's execution of muscle control. This will be referred to as muscle memory throughout this report. Procedural memory has no control over whether a task is being performed correctly, and will continue to develop no matter what action is being repeated. If a piano player practices bad technique while learning and practicing, muscle memory will develop around the bad habits. These habits are hard to reverse and hinder learning progression [13].

Muscle memory is an essential part of learning to play the piano and developing proper finger-note association. To play the piano most efficiently with fluid hand and finger motions, a player's fingers, on both hands, are assigned a number 1-5 that will correspond to a song or scale's notes on the keyboard [14]. The thumb is labeled 1, index finger 2, middle finger 3, ring finger 4, and the pinky 5; these can be referred to as the first, second, third, fourth, and fifth fingers and have equivalent associations for both the left and right hands, as can be seen in the Figure 4 below.

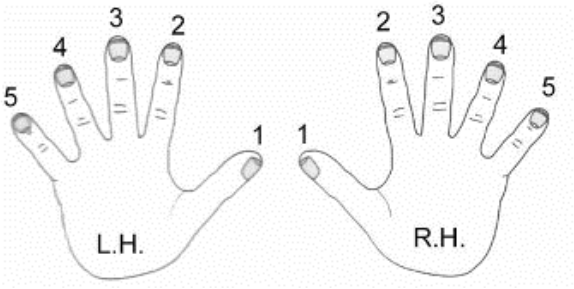


Figure 4: Left and Right Hand Number Association [14]

There are a variety of practice techniques that help students develop and refine piano-playing skills. Such are scales, finger exercises, and breaking up a song into measures or separate components [15]. Scales improve a piano student's muscle memory, coordination and concentration. Practicing scales is essential to learning to play the piano with the most natural and effective fingering. The repetition and symmetrical fingering performed while playing scales strengthens muscle memory and note-to-finger association [17].

Finger exercises improve and develop independence between each finger, also enhancing muscle memory and control [18]. When it comes to learning how to play a song, it is extremely helpful to break the song up into chunks and separate musical components, master a part of the piece, and then continue on to the next measure [19]. One way to do this is by separating the left and right hands and practicing each hand's part separately until the player is comfortable with that component. By mastering the left hand and right hand's musical progression

separately, achieving coordination between the fingers and hands will be much easier due to muscle memory developed between each separate hand and its accompanying music [17].

Figure 5: Finger Muscle Memory Cartoon [20]

2.3 Market Research

Learning to play the piano is no longer confined to a music teacher instructing piano students to practice scales and song assignments. In 2014, there are more options available to beginning piano students. The alternative to music lessons is technological piano-playing learning devices and downloadable software. Piano technology ranges from player pianos, digital pianos, and software that specializes in a variety of learning or musical aspects [21]. Modern software applications can convert music files to sheet music, download, unzip, and process music files from the Internet, as well as perform harmony analysis and chord names that a musician plays.

Many technological systems that are aimed to help beginners learn the piano are age specific. Some will specify an age range, such as the "Kids' Toy Piano" [22], while others incorporate games and lights to intrigue children learning the piano, such as Hammacher Schlemmer's "Children's Learn to Play Keyboard" [23]. The "Children's Learn to Play Keyboard" includes instructional software programmed with over 100 lessons for beginning piano students that range from learning basic piano skills, such as proper finger placement on the piano, to more complex lessons that help a child student learn to play songs and read music.

Figure 6: Children's Learn to Play Keyboard [23]

As can be seen in Figure 6 above, the “Children’s Learn to Play Keyboard” incorporates computer software and a physical keyboard to practice the lessons on. A similar product is MGA Entertainment’s “iTikes: A place where tech meets play” keyboard [24]. iTikes pedagogical system consists of a standalone keyboard that has the ability to control volume and a tempo/pitch tuner. The downside to this product is that it requires an Apple iPhone, iPad, or iPod touch to download the corresponding iTikes application [25]. The app is where musical games teach children limited piano skills. Although this product is meant to aid in the learning process of a beginning piano student, iTikes is appealing to the eye with its light-up major keys, but only includes 4 built-in songs and received a 2.5 stars out of 5 in the iTunes app store. The app is free and the keyboard can be found at stores such as Target, ToysR’Us, or Walmart, and is selling for $35.79 online [26]. The “Children’s Learn to Play Keyboard” has many more learning exercises than iTikes, and a USB cable connects the “Children’s Learn to Play Keyboard” computer software to its physical, 49-key device. Reviews indicate 5 out of 5 stars; the multimedia system costs $149.95 on Hammacher Schlemmer’s online site [23].

Figure 7: iTikes Keyboard with iPad Software Attached [26]

Apple devices, such as iPads, iPhones, and iPods, have multiple applications to help a beginner learn how to play piano; we will refer to the iPad as an example device throughout this section. Piano pedagogy apps for the iPad range in complexity, features, and price. Of the eight piano-learning iPad apps recommended by AJ Dellinger of MacLife, only one does not involve an interactive piano keyboard [27]. For these applications, the user, or student, will tap the iPad's interactive touch screen to "play" the piano notes. Examples of free iPad applications are "Steinway Etude", "Piano Lesson PianoMan", and "Piano...". Steinway Etude displays sheet music and a keyboard [Figure 8] that follows along with the musical score's measures and indicates which keys should be played during that time [2]. There is another interactive view that displays the musical notes to be played and the duration of the length of the note. This corresponds to musical components of the piece, such as tempo and rhythm via note notation, such as half, whole, or quarter notes.

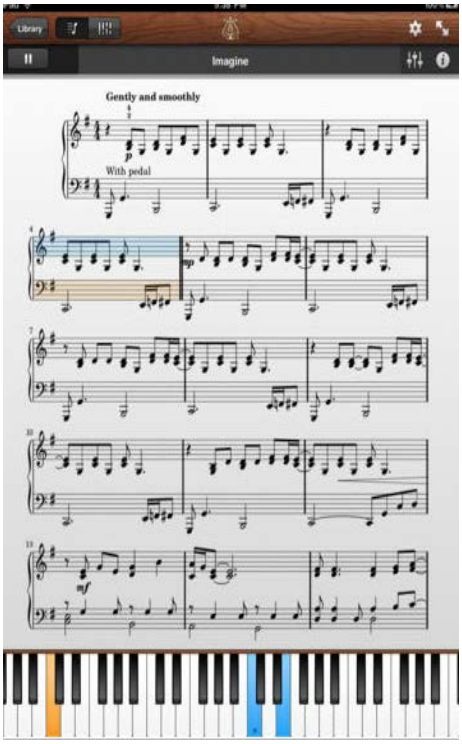


Figure 8: Steinway Etude Sheet Music [2]

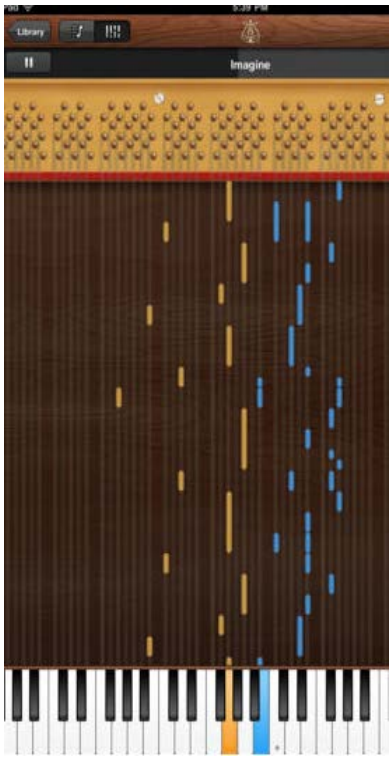


Figure 9: Steinway Etude Tempo and Rhythm View [2]

Piano Lesson PianoMan is advertised in Dellinger's article as a piano learning device, but the application's description and reviews infer a game similar to guitar hero [27]. No technical learning skills are mentioned. Piano... is an iPad app created by Obie Leff [28] that is described as an interactive piano with 50 pre-programmed animated songs and keys that light up with the music's rhythm [Figure 10].

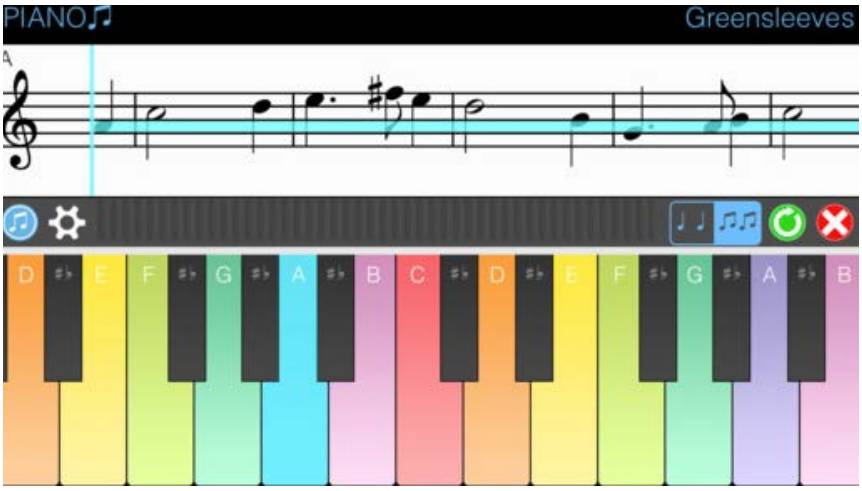


Figure 10: Piano... iPad Application [28]

Other piano pedagogy iPad applications are not free. Depending on the application's technicality, iPad apps that are aimed to teach beginners how to play piano range from $2.99 (SmileyApp's "Piano Tutor for iPad") to $19.99 (IK-Multimedia's iGrand Piano) [29, 30]. Piano Tutor for iPad has great user ratings. Customers boast about enjoying the piano-learning experience and the application's educational value that teaches the user how to read notes and practice rhythm [29]. IK-Multimedia's iGrand Piano sells for $19.99, and is worth its price [31]. iGrand Piano has "quality, variety, and professional features" including MIDI-controllable adjustable parameters and a low-latency touch screen response that makes it feel like the student is playing real piano keys. The user can choose from 18 different multi-sampled pianos such as the Classical Baby Grand and a Jazz Upright piano. The app offers a recorder, metronome, and multiple compatibility features for Audiobus and MIDI applications. When connected to a computer, the iGrand Piano can send its output an audio app such as AmpliTube and GarageBand where the sound can be filtered, recorded, and looped [30]. These audio application compatibility functions and an interface that implements the feeling of playing a natural piano gives the iGrand Piano application its credibility. Reviews claim a "tremendous piano app" but complaints about advertisements and a distorted output make an overall rating of 4.5 out of 5 stars [31]. The following figures show iPad screenshots of iGrand Piano's metronome, recording, tempo, and playback capabilities.

Figure 11: iGrand Piano; Metronome and Tempo [30]

Finally, the "PianoMaestro Learning System", created by Ken Ihara, is a computer software learning device that connects via a USB cable to a strip of lights that is placed upon the back of a full, real-life piano keyboard [1]. The lights correspond to what note should be played at what time by communicating with the computer software that a student has opened on their computer. The software translates standard MIDI files into a musical notation that then appears onscreen in the form of sheet music. The player is to read along with the music while being guided by PianoMaestro's light learning system. Combining computer software and a visual stimulant, as well as a real-life piano, a student learning the piano should experience a quicker and easier learning process than learning without the aid of the device.

Figure 12: PianoMaestro; Piano, Keyboard Light Strip, and Computer Software [1]

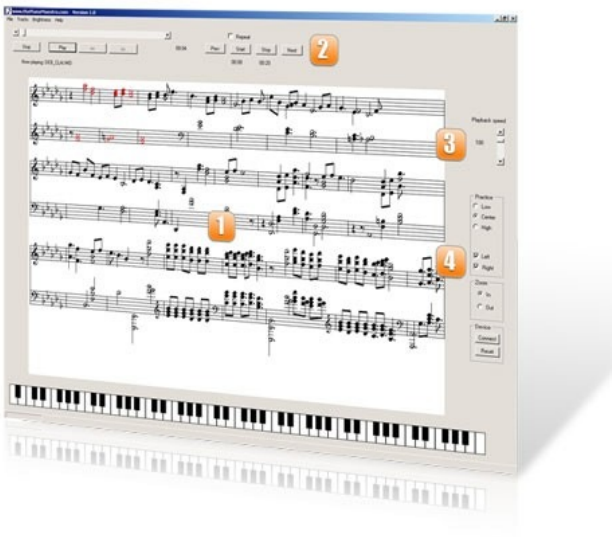


Figure 13: PianoMaestro's Musical Notation Output After Conversion of a MIDI File [1]

2.4 Haptic Technology

Computer scientists have developed numerous methods to emulate sounds and visuals with computers; however, it has been a harder task to replicate tactile signals [32, pg. 0]. The challenge behind stimulating the sense of touch originates from the limitations in the design of typical interfaces. For instance, it is hard to transmit the reality of playing tennis through a standard video game controller. The game controller cannot imitate the shape, texture, weight, forces and overall feel of a tennis racket; nevertheless, these controllers do offer some tactile feedback through vibrations. This example is just one of many applications in the field of haptic technology.

The science of haptics focuses on studying how the sense of touch works, understanding the manipulation of environments and the perception of motion under different circumstances. Like all other senses, the sense of touch is crucial so that humans can perceive their surroundings. What separates touch from the other senses is that humans can receive tactile feedback from any point on their body. There are many receptors and nerves scattered right under the skin. These receptors are distributed with varying densities in different areas of the body. The hand is the main body part used to touch surroundings and it contains 40 muscles that allow for a great deal dexterity [32, pg. 1]. According to Heller and Schiff, the human body can distinguish two different tactile stimuli that are within 5ms of each other. The sense of touch is perceived and processed at an impressive rate, which is five times faster than vision. [33, pg. 2] Not only is the response to touch fast, but also it is accurate. Humans are capable of detecting movements of 0.2μm [33, pg. 2]. A study by Robles de la Torre, PhD, states that losing the sense of touch can lead to implications such as “impairment of hand dexterity, loss of limb position perception, and the inability to walk” [33, pg. 3]. The study and development of haptic technologies is fundamental to the improvement of technological interactions since touch is an invaluable resource that the human body uses to react to stimuli.

The study of haptic technology is documented as early as the 1960's; however, haptic technology only reached new heights in development with the advancement of robotics [34]. The fields of robotics and haptics have been developed alongside, since both have overlapping goals and applications. Many of the early implementations of haptic technology were focused on creating a device that allowed remote control of certain machinery. This is known as teleoperation [34]. The purpose of these systems was to allow for a user to operate hazardous objects that were located at a distance. For example, this allowed for safe manipulation of nuclear materials through robotic arms. Eventually, haptic technology was introduced to other areas like medicine, aerospace, teaching, music and many others. But it was not until 1993 that the Artificial Intelligence Laboratory at MIT created the first device that allowed for manipulation of virtual objects [32, pg. 2]. This invention promoted a surge in the research of haptics. The application of haptics can be divided into two categories. The first category focuses on simulating the sensations of a certain action or object. The second category aims to provide motor feedback in order to control or regulate motion.

Haptic technology is present in many of the devices that are common to everyday use. Some of the most common application of haptics is in new cell phones. Today, many of the phones available on the market use a touch screen. There is haptic technology in many of these phones that attempts to improve the user experience by generating tactile feedback on the touch screen. The tactile feedback is provided through piezoelectric sensors to simulate the experience of pressing a real button. The haptic feedback in touch screens helps reduce user error and also reduces the time to finish a task [32, pg. 6]. For instance, FingerFlux is an application of haptic feedback on touch surfaces. It uses electromagnetic fields to guide the users when using a touch surface. Through user studies, FingerFlux showed that users drastically increased accuracy when pressing different items on the screen [35]. FingerFlux especially facilitated the use of the surface without the need to look at the screen.

Other notable devices that incorporate haptic technology are the PHANTOM and Cyber Grasp.

Figure 14: PHANTOM interface from SenseAble Technologies [37]

The PHANTOM device is a haptic tool that helps simulate different situations. It is controlled with the pen shaped arm that can be moved with different resistances. The PHANTOM can be programed to simulate movement under water, the feel of modeling clay or even the resistance of skin to a needle. This device uses three motors to generate the different forces required in the simulation. The maximum continuous force that the device can exert is .88N[38]. The device

has six degrees of freedom for position sensing, but only three degrees of freedom for providing force feedback. The limitation of this device is that it has reduced range of motion and it is focused on helping in specific technical training [32, pg. 4].

Figure 15: CyberGrasp system from Immersion Corporation [38]

The CyberGrasp is another type of haptic device that works through a mechanical exoskeleton. This glove-like contraption uses an exoskeleton to enforce different resistances for finger movements. The CyberGrasp system consists of the exoskeleton, a position tracker, and a CyberGlove [39]. Every component communicates with the CyberGrasp Force Control Unit (FCU). The exoskeleton is responsible for providing the force feedback through actuators. The exoskeleton can exert up to 12N of continuous force per finger [39]. The position tracker is a third party component that measures the position and orientation of the hand in space. The position tracker communicates with the FCU through a tracker interface unit. The CyberGlove is worn under the exoskeleton and it is responsible for measuring the joint angles of the fingers, hand and wrist. The main purpose of this device is to generate the resistance necessary to feel virtually generated objects. Through CyberGrasp, it is possible to feel a virtual model of a soccer ball. The five actuators in the device will create the necessary resistances to emulate the properties of touching a desired object [32, pg. 4]. The CyberGrasp and the PHANTOM interface fall under the category of haptic products that focus on simulating environments. The next haptic invention belongs to the category that is aimed to provide motor feedback rather than simulating a motion or texture.

Figure 16: PianoTouch developed in Georgia Tech [40]

In the realm of music, students from Georgia Tech have developed a device called PianoTouch. The goal of the device is to serve as a teaching tool for piano. PianoTouch is composed of three

main components: the actuators (10 small vibrators), a microprocessor with a Bluetooth module, and a laptop computer [41]. The laptop is responsible for producing the music and sending the synchronized vibration instructions to the Bluetooth module. The module then passes the information to the microprocessor so that it sends the necessary current to activate the actuators. The vibrations motors are set to vibrate for 350 ms for each note when they are activated. The goal of the device is to help with the passive learning of piano. This means that users could practice the piano while focused on other daily activities. The results from their user study showed that the tactile feedback allowed the subjects to play a song with less number of errors. A user of the study stated that PianoTouch was helpful for memorizing the finger order and patterns for playing. This left the users with the task of focusing on recalling the position for one finger because the rest of the fingers "fell into place". The device decreased the amount of information required to retrieve from memory. An issue with this device is that the vibrations proved to be distracting to some of the test subjects. In addition, the device itself was obtrusive and not very comfortable. Users who participated in the study suggested that the glove should be thin and breathable [41].

2.5 Design Approach for Haptic Systems

There are many approaches for designing haptic systems. The initial step for designing a haptic device involves defining the following [42, pg 115]:

1. The specific function of the device
2. A specific technical approach and market-oriented context

It is vital to establish the function of the haptic device at the beginning of the design process. The function determines the intended results from utilizing the device. The market-oriented context of a device is dependent on many factors such as: product competition, time frame of development, customer audience, personal resources and budget [42, pg 115]. These factors should serve as constraints in order to select the best method to developing the haptic system. Selecting the technical approach for a haptic application is dependent on the type of haptic feedback needed for the system.

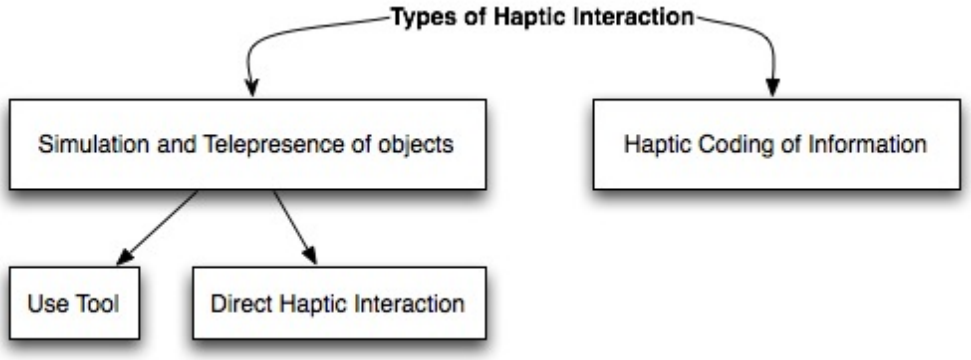


Figure 17: Types of Haptic Interaction [42]

The types of haptic feedback can be divided into two main categories (Figure 17) as follows: simulation and telepresence of objects and haptic coding of information [42, pg 117]. The first category involves recreating or interacting with objects that are either virtual or remotely accessible. This usually requires a tool that can interact with the user to control and interact with the simulation or telepresent object. This tool can be a joystick, a pen, or other objects that can be hand held. An example of a haptic device that falls under the first category is the PHANTOM device by SensAble. This first category also includes direct haptic interaction. In this subcategory, the user can feel the physical attributes of the object. Some of these attributes include: mass, texture, elasticity and plasticity [42, pg 117]. For direct haptic interaction, not only is it necessary to have a tool that is used to manipulate the object but also it is important to consider the methods needed to recreate the textures. For instance, CyberGrasp is a device that focuses on direct haptic interaction since it involves manipulation of 3D simulated objects.

The second category entails abstract information that is communicated through the sense of touch. This haptic interaction relies on the source of the information. The information could rely on single time events, spatial orientation, or volumetric information [42, pg 117]. A simple example of this is the vibrations setting on phones. If the phone is set to vibrate, then whenever there is an incoming call or message the phone will vibrate. This project corresponds to this category, since it develops a device that associates the pull of magnetic fields with correct finger positioning while playing piano.

The basic parts of a haptic system are the power supply, the computer controller, and the physical device [33, pg 70]. The physical device is constituted of sensor(s) and actuator(s). The sensors are used to record information from surrounding elements and the actuators provide a response to an electrical stimulus. An analog-to-digital converter (ADC) translates the information from the sensors to a digital value that is used by the computer. The computer processes the information and then provides the output to a digital-to-analog-converter (DAC). The DAC uses the digital information and converts it to an analog signal that can be used to control the actuators. The interaction among all the components of a basic haptic system are depicted in the following image:

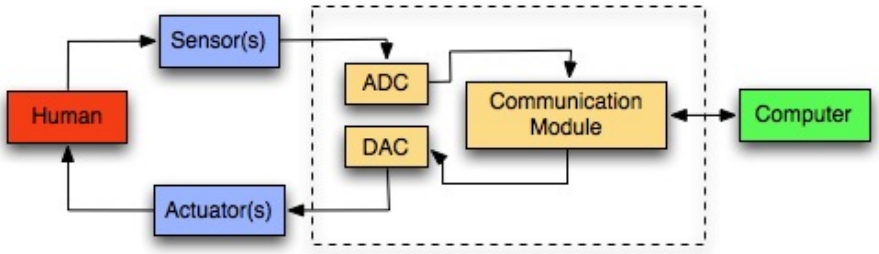


Figure 18: Basic representation of Haptic System [33, pg. 70]

The most important component in a haptic feedback system is the actuator, since it directly influences the haptic interaction with the user [33, pg 76]. In a haptic device, the actuator is the component that is responsible for exerting a stimulus on the human body in order to create a certain sensation. There are three most important actuators for haptic interfaces: electrical

actuators, pneumatic actuators, and hydraulic actuators. Some of the important factors to consider while selecting an actuator are the following: speed of operation (response time), safety, mechanical transparency, workspace, number of degrees of freedom, maximum applicable forces, and compactness [33, pg 76].

2.5.1 Electrical Actuators

Electrical actuators include: AC, DC, and stepper motors. These actuators are usually small and straightforward to install. The disadvantage with electrical actuators is that they only generate small torques in relation to their size and weight. They are also rigid so they cannot bend and it becomes hard to apply them to wearable devices [33, pg 76].

2.5.1.1 Electromagnets

This project aims to create a device that is small and as unobtrusive as possible. The device will attempt to guide the users fingers into correct position. In order to achieve this effect, our deice will attach magnets to the users fingertips and will implement electromagnets into the keys of our device. An electromagnet behaves similar to a permanent magnet. The difference is that an electrical current generates the magnetic field in an electromagnet. Electromagnets are usually composed of a conductive wire that is coiled around some ferromagnetic core. The purpose of the core is to intensify the strength of the magnetic field produced by the wire [43]. Ferromagnetism is a property attributed to materials that can be magnetized or that are strongly responsive to magnetic fields.

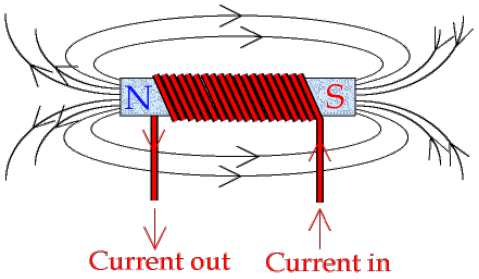


Figure 19: Magnetic field in electromagnet [44]

The diagram in Figure 19 shows the magnetic field formed as the current goes through the coil. The field emanates from two poles at the extremes of the core known as “north” and “south” or “positive” and “negative”. When two identical poles are close together they repel each other. When two different magnetic poles come in proximity then they attract each other. The unique characteristic of electromagnets is that it is possible to change the polarity and strength of the magnetic field by manipulating the direction and amplitude of the current. Electromagnets are powered with a battery or any other source that produces a current [43]. The following equation is used to relate the magnetic field with the current and the number of turns of the wire:

$$B = \frac{NI\mu}{L}$$

In the above equation, B represents the magnetic flux density in tesla [45]. N is the number of turns the wire has on the electromagnet and I is the current in amperes going through the wire. μ is the permeability of the electromagnet core material and it is represented in newton per square ampere. Finally, L is the length, in meters, of the magnetic field.

Electromagnets can be found in a wide range of electrical applications. A common use for electromagnets is in electric doorbells [44]. These doorbells use an electromagnet to pull a metal clapper against a bell. The electromagnet is turned on only when the doorbell is rung. Another popular use for electromagnets is in hard disks. Data is stored in a hard disk by magnetizing small segments on a thin metal surface into different patterns. The patterns saved onto the disks are representations of binary code, which can then be read and translated into useful information. There are many other uses for electromagnets, including, but not limited to, motors, magnetic locks, actuators, particle accelerators, and speakers.

One of the newer applications for electromagnets is in haptic technology. In haptic feedback systems, electromagnets are a popular choice for providing response. As mentioned in the haptic technology section, FingerFlux is a product that uses electromagnets to provide a haptic output. FingerFlux uses an array of electromagnets to control user interaction with a display [46]. The user needs to attach a small permanent magnet to one of their fingertips in order to interact with the haptic surface. The electromagnets grid is then controlled to either pull or repel the magnet on the user's finger. In this design, each magnet had a height of 34.5mm and the diameter was 19.5mm [46]. The magnets contained 3,500 turns of wire and were driven at 40V DC and 255 mA. The purpose of using electromagnets in this application is that the haptic feedback can be provided up to 35mm above the surface of the display. Figure 20 shows the electromagnet array of 12 x 19 used in FingerFlux.

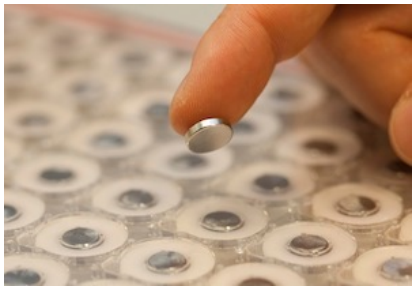


Figure 20: FingerFlux electromagnet array [47]

Another instance of electromagnets in a haptic feedback system is a mouse interface developed in Korea. The device uses an electromagnet to create attraction between an optical mouse and a ferromagnetic trackpad [45]. The purpose of this attraction is to create different types of responses for the user. For instance, the attraction can be used to generate higher friction when the user moves to certain areas on the screen. This would allow the user to locate certain objects on the screen fairly easily. The advantage provided in this haptic mouse is that it does not change the conventional mouse functionality. It also creates intuitive responses based on the different environments on the screen. The next improvement for this mouse is to use more than one electromagnet in order to provide control over more dimensions.

2.5.2 Pneumatic Actuators

These actuators usually use air pressure to convert energy into a mechanical motion. Pneumatic actuators are usually lightweight and are composed of a piston, a cylinder and valves or ports. The disadvantage of using pneumatic actuators is that they are stiff and they require lubrication to deal with static friction [33, pg 76].

2.5.3 Hydraulic Actuators

As the name suggests, hydraulic actuators work with a fluid, which is usually oil. They use this hydraulic pressure to generate the mechanical motion. They are bulky devices that can deal with heavyweights and high forces in haptic interactions. Their biggest disadvantage is the need to maintain and replace the liquid in the actuator [33, pg 77].

3. Project Goals

There have been other designs that aim to create devices that help improve the piano learning process and to increase the motivation for practicing the instrument. The approach for many of these products is to provide visual cues. This MQP aims to improve upon the design of many of the piano learning devices by introducing haptic feedback into the learning process, since haptic technology has shown to have a positive impact in learning applications.

The goal of this MQP is to implement and assess the viability of a haptic learning device to aid piano students develop good habits while learning finger positioning for piano. Overall, this device will focus on reducing errors in finger positioning on a keyboard. This will allow beginner students to shorten the amount of time it takes to develop the correct muscle memory associated with these actions. The device will consist of a keyboard that uses haptic feedback to enforce proper playing on a piano. The design approach to achieve these attributes and to fulfill the goal of the project will be described in the following section.

4. Preliminary Design Approach

Our design will use haptic technology to input data from a capacitive touch sensor that will be used to light the keyboard's LEDs and activate its electromagnets. A visual programming language will be used to output these statuses for each key. The project's firmware will be a microcontroller detecting sensor inputs and outputting the LED and electromagnet statuses for the keys.

The user will be wearing gloves with colors on the "fingernail" and magnets on the "finger pads". There will be a different color on each finger that match the five colors that the LEDs will display. To show which key is supposed to be played, an LED will light and the key's electromagnet will turn on. The LED color will indicate which finger should play the key for proper playing technique; when the key's electromagnet is turned on, the user will be able to feel which key is supposed to be played because of the newly present magnetic field tugging upon the glove's magnet. The LED and glove color coordination will assure that a player uses the correct finger to play the key.

Users will be able to between two modes, Follow and Play, and two piano exercises, Ode to Joy, a song composed by Beethoven [62], and the C Major Scale. The modes will determine the keyboard's functionality for each exercise. In Follow mode, the user must press the proper key before the program will continue on to the next note instruction. Once input data corresponding to the proper key press is received, the next LED and electromagnet output states will be implemented. Alternatively, play mode is independent of user input. The LED and electromagnet key commands will be sent to the keyboard continuously, at a specified timing interval, or tempo. The keyboard's user is expected to play along with the output instructions. Ode to Joy and the C Major Scale are the two melodies that will determine the keyboard's output progression. Each exercise has a key signature of C Major, which contains no sharps or flats. This enhances the key signature's simplicity because the player does not need to play any black keys on a piano keyboard; therefore C Major is a good beginning point for novice piano students.

4.1 Description of Device

Our system is composed of two parts: gloves and a keyboard. Gloves worn by the user will have small magnets attached to the fingertips. The keyboard box imitates 2 octaves of a piano keyboard. Our project will only be testing functionality with the 15 white keys of these octaves. The box will contain 15 electromagnets that provide haptic feedback, one for each key in our device. One multicolor LED will be located at the top of each key and one electromagnet will be located under each key. A microcontroller will be used to control all these components. The microcontroller will be located inside the box and it will be communicating with Max on a computer through a serial port. The interaction for these devices is depicted in 21.

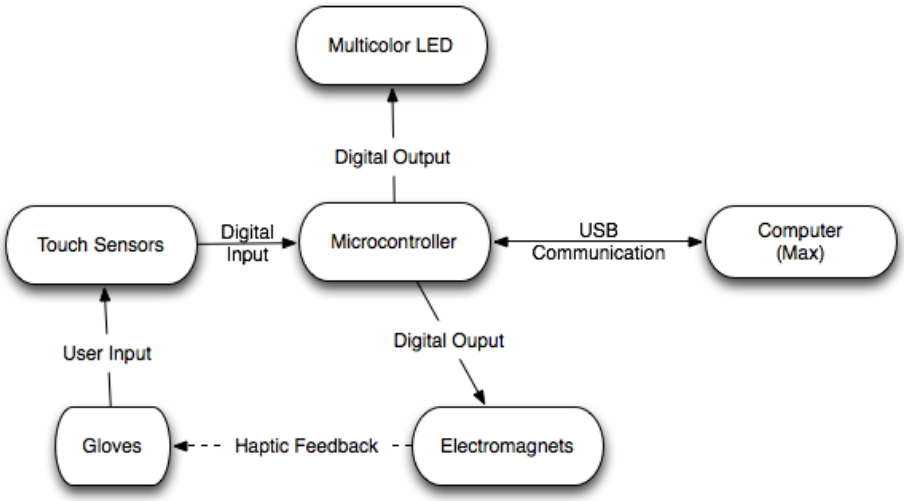


Figure 21: System Black Diagram

4.2 Multicolor LEDs

The LEDs wills serve as a visual aid for using the device. Each LED will be located at the top of each of the keys in our device. The LEDs will light up when the corresponding key must be played. We will use multicolor LEDs and there will be five possible colors for each LED. The color of the LED will be used to identify which finger should press which key. The user will be responsible for associating the correct hand to the necessary keys through deduction. The colors that are matched to each finger are shown on Table 1 below.

Color Name	Finger
Red	Thumb
Blue	Index
Green	Middle
Cyan	Ring
Magenta	Pinky

The multicolor LEDs, used to create the colors in the table above, have four leads. The following image shows the LED:

Figure 22: RGB LED [48]

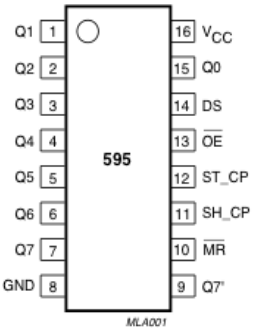
The longest lead is the common cathode for the LED. The pin furthest to the left is used to control the color red. The pin to the right of the cathode can control the color green and the last pin to the right is used to control the color blue. Each one of the color leads operates at its brightest when 20 mA are supplied to the pins. The red pin has a typical forward voltage of 2V while the green and blue pins have forward voltages of 3.2V. Each LED can create multiple colors based on which colored pins are being powered. The table for creating the colors is shown in Figure 23.

Colour Truth Table

red	green	blue	
ON	ON	OFF	yellow
OFF	ON	ON	cyan
ON	OFF	ON	magenta
ON	ON	ON	white

Figure 23: Color Truth Table for LEDs [49]

Our design requires 15 of these LEDs. In order to avoid using one digital port for each pin on the LEDs, they will be controlled using shift registers. This allows us to reduce the amount of digital ports used on our microcontroller to three. In addition to the three digital ports, to implement this design we require: a connection to the 5V pin, the ground pin (GND), 45 current limiting resistors, and 6 8-bit shift registers. The shift register chip that we will be using is the 74HC595. The description for each pin is shown in Figure 24.



PINS 1-7, 15	Q0 " Q7	Output Pins
PIN 8	GND	Ground, Vss
PIN 9	Q7"	Serial Out
PIN 10	MR	Master Reclear, active low
PIN 11	SH_CP	Shift register clock pin
PIN 12	ST_CP	Storage register clock pin (latch pin)
PIN 13	OE	Output enable, active low
PIN 14	DS	Serial data input
PIN 16	Vcc	Positive supply voltage

Figure 24: 74HC595 Schematic [50]

There are 16 total pins on the chip. As shown in the figure, pins 1-7 and 15 are the output ports. The three pins that control the shifting process are pins 11,12,14. These pins are commonly known as the clockPin, the latchPin, and the dataPin respectively. The clockPin gives the signal of when you are going to send a new bit. When the clock is set to high, it sends that bit to the dataPin so that it can be shifted in. The latchPin is turned on when all the bits have been shifted in and the data is ready to be outputted. For our design, we will first shift in the data for the LEDs that should be turned on and then we will enable the latchPin so that the LEDs change state. The GND pin goes to ground and the Vcc pin will be powered by the 5V pin from the microcontroller. The output enable will be set to low since it is active low and we always want to display an output. The master clear pin will be grounded. The chip only has 8 bits of output; however it is possible to add more shift registers in order to satisfy the amount of outputs. We can do this by connecting another chip to pin 9. This pin is the serial data out port. For our design, we need to do connect six shift registers together. The circuit for two shift register is shown in Figure 25.

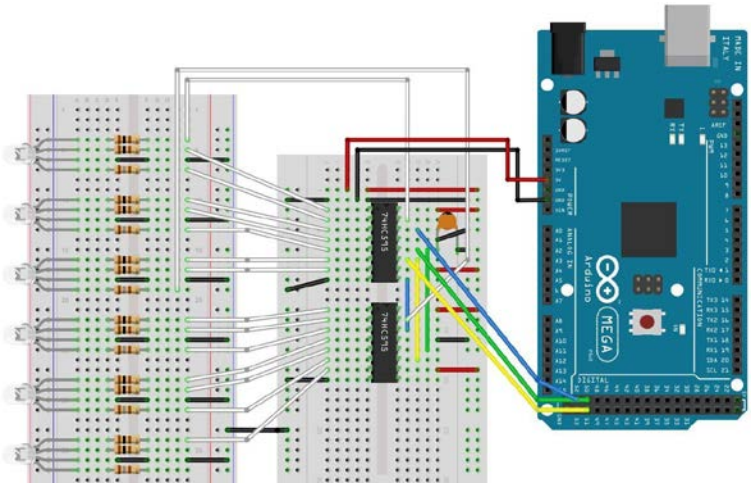


Figure 25: Breadboard Circuit for Two Shift Registers

The circuit above shows the setup of two shift registers with the RGB LEDs and the microcontroller. The black wires are GND connections and the red wires are the 5V connections. The latchPin is coded with the green wire. A 0.1µF capacitor is placed on this pin to prevent any flicker while latching. The yellow wires are the clockPin connections. It is important that all shift registers are connected to the same clockPin, latchPin, GND and Vcc. The blue wires are the dataPin connections. As shown in the image, the serial out of the first chip goes into the serial input of the second chip. This extends the number of bits available as outputs. Finally the white wires are the output connections to the LED pins. Each LED has a ground connection and one current regulating resistor in series with each pin.

4.3 Touch Sensors

The device will take user inputs through capacitive touch sensors. The surface of each key will be a touch sensor. The touch sensors will be used to determine when there are key presses in order to generate the sound and to control the next instruction for the device. The sensor will work by measuring the capacitance of the human body. The implementation of this technology will be done with the capacitive sensing library for Arduino.

Our device will require 15 touch sensors. For each touch sensor, we need two pins on our microcontroller. One pin acts as the send pin and the other as the receive pin. The microcontroller continuously changes the send pin state and then measures the time it take for the receive pin to match the state. This delay is stored in the microcontroller and provided as an output with arbitrary units. This design requires a medium to high value resistor that will be placed between the two pins. This resistor is used to regulate the sensitivity, because it influences the delay between the send pin changing and the receive pin changing. A typical value for the resistor in this application is 1MΩ. A visual representation of one touch sensor can be seen in the following image.

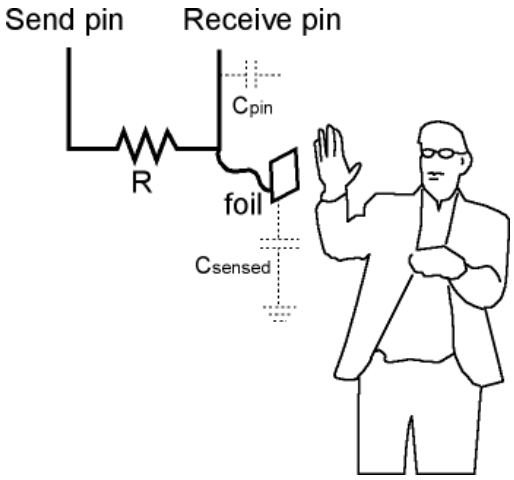


Figure 26: Capactivie Touch Sensor Circuit [51]

The receive pin senses touch through a piece of metal that is connected to the circuit with a wire. This metal can be covered with plastic or other materials in order to conceal the metal if

desired. A small capacitor should be placed between the sensor pin and GND in order to improve stability. A good value for this capacitor is 100 pF. It is also desired to add a capacitor in parallel with the user's body and GND to stabilize the reading even further. We will try to use aluminum or thin copper sheets for our touch sensor. These are good alternatives since they are easy to acquire and it is easy to shape them into piano keys.

4.4 Electromagnets

The electromagnets are the components responsible for creating the haptic feedback on the device. One electromagnet will be placed under each key. The electromagnets will provide the feedback to the user through the permanent magnets located on the fingertips of the gloves. The electromagnets will work by pulling the users finger to the desired keys. The main considerations for picking our electromagnet are the size, the magnetic field strength and the power requirements.

Our system will require 15 electromagnets. Each electromagnet needs to have a diameter smaller than 23.5mm in order to fit inside a piano key. The electromagnet that we will be suing has a diameter of 20mm and is depicted in the next image.

Figure 27: Electromagnet [52]

This electromagnet has a holding force of 25N or 5.6lbs that should be enough to provide the haptic feedback in our system. It is important to point out that the magnetic field strength drops off almost exponentially over distance [53]. The power consumption of this electromagnet is 3W. It operates at 12V DC at .25A. In order to power all the electromagnets our system demands a power supply. Our microcontroller will not be able to power the electromagnets, since they require more current that can be provided through the digital ports. It is risky to power the electromagnets directly through the microcontroller pins. When the electromagnet turns off, a large voltage is generated across the magnet that can easily damage the microcontroller. We avoid these issues by controlling our electromagnets with an NPN transistor and also placing a protective diode in parallel with the electromagnet. The driving circuit for one electromagnet is shown in Figure 28.

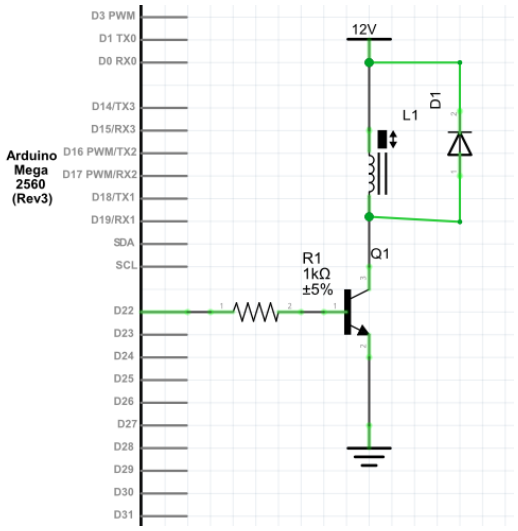


Figure 28: Circuit to Drive One Electromagnet [54]

The base of the transistor is connected to the digital port of the microcontroller. The Arduino will be programed to provide the current required to switch on the NPN transistor. The transistor that we will use is the TIP120. The electromagnet will be connected to an external power supply and to the collector of the transistor. A 1N4001 diode will be placed in parallel to the electromagnet to protect the circuit from voltage spikes. Each electromagnet requires 12 V DC and 250 mA; therefore we will need a 12V DC power supply that can provide at least 3.75 A.

4.5 Microcontroller

Our haptic feedback system requires a microcontroller in order to manage all the separate components in the device. One requirement for our microcontroller is to have enough pins to control the inputs and outputs of our system. In our system we need a total of 48 digital I/O pins (15 pins for the electromagnets, 30 pins for the touch sensors, and 3 pins for the LED shift register). Our team is most familiar with the Arduino environment; therefore, we decided to select one of their boards. The board we selected is the Arduino Mega 2560.

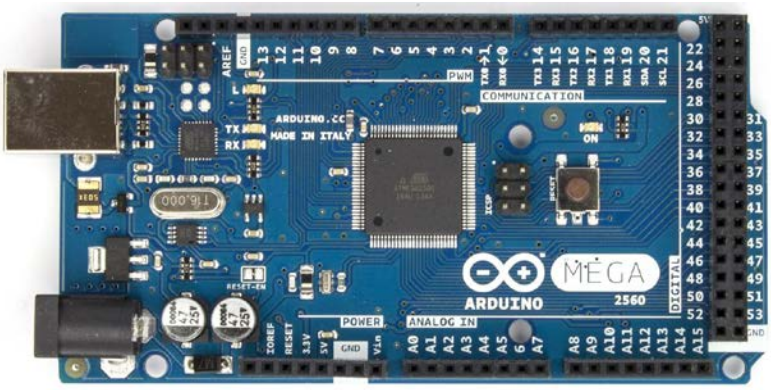


Figure 29: Arduino Mega2560 [55]

Figure 29, above, shows the Arduino Mega board. Some of the specifications for the board are shown below.

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Figure 30: Arduino Mega2560 Specification [55]

The board operates with the ATmega2560 chip. The board can be powered with the USB connection; however, this limits the current available to the ports. In order to power all our circuit components we will be using an AC to DC adapter with 12V and 2A. The DC current available for each port is 40 mA.

The microcontroller in the keyboard will communicate with the computer through a serial port. This means that pin 0 and pin 1 need to be explicitly used for the communication and cannot be used to control any components. The board will receive information from one input, the touch sensors. The data read from the sensors will be passed on to the computer as either a high or a low value. The microcontroller will receive data from the computer to control the two outputs. The data received to control the electromagnets will be a sequence of 1's and 0's to control the state of each port. The computer will provide the data to control the LEDs as a binary sequence that will be passed on to the shift registers.

4.6 Software: Max

Max is a visual programming language useful for building audio MIDI, video, and graphic interactive software programs, or for controlling the hardware of a physical application [56]. The language is split into three separate components that work together, Max, MSP, and Jitter. Max controls MIDI and discrete operations. MSP is the audio and signal processor of the program; Jitter renders graphics and video manipulation. Neither MSP nor Jitter will be applied to this project. The Max programming language can be incorporated into projects that include hardware interfaces, such as an Arduino microcontroller.

Patches are Max programs that are created by connecting interface objects together [57]. The objects within a patch have text names and different attributes to define their functionality [58]. Interface objects include dials, sliders, graphic keyboards, buttons, toggles, and switches that are manipulated by the computer's mouse or keyboard to store and send messages. Max

messages are passed between objects along a patch's virtual "cord". Objects have inlets and outlets that patch cords are connected to in order for objects to send and receive messages. Any data type that is passed between Max objects is referred to as a message, and can be in integer, float, bang, symbol, or list form [59]. Bang messages are specific to Max; they instruct objects to execute their functions. The programming language's order of operation is right-to-left and bottom-to-top [58]. This means that patch cord messages will be sent in the order of which cord connects to the right-most object first. Objects follow Max's right-to-let order of operation through their inlets and outlets. An object will accept and store messages in its right inlet, while the left inlet will receive messages from cords and trigger the object's operation

Max controls all of the inputs and outputs of our system's design [Figure 21]. While active, the software will constantly be on the lookout for input data from the device's touch sensors. Detecting the user's touch will be the only input data transferred from the keyboard to Max. Our design has two outputs that will be implemented by the Arduino microcontroller once it receives an execution command from Max. These outputs will control the LEDs and electromagnets of each key on the device. Sensor feedback will determine which key has been pressed by the user, and will affect all of the other components under Max's control. The following Max patches will control the keyboard's functionality: the user interface, serial data, exercise/song modes, and key states.

Within each music patch, send and receive message objects carry input and output data that will be evaluated based on the patch's mode and note sequence, whether it be the C Major scale or Ode to Joy song. Sensor data retrieves information regarding which key was just pressed, and the program determines if the input is correct; the LED and electromagnet output states associated with the key are also evaluated. Based on the mode and input data, the active music patch will output new LED and electromagnet commands for the next key in the note sequence. The keyboard's output states will change, alerting the user of the next note in the patch's musical progression.

4.7 Glove

Our haptic system uses electromagnets in the piano keys to provide the tactile stimulation. In order to transmit this information, it is necessary for the user to have a magnet attached to each one of their fingertips. Our device will require the user to wear gloves that will have disk magnets attached to each one of the fingertips. The gloves should be thin and comfortable. We will attach color strips to the top area of the fingers corresponding to the available LED colors. This will allow the users to match the correct finger with the correct key. The permanent magnets need to be as unobtrusive as possible while still providing enough pull from the electromagnets. The magnets that we will be using are shown in Figure 31. We will use magnets that have .375" inches in diameter and .031" in thickness. The pull strength for this magnet on a flat surface made of steel is 7.166 lbs. The magnets are cheap so we can easily try different sizes and strengths in order to find the optimal magnet for our device.

Figure 31: Magnet next to penny for scale [60]

4.8 Keyboard structure

The last component in our device is the keyboard box. The keyboard box is the enclosure that holds the other components. The box should be made out of a non-metallic material. We will use either wood or plastic. The topside of the box will be covered with an image for the two octaves of a piano keyboard. The metal pieces for the touch sensor will cover the white keys. The LEDs are the other visible component on the box, since they will be placed at the top of each key. There will be power cables coming out of the back of the box and the USB cable that will be connected to the computer. The dimensions of the box need to be at least large enough to fit two octaves. This requires that the length of the box should be at least 352.5mm and that the width of the box is at least 15cm.

5. Hardware Implementation

The hardware of this project was initially developed as independent components. The components of this project are the touch sensors, LED circuit, and electromagnet circuit. Once all the components were tested separately they were combined together to test full operation of the system. The overall system block diagram for our project is shown in Figure 21. The Arduino IDE used in our project is version 1.0.6. The link for a download of the software can be found in Appendix A.

5.1 Touch Sensors

The touch sensors serve as the only user input in our system. One instance of the touch sensor was tested initially. Once all the desired functionality was achieved in one touch sensor, the remaining instances were implemented and tested together. The final design requires a total of 15 touch sensors, one sensor for each white key in two octaves of a piano keyboard. The Arduino program used to control the touch sensors uses the capacitive touch-sensing library developed by Paul Badger. The name of the library is CapSense. The link and instructions to install this library are in the Arduino website and are also detailed in Appendix A.

5.1.1 Initial Tests and Implementation

The circuit for one touch sensor consists of a resistor and a piece of metal. Each end of the resistor was connected to a digital pin on the microcontroller. One of the pins acts as the send pin and the other as the receive pin. The end of the resistor that is connected to the receive pin is the sensor. We determined that as the resistance increased, the sensitivity of the touch sensor increased, but it also lowered response time. For our project, we wanted the touch sensors to react to direct touch and to have close to immediate response time. We use a 1 MΩ resistor to achieve this operation. We decided that copper would be the material used for the touch sensor since it provided consistent readings from our sensors and it did not interfere with the magnetic field of the electromagnets.

The program for the first touch sensor implements one touch sensor and uses the touch sensor to control an LED. The program uses the sample code from the CapSense library to setup the structure of the program. The full code for this program can be found in Appendix B.1. In order to use the CapSense library it is necessary to include the header for the library. This is done with the first line of code in the program #include <CapacitiveSensor.h>. This grants access to the functions in that library. The following line creates an instance of the touch sensor library in order to setup one touch sensor:

```
CapacitiveSensor   cs_4_6 = CapacitiveSensor(4,6);
```

This sets up one touch sensor that uses pin 4 and pin 6. With this instance, the sensor pin is pin 6. The setup for this program initializes the serial port in order to view sensor values

in serial monitor and then sets our LED pin as an output. The main loop of the program works as described by the following flow diagram.

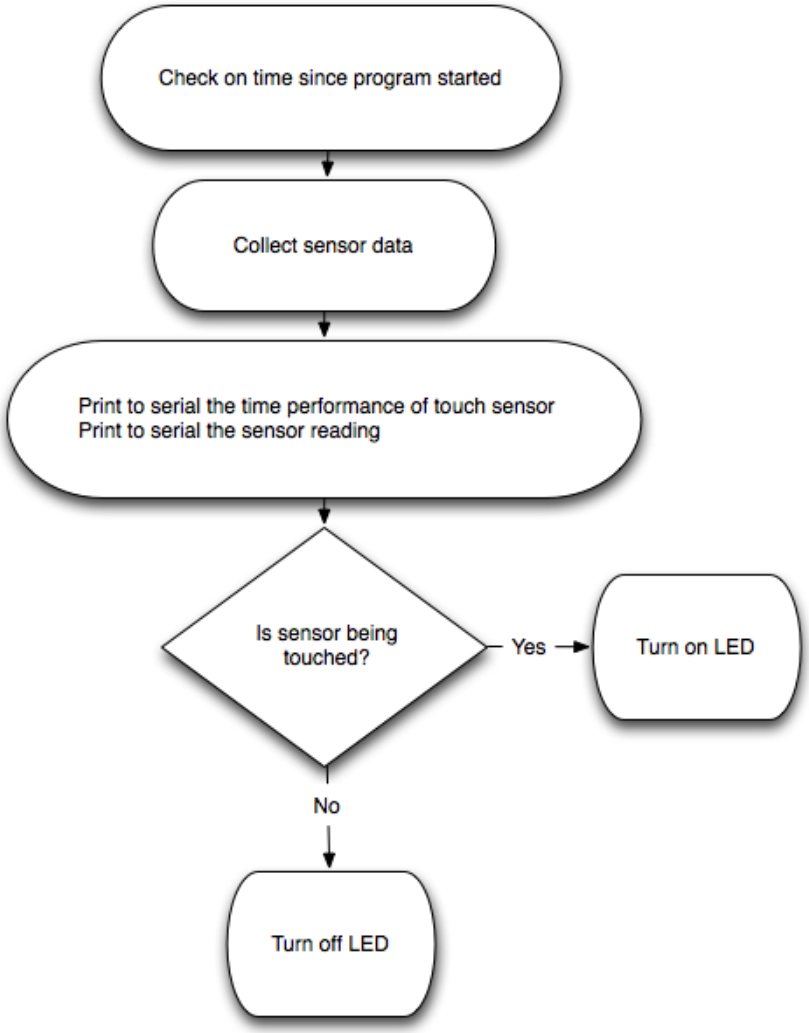


Figure 32: Flow diagram for initial touch sensor test

The flow diagram in Figure 32 shows the operation of the program used to test the first touch sensor. This program provides the performance of the touch sensor and the data reading from the touch sensor. The 'if' statement in the program checks to see if the sensor is being touched by comparing the current reading to a user defined threshold. We found our threshold by looking at the values read by the sensor when touched and then determined a value that marked a touch event. In our tests the value was set to 250. The data for the touch sensor returned values lower than the threshold when there was no contact with the sensors.

5.1.2 Final Implementation

After the initial tests, we added the remaining instances of the touch sensors to our system. In our final design, we determined that by having three different common send pins it was possible to connect the touch sensors in groups of five without increasing response time significantly. Each touch sensor had a copper piece attached as the sensor surface. The copper piece was cut to match the size of a white key of piano and it had wire soldered to one end. The specific connection pins for the sensors to the Arduino are detailed in a later section.

In order to implement 15 touch sensors the program included 15 instances of the CapacitiveSensor [Appendix B.2]. These instances were created such that the send pin was shared with 5 other sensors at a time. This program used the same code flow as described for one touch sensor and then repeated this for 14 other sensors. An important factor to notice is that the sensitivity threshold is different for 15 touch sensors than for one touch sensor. An addition to this program is the code that prints the readings for the touch sensors. This code lays out the foundation for the structures used to send out sensor readings to Max for processing. The code compares the reading for each sensor with the threshold. If the reading is higher then it writes 1 to an array, otherwise it remains 0. The array is printed at the end of the program and it shows the sensor states as on or off.

The final step in the sensor implementation develops the code to fully transmit the desired data through the serial port into Max [Appendix B.3]. The code reads data from the touch sensor and then compares it to the threshold. If the sensor is ‘ON’ then it outputs the corresponding character to the serial port otherwise it prints out the ‘OFF’ character.

Table 2, on the following page, shows the characters that are correspondent to each touch sensor state.

Sensor #	Sensor is ON ASCII output	Sensor is OFF ASCII output
1	'A'	'a'
2	'B'	'b'
3	'C'	'c'
4	'D'	'd'
5	'E'	'e'
6	'F'	'f'
7	'G'	'g'
8	'H'	'h'
9	'I'	'i'
10	'J'	'j'
11	'K'	'k'
12	'L'	'l'
13	'M'	'm'
14	'N'	'n'
15	'O'	'o'

Table 2: ASCII to Max Conversion Factors

The characters are unique and are sent as ASCII numbers to Max. This allows for unique identification for each sensor and for the state of each sensor. The program also controls the number of ‘ messages sent to max. The progression for printing out one sensor reading to max is shown in the flow chart below.

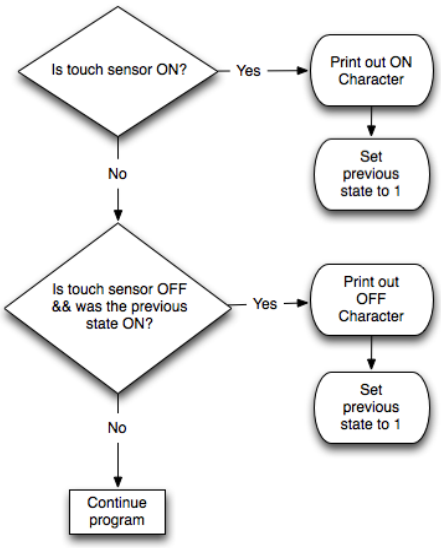


Figure 33: Print out logic for sensors

The diagram in Figure 33 shows the logic that governs the passing of data to Max. This logic is repeated for each sensor. The purpose of keeping track of the previous state is to limit the off messages sent to Max. We only print one off messages every time the touch sensor has no longer received input. This prevents Max from receiving repeated data for off messages.

5.2 LED Circuit

The LEDs are one of the outputs to our system. We implemented the LEDs following a similar progression as the touch sensors. The initial test only tried one LED. Then the LED was tested using a shift register and finally the rest of the LEDs and shift registers were combined for the final implementation.

5.2.1 Initial Tests and Implementation

The LEDs were implemented first by controlling one LED directly with the microcontroller. Each anode of the LED was in series with a resistor and connected to a digital pin on the microcontroller. The resistor values were determined to be 100Ω for green and blue and 200Ω for red. This provided the optimal output of colors for the LEDs. In this initial setup with one LED it was simple to test the different colors by changing the digital pins from low to high in the Arduino program. This program declared three output pins and then they were set manually to output the desired color. These tests showed that the LED was really bright and also required a diffuser to properly mix the colors.

The next step in implementing the LEDs was to set up one LED and control it using a shift register. The proposed approach would use the 74HC595 shift register chips; however, it was not possible to obtain these and they were replaced with 74HC164 shift registers. The difference between these two models is that the 74HC164 does not have a latch pin and there is no serial out pin. The 74HC164 cannot be concatenated like the 74HC595. The disadvantage of

this is that using the 74HC164 required more digital pins on the Arduino than using the 74HC595. This was not a major issue since we saved many pins with our changes to the sensor implementation. The circuit used to test the 74HC164 chip with one LED is shown next.

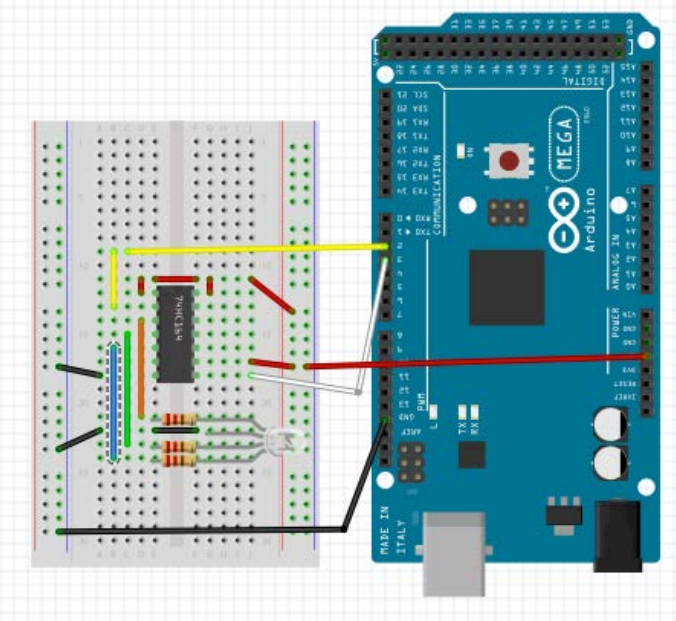


Figure 34: One LED with one 74HC164

Figure 34 shows the circuit connection used to test one LED with the 74HC164. The yellow wire is the data connection for the information shifted out. The white wire is the clock connection that controls when the values are shifted out. The black wires establish the common ground for this circuit and the red wires provide 5V to the LEDs. Each anode of the LED is then connected to an output from the shift register. The code that controls the operation of this circuit is attached in Appendix C.1. The purpose of this code is to control the LEDs by reading data from the serial monitor, since this is the method used to communicate with Max.

The code that controls the LED starts off by shifting out off messages to the LED. This ensures that when the system starts, the LED will be off. The main loop of the program is depicted in the following flow chart.

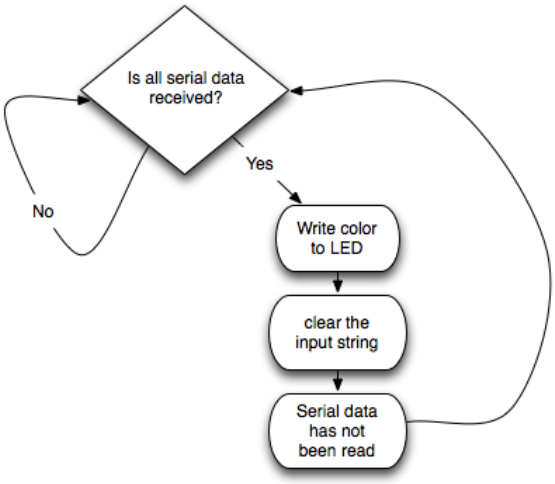


Figure 35: Check to see if serial data is fully received

Figure 35 shows the operation of the main loop in the LED test program. This loop checks a flag to see if serial data has been provided to the system. If nothing has been received then it waits. If a string has been provided then it send this string to the pinWrite function. The flag for this loop is set using the serialEvent() function shown next.

```
void serialEvent() {// checks to see if serial event is occurring
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();  // read characters
    // add it to the inputString:
    inputString += inChar; // write characters to a string
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {// stop reading when newline is received
      stringComplete = true;
    }
  }
}
```

This function is called every time there is a serial event while the Arduino is running. This means that when the user provides any input through the serial monitor this function is called to read the bytes. It read the bytes as characters and appends them to a string. This process continues until the newline character is sent. Once the newline character is received, the flag for stringComplete is set to true so that the program can use the data to control the LEDs.

This program is implemented to work with a certain format of serial inputs. The format required consists of a capital letter followed by two integers. The last character after the integers should always be new line in order to finish reading the serial message. The capital letters that are allowed are R, G, B, C, M and O. These letters represent Red, Green, Blue, Cyan, Magenta and Off. These determine the output that the LED will display. The integers are used to control which

key number will be controlled. The operation of the integers is explained later in the final implementation of the LED circuit.

When a valid serial message is completely received the main loop calls the function pinWrite(String whichKey). The argument for this function is a string. In this program we pass the received serial message to this function. The function uses this string to determine the output for the LED. The function checks the first letter of the string to determine the desired setting for the LED. If the string matches then it calls the colorWrite(char whichColor) function to determine the color represented by the letter. The colorWrite function uses a switch statement to convert the string into binary output for the string. The cases for the switch statement as well as the outputs are shown in the table below.

String Cases	Color	Binary Representation
R	RED	B001
G	GREEN	B010
B	BLUE	B100
C	CYAN	B110
M	MAGENTA	B101
O	OFF	B000

Table 3: Arduino LED Color Representations

The binary representation of the string is the data that is passed out to the shift register. The bits represent the pins on the LED. The least significant bit controls red. The most significant bit controls blue. And the center bit controls green. After the function sets the desired binary representation, it sends this data to the shiftOut function that is standard in the Arduino library. The shiftOut function takes four arguments: dataPin, clockPin, bitOrder and value. The first two are set to the data and clock pin connected to our shift register. The bit order is set to MSBFIRST because we want the most significant bit to be displayed in the last shift register output. The data that we provide is the binary value that we set previously. In conclusion, this program allows control of the LED though the shift register when provided with a valid instruction through the serial monitor.

5.2.2 Final Implementation

The initial implementation of the LED shift register created the framework for the rest of the LEDs; however, some adjustments were introduced to handle the additional LEDs. In order to control the 15 LEDs of our system we required six 74HC164 shift registers. We controlled the shift registers in three groups. Each group of shift register consisted of five LEDs and two shift registers. The final connections to the Arduino digital pins are detailed later in a separate section.

We controlled the LEDs using a similar program as the one described above for a single LED. The expected input continues to consist of a capital letter and two numbers. In this system it is necessary to know the key number that needs to be controlled, so that the correct LED is modified. This program begins by initializing the clock and data pins for each shift register [Appendix C.2].. Then it calls the function turnAllOff(). This function ensures that all LED's are off at the start of the program. The main loop of the program is handled with the same loop that is mentioned in the initial test with one LED. The first difference in the flow of this program is that it calls the function getKeyNumber(String whichKey) when it finds a match in the pinWrite() function. A flow diagram depicting the overall flow of the program is depicted next.

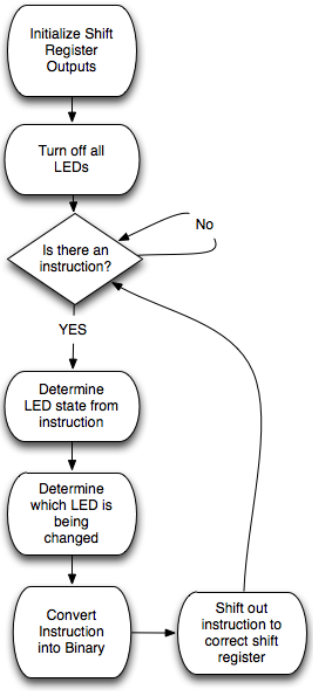


Figure 36: Shift Registers Control Flow

The diagram in Figure 36 shows the flow of tasks performed to write to an LED. In our program, Max provides the instruction to control the LEDs through the serial port.

5.3 Electromagnet Circuit

The electromagnets are our second system output. The circuit for the electromagnets was tested first individually and then as a whole. Controlling the electromagnets required a short modification to the program created for the LEDs.

5.3.1 Initial Tests and Implementation

The electromagnet circuit was first tested without an electromagnet since the electromagnets arrived at a later date than expected. For the first test, the electromagnet was replaced with a red LED in series with a resistor in order to verify that the circuit had adequate functionality. The LED successfully lit up during the preliminary tests to show that the circuit worked. The diagram of the circuit can be found previously in Figure 28. Controlling the electromagnet requires one line of code. In order to turn on the electromagnet the pin connected to the transistor is set to high and to turn the electromagnet off, the pin should be set to low. During our tests, we noticed that the electromagnets heat up a lot if left on for extended periods of time.

5.3.2 Final Implementation

The final implementation for the electromagnet circuit creates 15 total instances of the circuit in Figure 28. The electromagnets are controlled within the LED code. They are turned on right after the LEDs. The function to turn on the electromagnet is pullOn() and the function to turn it off is pullOff(). They both consist of one line of code.

```
digitalWrite((thisKey+19),HIGH);
```

This line of code turns on the electromagnet based on the integers provided in the instruction from Max. We add 19 because the electromagnet connections start on pin 20 as shown in the next section.

5.4 Arduino Microcontroller Connections

The microcontroller connections were set up on digital pins that were not used for serial communication. Throughout the entire implementation process, pin 0 and pin 1 have been set to be the send and receive pin for serial data. The rest of the pin organization can be seen in the following table.

Digital Pin	Connection Purpose
0,1	Serial Communication
2	Touch Sensor Receive 1
3,4,5,6,7	Touch Sensors for Keys 1-5
8	Touch Sensor Receive 2
9,10,11,12,13	Touch Sensors for Keys 6-10
14	Touch Sensor Receive 3
15,16,17,18,19	Touch Sensors for Keys 11-15
20-34	Electromagnets
35-36	Shift Register 1
37-38	Shift Register 2
39-40	Shift Register 3
41-42	Shift Register 4
43-44	Shift Register 5
45-46	Shift Register 6
Vin	Input for Power supply
GND	Provides ground for circuits
5V	Uses Voltage regulator to output 5V

The final program that combines all the components and runs with the connections listed in the table is attached in Appendix D.

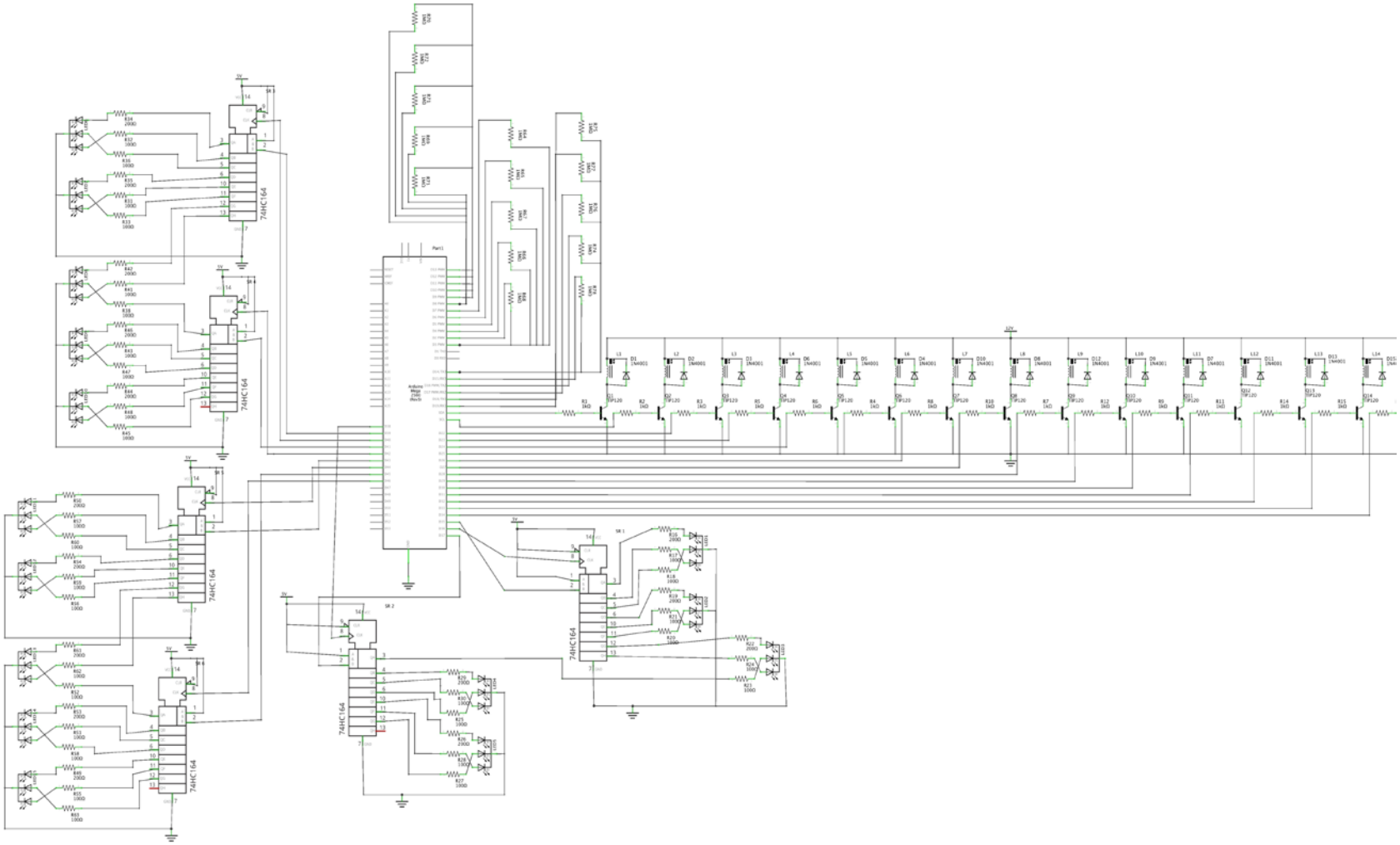


Figure 37: Full Circuit Schematic

Figure 37 shows the full circuit schematic for our device with all the connections of the components to the Arduino. We created a layout for a PCB for our circuit with software called Fritzing. The PCB layout we developed is shown in Figure 38.

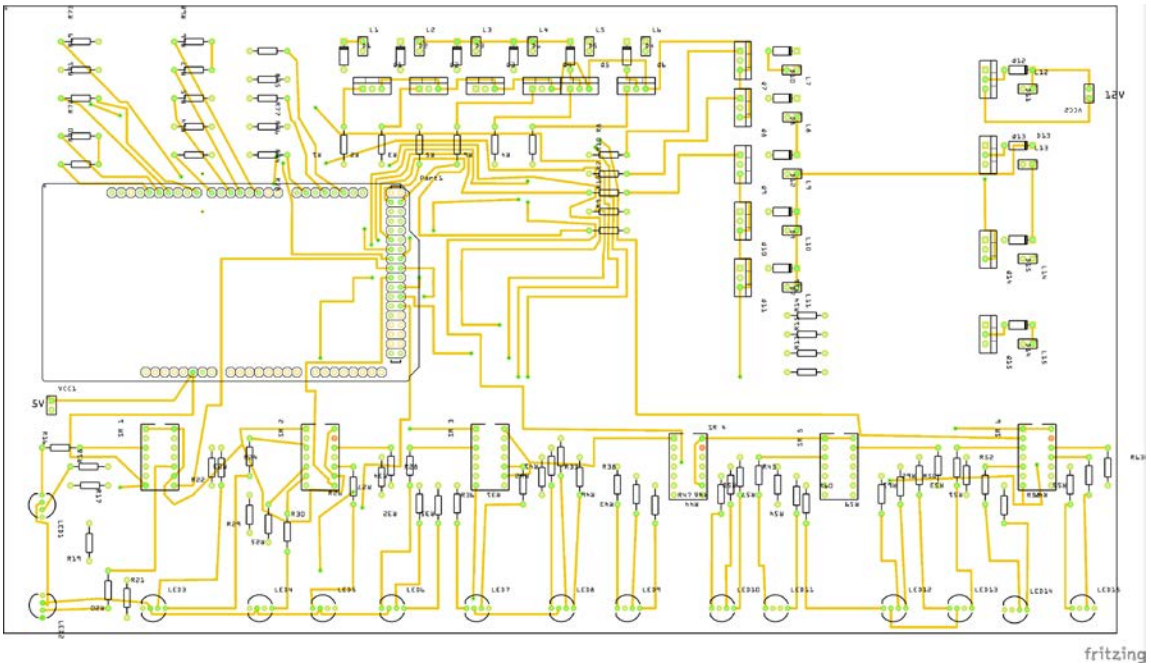


Figure 38: PCB layout top layer designed using Fritzing

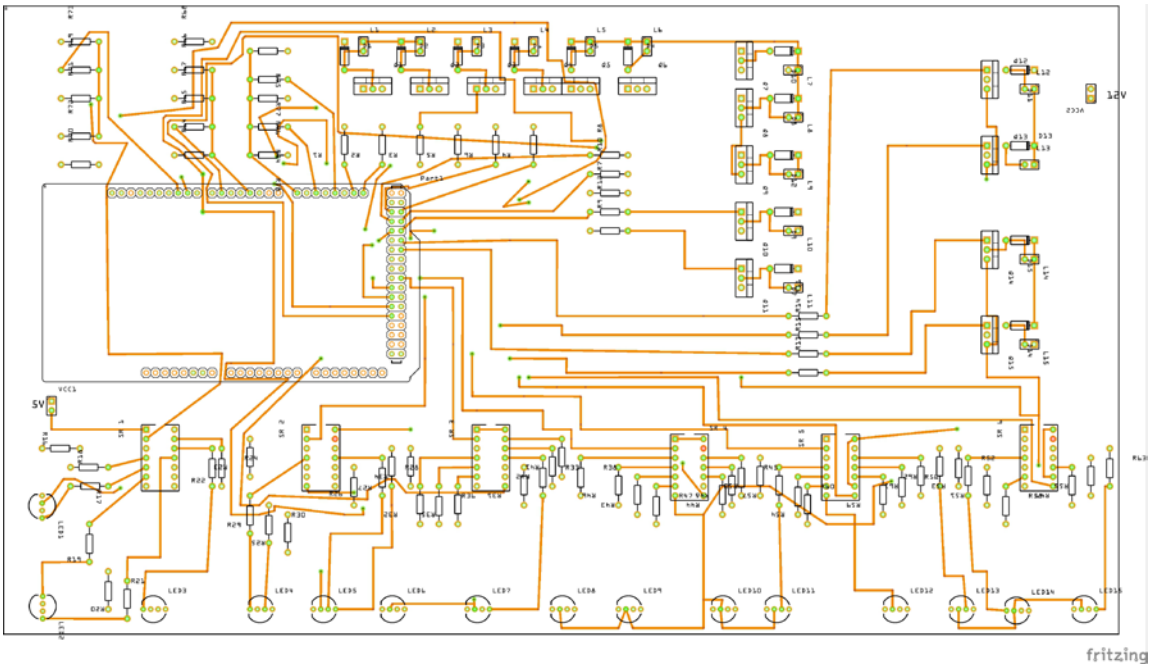


Figure 39: PCB layout bottom layer designed using Fritzing

5.5 Box Construction

The box was created using pine and the design plans were developed in SolidWorks. The design plan for the Piano box is attached in Appendix E. The materials required to assemble the piano box are shown in the next table.

Item	Quantity
1”x6”x8’ No. 2 Pine Board	1pc
1”x12”x6’ No.2 Pine Board	1pc
1”x36” Square Wood Dowel	3pc
36”x3/8” Round Rod Aluminum	1pc
2’x4’ Ice White Acrylic Panel	1pc
3/8” Nylon Washer	50pc
1/4”x6’ Threaded Rod	2pc
¼-20 Steel Nuts	50pc
1/4” Wide Washer	1box
1/4”x36” Round Wood Dowel	3pc
#10 Plate Joining Biscuit	1box

The necessary tools for construction included:

- 12in. Sliding Dual Compound Miter Saw with Laser
- Table Saw with Adjustable angle
- Drill press with configurable depth and speed
- Router
- Palm Sander
- Hand Planer
- Hack Saw
- Various Screwdrivers
- Hammer
- Corded Drill
- 1/4” Brad Point Drill Bit
- 13/64” Brad Point Drill Bit
- 3/8” Brad Point Drill bit

- 13/16” Flat Wood Bit
- 4” C-Clamp
- 2 Adjustable ¾” Pole Clamps

6. Max Software Implementation

The software implementation of our product establishes communication between the keyboard’s hardware, the Arduino microcontroller, and the Max virtual programming language. Multiple Max patches control the keyboard’s functionality. The program’s patches can be recognized by the Max file extension (.maxpat), and are named UI_homeScreen, pianoPoll., keyControl, C_MajorScale_Follow, C_MajorScale_Play, odeToJoy_Follow, and odeToJoy_Play. Only the user interface patch, UI_homeScreen, will be visible to the piano keyboard player [Appendix F.1].

The Arduino microcontroller receives sensor data from the keyboard and sends it to the Max patch pianoPoll [Appendix F.2]. PianoPoll deciphers which key was pressed and outputs the MIDI pitch value corresponding to that key’s musical note. When a key press is detected, pianoPoll sends the input data to be processed by keyControl [Appendix F.2]. KeyControl contains patchwork that executes commands for each key. Output data is relayed back to pianoPoll to output the key’s MIDI pitch through the computer’s speakers. The keyControl patch also sends its output to the Max patch associated with the current mode and piano lesson chosen by the user. These four patches, C_MajorScale_Follow, C_MajorScale_Play, odeToJoy_Follow, and odeToJoy_Play, are referred to as “music” patches and contain execution instructions for the patch’s lesson and mode [Appendix F.3]. Each patch described in this paragraph is executed “behind the scenes” in Max, meaning that the user is unaware of their presence.

The user interface patch, UI_homeScreen, is the only interactive window within the keyboard’s program [Appendix F.1]. Here, the player chooses a serial port, lesson and mode. Two interactive objects, a button and a toggle, and three selection menus are located in the user interface. When the button is pressed, a list of serial port options is displayed in the Max console, located on the right side of the window. One of the listed serial ports will be connected to the keyboard via a USB connection. From the drop-down menu beneath the display-serial button, the user must choose the letter corresponding to the correct serial port. Beneath the serial port options and instructions are two drop-down menus that are used to select a lesson and mode. Once selected, the corresponding music patch is activated. Now, the player can click the large toggle button at the top of the user interface patch to begin playing, a state that will be referred to as play mode.

Once in play mode, the keyboard receives its first instruction from the activated music patch. The output instruction will turn on the electromagnet and LED of the key that corresponds to the first pitch in the lesson’s note sequence.

If in Follow Mode, the next output instruction is not sent until the program receives input data from the activated key's touch sensor. Once received, the next key in the music patch's note sequence will be turned on. In Play Mode, instructions are output based on a specified time interval, or tempo. Input data from the touch sensors is still received and processed, but is completely independent of the output states associated with the music patch's note sequence and key progression.

While the keyboard and computer software exchange data, the user is still looking at the user interface home screen. On the right side of the screen, a musical staff receives input data that corresponds to the current key's pitch and outputs a visual representation of the note on the musical staff. On the left side of the home screen, there is a visual of the left and right hands with numbered fingers, a keyboard with labeled note keys, and a musical staff showing the bass and treble clef. A beginning piano player can use these images as a reference to which hand is associated with the upper and lower halves of the keyboard, as well as where the notes' pitch value lies on the musical staff.

7. Recommendations

Ideas for future development of the keyboard, and other haptic learning devices, occurred to us while we worked on the project. Throughout building the keyboard and analyzing the final product's functionality, we thought of how this could influence haptic technology in the future. There are many improvements that could be made to improve the keyboard's functionality and influence its worth as a learning device.

One recommendation would be to add more electromagnets per key. This would enhance haptic feedback control of the over the key's surface, such as creating a magnetic grid with altering polarities beneath the keys. The haptic device would have more degrees of control over positioning the user's fingers for correct piano playing. The opposing forces would also enhance a player's ability to determine which key should be pressed. Changing the polarity of a key could also encourage a player to lift their finger from the key at the right time and would greatly reduce the resistance between a key and the player's fingers when trying to lift their finger from the key just pressed. This is would be a helpful improvement because the ferromagnetic material of the electromagnets is naturally attracted to the magnets in the gloves.

The touch sensors in our design worked adequately, but after testing we realized that replacing them with force sensors under the keys would be more suitable for haptic technology. Force sensors would allow better tracking of user input and could increase the sensation of playing a piano. With force sensors, users would be required to press a key to generate sound, rather than by simply touching the key. The user would also be able to rest their fingers on the keys without generating input signals.

Modifications to our software would include a friendlier user interface and a more thorough analysis of user performance, such as keeping track of wrong key presses corresponding to each finger. More intense data analysis via the computer software could enhance the

keyboard's educational abilities by providing users with feedback regarding their playing technique and common mistakes.

In the future, the keyboard could be used in a user study to test the efficiency of haptic technology as an educational tool. Following the progression of beginning-level piano students using the keyboard would provide feedback for both the keyboard's functionality and the plausibility of creating other haptic learning-aid technologies. In addition to testing the exercises we implemented, another interesting study would be to test the haptic technology using a larger variety of songs and exercises that increase in difficulty. Advanced piano students may be asked to use the keyboard as an aid for finger exercises and assess their performance. This would show if the device has potential to be used as a viable option for users of different skill levels.

8. Results and Conclusions

The outcome of this project created a proof of concept for a haptic learning device. We tested the operation of the technology with our keyboard. Our keyboard prototype shows functionality between its components. The keyboard we created can be programed to run a C Major scale exercise and provide instructions for playing Ode to Joy. Haptic technology continues to show that it has potential to become a viable alternative for learning applications; however, designing a system that achieves this successfully requires extensive testing and design changes.

The end result of our device performed as expected in some respects. The gloves proved to be a challenging component to design. It is important that they do not interfere with the capacitance reading of the touch sensors. The electromagnets and the magnets interacted as expected; however, the strength of the pull from the electromagnets was not as noticeable as anticipated. Overall, the device functioned properly and could be used for user testing with some of the improvements mentioned previously.

This project created a foundation for the design of the haptic learning technology, but would require some user testing to show its applicability to teaching an instrument. Learning the piano is an activity that continues to interest many and it would be beneficial to continue to explore alternatives to improving the learning experience in a unique way.

9. References

[1] *Ridden, Paul. "PianoMaestro Guides Pianists through the Music." Gizmag. Gizmag, 1 Apr. 2011. Web. 06 Oct. 2014. <http://www.gizmag.com/pianomaestro-guides-pianists-through-the-music/18297/picture/132550/>.*

[2] *Steinway Etude. Computer software. Apple App Store. Vers. 2.2. Steinway Musical Instruments, 24 Sept. 2012. Web. 07 Oct. 2014. <https://itunes.apple.com/us/app/steinway-etude/id430004407?mt=8>.*

[3] *Georgia Tech PianoTouch*

[4] *“Pedagogy.” The Free Dictionary. Farlex, 2008. Web. 06 Oct. 2014. <http://www.thefreedictionary.com/pedagogy>.*

[5] *“Piano Pedagogy." Wikipedia. Wikimedia Foundation, 10 July 2014. Web. 07 Oct. 2014. <http://en.wikipedia.org/wiki/Piano_pedagogy>.*

[6] *Lewis, Martha Beth. "Music Biography." Music Biography. Summy-Birchard (Warner) and E.C. Schirmer, 13 Mar. 2001. Web. 07 Oct. 2014. <http://www.marthabeth.com/music_bio.html>.*

[7] *Lewis, Martha Beth, Ph.D. "Music Pedagogy QA." Music Pedagogy QA. Summy-Birchard (Warner) and E.C. Schirmer, 31 July 2014. Web. 06 Oct. 2014. <http://www.marthabeth.com/pedagogy_QA.html>.*

[8] *Graf, David. "Piano Fingering, Piano Technique, Piano Posture." True Piano Lessons.com. True-piano-lessons.com, Fall 2014. Web. 07 Oct. 2014. <http://www.true-piano-lessons.com/piano-fingering.html>.*

[9] *Green, Andrew. "Practice Makes Perfect." Piano Lessons:. Simon Probert, Jan. 2007. Web. 07 Oct. 2014. <http://www.piano-lessons.net/news_item.php?id=9>.*

[10] *Chang, C. C. "What Is Piano Technique?" Web log post. Fundamentals of Piano Practice. Web. 06 Oct. 2014. <http://fundamentalpiano1.blogspot.com/2006/03/what-is-piano-technique.html>.*

[11] *"Do You Have Piano Hands?" N.p., n.d. Web. 06 Oct. 2014. http://www.heykiki.com/blog/2012/10/04/do-you-have-piano-hands/*

[12] *Sharlene. "Lesson 2: How to Sit at the Piano." How to Sit Properly at the Piano. Epianostudio, Apr. 2013. Web. 07 Oct. 2014. <http://www.epianostudio.com/2008/10/21/lesson-how-to-sit-at-the-piano/>.*

[13] *Dachis, Adam. "How Muscle Memory Works and How It Affects Your Success." Lifehacker. Kinja, 6 May 2011. Web. 07 Oct. 2014. <http://lifehacker.com/5799234/how-muscle-memory-works-and-how-it-affects-your-success>.*

[14] *Gleaves, Tiana. "Basic Piano Fingerings for the 12 Major Scales." Free Online Piano Lessons Discover How To Play Piano in a Way That Fits Your Busy Lifestyle Basic Piano Fingerings for the 12 Major Scales Comments. Piano Lessons Central, 30 July 2014. Web. 07 Oct. 2014. <http://www.piano-lessons-central.com/piano-scales/piano-fingerings/>.*

[15] *Frantz, Albert. "Efficient Piano Practice." 10 Expert Tips. Key-notes LLC, 2014. Web. 06 Oct. 2014. <http://www.key-notes.com/efficient-piano-practice.html>.*

[16] *Lewander, Maria. "The C Major Scale." Online Piano Coach. OnlinePianoCoach.com, 2014. Web. 07 Oct. 2014. <http://www.onlinepianocoach.com/c-major-scale.html>.*

[17] *Copp, Evan A. "Benefits of Learning Scales." Netplaces. About.com, 2014. Web. 6 Oct. 2014. <http://www.netplaces.com/piano/scaling-mountains/benefits-of-learning-scales.htm>.*

[18] *Frantz, Albert. "Efficient Piano Practice." Piano Fingering. Key-notes LLC, 2014. Web. 07 Oct. 2014. <http://www.key-notes.com/piano-fingering.html>.*

[19] *David, and Ido. "The Best Piano Exercises for Beginners." The Best Piano Exercises for Beginners. Piano-Play-It.com, 2013. Web. 07 Oct. 2014. <http://www.piano-play-it.com/piano-exercises.html>.*

[20] *Beatty, Jessica. "I Teach Piano (:." Pinterest. Pinterest, 2014. Web. 07 Oct. 2014. <http://www.pinterest.com/jessicabeatty20/i-teach-piano/>.*

[21] *Pianonet. "New Technology in Pianos." Your Comprehensive Guide to Everything about Pianos. National Piano Foundation, 2014. Web. 07 Oct. 2014. <http://pianonet.com/resources/publications/new-technology-in-pianos/>.*

[22] *"Play Musical Piano For Kids Children's Learning Toy Machine." Aliexpress.com. Ali Express. Web. 07 Oct. 2014. <http://www.aliexpress.com/item/Russian-Piano-Toys-Electrical-Piano-Musical-Toys-For-Kids-Music-Piano-Keyboard-Instruments-For-Children-Kids/725324095.html>.*

[23] *The Children's Learn To Play Keyboard. Digital image. Hammacher Schlemmer. Hammacher Schlemmer & Company, INC, 2014. Web. 07 Oct. 2014. <http://www.hammacher.com/Product/81525?cm_cat=ProductSEM&cm_pla=AdWordsPLA&source=PRODSEM&gclid=CjwKEAjwp7WgBRCRxMCLx8mMnDMSJADncxS2UOConeCBc3JqThcWGw8K47DiAaSklBKoohbFPixpsRoCaATw_wcB>.*

[24] *iTikes. "Welcome to ITikes: A Place Where Tech Meets Play." Itikes.com. TM & MGA Entertainment, Inc, 2014. Web. 06 Oct. 2014. <http://www.itikes.com/>.*

[25] *MGAE. ITikes | Create Piano. Computer software. Apple App Store. Vers. 1.2. MGA Entertainment Inc., 6 Nov. 2012. Web. 06 Oct. 2014. <https://itunes.apple.com/us/app/itikes-i-create-piano/id540567773?mt=8>.*

[26] *Little Tikes, ITikes. "Little Tikes ITikes Keyboard, White/Blue." Walmart.com. Walmart, 2014. Web. 07 Oct. 2014. <http://www.walmart.com/ip/Little-Tikes-iTikes-Keyboard-White-Blue/21007393>.*

[27] *Dellinger, AJ. "8 Apps That Teach You How to Play Piano." Mac|Life. Mac|Life, 26 Sept. 2013. Web. 07 Oct. 2014. <http://www.maclife.com/article/gallery/8_apps_teach_you_how_play_piano#slide-0>.*

[28] *Leff, Obie. Piano... Computer software. Apple App Store. Vers. 2.0. Obie LEFF, 5 Aug. 2014. Web. 06 Oct. 2014. <https://itunes.apple.com/us/app/piano.../id445298897?mt=8>.*

[29] *SmileyApps, LLC. Piano Tutor for IPad. Computer software. Apple App Store. Vers. 7.1. SmileyApps, LLC, 7 Nov. 2013. Web. 06 Oct. 2014. <https://itunes.apple.com/us/app/piano-tutor-for-ipad/id364898961?mt=8>.*

[30] *"The Concert Quality Piano App for IPad." IK Multimedia. IK Multimedia., 2014. Web. 06 Oct. 2014. http://www.ikmultimedia.com/products/igrandipad/*

[31] *IK Multimedia. IGrand Piano for IPad. Computer software. Apple App Store. Vers. 1.1.1. IK Multimedia, 18 Sept. 2013. Web. 06 Oct. 2014. <https://itunes.apple.com/us/app/igrand-piano-for-ipad/id562917936?mt=8>.*

[32] *Harris, William. "How Haptic Technology Works" 30 June 2008. HowStuffWorks.com. <http://electronics.howstuffworks.com/everyday-tech/haptic-technology.htm> 17 September 2014.*

[33] *Saddik, Abdulmotaleb El. Haptics Technologies: Bringing Touch to Multimedia. Heidelberg: Springer, 2011. Print.*

[34] *Stone, Robert J. "Haptic Feedback: A Potted History, From Telepresence to Virtual Reality." http://www.dcs.gla.ac.uk/~stephen/workshops/haptic/papers/stone.pdf. 17 Sept. 2014.*

[35] *"FingerFlux: Near-surface Haptic Feedback on Tabletops." YouTube. YouTube, 16 Oct. 2011. Web. 19 Sept. 2014.*

[36] *PHANTOM OMNI® HAPTIC DEVICE. Digital image. SensAble, 1 July 2007. Web. <http://www.dentsable.com/haptic-phantom-omni.htm>.*

[37] *"PHANTOM Omni® Haptic Device." PHANTOM OMNI. SenseAble, 2009. Web. 08 Oct. 2014. <http://www.dentsable.com/haptic-phantom-omni.htm>*

[38] *Solon, Olivia. CyberGlove. Digital image. Stroke Victims Video Games. WIRED, 16 May 2011. Web. 17 Sept. 2014 <http://www.wired.co.uk/news/archive/2011-05/16/stroke-victims-video-games>.*

[39] *“CyberGrasp™ System V2.0." CyberGraspTM System V2.0 . CyberGlove Systems LLC, 2007. Web. 8 Oct. 2014.*

[40] *Mobile Music Touch Works with a Computer, MP3 Player or Even a Smart Phone. Digital image. Science Data Base. Mumbai Mirror, 19 July 2012. Web. 17 Sept. 2014. <http://www.sciencedatabase.com/2012_09_01_archive.html>.*

[41] *Huang, Kevin, Ellen Y. Do, and Thad Starner. PianoTouch: A Wearable Haptic Piano Instruction System For Passive Learning of Piano Skills. IEEE, 28 Sep. 2008. Web. 17 Sept. 2014. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4911582>.*

[42] *Kern, Thorsten A. Engineering Haptic Devices: A Beginner's Guide for Engineers. Dordrecht: Springer, 2009. Print.*

[43] *Brain, Marshall, and Lance Looper. "How Electromagnets Work?" HowStuffWorks. HowStuffWorks.com, 1 Apr. 2000. Web. 30 Sept. 2014. <http://science.howstuffworks.com/electromagnet.htm>.*

[44] *France, Colin. Magnetic field produced by Electromagnet. Digital image. GCSE PHYSICS - Electromagnetism. GCSE, 2014. Web. 01 Oct. 2014. <http://www.gcsescience.com/pme5.htm>.*

[45] *Park, Wanjoo, Sehyoung Park, Laehyun Kim, and Seungjae Shin. "Haptic Mouse Interface Actuated by an Electromagnet." IEEE Xplore. IEE, July 2011. Web. 01 Oct. 2014. <http://ieeexplore.ieee.org/xpl/abstractCitations.jsp?arnumber=5989053>.*

[46] *Weiss, Malte, Chat Wacharamanotham, Simon Voelker, and Jan Borchers. "FingerFlux: Nearsurface Haptic Feedback System on Tabletops." ACM. RWTH Aachen University, 19 Oct. 2011. Web. 8 Oct. 2014.*

[47] *Weiss, Malte. "FingerFlux: Near-surface Haptic Feedback." Digital image. The Media Computing Group. 28 Oct. 2011. Web. 01 Oct. 2014. <http://hci.rwth-aachen.de/fingerflux>.*

[48] *"LED - RGB Clear Common Cathode." SparkFun Electronics. Spark Fun, 2014. Web. 20 Oct. 2014. <https://www.sparkfun.com/products/105>.*

[49] *Oomlout. "RGB LED Tutorial (using an Arduino) (RGBL)." Instructables. Instructables, 2009. Web. 20 Oct. 2014. <http://www.instructables.com/id/RGB-LED-Tutorial-using-an-Arduino-RGBL/?ALLSTEPS>.*

[50] *Maw, Carlyn, and Tom Igoe. "Serial to Parallel Shifting-Out with a 74HC595." Arduino. Arduino, Nov. 2006. Web. 20 Oct. 2014. <http://arduino.cc/en/tutorial/ShiftOut>.*

[51] *Badger, Paul. "CapacitiveSensor." Arduino Playground. Arduino, n.d. Web. 20 Oct. 2014. <http://playground.arduino.cc/Main/CapacitiveSensor?from=Main.CapSense>.*

[52] *"5.6lbs DC 12V Holding Electromagnet Lift Solenoid." Suntek Store. Suntek Store, 2014. Web. 20 Oct. 2014. <http://www.suntekstore.com/goods.php?id=14002885&utm_source=gbus&utm_medium=paid>*

[53] *"Magnet Frequently Asked Questions." Integrated Magnetics. Integrated Magnetics, n.d. Web. 20 Oct. 2014. <http://www.intemag.com/faqs.html#strengthdistance>.*

[54] *Jts3k. "Controlling Solenoids with Arduino." Instructables. Instructables, 2010. Web. 20 Oct. 2014. <http://www.instructables.com/id/Controlling-solenoids-with-arduino/?ALLSTEPS>.*

[55] *"ArduinoBoardMega2560." Arduino. Arduino, 2014. Web. 20 Oct. 2014. <http://arduino.cc/en/Main/arduinoBoardMega2560>.*

[56] *Instructable is part of a 3-part workshop run at Women's Audio Misson Ghassaei, Amanda. "Intro to MaxMSP." Instructables.com. Autodesk, Inc, n.d. Web. 13 Oct. 2014.*

[57] *Spitz, Andrew. "How to Set Up Arduino with Max/MSP { Sound + Tutorial }." { Sound + Design }. Creative Commons Attribution-Noncommercial-Share Alike, 1 Mar. 2009. Web. 20 Oct. 2014. <http://www.soundplusdesign.com/?p=1305>.*

[58] *Kothman, Keith. "(maxmsp) Programming and Max Basics." TeachingMusic. TeachingMusic, 12 Jan. 2011. Web. 20 Oct. 2014. <http://teachingmusic.keithkothman.com/2011/01/compmus3-programming-and-max-basic/>.*

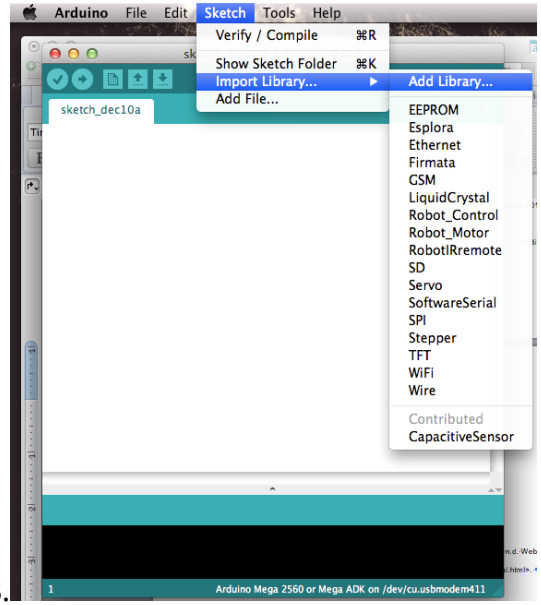
[59] *"Max Basic Tutorial 5: Message Order and Debugging." Max Basic Tutorial 5: Message Order and Debugging. Cycling '74, n.d. Web. 20 Oct. 2014. <http://www.cycling74.com/docs/max5/tutorials/max-tut/basicchapter05.html>.*

[60] *"D032C." Amazing Magnets. Amazing Magnets, 2014. Web. 20 Oct. 2014. <https://www.amazingmagnets.com/show-decimal-d032c.aspx>.*

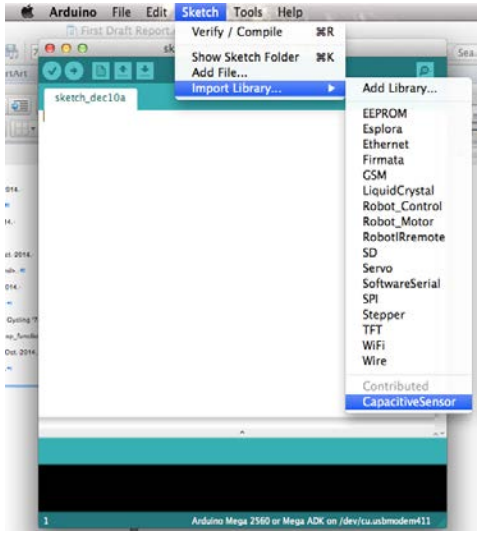
10. Appendices

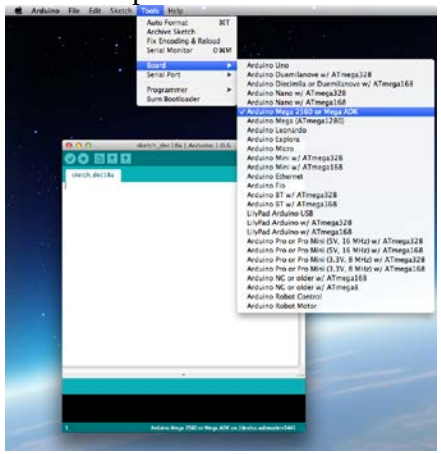
Appendix A: Arduino Software Installation and Setup

1.Download link for Arduino IDE 1.0.6: http://arduino.cc/en/Main/Software
2.Download link for CapSense : http://playground.arduino.cc/Main/CapacitiveSensor?from=Main.CapSense

3. 

4.Select the library that was downloaded in step one.
5.Click on choose
6.The library should appear in the import library menu

7. 

8.Click to import the library.
9.To set up the environment for the Arduino Mega:

10. Go to the Tools menu and then move down to the Board option and select the Arduino Mega option.
11. To upload our project:
12. Copy Appendix G to the open window.
13. Connect the device with the USB port
14. And Click on the arrow facing right on the top left corner of the window.

The Program should be uploaded and will run with the final circuit described in this report.

Appendix B: Touch Sensors

1. Example Code for One Touch Sensor

```
#include <CapacitiveSensor.h>

int redLED = 12;
const int sensitivity = 200; // threshold to determine touch
const int samples = 30; //number of sample reading for touch sensor                    //reading

CapacitiveSensor   cs_4_6 = CapacitiveSensor(4,6);// 1M resistor                                    //between pins 4 &
6, pin 6 is sensor pin, add                                                                        //a wire and or foil


void setup()
{
  Serial.begin(9600);
  pinMode(redLED,OUTPUT);
}

void loop()
{
    long start = millis(); // stores the number of milliseconds since                              //the program started

    long total1 =  cs_4_6.capacitiveSensor(samples);                                               // stores the sensor reading
for the requested number of samples

    Serial.print(millis() - start); // check on performance                                                    //in
milliseconds
    Serial.print("\t");// tab character for debug window                                                     //spacing


    Serial.print(total1);    // print sensor output 1
    Serial.println("\t");

    if ((total1 > sensitivity)) // if sensor is being touched
      digitalWrite(redLED, HIGH); // turn on LED
    else if (total1 <= sensitivity) // else
      digitalWrite(redLED, LOW); // turn off LED

    delay(0);// arbitrary delay to limit data to serial port
}
```

2. Setting the Touch Sensor's Sensitivity for 15 Keys

```
#include <CapacitiveSensor.h>

const int samples = 60;
const int sensitivity = 250;
int redLED = 13;
//int greenLED = 11;
//int blueLED = 10;
CapacitiveSensor   cs_2_3 = CapacitiveSensor(2,3);        // 1M resistor between pins 2 & 3, pin 3 is sensor pin, add a wire and or foil
CapacitiveSensor   cs_2_4 = CapacitiveSensor(2,4);        // 1M resistor between pins 2 & 4, pin 4 is sensor pin, add a wire and or foil
CapacitiveSensor   cs_2_5 = CapacitiveSensor(2,5);        // 1M resistor between pins 2 & 5, pin 5 is sensor pin, add a wire and or foil
CapacitiveSensor   cs_2_6 = CapacitiveSensor(2,6);  //4
CapacitiveSensor   cs_2_7 = CapacitiveSensor(2,7);  //5
CapacitiveSensor   cs_8_9 = CapacitiveSensor(8,9);   //6
CapacitiveSensor   cs_8_10 = CapacitiveSensor(8,10);   //7
CapacitiveSensor   cs_8_11 = CapacitiveSensor(8,11);  //8
CapacitiveSensor   cs_8_12 = CapacitiveSensor(8,12);  //9
CapacitiveSensor   cs_8_13 = CapacitiveSensor(8,13);   //10
CapacitiveSensor   cs_14_15 = CapacitiveSensor(14,15);  //11
CapacitiveSensor   cs_14_16 = CapacitiveSensor(14,16);  //12
CapacitiveSensor   cs_14_17 = CapacitiveSensor(14,17);  //13
CapacitiveSensor   cs_14_18 = CapacitiveSensor(14,18);  //14
CapacitiveSensor   cs_14_19 = CapacitiveSensor(14,19);  //15

void setup()
{
  //cs_4_6.set_CS_AutocaL_Millis(0xFFFFFFFF);     // turn off autocalibrate on channel 1 - just as an example
  //cs_4_8.set_CS_AutocaL_Millis(0xFFFFFFFF);
  Serial.begin(9600);
  pinMode(redLED,OUTPUT);
//   pinMode(greenLED,OUTPUT);
//   pinMode(blueLED,OUTPUT);
}

void loop(){

    long start = millis();
    long touch1 =  cs_2_3.capacitiveSensor(samples);
    long touch2 =  cs_2_4.capacitiveSensor(samples);
    long touch3 =  cs_2_5.capacitiveSensor(samples);
    long touch4 =  cs_2_6.capacitiveSensor(samples);
    long touch5 =  cs_2_7.capacitiveSensor(samples);
    long touch6 =  cs_8_9.capacitiveSensor(samples);
    long touch7 =  cs_8_10.capacitiveSensor(samples);
    long touch8 =  cs_8_11.capacitiveSensor(samples);
    long touch9 =  cs_8_12.capacitiveSensor(samples);
    long touch10 =  cs_8_13.capacitiveSensor(samples);
```

```
    long touch11 =  cs_14_15.capacitiveSensor(samples);
    long touch12 =  cs_14_16.capacitiveSensor(samples);
    long touch13 =  cs_14_17.capacitiveSensor(samples);
    long touch14 =  cs_14_18.capacitiveSensor(samples);
    long touch15 =  cs_14_19.capacitiveSensor(samples);

    int sensors[15] = {0};

    Serial.print(millis() - start); // check on performance in milliseconds
    Serial.print("\t"); // tab character for debug window spacing

  //  Serial.print(touch1);    // print sensor output 1
  //  Serial.print("\t");
  //  Serial.print(touch2);    // print sensor output 2
  //  Serial.print("\t");
  //  Serial.print(touch3);    // print sensor output 3
  //  Serial.print("\t");
  //  Serial.print(touch4);    // print sensor output 4
  //  Serial.print("\t");
  //  Serial.print(touch5);    // print sensor output 5
  //  Serial.print("\t");
  //  Serial.print(touch6);    // print sensor output 6
  //  Serial.print("\t");
  //  Serial.print(touch7);    // print sensor output 7
  //  Serial.print("\t");
  //  Serial.print(touch8);    // print sensor output 8
  //  Serial.print("\t");
  //  Serial.print(touch9);    // print sensor output 9
  //  Serial.print("\t");
  //  Serial.print(touch10);    // print sensor output 10
  //  Serial.print("\t");
  //  Serial.print(touch11);    // print sensor output 11
  //  Serial.print("\t");
  //  Serial.print(touch12);    // print sensor output 12
  //  Serial.print("\t");
  //  Serial.print(touch13);    // print sensor output 13
  //  Serial.print("\t");
  //  Serial.print(touch14);    // print sensor output 14
  //  Serial.print("\t");
  //  Serial.print(touch15);    // print sensor output 15
  //  Serial.println("\t");

    if (touch1 > sensitivity) // Check reading with threshold
      sensors[0] = 1;                                                                                    // Output 1 if sensor
is touched
    if (touch2 > sensitivity)
      sensors[1] = 1;
    if (touch3 > sensitivity)
      sensors[2] = 1;
    if (touch4 > sensitivity)
```

```
    sensors[3] = 1;
  if (touch5 > sensitivity)
    sensors[4] = 1;
  if (touch6 > sensitivity)
    sensors[5] = 1;
  if (touch7 > sensitivity)
    sensors[6] = 1;
  if (touch8 > sensitivity)
    sensors[7] = 1;
  if (touch9 > sensitivity)
    sensors[8] = 1;
  if (touch10 > sensitivity)
    sensors[9] = 1;
  if (touch11 > sensitivity)
    sensors[10] = 1;
  if (touch12 > sensitivity)
    sensors[11] = 1;
  if (touch13 > sensitivity)
    sensors[12] = 1;
  if (touch14 > sensitivity)
    sensors[13] = 1;
  if (touch15 > sensitivity)
    sensors[14] = 1;

  for (int i = 0; i < 15; i = i + 1) { // prints the reading for each
                                                                                                    //sensor
    Serial.print(sensors[i]);
  }
   Serial.println("\t");


  delay(0);  // arbitrary delay to limit data to serial port
}
```

3. Serial Port Connection

Sending Unique Readings to the Touch Sensors via Serial Port

```
#include <CapacitiveSensor.h>

const int samples = 60; // number of samples collected by touch sensor
const int sensitivity = 250; // threshold for sentisty of sensors
int state[15] = {0}; // record previous state for touch sensor 1= ON, 0=OFF
```

```
CapacitiveSensor   cs_2_3 = CapacitiveSensor(2,3); // 1M resistor between pins 2 & 3, pin 3 is sensor pin, add a wire and or foil
CapacitiveSensor   cs_2_4 = CapacitiveSensor(2,4); // 1M resistor between pins 2 & 4, pin 4 is sensor pin, add a wire and or foil
CapacitiveSensor   cs_2_5 = CapacitiveSensor(2,5); // 1M resistor between pins 2 & 5, pin 5 is sensor pin, add a wire and or foil
CapacitiveSensor   cs_2_6 = CapacitiveSensor(2,6);  //4
CapacitiveSensor   cs_2_7 = CapacitiveSensor(2,7);  //5
CapacitiveSensor   cs_8_9 = CapacitiveSensor(8,9);   //6
CapacitiveSensor   cs_8_10 = CapacitiveSensor(8,10);   //7
CapacitiveSensor   cs_8_11 = CapacitiveSensor(8,11);  //8
CapacitiveSensor   cs_8_12 = CapacitiveSensor(8,12);  //9
CapacitiveSensor   cs_8_13 = CapacitiveSensor(8,13);   //10
CapacitiveSensor   cs_14_15 = CapacitiveSensor(14,15);  //11
CapacitiveSensor   cs_14_16 = CapacitiveSensor(14,16);  //12
CapacitiveSensor   cs_14_17 = CapacitiveSensor(14,17);  //13
CapacitiveSensor   cs_14_18 = CapacitiveSensor(14,18);  //14
CapacitiveSensor   cs_14_19 = CapacitiveSensor(14,19);  //15

void setup()
{

   Serial.begin(9600);

}

void loop(){

    long start = millis();// record time program is running
    long touch1 =  cs_2_3.capacitiveSensor(samples); // read data from sensors
    long touch2 =  cs_2_4.capacitiveSensor(samples);
    long touch3 =  cs_2_5.capacitiveSensor(samples);
    long touch4 =  cs_2_6.capacitiveSensor(samples);
    long touch5 =  cs_2_7.capacitiveSensor(samples);
    long touch6 =  cs_8_9.capacitiveSensor(samples);
    long touch7 =  cs_8_10.capacitiveSensor(samples);
    long touch8 =  cs_8_11.capacitiveSensor(samples);
    long touch9 =  cs_8_12.capacitiveSensor(samples);
    long touch10 =  cs_8_13.capacitiveSensor(samples);
    long touch11 =  cs_14_15.capacitiveSensor(samples);
    long touch12 =  cs_14_16.capacitiveSensor(samples);
    long touch13 =  cs_14_17.capacitiveSensor(samples);
    long touch14 =  cs_14_18.capacitiveSensor(samples);
    long touch15 =  cs_14_19.capacitiveSensor(samples);



    if (touch1 > sensitivity){ // is the touch sensor being touched?
      Serial.write('A'); // Yes, so write out 'A'
      state[0] = 1; // set previous state to On
    }
    else if ((touch1 <= sensitivity) && (state[0] == 1)){ // is the touch not being touched and is the previous state on
```

```
  Serial.write('a'); //Write out 'a'
  state[0]= 0; // set previous state to Off
}

if (touch2 > sensitivity){
  Serial.write('B');
  state[1] = 1;
}
else if ((touch2 <= sensitivity) && (state[1] == 1)){
  Serial.write('b');
  state[1]= 0;
}

if (touch3 > sensitivity){
  Serial.write('C');
  state[2] = 1;
}
else if ((touch3 <= sensitivity) && (state[2] == 1)){
  Serial.write('c');
  state[2]= 0;
}

if (touch4 > sensitivity){
  Serial.write('D');
  state[3] = 1;
}
else if ((touch4 <= sensitivity) && (state[3] == 1)){
  Serial.write('d');
  state[3]= 0;
}

if (touch5 > sensitivity){
  Serial.write('E');
  state[4] = 1;
}
else if ((touch5 <= sensitivity) && (state[4] == 1)){
  Serial.write('e');
  state[4]= 0;
}

if (touch6 > sensitivity){
  Serial.write('F');
  state[5] = 1;
}
else if ((touch6 <= sensitivity) && (state[5] == 1)){
  Serial.write('f');
  state[5]= 0;
}

if (touch7 > sensitivity){
```

```
  Serial.write('G');
  state[6] = 1;
}
else if ((touch7 <= sensitivity) && (state[6] == 1)){
  Serial.write('g');
  state[6]= 0;
}

if (touch8 > sensitivity){
  Serial.write('H');
  state[7] = 1;
}
else if ((touch8 <= sensitivity) && (state[7] == 1)){
  Serial.write('h');
  state[7]= 0;
}

if (touch9 > sensitivity){
  Serial.write('I');
  state[8] = 1;
}
else if ((touch9 <= sensitivity) && (state[8] == 1)){
  Serial.write('i');
  state[8]= 0;
}

if (touch10 > sensitivity){
  Serial.write('J');
  state[9] = 1;
}
else if ((touch10 <= sensitivity) && (state[9] == 1)){
  Serial.write('j');
  state[9]= 0;
}
if (touch11 > sensitivity){
  Serial.write('K');
  state[10] = 1;
}
else if ((touch11 <= sensitivity) && (state[10] == 1)){
  Serial.write('k');
  state[10]= 0;
}

if (touch12 > sensitivity){
  Serial.write('L');
  state[11] = 1;
}
else if ((touch12 <= sensitivity) && (state[11] == 1)){
  Serial.write('l');
  state[11]= 0;
```

```
    }

    if (touch13 > sensitivity){
      Serial.write('M');
      state[12] = 1;
    }
    else if ((touch13 <= sensitivity) && (state[12] == 1)){
      Serial.write('m');
      state[12]= 0;
    }

    if (touch14 > sensitivity){
      Serial.write('N');
      state[13] = 1;
    }
    else if ((touch14 <= sensitivity) && (state[13] == 1)){
      Serial.write('n');
      state[13]= 0;
    }

    if (touch15 > sensitivity){
      Serial.write('O');
      state[14] = 1;
    }
    else if ((touch15 <= sensitivity) && (state[14] == 1)){
      Serial.write('o');
      state[14]= 0;
    }

    delay(0); // arbitrary delay to limit data to serial port
}
```

Appendix C: Receive LED Control Data

1. LED Shift Register Control via Serial Port Connection

```
//Touch Sensor variables

int started = 0;

//Shift Register Variables
const int clockPin = 35; //Pin connected to clock pin (SH_CP) of 74HC595
const int dataPin = 36; //Pin connected to Data in (DS) of 74HC595

//Serial Variables
```

```
String inputString = "";         // a string to hold incoming data
boolean stringComplete = false;  // whether the string is complete

void setup() {
  //set pins to output because they are addressed in the main loop
  pinMode(dataPin, OUTPUT);
  pinMode(clockPin, OUTPUT);

  Serial.begin(9600);
  inputString.reserve(200);
  shiftOut(dataPin, clockPin, MSBFIRST, B00000); // initialize LED outputs to off

}


void loop() {

    if (stringComplete) {         // start reading serial values
      pinWrite(inputString); //call function to control LEDs
      // clear the string:
      inputString = "";
      stringComplete = false;
   }
}



//*****************************METHODS*****************************//


void pinWrite(String whichKey) {

  int key;

  if(whichKey.startsWith("R")){ //do we want to turn on red?
    Serial.print("Red");
    colorWrite('R');
  }
  if(whichKey.startsWith("G")){
    Serial.print("Green");
    colorWrite('G');
  }

  if(whichKey.startsWith("B")){
    Serial.print("Blue");
    colorWrite('B');
  }

  if(whichKey.startsWith("C")){
    Serial.print("Cyan");
```

```
    colorWrite('C');
  }

  if(whichKey.startsWith("M")){
    Serial.print("Magenta");
    colorWrite('M');
  }

  if(whichKey.startsWith("O")){
    Serial.print("Off");
    colorWrite('O');
  }
}

//Get data from serial (MAX)
void serialEvent() {        // checks to see if serial event is occuring
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();  // read characters
    // add it to the inputString:
    inputString += inChar; // write characters to a string
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') { // stop reading when newline is received
      stringComplete = true;
    }
  }
}
//turns on correct color
void colorWrite(char whichColor) {
// the bits you want to send
  byte bitsToSend = B000;

  switch(whichColor){
    case 'R':
     bitsToSend = B001; //Red on lED
     break;
    case 'G':
     bitsToSend = B010; //Green on LED
     break;
    case 'B':
     bitsToSend = B100;//Blue on LED
     break;
    case 'C':
     bitsToSend = B110;//Cyan on LED
     break;
    case 'M':
     bitsToSend = B101;//Magenta on LED
     break;
    case 'O':
```

```
      bitsToSend = B000;// LED OFF
      break;
}
    // shift the bits out:
    shiftOut(dataPin, clockPin, MSBFIRST, bitsToSend);

  }
```

2. Activating the 15 RGB LEDs via Shift Register and Serial Port Connection

```
//Clock and Data Pin for first Shift Register
const int clockPin1 = 35;
const int dataPin1 = 36;

//Clock and Data Pin for second Shift Register
const int clockPin2 = 37;
const int dataPin2 = 38;

//Clock and Data Pin for third Shift Register
const int clockPin3 = 38;
const int dataPin3 = 39;

//Clock and Data Pin for fourth Shift Register
const int clockPin4 = 40;
const int dataPin4 = 41;

//Clock and Data Pin for fifth Shift Register
const int clockPin5 = 42;
const int dataPin5 = 43;

//Clock and Data Pin for sixth Shift Register
const int clockPin6 = 44;
const int dataPin6 = 45;

//shifRegister Data
word shift1= 0;
word shift2= 0;
word shift3= 0;

String inputString = "";         // a string to hold incoming data
boolean stringComplete = false;  // whether the string is complete

void setup() {
  pinMode(dataPin1, OUTPUT);
  pinMode(clockPin1, OUTPUT);
  pinMode(clockPin2, OUTPUT);
  pinMode(dataPin2, OUTPUT);
  pinMode(dataPin3, OUTPUT);
  pinMode(clockPin3, OUTPUT);
  pinMode(clockPin4, OUTPUT);
```

```
  pinMode(dataPin4, OUTPUT);
  pinMode(dataPin5, OUTPUT);
  pinMode(clockPin5, OUTPUT);
  pinMode(clockPin6, OUTPUT);
  pinMode(dataPin6, OUTPUT);

  digitalWrite(clockPin1,LOW);
  digitalWrite(clockPin2,LOW);
  digitalWrite(dataPin1,LOW);
  digitalWrite(dataPin2,LOW);

  Serial.begin(9600);
  inputString.reserve(200);
}
void loop() {
  if (stringComplete) {
      //Serial.println(inputString);
        pinWrite(inputString);
        // clear the string:
        inputString = "";
        stringComplete = false;
     }
}
void pinWrite(String whichKey) {
  int key;

  if(whichKey.startsWith("R")){
    Serial.print("Red");
    key = getKeyNumber(whichKey);
    colorWrite('R', key);
    Serial.print("\t");
    Serial.println(key);
  }
  if(whichKey.startsWith("G")){
    Serial.print("Green");
    key = getKeyNumber(whichKey);
    colorWrite('G',key);
    Serial.print("\t");
    Serial.println(key);
  }
  if(whichKey.startsWith("B")){
    Serial.print("Blue");
    key = getKeyNumber(whichKey);
    colorWrite('B',key);
    Serial.print("\t");
    Serial.println(key);
  }
  if(whichKey.startsWith("C")){
    Serial.print("Cyan");
    key = getKeyNumber(whichKey);
```

```
    colorWrite('C',key);
    Serial.print("\t");
    Serial.println(key);
  }
  if(whichKey.startsWith("M")){
    Serial.print("Magenta");
    key = getKeyNumber(whichKey);
    colorWrite('M',key);
    Serial.print("\t");
    Serial.println(key);
  }
  if(whichKey.startsWith("O")){
    Serial.print("Off");
    key = getKeyNumber(whichKey);
    colorWrite('O',key);
    Serial.print("\t");
    Serial.println(key);
  }
}
//gets the number for the key that needs to be controlled
int getKeyNumber(String getKey){
  String number = "";
  int keyNum;

  number += (char)getKey.charAt(1);
  number += (char)getKey.charAt(2);
  keyNum = number.toInt();
  number = "";
  return keyNum;
}

//Get data from serial (MAX)
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
//turns on correct color
void colorWrite(char whichColor, int thisKey) {
// the bits you want to send
  byte bitsToSend = B000;
  word shifth = 0;
```

```
word bitMask = getBitMask(thisKey);
Serial.print("\t");
Serial.print(bitMask,BIN);

switch(whichColor){
  case 'R':
    bitsToSend = B001;
    break;
  case 'G':
    bitsToSend = B010;
    break;
  case 'B':
    bitsToSend = B100;
    break;
  case 'C':
    bitsToSend = B110;
    break;
  case 'M':
    bitsToSend = B101;
    break;
  case 'O':
    bitsToSend = B000;
    break;
}
Serial.print("\t");
Serial.print(bitsToSend,BIN);

if(thisKey >=1 && thisKey < 6){
shift1 = bitMask & shift1;
shifth = bitsToSend << ((thisKey -1)*3);
shift1 = shift1 | shifth;
shiftBits(shift1, thisKey);
Serial.print("\t");
Serial.print("SHIFT1: ");
Serial.print(shift1,BIN);
}
if(thisKey >=6 && thisKey < 11){
shift2 = bitMask & shift2;
shifth = bitsToSend << (((thisKey-5) -1)*3);//need to correct key number for second shift register group
shift2 = shift2 | shifth;
shiftBits(shift2, thisKey);
Serial.print("\t");
Serial.print("SHIFT2: ");
Serial.print(shift2,BIN);
}
if(thisKey >=11 && thisKey < 16){
shift3 = bitMask & shift3;
shifth = bitsToSend << (((thisKey-10) -1)*3); //need to correct key number for third shift register group
shift3 = shift3 | shifth;
shiftBits(shift3, thisKey);
```

```
  Serial.print("\t");
  Serial.print("SHIFT3: ");
  Serial.print(shift3,BIN);
  }
}
word getBitMask(int keyNum){
  if((keyNum == 1) || (keyNum == 6) || (keyNum == 11))
    return 0xFFF8;//65528; //1111111111111000
  if((keyNum == 2) || (keyNum == 7) || (keyNum == 12))
    return 0xFFC7;//65479; //1111111111000111
  if((keyNum == 3) || (keyNum == 8) || (keyNum == 13))
    return 0xFE3F;//65087; //1111111000111111
  if((keyNum == 4) || (keyNum == 9) || (keyNum == 14))
    return 0xF1FF;//61951; //1111000111111111
  if((keyNum == 5) || (keyNum == 10) || (keyNum == 15))
    return 0x8FFF;//36863; //1000111111111111
}
void shiftBits(word bytes, int thisShiftReg){

  if(thisShiftReg >=1 && thisShiftReg < 6){
  // shift the bits out:
  shiftOut(dataPin1, clockPin1, MSBFIRST, lowByte(bytes));
  shiftOut(dataPin2, clockPin2, MSBFIRST, highByte(bytes));
  }
  if(thisShiftReg >=6 && thisShiftReg < 11){
  // shift the bits out:
  shiftOut(dataPin3, clockPin3, MSBFIRST, lowByte(bytes));
  shiftOut(dataPin4, clockPin4, MSBFIRST, highByte(bytes));
  }
  if(thisShiftReg >=11 && thisShiftReg < 16){
  // shift the bits out:
  shiftOut(dataPin5, clockPin5, MSBFIRST, lowByte(bytes));
  shiftOut(dataPin6, clockPin6, MSBFIRST, highByte(bytes));
  }
}
void turnAllOff(){
  pinWrite("O01");
  pinWrite("O02");
  pinWrite("O03");
  pinWrite("O04");
  pinWrite("O05");
  pinWrite("O06");
  pinWrite("O07");
  pinWrite("O09");
  pinWrite("O10");
  pinWrite("O11");
  pinWrite("O12");
  pinWrite("O13");
  pinWrite("O14");
  pinWrite("O15");
```

```
}
```

Appendix D: Full Program Execution

Control the Keyboard's 15 electromagnets, 15 RGB LEDs and 15 Capacitive Touch Sensors

```
#include <CapacitiveSensor.h>

//Touch Sensor variables
CapacitiveSensor   cs_2_3 = CapacitiveSensor(2,3);        // 1M resistor between pins 2 & 3, pin 3 is sensor pin, add a wire and or foil
CapacitiveSensor   cs_2_4 = CapacitiveSensor(2,4);        // 1M resistor between pins 2 & 4, pin 4 is sensor pin, add a wire and or foil
CapacitiveSensor   cs_2_5 = CapacitiveSensor(2,5);        // 1M resistor between pins 2 & 5, pin 5 is sensor pin, add a wire and or foil
CapacitiveSensor   cs_2_6 = CapacitiveSensor(2,6);  //4
CapacitiveSensor   cs_2_7 = CapacitiveSensor(2,7);  //5
CapacitiveSensor   cs_8_9 = CapacitiveSensor(8,9);   //6
CapacitiveSensor   cs_8_10 = CapacitiveSensor(8,10);   //7
CapacitiveSensor   cs_8_11 = CapacitiveSensor(8,11);  //8
CapacitiveSensor   cs_8_12 = CapacitiveSensor(8,12);  //9
CapacitiveSensor   cs_8_13 = CapacitiveSensor(8,13);   //10
CapacitiveSensor   cs_14_15 = CapacitiveSensor(14,15);  //11
CapacitiveSensor   cs_14_16 = CapacitiveSensor(14,16);  //12
CapacitiveSensor   cs_14_17 = CapacitiveSensor(14,17);  //13
CapacitiveSensor   cs_14_18 = CapacitiveSensor(14,18);  //14
CapacitiveSensor   cs_14_19 = CapacitiveSensor(14,19);  //15

const int samples = 60;
const int sensitivity = 300;

int state[15] = {0};

//Shift REgister Variables
//Clock and Data Pin for first Shift Register
const int clockPin1 = 35;//white wire
const int dataPin1 = 36; //yellow wire

//Clock and Data Pin for second Shift Register
const int clockPin2 = 37;
const int dataPin2 = 38;

//Clock and Data Pin for third Shift Register
const int clockPin3 = 39;
const int dataPin3 = 40;

//Clock and Data Pin for fourth Shift Register
const int clockPin4 = 41;
const int dataPin4 = 42;
```

```
//Clock and Data Pin for fifth Shift Register
const int clockPin5 = 43;
const int dataPin5 = 44;

//Clock and Data Pin for sixth Shift Register
const int clockPin6 = 45;
const int dataPin6 = 46;

//electromagnets
const int e1 = 20;
const int e2 = 21;
const int e3 = 22;
const int e4 = 23;
const int e5 = 24;
const int e6 = 25;
const int e7 = 26;
const int e8 = 27;
const int e9 = 28;
const int e10 = 29;
const int e11 = 30;
const int e12 = 31;
const int e13 = 32;
const int e14 = 33;
const int e15 = 34;


//shifRegister Data
word shift1= 0;
word shift2= 0;
word shift3= 0;

//Serial Variables
String inputString = "";         // a string to hold incoming data
boolean stringComplete = false;  // whether the string is complete

void setup() {
  //set pins to output because they are addressed in the main loop
  pinMode(dataPin1, OUTPUT);
  pinMode(clockPin1, OUTPUT);
  pinMode(clockPin2, OUTPUT);
  pinMode(dataPin2, OUTPUT);
  pinMode(clockPin3, OUTPUT);
  pinMode(dataPin3, OUTPUT);
  pinMode(clockPin4, OUTPUT);
  pinMode(dataPin4, OUTPUT);
  pinMode(clockPin5, OUTPUT);
  pinMode(dataPin5, OUTPUT);
  pinMode(clockPin6, OUTPUT);
  pinMode(dataPin6, OUTPUT);
```

```
  pinMode(e1, OUTPUT);
  pinMode(e2, OUTPUT);
  pinMode(e3, OUTPUT);
  pinMode(e4, OUTPUT);
  pinMode(e5, OUTPUT);
  pinMode(e6, OUTPUT);
  pinMode(e7, OUTPUT);
  pinMode(e8, OUTPUT);
  pinMode(e9, OUTPUT);
  pinMode(e10, OUTPUT);
  pinMode(e11, OUTPUT);
  pinMode(e12, OUTPUT);
  pinMode(e13, OUTPUT);
  pinMode(e14, OUTPUT);
  pinMode(e15, OUTPUT);

  turnAllOff();

  Serial.begin(9600);
  inputString.reserve(200);
}


void loop() {
    long start = millis();
    long touch1 =  cs_2_3.capacitiveSensor(samples);
    long touch2 =  cs_2_4.capacitiveSensor(samples);
    long touch3 =  cs_2_5.capacitiveSensor(samples);
    long touch4 =  cs_2_6.capacitiveSensor(samples);
    long touch5 =  cs_2_7.capacitiveSensor(samples);
    long touch6 =  cs_8_9.capacitiveSensor(samples);
    long touch7 =  cs_8_10.capacitiveSensor(samples);
    long touch8 =  cs_8_11.capacitiveSensor(samples);
    long touch9 =  cs_8_12.capacitiveSensor(samples);
    long touch10 =  cs_8_13.capacitiveSensor(samples);
    long touch11 =  cs_14_15.capacitiveSensor(samples);
    long touch12 =  cs_14_16.capacitiveSensor(samples);
    long touch13 =  cs_14_17.capacitiveSensor(samples);
    long touch14 =  cs_14_18.capacitiveSensor(samples);
    long touch15 =  cs_14_19.capacitiveSensor(samples);

    if ((touch1 > sensitivity)&& (state[0] == 0)){
      Serial.write('A');
      state[0] = 1;
    }
    else if ((touch1 <= sensitivity) && (state[0] == 1)){
      Serial.write('a');
      state[0]= 0;
    }
```

```
if ((touch2 > sensitivity)&& (state[1] == 0)){
  Serial.write('B');
  state[1] = 1;
}
else if ((touch2 <= sensitivity) && (state[1] == 1)){
  Serial.write('b');
  state[1]= 0;
}

if ((touch3 > sensitivity)&& (state[2] == 0)){
  Serial.write('C');
  state[2] = 1;
}
else if ((touch3 <= sensitivity) && (state[2] == 1)){
  Serial.write('c');
  state[2]= 0;
}

if((touch4 > sensitivity)&& (state[3] == 0)){
  Serial.write('D');
  state[3] = 1;
}
else if ((touch4 <= sensitivity) && (state[3] == 1)){
  Serial.write('d');
  state[3]= 0;
}

if((touch5 > sensitivity)&& (state[4] == 0)){
  Serial.write('E');
  state[4] = 1;
}
else if ((touch5 <= sensitivity) && (state[4] == 1)){
  Serial.write('e');
  state[4]= 0;
}

if((touch6 > sensitivity)&& (state[5] == 0)){
  Serial.write('F');
  state[5] = 1;
}
else if ((touch6 <= sensitivity) && (state[5] == 1)){
  Serial.write('f');
  state[5]= 0;
}

if((touch7 > sensitivity)&& (state[6] == 0)){
  Serial.write('G');
  state[6] = 1;
}
else if ((touch7 <= sensitivity) && (state[6] == 1)){
```

```
  Serial.write('g');
  state[6]= 0;
}

if((touch8 > sensitivity)&& (state[7] == 0)){
  Serial.write('H');
  state[7] = 1;
}
else if ((touch8 <= sensitivity) && (state[7] == 1)){
  Serial.write('h');
  state[7]= 0;
}

if((touch9 > sensitivity)&& (state[8] == 0)){
  Serial.write('I');
  state[8] = 1;
}
else if ((touch9 <= sensitivity) && (state[8] == 1)){
  Serial.write('i');
  state[8]= 0;
}

if((touch10 > sensitivity)&& (state[9] == 0)){
  Serial.write('J');
  state[9] = 1;
}
else if ((touch10 <= sensitivity) && (state[9] == 1)){
  Serial.write('j');
  state[9]= 0;
}

if((touch11 > sensitivity)&& (state[10] == 0)){
  Serial.write('K');
  state[10] = 1;
}
else if ((touch11 <= sensitivity) && (state[10] == 1)){
  Serial.write('k');
  state[10]= 0;
}

if((touch12 > sensitivity)&& (state[11] == 0)){
  Serial.write('L');
  state[11] = 1;
}
else if ((touch12 <= sensitivity) && (state[11] == 1)){
  Serial.write('l');
  state[11]= 0;
}

if((touch13 > sensitivity)&& (state[12] == 0)){
```

```
      Serial.write('M');
      state[12] = 1;
    }
    else if ((touch13 <= sensitivity) && (state[12] == 1)){
      Serial.write('m');
      state[12]= 0;
    }

    if ((touch14 > sensitivity)&& (state[13] == 0)){
      Serial.write('N');
      state[13] = 1;
    }
    else if ((touch14 <= sensitivity) && (state[13] == 1)){
      Serial.write('n');
      state[13]= 0;
    }

    if ((touch15 > sensitivity)&& (state[14] == 0)){
      Serial.write('O');
      state[14] = 1;
    }
    else if ((touch15 <= sensitivity) && (state[14] == 1)){
      Serial.write('o');
      state[14]= 0;
    }

    if (stringComplete) {
    //Serial.println(inputString);
      pinWrite(inputString);
      // clear the string:
      inputString = "";
      stringComplete = false;
    }
}



//*****************************METHODS*****************************//

void pinWrite(String whichKey) {

  int key;

  if(whichKey.startsWith("R")){
    key = getKeyNumber(whichKey);
    colorWrite('R', key);
    pullOn(key);
  }
  if(whichKey.startsWith("G")){
    key = getKeyNumber(whichKey);
```

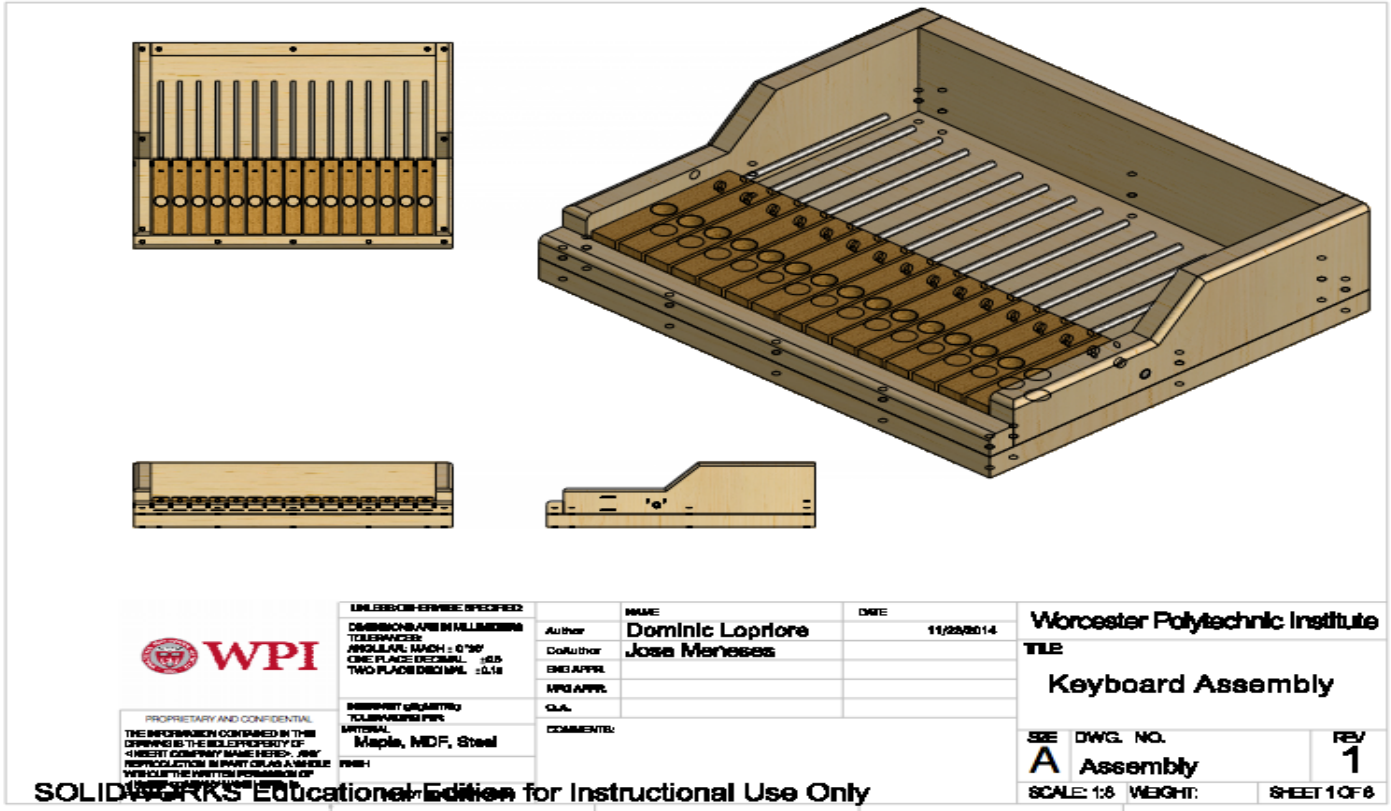
```
    colorWrite('G',key);
    pullOn(key);
  }

  if(whichKey.startsWith("B")){
    key = getKeyNumber(whichKey);
    colorWrite('B',key);
    pullOn(key);
  }

  if(whichKey.startsWith("C")){
    key = getKeyNumber(whichKey);
    colorWrite('C',key);
    pullOn(key);
  }

  if(whichKey.startsWith("M")){
    key = getKeyNumber(whichKey);
    colorWrite('M',key);
    pullOn(key);
  }

  if(whichKey.startsWith("O")){
    key = getKeyNumber(whichKey);
    colorWrite('O',key);
    pullOff(key);
  }
}

//gets the number for the key that needs to be controlled
int getKeyNumber(String getKey){
  String number = "";
  int keyNum;

  number += (char)getKey.charAt(1);
  number += (char)getKey.charAt(2);
  keyNum = number.toInt();
  number = "";
  return keyNum;

}

//Get data from serial (MAX)
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
```

```
    // so the main loop can do something about it:
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
//turns on correct color
void colorWrite(char whichColor, int thisKey) {
// the bits you want to send
  byte bitsToSend = B000;
  word shifth = 0;
  word bitMask = getBitMask(thisKey);

  switch(whichColor){
    case 'R':
     bitsToSend = B001;
     break;
    case 'G':
     bitsToSend = B010;
     break;
    case 'B':
     bitsToSend = B100;
     break;
    case 'C':
     bitsToSend = B110;
     break;
    case 'M':
     bitsToSend = B101;
     break;
    case 'O':
     bitsToSend = B000;
     break;
  }

  Serial.print("\t");
  Serial.print(bitsToSend,BIN);

  if(thisKey >=1 && thisKey < 6){
  shift1 = bitMask & shift1;
  shifth = bitsToSend << ((thisKey -1)*3);
  shift1 = shift1 | shifth;
  shiftBits(shift1, thisKey);
  }

  if(thisKey >=6 && thisKey < 11){
  shift2 = bitMask & shift2;
  shifth = bitsToSend << (((thisKey-5) -1)*3);//need to correct key number for second shift register group
  shift2 = shift2 | shifth;
  shiftBits(shift2, thisKey);
  }
```

```
  if(thisKey >=11 && thisKey < 16){
  shift3 = bitMask & shift3;
  shifth = bitsToSend << (((thisKey-10) -1)*3); //need to correct key number for third shift register group
  shift3 = shift3 | shifth;
  shiftBits(shift3, thisKey);
  }
}

word getBitMask(int keyNum){
  if((keyNum == 1) || (keyNum == 6) || (keyNum == 11))
    return 0xFFF8;//65528; //1111111111111000
  if((keyNum == 2) || (keyNum == 7) || (keyNum == 12))
    return 0xFFC7;//65479; //1111111111000111
  if((keyNum == 3) || (keyNum == 8) || (keyNum == 13))
    return 0xFE3F;//65087; //1111111000111111
  if((keyNum == 4) || (keyNum == 9) || (keyNum == 14))
    return 0xF1FF;//61951; //1111000111111111
  if((keyNum == 5) || (keyNum == 10) || (keyNum == 15))
    return 0x8FFF;//36863; //1000111111111111
}

void shiftBits(word bytes, int thisShiftReg){

  if(thisShiftReg >=1 && thisShiftReg < 6){
  // shift the bits out:
  shiftOut(dataPin1, clockPin1, MSBFIRST, lowByte(bytes));
  shiftOut(dataPin2, clockPin2, MSBFIRST, highByte(bytes));
  }

  if(thisShiftReg >=6 && thisShiftReg < 11){
  // shift the bits out:
  shiftOut(dataPin3, clockPin3, MSBFIRST, lowByte(bytes));
  shiftOut(dataPin4, clockPin4, MSBFIRST, highByte(bytes));
  }

  if(thisShiftReg >=11 && thisShiftReg < 16){
  // shift the bits out:
  shiftOut(dataPin5, clockPin5, MSBFIRST, lowByte(bytes));
  shiftOut(dataPin6, clockPin6, MSBFIRST, highByte(bytes));
  }
}

//Electromagnet On
void pullOn(int thisKey){
  digitalWrite((thisKey+19),HIGH);
}
//Electromagnet Off
void pullOff(int thisKey){
  digitalWrite((thisKey+19),LOW);
```

```
}

//Turn LED and ELectromagnets off
void turnAllOff(){
  pinWrite("O01");
  pinWrite("O02");
  pinWrite("O03");
  pinWrite("O04");
  pinWrite("O05");
  pinWrite("O06");
  pinWrite("O07");
  pinWrite("O09");
  pinWrite("O10");
  pinWrite("O11");
  pinWrite("O12");
  pinWrite("O13");
  pinWrite("O14");
  pinWrite("O15");
}
```

Appendix E: Solid Works Keyboard Design Plans



	NAME	DATE
Author	Dominic Lopriore	11/22/2014
CoAuthor	Jose Meneses	
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
TOLERANCES:
ANGULAR: MACH ± 0°30'
ONE PLACE DECIMAL ±0.5
TWO PLACE DECIMAL ±0.1
MATERIAL: Maple, MDF, Steel
FINISH

PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

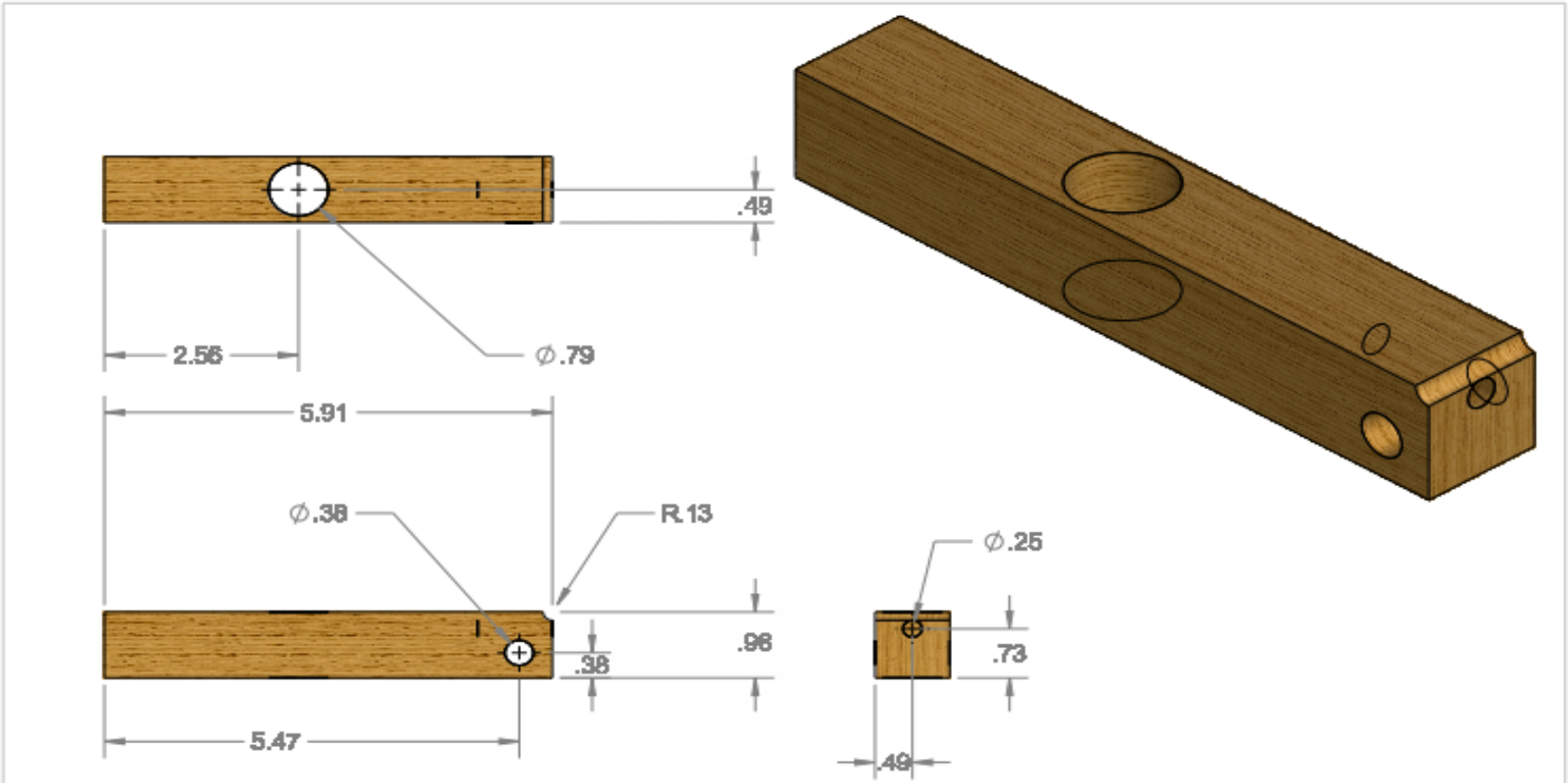
Worcester Polytechnic Institute

TITLE: Keyboard Assembly

SIZE	DWG. NO.	REV
A	Assembly	1

SCALE: 1:8 WEIGHT: SHEET 1 OF 6

5 4 3 2 1



WPI

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MILLIMETERS

TOLERANCES:

ANGULAR: MACH ± 0°30'

ONE PLACE DECIMAL ±0.5

TWO PLACE DECIMAL ±0.13

MATERIAL: Maple, MDF, Steel

FINISH

	NAME	DATE
Author	Dominic Lopriore	11/23/2014
CoAuthor	Jose Meneses	
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS:

Worcester Polytechnic Institute

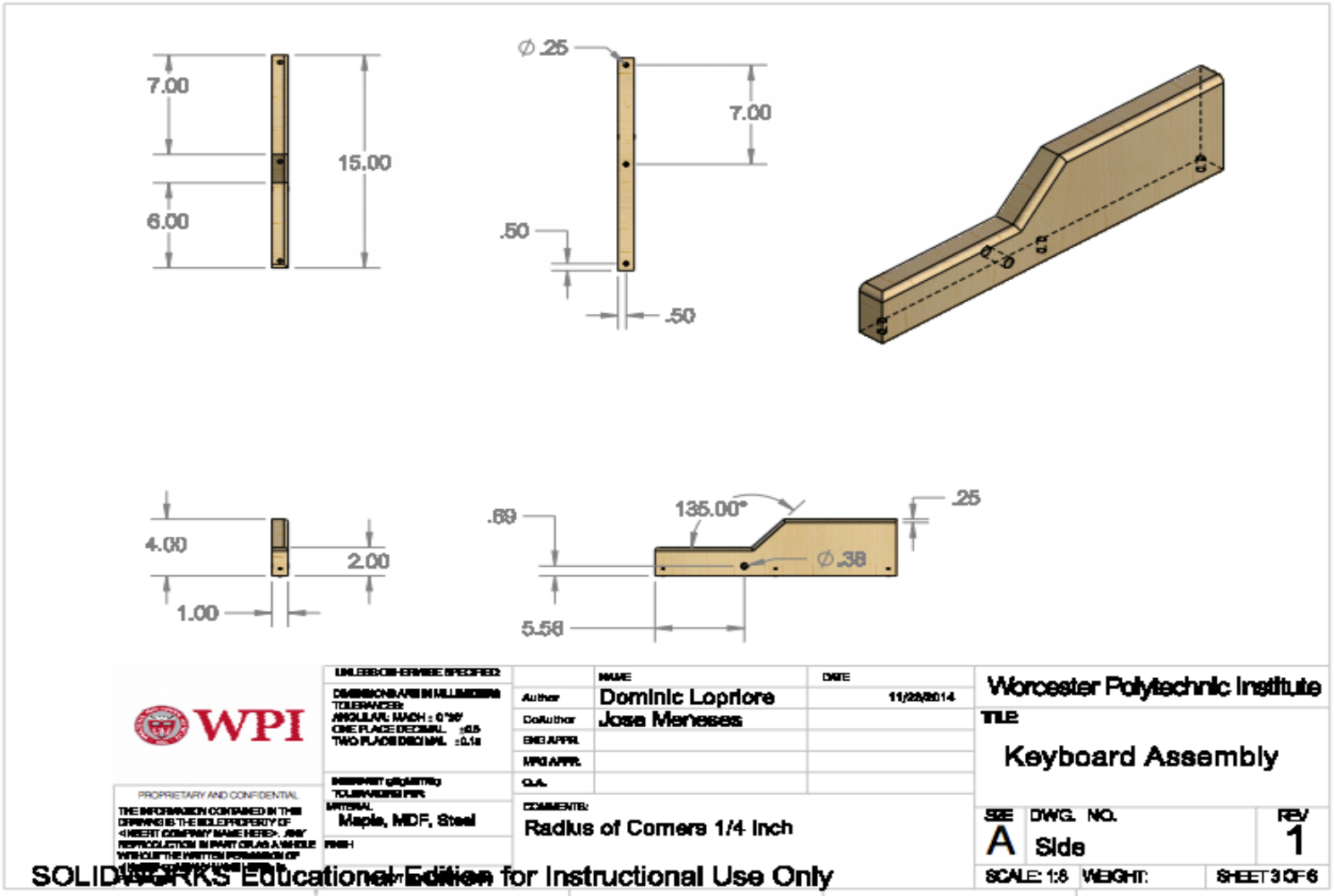
TITLE

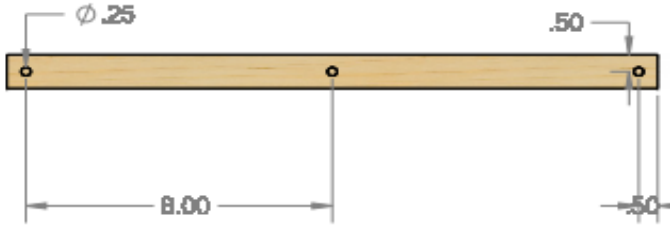
Keyboard Assembly

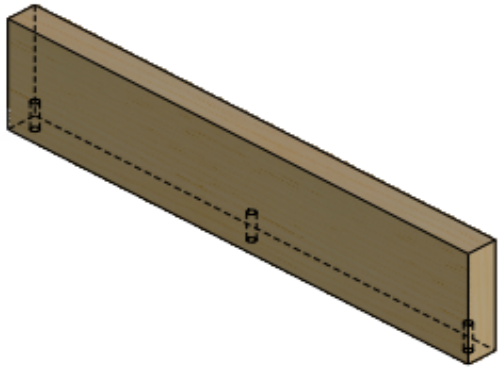
SIZE	DWG. NO.	REV
A	Key	1

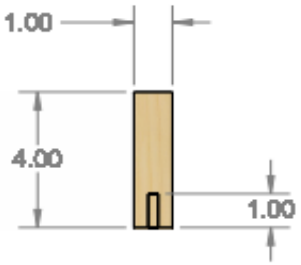
SCALE: 1:8 WEIGHT: SHEET 2 OF 6

5 4 3 2 1


7.00
15.00
6.00
Ø .25
7.00
.50
.50
4.00
2.00
1.00
.69
135.00°
.25
Ø .38
5.56
WPI
PROPRIETARY AND CONFIDENTIAL
Maple, MDF, Steel
Author
Dominic Lopriore
11/25/2014
CoAuthor
Jose Meneses
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
Radius of Corners 1/4 Inch
Worcester Polytechnic Institute
TITLE:
Keyboard Assembly
SIZE
A
DWG. NO.
Side
REV
1
SCALE: 1:8
WEIGHT:
SHEET 3 OF 6
SOLIDWORKS Educational Edition for Instructional Use Only







WPI

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE>.

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MILLIMETERS

TOLERANCES:

ANGULAR: MACH ± 0°30'

ONE PLACE DECIMAL ±0.5

TWO PLACE DECIMAL ±0.13

INTERPRET GEOMETRIC TOLERANCING PER:

MATERIAL: Maple, MDF, Steel

FINISH

	NAME	DATE
Author	Dominic Lopriore	11/22/2014
CoAuthor	Jose Meneses	
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS: Radius of Corners 1/4 Inch

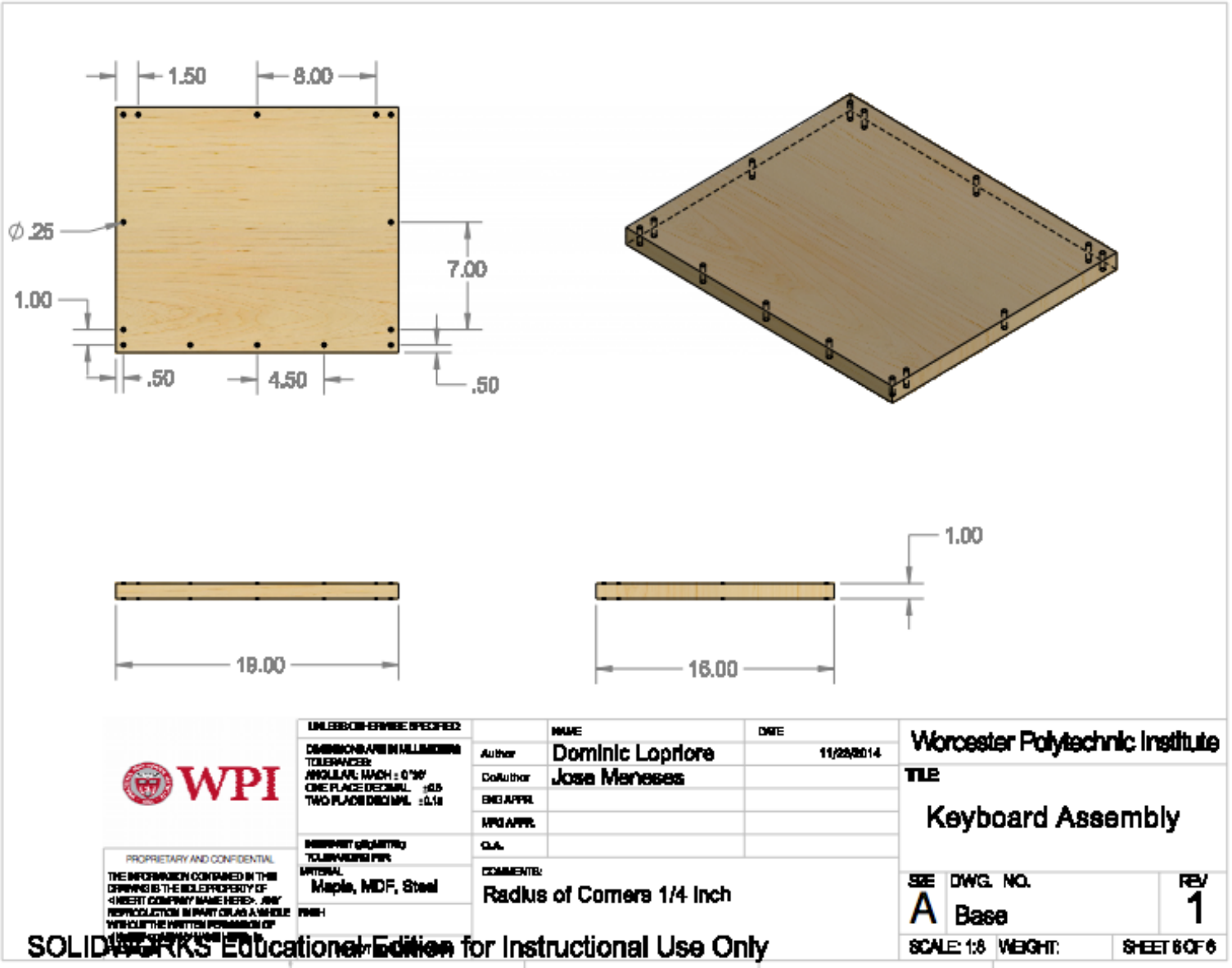
Worcester Polytechnic Institute

TITLE: Keyboard Assembly

SIZE	DWG. NO.	REV
A	Back Panel	1

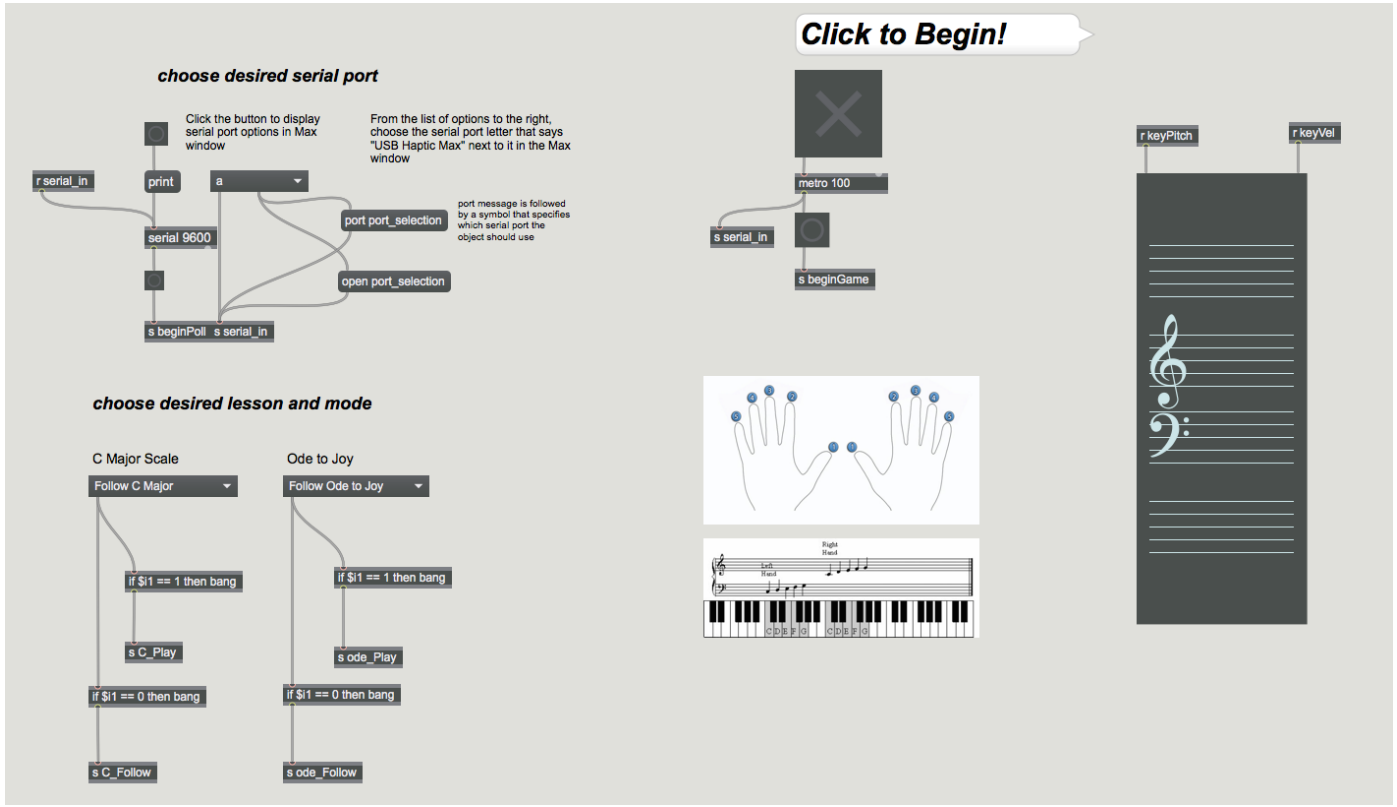
SCALE: 1:8 WEIGHT: SHEET 5 OF 6

5 4 3 2 1


1.50
8.00
Ø .25
1.00
7.00
.50
4.50
.50
1.00
19.00
16.00
WPI
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
TOLERANCES:
ANGULAR: MACH ± 0°30'
ONE PLACE DECIMAL ±0.5
TWO PLACE DECIMAL ±0.13
MATERIAL
Maple, MDF, Steel
FINISH
NAME
DATE
Author
Dominic Lopriore
11/22/2014
CoAuthor
Jose Meneses
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
Radius of Corners 1/4 Inch
Worcester Polytechnic Institute
TITLE:
Keyboard Assembly
SIZE
A
DWG. NO.
Base
REV
1
SCALE: 1:8
WEIGHT:
SHEET 6 OF 6
SOLIDWORKS Educational Edition for Instructional Use Only
5
4
3
2
1

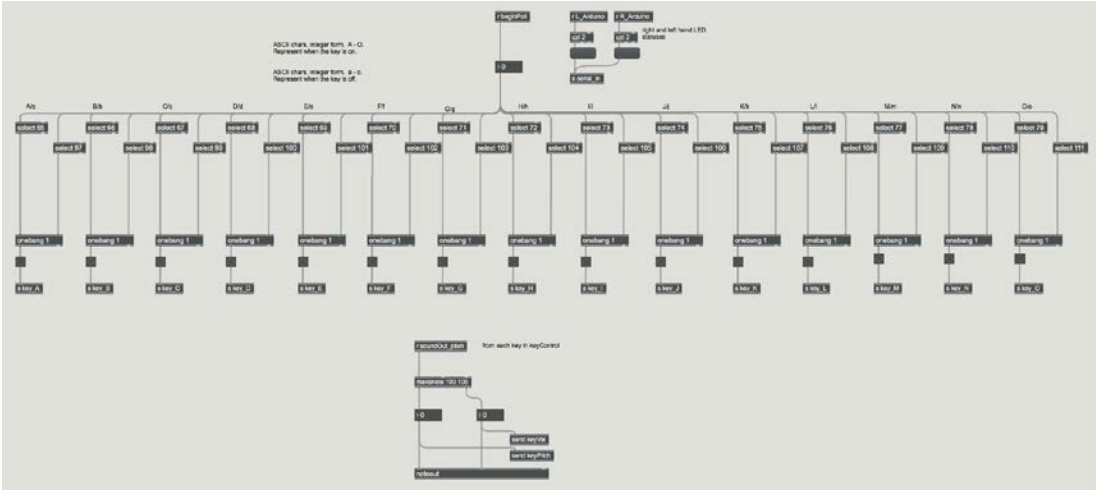
Appendix F: Max Software

1. User Interface

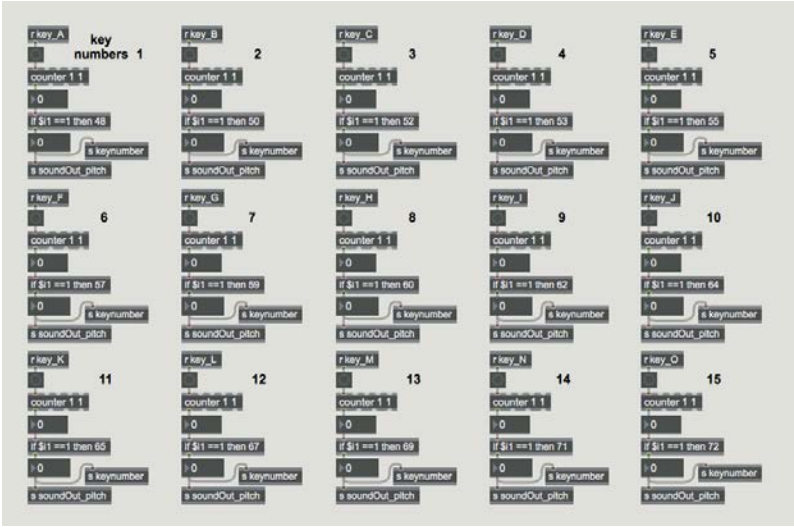


UI_homeScreen.maxpat

2. Data Control

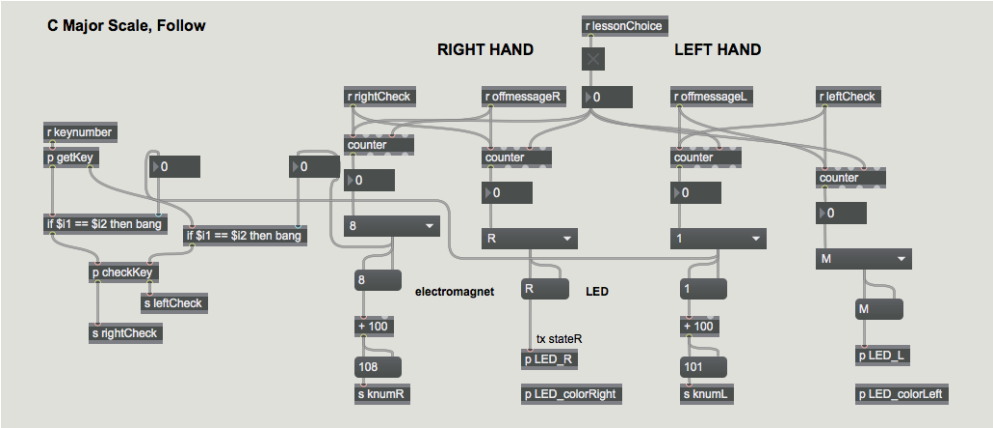


pianoPoll.maxpat

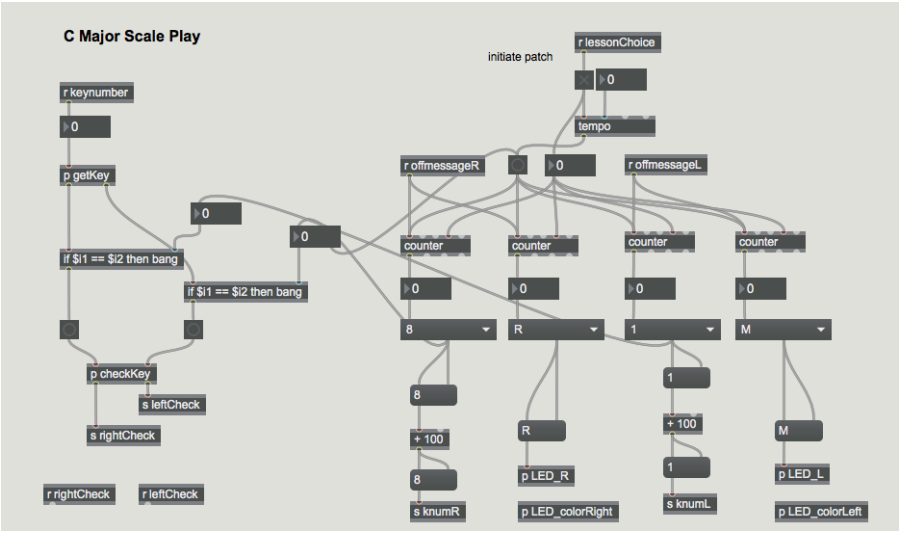


keyControl.maxpat

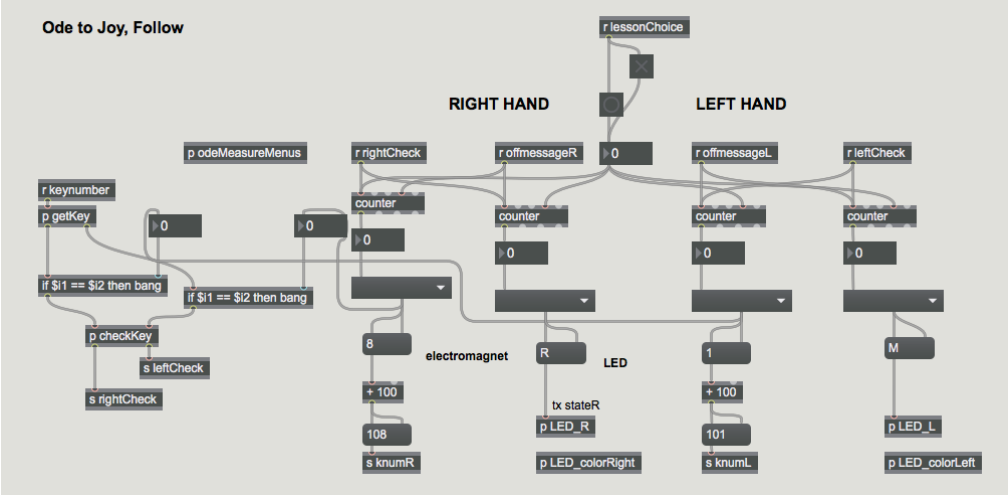
3. Music Patches



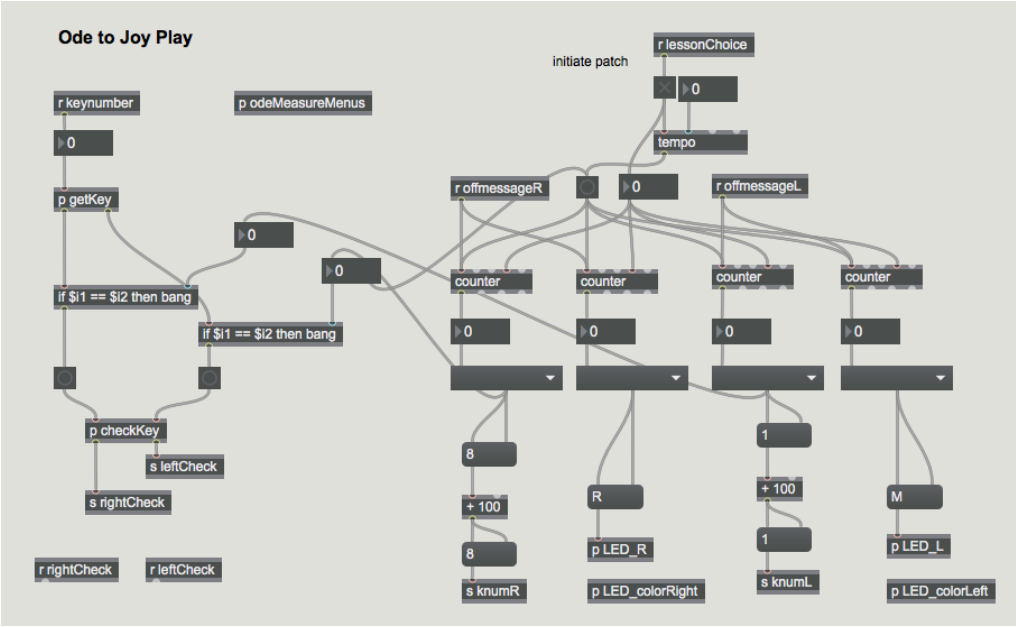
C_Major_Scale_Follow.maxpat



C_Major_Scale_Play.maxpat

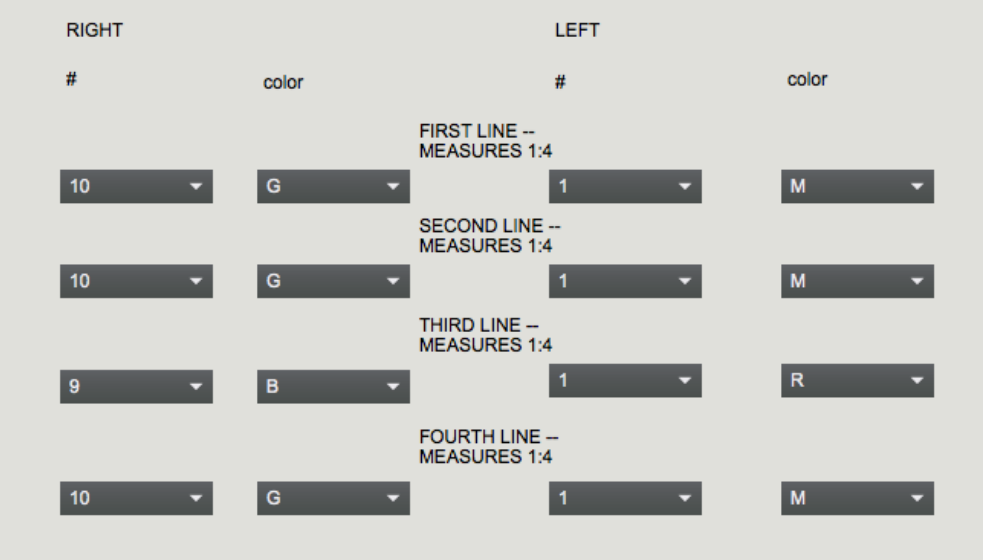


odeToJoy_Follow.maxpat

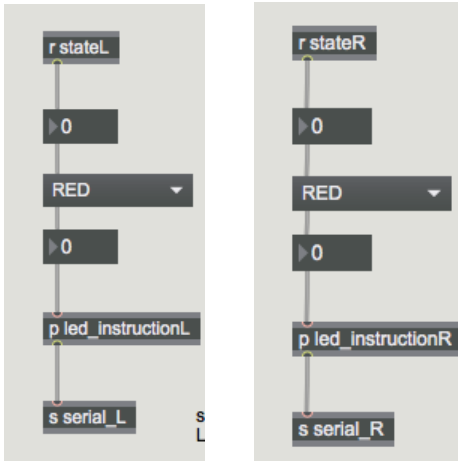


odeToJoy_Play.maxpat

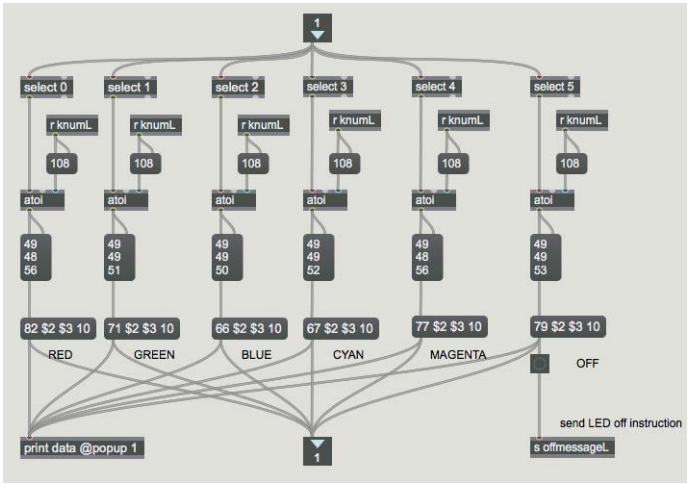
Music Patchers (Sub-patches)

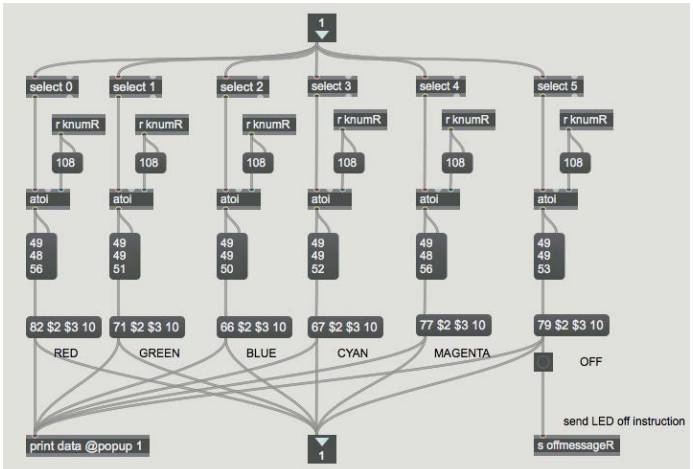


odeMeasureMenus

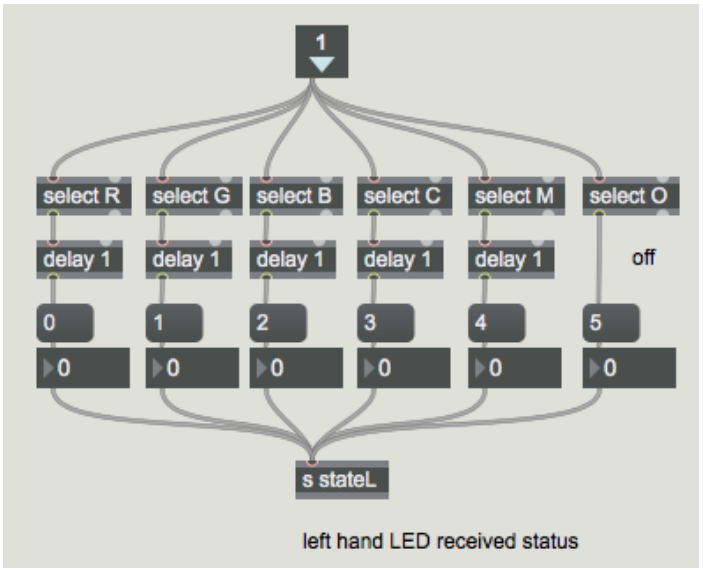


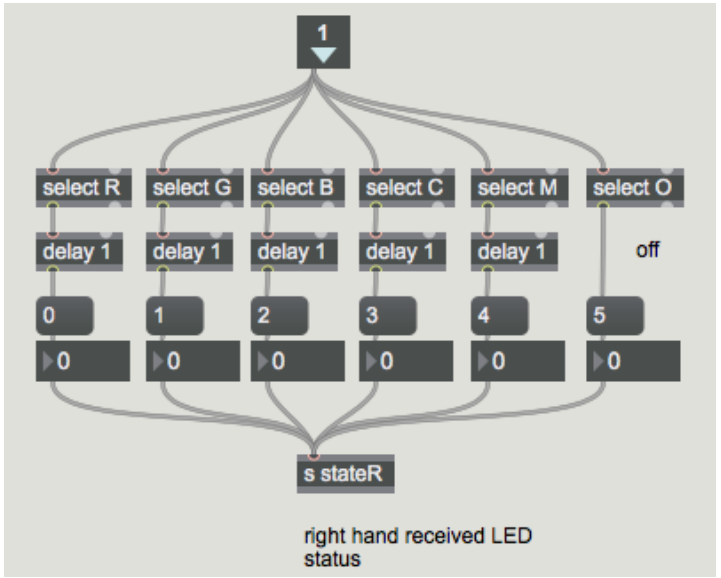
LED_colorLeft/Right



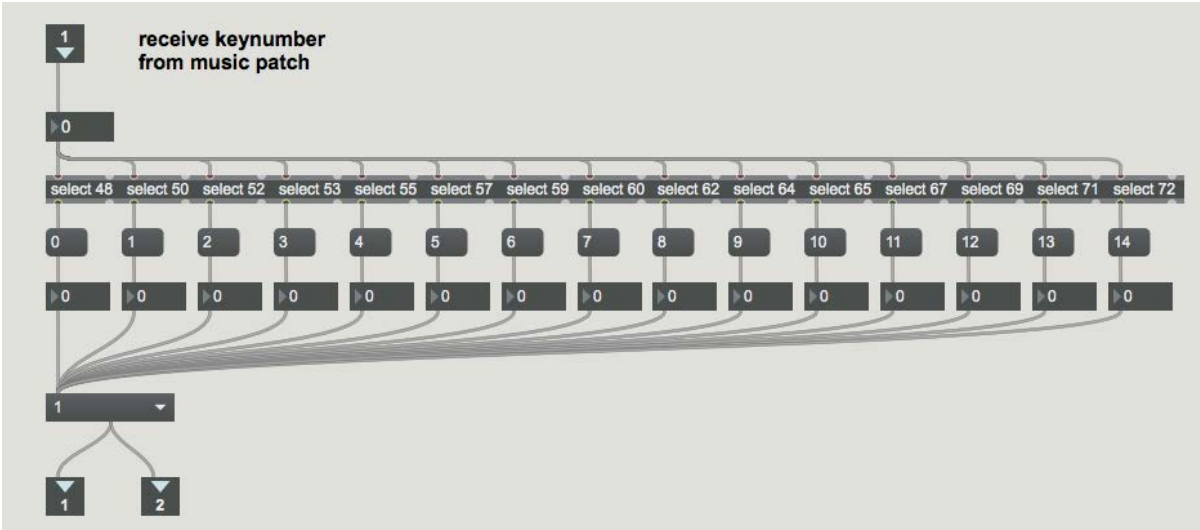


led_instructionL/R

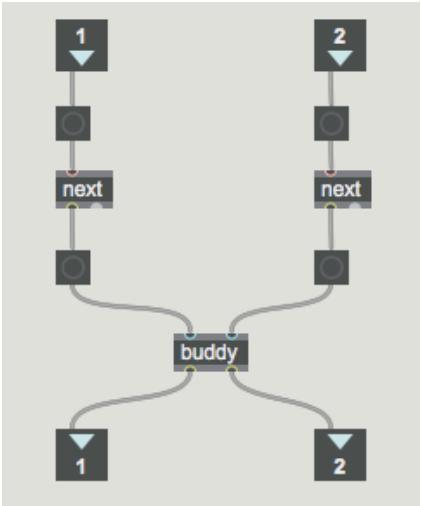




LEDstateRx_L/R.maxpat



getKey.maxpat



checkKey.maxpat