DESIGN AND DEVELOPMENT OF A RECONFIGURABLE SURFACE PROTOTYPE FOR ORTHOTICS POSTURE CORRECTION

by

Nathaniel Loewen and Parker Wise

Submitted in partial fulfillment of the
requirements for Departmental Honors in
the Department of Engineering
Texas Christian University
Fort Worth, Texas

May 8, 2017

DESIGN AND DEVELOPMENT OF A RECONFIGURABLE SURFACE PROTOTYPE FOR ORTHOTICS POSTURE CORRECTION

Project Approved:

Supervising Professor: Stephen Weis, Ph.D.

Department of Engineering

Robert Bittle, Ph.D.

Department of Engineering

Nick Bontrager, M.F.A.

Department of Studio Art

ABSTRACT

The purpose of this paper is to detail the design and development of a proof-of-concept prototype for a digitally reconfigurable surface for orthotic posture correction. To design an effective proof-of-concept, research was conducted to determine the components of an effective and practical design. After research, the development of the design consisted of building a proof-of-concept prototype and integrating it with a digital-user-interface via the use of Arduino and the Arduino environment on a personal computer. The proof-of-concept prototype allows for an array of nine shafts to be moved up or down via user input. It was successfully demonstrated for our orthotics specialist and TCU faculty and staff at the College of Science and Engineering Student Research Symposium. This research provides information and designs that will further supplement and inspire future developments of reconfigurable surfaces as well as production of custom orthotics.

Table of Contents

1. INTRODUCTION

This paper presents the design and research of part of a larger project to create a device capable of custom orthotic insert creation for posture correction. The device must be controlled digitally and support a 250-pound person. The part presented in this paper is the development of a proof-of-concept prototype and research of reconfigurable devices and improvements for the future.

Orthotic inserts are made normally in three different ways, there are digital foot scanners, foam impressions, and castings [1]. Digital foot scanners take the different pressures that vary across the foot and create an insert based on that scan. These do not provide accurate data and do not provide the inserts one needs. Foam impressions are made by a sitting or standing person pressing their feet into a box of foam. Castings are made in various ways by using casting gauze to create a copy of the shape of the foot. These last two techniques require training and experience before an Orthotist is able to create effective inserts. There can be problems from the feet not being in the correct position and the inserts can be incorrect.

Alan Hood, a licensed Orthotist and the owner of the Center for Postural Correction in Fort Worth, TX, uses foam impressions to get a starting position of the foot. He then will adjust the same feet of a person standing on the ground until he can see that the patient is standing with correct posture. Hood uses his experience and intuition and thinks of creating orthotic inserts as an art. It would be impossible for him to impart his knowledge directly, because a lot of what he sees is based on experience and happens in his mind. It is our hope that a device similar to ours would help Hood and other Orthotists to teach new people in the field how to create effective inserts. If an Orthotist can examine how the foot needs to change from when a person is standing with his or her normal posture to when that same person is standing with correct posture, it will give them a great advantage in the design and creation of custom orthotic inserts. One will not have to experience unreliable measuring devices, and Orthotists will have an easier time treating their patients.

2. BACKGROUND

The general idea for our design came from a simple pin art toy seen in figure 1. It is a general toy that makes a 3D model of any object that has manually reconfigured its pins. A more in-depth design came from researching MIT Tangible Media Group's reconfigurable surface project, inFORM [2]. It achieves the goal of creating a reconfigurable surface with the main application being digital media. It captures spatial images from a Kinect camera and translates that to the reconfigurable surface (Figure 2). Three-dimensional CAD drawings can also be translated onto the surface allowing inventors to see visual representations of their designs. According to the Tangible Media Group's website, one of the main applications is using the surface to convert geospatial data from a computer program and allow that to be visually recreated by the surface. This is especially useful with designs made by architects and city

planners. This design inspired our design: a reconfigurable surface made of an array of closely packed pins.

Figure 1: Common pin art toy. Image credit: Google Images

Figure 2: InFORM Surface projecting hands holding a flashlight. Image credit: InFORM. http://tangible.media.mit.edu/project/inform/.

To achieve our goal we had to consider ways to actuate each of these pins as well as brake them at the desired height. We also had to consider that these pins would be under a considerable amount of downward force since a person would need to be standing on the surface as it was moving, so these actuation and braking methods would have to withstand the force of a 250-lb. human being spread out across each pin. If the weight was evenly distributed across a 9 x 9 array of pins, that would mean each actuation and braking method would have to withstand at minimum 250/81 = 3.1 lbs. of force. At maximum, we estimate that each pin needs to hold 25 lbs. of force meaning that we would expect a minimum of 10 pins being in contact with the foot at any given time.

We investigated four unique ways to actuate pins in an array. Hydraulic/pneumatic actuators, piezoelectric motors, screws driven by a DC motor or stepper motor, and screws driven by a 3 – axis robotic controlled motor. Similarly, we investigated four different ways to brake an array of pins and create a stable reconfigurable surface. They were pressure (pneumatic or hydraulic), closely packed pins that utilized the friction between each pin, threaded rods, and solder.

2.1 ACTUATION

2.1.1 Hydraulic Motors

Hydraulic actuators are frequently used in manufacturing. We focused on the hydraulic motor. A hydraulic motor works by utilizing a rotating shaft to convert hydraulic energy into mechanical energy. Highly pressurized liquid, usually water or oil, is pumped through the motor and this buildup in pressure creates the power required to drive the motor [3]. The motors range in size from small (fitting rods around half of an inch) to large enough to run heavy industrial equipment. Northern Tool and Equipment makes a small hydraulic motor with up to 4,000 psi as seen in figure 3.

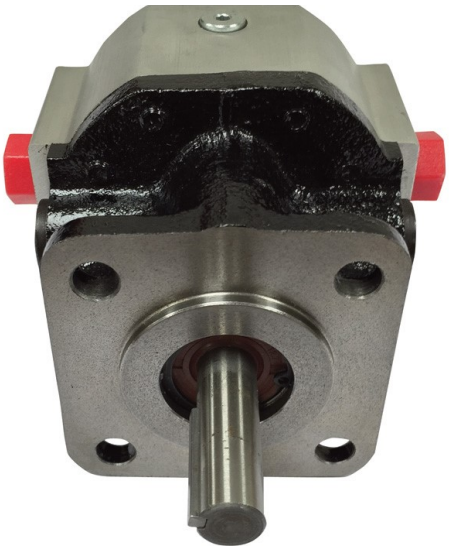

Figure 3: Small hydraulic motor. Image credit: Northern Tool and Equipment.
https://dta.eu/hydraulics/hydraulic-motors

2.1.2 Hydraulic Cylinders

Hydraulic cylinders are also very popular. Instead of the hydraulic energy being converted into rotating mechanical energy [3], it is used to linearly move a piston up and down with extreme force.

2.1.3 Pneumatic Cylinders

Pneumatic actuators work very similarly to hydraulic actuators with the main difference being the type of fluid that is converting the mechanical energy. In the case of the pneumatic actuator, the fluid is compressed air. Figure 4 shows an example of a pneumatic cylinder that we purchased to understand how it works. This actuator in figure 4 is double action meaning that there are two places where the air compressor would connect to the cylinder. The connection on the bottom of the cylinder pushes the pin up, while the connection on the top forces the pin back down.

Figure 4: Small dual action pneumatic cylinder purchased from Marlin P. Jones http://www.mpja.com.

2.1.4 Piezo-Electric Motors

Piezo-electric motors were another actuation option. Piezo-electricity is the electric charge emitted from a piezoelectric crystal when it is deformed in some way. Figure 5 shows multiple ways of how piezoelectric crystals can work. If you manually contort the crystals, they will induce a voltage on their ends in respect to whether the crystal was elongated or compressed (generator action) [4]. On the other hand, if you apply a voltage to the ends of the crystal, the crystal will elongate or compress depending on the polarity of the voltage applied (motor action) [4]. We were interested in using the crystals as a motor to variably move our surface up and down.

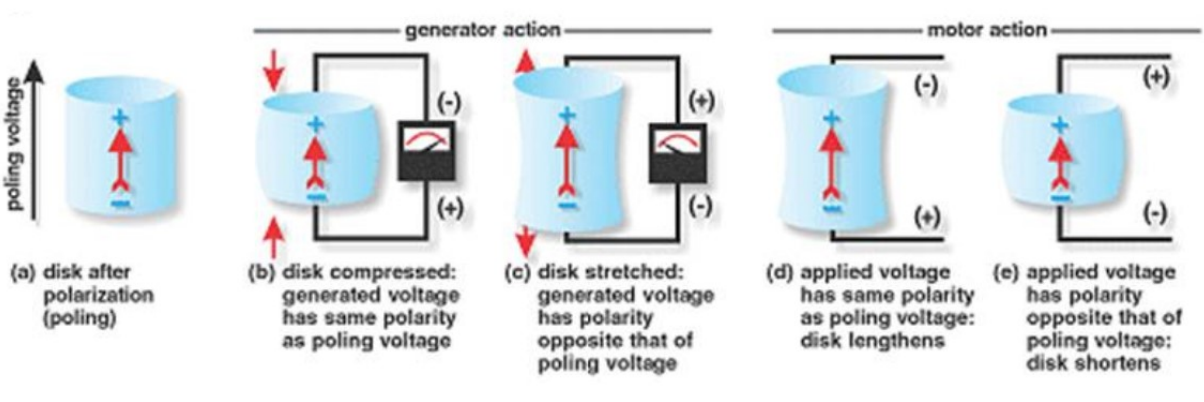

Figure 5:Diagram explaining both generator action and motor action with piezoelectric crystals. Image Credit: Advances in Ceramics – Electric and Magnetic Ceramics

2.1.5 DC Motors

Probably the most common method we could find for actuation was the use of a simple DC motor. A DC motor runs on direct current which allows the option to be powered by batteries. A DC motor converts electrical power from the battery to a mechanical force.
The DC motor has two main components, the stator and armature. The stator is the permanent magnet and the armature is coiled wire within that magnetic field of the stator. The length of the wire, current through the wire, and strength of magnetic field are what change the speed of the motor [5]. Figure 6 shows a 12V, 5 rpm motor sold by Marlin P. Jones and Associates.

Figure 6: Common pin art toy. Image credit: Marlin P. Jones. http://www.mpja.com

2.1.6 Stepper Motors

DC Stepper motors vary from DC motors in several ways. Rather than run on direct current, the stepper motor runs on pulses. Instead of constant rotation like a DC motor, these pulses allow the stepper motor to move in precise angular increments [6]. This makes it much easier to control than a DC motor and therefore better for control system applications.

2.1.7 3 – Axis Robot

Probably the most interesting way to actuate an array of pins that we found was via the use of a 3 - axis robot. A patent made by Jacques Berteau shown in Figure 7 describes the implementation of the 3 – axis robot [7].

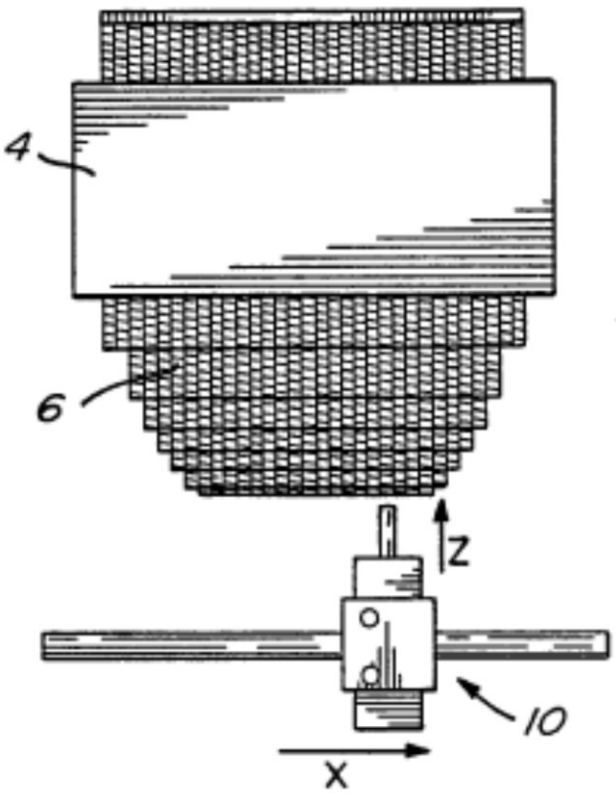

Figure 7: Common pin art toy. Image credit: Variable – Shape Mold

An array of threaded rods hang above the robotic motor. The user adjusts the motor to the desired rod either manually or with a computer program. Once the motor is under the desired rod, the motor translates upward and clamps onto the desired rod. The motor then rotates clockwise or counter-clockwise depending if the user wants the rods to translate up or down [7]. The rods act as each other's brakes by being threaded together.

2.2 BRAKES

2.2.1 Pressure

Hydraulic and pneumatic actuators are unique compared to the other types of actuators in that they also act as their own brake. The pressure pushing each individual pin in the array up can also control how far down each pin drops when someone is standing on the surface. In theory, by increasing or decreasing the pressure in each actuator one can variably adjust the height of each pin.

2.2.2 Closely Packed Pins

The thought of packing the pins so close together that they acted as their own brake was an early approach. To pack the pins close together and use friction to keep them in place, we would need as much surface area as possible of each pin touching each other. We also needed

each pin to touch as many other pins as possible. We determined that a hexagon shaped pin would be the best. Figure 8 provides a representation of our thoughts. However, it doesn't matter what shape the pins are if there is nothing forcing them against each other. An option to keep the pins packed closely together is to arrange them in an enclosed area. Figure 9 is an image from a patent from 1971 by David H. Humphrey and explains the idea of using some sort of clamp to pack the pins together. This would allow us to remove a wall when we needed to add or remove pins and clamp it back on when ready to reconfigure the surface [8].

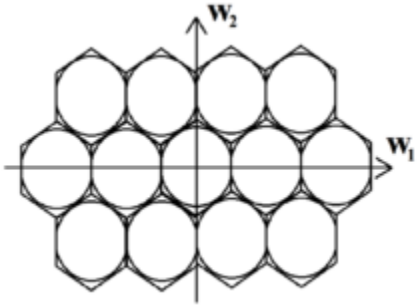

Figure 8: Closely packed hexagons share more surface area than circles. Image credit: Google Images

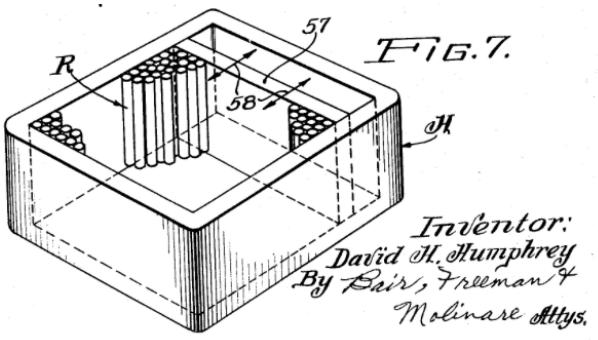

Figure 9: Adding an adjustable wall allows for pins to be compressed to any specifications. Image credit: Mold Forming Device

2.2.3 Threaded Rods

Another early idea we considered was using threaded rods instead of pins hoping that the threads could act as their own brake. For example, if we were to pack the rods close enough together that they were interlocked between threads, we thought that might be enough holding force to contain the weight of human being. Instead of the rods touching, another option for using the threads as brakes would be to simply utilize the connection between the thread of the rods and the thread of a metal base plate. Figure 10 shows an early collar we manufactured to demonstrate threaded rod placement in a metal plate.

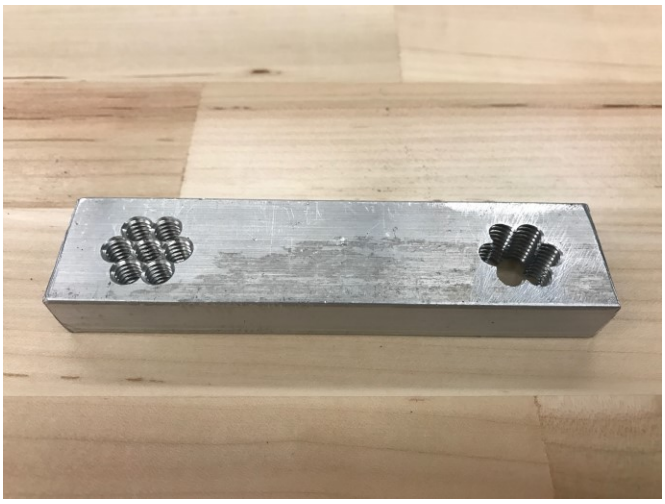

Figure 10: Holes on the left utilize the braking between rod and plate. Holes on the right utilize the braking between rods.

If the rods were threaded through a surface such as a metal plate, there could be enough stopping power between the rod and the plate to hold a considerable amount of weight without stripping the rod. This would depend on the geometry of the rods and threads.

2.2.4 Solder

Probably the most interesting brake concept we found was braking via the use of solder. A design by Benjamin J. Peters at MIT utilized low temperature melt solder and resistors to create brakes for each pin in a similar attempt at a reconfigurable surface [9]. A diagram he made to explain how this braking process works can be seen in Figure 11.

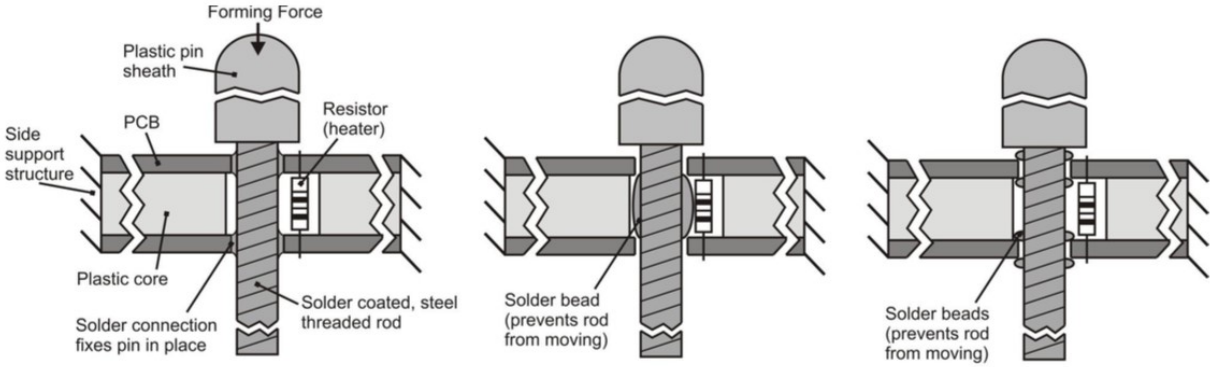

Figure 11: Solder braking method. When solder is heated, the rods translate up and down. When no heat is applied, the solder either connects to the PCB or clumps, causing a brake. Image credit: Design and Fabrication of a Digitally Reconfigurable Surface. http://www.bnpeters.com/digitally-reconfigurable-mold.html

2.3 CONTROLS

The early pin type tools were controlled manually. The first known reconfigurable pin type tool used shims to control the position of pins before screws and threaded pins were used to control position. With the invention of computers and computer numeric controls, the position of

the pins in different types of tools and surfaces could be controlled with computer programming [10]. These programs can be realized by tools such as microcontrollers, FPGAs, and building one's own digital circuit.

3. DESIGN PROCESS

3.1 Research Meeting

Because our research resulted in multiple options for both actuating and braking the pins, we had to down select our options and determine what we thought we should spend our time developing. We began by holding a meeting between us, our professor Dr. Stephen Weis, the two machinists from our machine shop David Yale and Mark Roegels, and an orthotics specialist Alan Hood. We presented our research and were able to get insight into how a finished design would ideally look. Our orthotics specialist Alan Hood said that ideally, the end design should have 9 pins per square inch as well as the option to move pins both in groups and individually. Our end design would also have to take speed into account, because if someone was standing on the pins we would want the pins to translate upwards quick enough so they wouldn't be standing on them for long amounts of time but also slowly enough that we could keep the movement very precise.

A discussion of controls took place at this meeting. It was clear that everybody involved wanted a digitally controlled device with a user-friendly interface. There were two main options to achieve this goal. The first was to design our own digital circuit and either create our own software or a manual switch or button panel as the control interface. The other option would be to use a microcontroller and use that controller to implement a program and use either a laptop or tablet as the interface.

To best convey our ideas, we decided to move forward with developing a proof-of-concept that would be able to showcase the design of a reconfigurable surface tailored for orthotic posture correction. We wanted our prototype to show that a digitally reconfigurable surface is achievable, which meant focusing on the mechanics of the design such as developing working actuation and braking methods as well as the ability to control each pin digitally with precision.

3.2 Preliminary Design

The braking and control methods used would have to be compatible with our actuation method. The first concept ruled out for actuation of our pins was the 3 – axis robotic motor. While interesting, we deemed it too complex for our design. We also thought it would be too slow and would limit us to only moving a single pin at a time. Piezo-electric motors were also quickly ruled out. We like the idea that they were small and could allow us to pack the pins extremely close together, but we determined that they would not be powerful enough to move pins while a person was standing on them. We thought hydraulic actuators were too complex for us to develop a working proof of concept in a reasonable amount of time so we decided not to move forward with that idea even though we thought it could be possibly used in the future for our application. Pneumatic actuators, however, intrigued us. It would be one of the most powerful actuators, given that even small pneumatic cylinders can provide up to 200 psi. Spread that out over a large surface area with numerous actuators and it would be more than enough pressure to hold up a human being. Our biggest question with it was accuracy. One of the most

important aspects of our design is that it remains completely stationary when it is braked. We were afraid that the pressurized air would not be stable enough to keep the person completely still on the surface. This is more of an issue of braking rather than actuation, so if we had more time and wanted to consider ways to better brake pneumatically actuated pins there might be a good solution. However, for the sake of time and resources, we decided to consider other alternatives.

The idea of using DC motors to translate the pins was probably the simplest we found and this made it desirable. DC motors can be inexpensive, they are powerful, and we can purchase them specified with virtually any rotational speed we desired. They run on batteries or a compact power supply which are both inexpensive and easily accessible and can be small enough to be placed in arrays that would allow us to complete our proof of concept. They are what we are most familiar with and we predicted that they would require less time troubleshooting compared to the other actuation methods. Realizing that if we decided to move forward with DC motors, we would most likely be working with threaded rods rather than simple pins. This is because the DC motors rotate rather than translate upwards.

Since we decided to go with the DC motors as our actuators we were left with few options for our braking. Fluid pressure would not be an option because we weren't using hydraulic or pneumatic actuators. The solder braking method also seemed too complex for our proof of concept. Once we can develop our proof of concept and show that a reconfigurable surface for orthotic posture correction was possible, it could be design idea worthy of further development. Because we decided on using threaded rods, we decided for our proof of concept that utilizing the connection between the threads of the rods and a metal base plate would be enough to brake our design. A simple diagram explaining our preliminary design idea can be seen in figure 12.

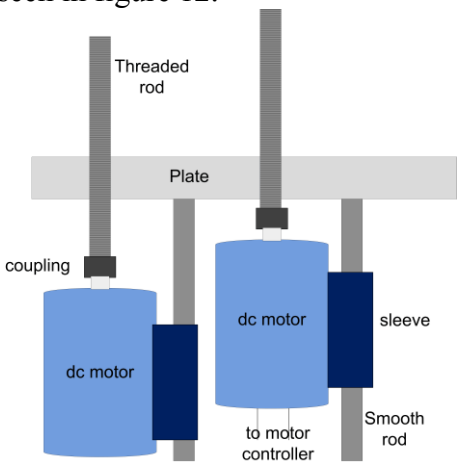

Figure 12: Preliminary Design Drawing: As the motors rotate, they translate up and down with the threaded rods.

This side view diagram shows both our actuation and braking methods in practice. The DC motors are connected to the threaded rods by a coupler. Figure 13 shows the coupler we purchased from Marlin P. Jones and how it connects the motors to the threaded rods.

Figure 13: A coupler connects the head of the DC motor to the threaded rod.

To avoid having the motors and rods spin in place rather than translate upwards, we must add in an anchor to the motors. The sleeve in figure 12 is connected to each DC motor and keeps it from spinning in place. The sleeve, wrapped around a smooth rod allows the motor and threaded rod to translate up or down (depending on rod rotation direction) and effectively create a reconfigurable surface made of threaded rods.

A diagram of the control system established in the preliminary design can be seen below in Figure 14. An Arduino Uno microcontroller was used to implement a control program. Arduino's C-based software would allow us to write a program and easily upload it to the microcontroller. The microcontroller uses compatible motor driver circuitry, referred to as "motor shields", to drive the DC motors.

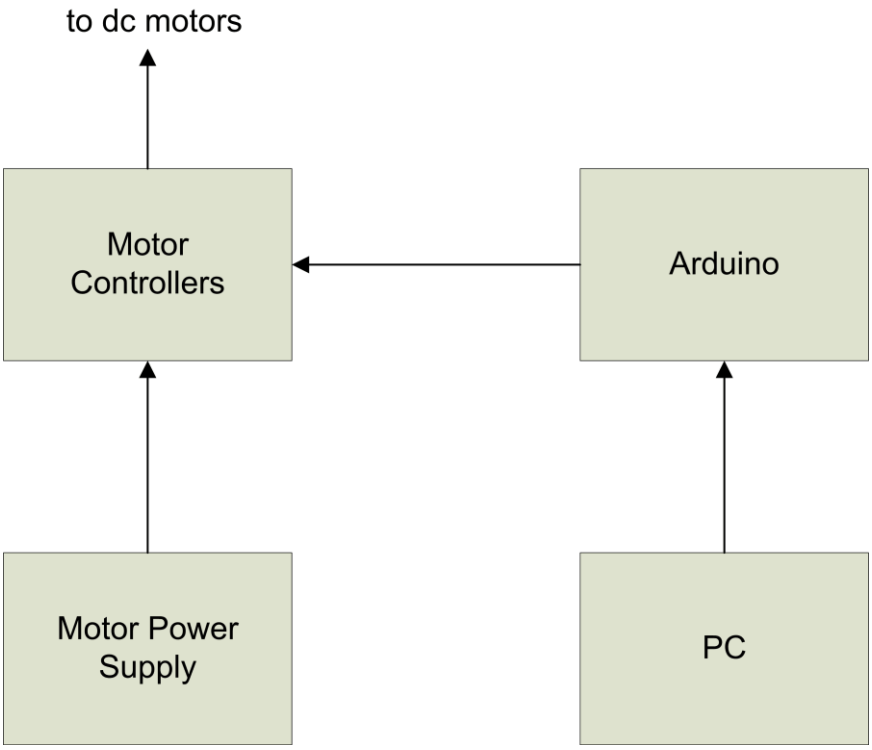

Figure 14: Preliminary design flow chart. A personal computer sends commands to the Arduino and its connected motor controllers. These motor controllers translate the commands sent from the PC and controls the individual DC motors.

3.3 Final Device Design

Taking it one step further our proof-of-concept needed to be tailored to an individual standing on the surface. It would be difficult and uncomfortable for a person to stand on thin threaded rods so we added a “shaft” to encase the rods. This was made from plastic hexagonal shafts that were threaded using a tap and placed over the threaded rods (figure 15).

Figure 15: Plastic hexagonal shafts are tapped and screwed onto the threaded rods.

To allow for these plastic hexagonal shafts to translate upwards we had to slightly alter our preliminary design. We took the metal plate and bored holes in it so that each motor would be fixed in place (Figure 16).

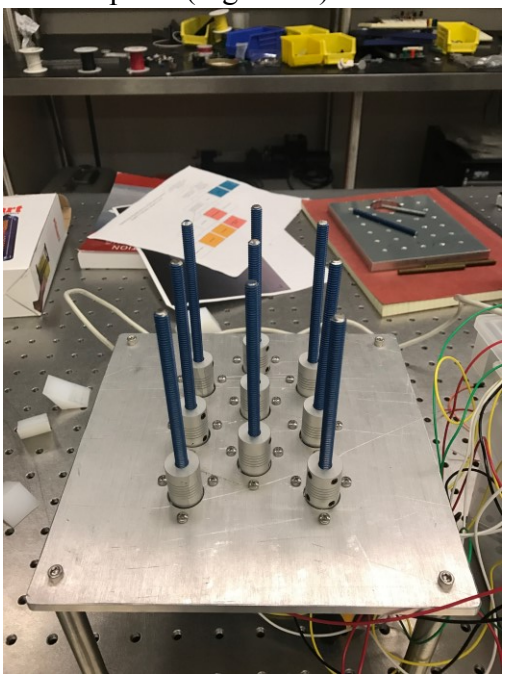

Figure 16: Each motor is hanging beneath the metal plate, held in place by three screws. The rods and couplers protrude through the plate.

After placing each hexagonal shaft on each threaded rod, we packed them together tightly using industrial bars made by 80/20 Inc. that we had in our lab. This material allowed us to create an adjustable square “collar” around the group of hexagonal shafts, thus keeping them tightly packed. Figure 17 shows the 80/20 collar holding the hexagonal shafts in place.

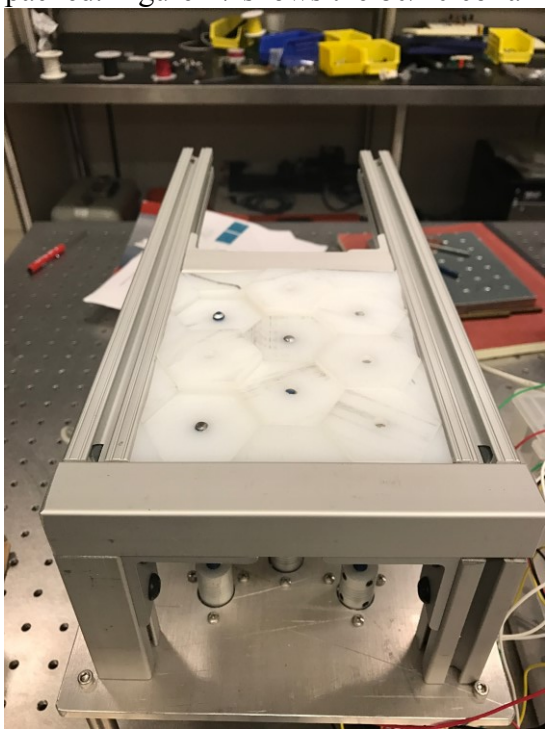

Figure 17: Collar made of 80/20 material holding the hexagonal shafts in place. The back side of the 80/20 collar is adjustable.

Our final design consisted of an array of nine 12V DC motors that run at 5 rpm. This is very slow and allows for a very precise movement of the surface. The hexagonal shafts are 1 5/8 inches from the flat to flat. The motors are spaced 1.63 inches from center to center vertically. The middle column of motors was staggered by .8125 inches from center of the corresponding motors in each column. The horizontal spacing is 1.415 inches center to center. This configuration was chosen so each hexagonal shaft would be just touching each other. We chose this many motors because we viewed it as sufficient enough to demonstrate the proof of concept of our reconfigurable surface. The ¼-20 threaded rods are 3.66 inches long. Each hexagonal shaft is 1 inch long. We chose this length because we wanted each shaft to be long enough to translate 1 inch up and down.

3.4 Final Controls

The device was controlled digitally using the system established in the preliminary design. The Arduino Uno compatible motor drivers(referred to as shields by the Arduino community) used were v2 Motor Shields from Adafruit. One Adafruit v2 Motor Shield can control four DC motors at once. Stacking headers are available to stack these motor shields and used up to 32 shields at once. Since our design consists of nine DC motors, we stacked three shields on top of the microcontroller. The bottom two shields control four motors each, and the

top shield controlled the final motor. Figure 18 below shows the microcontroller and motor shield circuit.

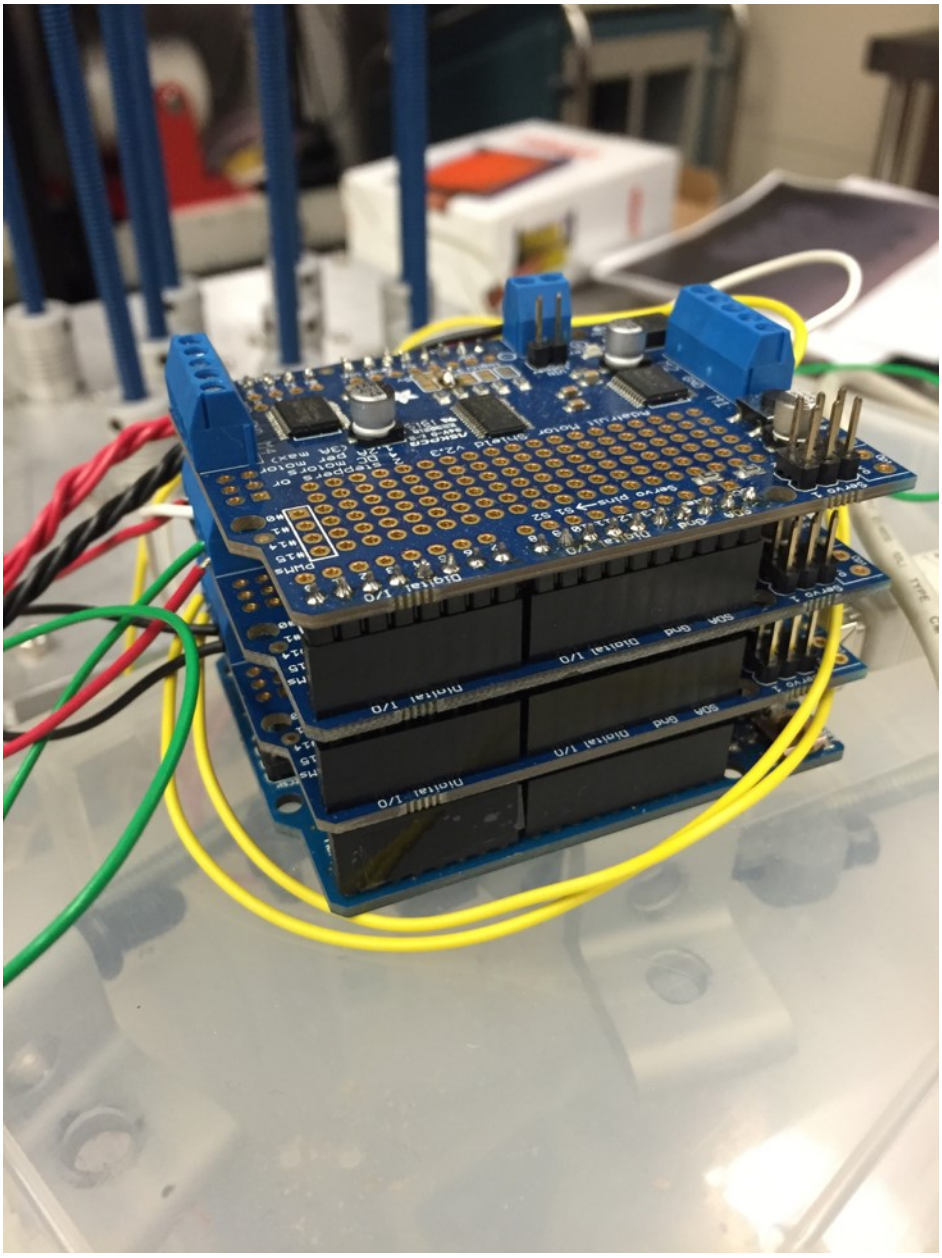

Figure 18: Motor Shield and Microcontroller Block

Arduino's program software was used in conjunction with Adafruit programs to control the motors. Each motor shield has address jumpers on the lower edge of the board. The I2C base address is 0x60, which has no address jumpers. An I2C address of 0x70 calls all the motors. The motor shield in figure 19 with the address of 0x61 has farthest right of five address jumpers for a binary address of 00001. Each address jumper represents one digit of a five-digit binary number. Adding one to binary address, adds one to the base I2C address, so the address 0x61 has the binary address 00010. 0x7F is the highest address that can be used and is achieved through jumping all of the jumping addresses [11].

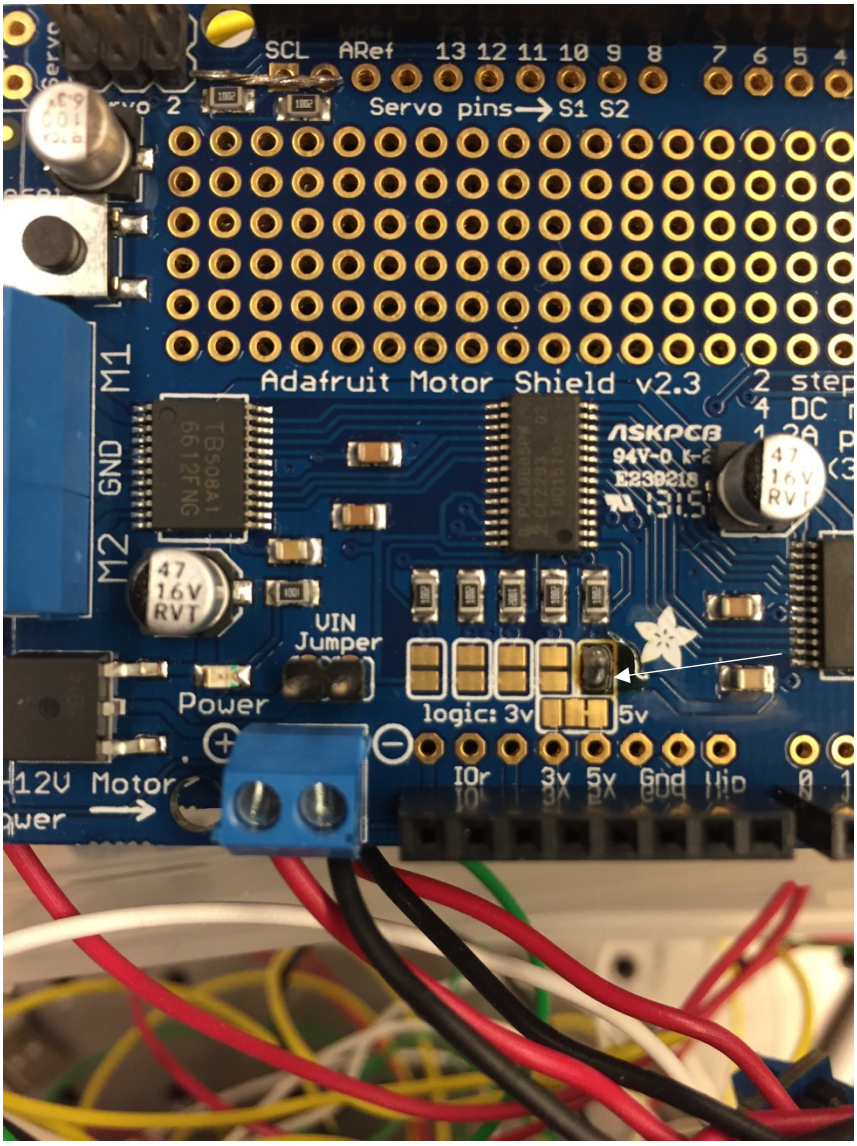

Figure 19: 0x61 I2C Address Jumpers

The program used to control the motors was adapted from an example taken from the Adafruit motor shield library called DC Motor Test. Each motor was connected to the ports of the motor shields and the program addresses each one based on the specific shield. The motors were named M1-M9. The program reads in user-entered characters from the keyboard and uses those characters as the parameters for conditional statements. The statements will run specific motors either forward or backwards if they are true. There are 19 different possible parameters and conditional statements. One for each direction of each motor and then the space bar is used to stop all of the motors. The user enters one character at a time and presses enter to send the character to the program. Table 1 shows the different characters for each motor and direction. The code for the entire program can be seen in the Appendix.

Motor	Up	Down	Motor	Up	Down	Motor	Up	Down
M1	q	a	M4	r	f	M7	u	j
M2	w	s	M5	t	g	M8	i	k
M3	e	d	M6	y	h	M9	o	l

Table 1: Characters used to Control Each Motor

4. FUTURE

4.1 Expanding Size and Precision

The current design is a prototype to show proof of concept. It does not have the size nor strength to allow for a human to stand on the surface. The spacing of the pins does not provide precise enough points to correctly study the shape of a foot. Both problems need further research, but we foresee expanding the size to provide problems than improving precisions.

To allow a human to stand on the surface, two factors must be taken into account. The amount of space taken up by the feet and width of stance of the human is the first factor. This problem has a simple solution. Most pin arrays can be designed to have outer dimensions large enough for any size human to have space to stand on. The other factor is the weight of the patient. The weight will be distributed among the different pins. The spacing and material of the pins must be able to hold the weight of a typical human.

Improving the precision will require a lot of further research. With our current design, we are limited by the size of the motors. In order to provide a closely packed pin matrix, a different actuation design and/or method must be used. Vertical staggering of the DC motors to shorten the space in between motors is one option, but the space is still confined by the size of the motors. Using a robot arm to move one pin at a time is an option, but it will sacrifice speed and make the device difficult to use. Flexible drive shafts have been looked at as an option, but they are complex to implement, have a high cost, and can easily be damaged.

Another part of the precision is the rate that the pins move up and down. With threaded pins, like the ones we use now, smaller thread spacing will allow for more precise positions, but it will slow down the rate at which the pins move. For the smaller thread space and a robot arm actuation method, speed would be sacrificed for precision.

4.2 Improving Control

With our current system, we are limited to the use of ninety-five printable characters as parameters for conditional statements. This means one could only control both directions of forty-seven different pins. Another limitation of the current system is that only one pin at a time can be controlled. A possible solution to these problems would be to design our own digital controller. This would also allow us to build a more user-friendly interface.

It is difficult to use any other device other than a computer with the Arduino microcontroller. The interface we were able to build was crude. There was no visual element to be able to see which pins are being controlled as one uses it. A switch or button panel could be created to control the pins. Each button or switch would be in same relative position of its corresponding pin. Software could be developed on a tablet and function the same way the button panel would.

4.3 Capturing Data

A device to help Orthotists create customized orthotic inserts will not be able to achieve its purpose without capturing data. Most devices now use a 2D image based on the different pressure exerted by the feet to extrapolate a 3D image [10]. Further research into capturing a direct 3D image is important to the future of this device. Two main ideas have been discussed. One uses the relative movement of the pins to capture data and the other directly captures a 3D image.

It would be possible to start with an imprint of the foot either in a foam or by also spring loading the pins to create a good starting position for the foot. If one knows the speed of rotation of threaded pins and the spacing of the threads, they can use that to determine the amount a pin has moved relative to its starting position. This data can be used to build an insert.

Devices such as the, Xbox Kinect, have been researched for the use of 3D imaging. They could be used to capture the image of the final position of the pins to create a 3D map of what the inserts should look like. They can also map the feet and the starting position of the feet to help establish a starting position of the feet [1].

5. CONCLUSION

Overall, we were able to develop a proof of concept prototype that allowed for a user to variably control an array of nine motors. We presented this design in the TCU College of Science and Engineering (CSE) Student Research Symposium (SRS) and were pleased with the responses we received. We also presented the design to our orthotics specialist Alan Hood and his feedback was reassuring. He likes the design and believes that the proof of concept, if developed into a full scale prototype, can a valuable tool with current orthotics development. In order for the design to be developed into a full scale prototype more research and development would have to occur over the coming years from other students as well as implementation and continued development of Steven Culver's research on 3-D image capture [1].

6. References

[1] S. Culver, “Use of An Xbox Kinect as a 3D Scanner for the Manufacturing of Custom Orthotic Insole.”

[2] D. Leithinger, S. Follmer, A. Olwal, A. Hogge and H. Ishii, "inFORM", *Tangible.media.mit.edu*, 2017. [Online]. Available: http://tangible.media.mit.edu/project/inform/. [Accessed: Sep- 2015].

[3] "Hydraulic Motors | Parts and Components | DTA Hydraulics", *Dta.eu*, 2017. [Online]. Available: https://dta.eu/hydraulics/hydraulic-motors/. [Accessed: Feb- 2017].

[4] I. Patel, "Ceramic Based Intelligent Piezoelectric Energy Harvesting Device", *Advances in Ceramics - Electric and Magnetic Ceramics, Bioceramics, Ceramics and Environment*, 2011.

[5] S. J. Chapman, *Electric Machinery Fundamentals*, 5th ed., McGraw-Hill, pp. 650-651 (2012).

[6] V. Athani, *Stepper motors*, 1st ed. New Delhi: New Age International (P) Ltd., Publishers, 2008, pp. 1-2.

[7] J. Berteau, "Variable - Shape mold", 5,330,343, 1994.

[8] D. Humphrey, "Mold Forming Device", 3596869 A, 1971.

[9] B. Peters, "Design and Fabrication of a Digitally Reconfigurable Surface", 2011.

[10] C. Munro and D. Walczyk, "Reconfigurable Pin-Type Tooling: A Survey of Prior Art and Reduction to Practice", *Journal of Manufacturing Science and Engineering*, vol. 129, no. 3, pp. 551-565, 2007.

[11] “Adafruit Motor Shield V2 for Arduino,” *Stacking Shields | Adafruit Motor Shield V2 for Arduino | Adafruit Learning System*. [Online]. Available: https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/stacking-shields. [Accessed: 10-Mar-2017].

7. Appendix

7.1 Motor Control Code

```
#include <Wire.h>
#include <Adafruit_MotorShield.h> //Adafruit Motor Shield Library
#include "utility/Adafruit_MS_PWMServoDriver.h"//this utility and the above

// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMSbot = Adafruit_MotorShield(); //bottom shield with no address
//jumpers jumped
Adafruit_MotorShield AFMSmid = Adafruit_MotorShield(0x61); //middle shield with the far
//right address jumpers jumped
Adafruit_MotorShield AFMStop = Adafruit_MotorShield(0x62); //top shield with second from
//the right address jumpers jumped
// Or, create it with a different I2C address (say for stacking)
// Adafruit_MotorShield AFMS = Adafruit_MotorShield(0x61);

// Select which 'port' M1, M2, M3 or M4. In this case, M1
Adafruit_DCMotor *M1 = AFMSbot.getMotor(1);
Adafruit_DCMotor *M2 = AFMSbot.getMotor(2);
Adafruit_DCMotor *M3 = AFMSbot.getMotor(3);
Adafruit_DCMotor *M4 = AFMSbot.getMotor(4);
Adafruit_DCMotor *M5 = AFMSmid.getMotor(1);
Adafruit_DCMotor *M6 = AFMSmid.getMotor(2);
Adafruit_DCMotor *M7 = AFMSmid.getMotor(3);
Adafruit_DCMotor *M8 = AFMSmid.getMotor(4);
Adafruit_DCMotor *M9 = AFMStop.getMotor(1);


void setup() {
  Serial.begin(9600);           // set up Serial library at 9600 bps

  AFMSbot.begin();
  AFMSmid.begin();
  AFMStop.begin();// create with the default frequency 1.6KHz
  //AFMS.begin(1000);  // OR with a different frequency, say 1KHz

  // Set the speed to start, from 0 (off) to 255 (max speed)
  M1->setSpeed(255);
  M1->run(FORWARD);
  // turn on motor
  M1->run(RELEASE);
```

```
  M2->setSpeed(255);
  M2->run(FORWARD);
  M2->run(RELEASE);

  M3->setSpeed(255);
  M3->run(FORWARD);
  M3->run(RELEASE);

  M4->setSpeed(255);
  M4->run(FORWARD);
  M4->run(RELEASE);

  M5->setSpeed(255);
  M5->run(FORWARD);
  M5->run(RELEASE);

  M6->setSpeed(255);
  M6->run(FORWARD);
  M6->run(RELEASE);

  M7->setSpeed(255);
  M7->run(FORWARD);
  M7->run(RELEASE);

  M8->setSpeed(255);
  M8->run(FORWARD);
  M8->run(RELEASE);

  M9->setSpeed(255);
  M9->run(FORWARD);
  M9->run(RELEASE);
}

void loop() {

  //Serial.println("Motor 1 press 'q' for up and 'a' for down");
  //Serial.println("Motor 2 press 'w' for up and 's' for down");
  Serial.println("Press ENTER to send command");
  Serial.println("Press SPACE to stop");
  Serial.println(); //these three lines allow for user inputs

  int speed;
  int data;

  char c;
```

```
while(true)
{
 while (Serial.available() > 0) // this while loop allows for a character to be read from the user
 {
  c = Serial.read();
 }


 if (c == 'a'){
  M1->run(FORWARD);
  M2->run(RELEASE);
  M3->run(RELEASE);
  M4->run(RELEASE);
  M5->run(RELEASE);
  M6->run(RELEASE);
  M7->run(RELEASE);
  M8->run(RELEASE);
  M9->run(RELEASE);
 }

 else if (c == 'q'){
  M1->run(BACKWARD);
  M2->run(RELEASE);
  M3->run(RELEASE);
  M4->run(RELEASE);
  M5->run(RELEASE);
  M6->run(RELEASE);
  M7->run(RELEASE);
  M8->run(RELEASE);
  M9->run(RELEASE);
 }

 else if (c == 's'){
  M1->run(RELEASE);
  M2->run(FORWARD);
  M3->run(RELEASE);
  M4->run(RELEASE);
  M5->run(RELEASE);
  M6->run(RELEASE);
  M7->run(RELEASE);
  M8->run(RELEASE);
  M9->run(RELEASE);
 }

 else if (c == 'w'){
  M1->run(RELEASE);
```

```
  M2->run(BACKWARD);
  M3->run(RELEASE);
  M4->run(RELEASE);
  M5->run(RELEASE);
  M6->run(RELEASE);
  M7->run(RELEASE);
  M8->run(RELEASE);
  M9->run(RELEASE);
 }

 else if (c == 'd'){
  M1->run(RELEASE);
  M2->run(RELEASE);
  M3->run(FORWARD);
  M4->run(RELEASE);
  M5->run(RELEASE);
  M6->run(RELEASE);
  M7->run(RELEASE);
  M8->run(RELEASE);
  M9->run(RELEASE);
 }

 else if (c == 'e'){
  M1->run(RELEASE);
  M2->run(RELEASE);
  M3->run(BACKWARD);
  M4->run(RELEASE);
  M5->run(RELEASE);
  M6->run(RELEASE);
  M7->run(RELEASE);
  M8->run(RELEASE);
  M9->run(RELEASE);
}

else if (c == 'f'){
  M1->run(RELEASE);
  M2->run(RELEASE);
  M3->run(RELEASE);
  M4->run(FORWARD);
  M5->run(RELEASE);
  M6->run(RELEASE);
  M7->run(RELEASE);
  M8->run(RELEASE);
  M9->run(RELEASE);
}
```

```
else if (c == 'r'){
   M1->run(RELEASE);
   M2->run(RELEASE);
   M3->run(RELEASE);
   M4->run(BACKWARD);
   M5->run(RELEASE);
   M6->run(RELEASE);
   M7->run(RELEASE);
   M8->run(RELEASE);
   M9->run(RELEASE);
}

else if (c == 'g'){
   M1->run(RELEASE);
   M2->run(RELEASE);
   M3->run(RELEASE);
   M4->run(RELEASE);
   M5->run(FORWARD);
   M6->run(RELEASE);
   M7->run(RELEASE);
   M8->run(RELEASE);
   M9->run(RELEASE);
}

else if (c == 't'){
   M1->run(RELEASE);
   M2->run(RELEASE);
   M3->run(RELEASE);
   M4->run(RELEASE);
   M5->run(BACKWARD);
   M6->run(RELEASE);
   M7->run(RELEASE);
   M8->run(RELEASE);
   M9->run(RELEASE);
}

else if (c == 'h'){
   M1->run(RELEASE);
   M2->run(RELEASE);
   M3->run(RELEASE);
   M4->run(RELEASE);
   M5->run(RELEASE);
   M6->run(FORWARD);
   M7->run(RELEASE);
   M8->run(RELEASE);
   M9->run(RELEASE);
```

```
}

else if (c == 'y'){
   M1->run(RELEASE);
   M2->run(RELEASE);
   M3->run(RELEASE);
   M4->run(RELEASE);
   M5->run(RELEASE);
   M6->run(BACKWARD);
   M7->run(RELEASE);
   M8->run(RELEASE);
   M9->run(RELEASE);
}

else if (c == 'j'){
   M1->run(RELEASE);
   M2->run(RELEASE);
   M3->run(RELEASE);
   M4->run(RELEASE);
   M5->run(RELEASE);
   M6->run(RELEASE);
   M7->run(FORWARD);
   M8->run(RELEASE);
   M9->run(RELEASE);
}

else if (c == 'u'){
   M1->run(RELEASE);
   M2->run(RELEASE);
   M3->run(RELEASE);
   M4->run(RELEASE);
   M5->run(RELEASE);
   M6->run(RELEASE);
   M7->run(BACKWARD);
   M8->run(RELEASE);
   M9->run(RELEASE);
}

else if (c == 'k'){
   M1->run(RELEASE);
   M2->run(RELEASE);
   M3->run(RELEASE);
   M4->run(RELEASE);
   M5->run(RELEASE);
   M6->run(RELEASE);
   M7->run(RELEASE);
```

```
  M8->run(FORWARD);
  M9->run(RELEASE);
}

else if (c == 'i'){
  M1->run(RELEASE);
  M2->run(RELEASE);
  M3->run(RELEASE);
  M4->run(RELEASE);
  M5->run(RELEASE);
  M6->run(RELEASE);
  M7->run(RELEASE);
  M8->run(BACKWARD);
  M9->run(RELEASE);
}

else if (c == 'l'){
  M1->run(RELEASE);
  M2->run(RELEASE);
  M3->run(RELEASE);
  M4->run(RELEASE);
  M5->run(RELEASE);
  M6->run(RELEASE);
  M7->run(RELEASE);
  M8->run(RELEASE);
  M9->run(FORWARD);
}

else if (c == 'o'){
  M1->run(RELEASE);
  M2->run(RELEASE);
  M3->run(RELEASE);
  M4->run(RELEASE);
  M5->run(RELEASE);
  M6->run(RELEASE);
  M7->run(RELEASE);
  M8->run(RELEASE);
  M9->run(BACKWARD);
}

 else if (c == ' '){
  M1->run(RELEASE);
  M2->run(RELEASE);
  M3->run(RELEASE);
  M4->run(RELEASE);
  M5->run(RELEASE);
```

```
      M6->run(RELEASE);
      M7->run(RELEASE);
      M8->run(RELEASE);
      M9->run(RELEASE);
    }
  }
}
```