Clemson University
TigerPrints

All Theses | Theses

8-2015

A Scalable and Low-Cost Interactive Shape-Changing Display

Amith Mysore Vijaykumar
Clemson University

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

Recommended Citation
Vijaykumar, Amith Mysore, "A Scalable and Low-Cost Interactive Shape-Changing Display" (2015). *All Theses*. 2498.
https://tigerprints.clemson.edu/all_theses/2498

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

A Scalable and Low-Cost Interactive shape-changing display

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Amith Mysore Vijaykumar
August 2015

Accepted by:
Dr. Ian D. Walker, Committee Chair
Dr. Keith E. Green
Dr. Richard E. Groff

Abstract

Research in new display technologies has garnered great interest in the recent years. Curved screens and foldable displays have already been commercialized. However a relatively new field of research is in dynamic shape-changing or shape-shifting displays. These displays utilize the ability to change their shape dynamically as another dimension of representing visual data. These displays potentially augmented with colors, can help visualize three dimensional data such as terrains, city and building plans, and medical data. They can also be used in new ways of Human-Computer Interaction by developing user interfaces that transform physically based on the scenario.

While there is research being done on new ways of using shape displays for interaction and manipulation, not much focus has been given to the issue of cost and scalability. The general shape displays which are currently being developed have individual pixels which need to be actuated. The commercial linear actuators which are used in these displays are extremely expensive and are not meant for such purposes. This thesis presents a design for a dynamic pixel-based shape changing display which focuses on cost and scalability by using custom designed modular actuators and sensor packages.

Dedication

I dedicate this to my parents Vijayakumar and Jayashree, and my brother Rohith.

Acknowledgments

First and foremost I offer my sincerest gratitude to my academic and research advisor, Professor Dr. Ian D. Walker, who has supported me thoughout my thesis with his patience and knowledge whilst allowing me the room to work in my own way. I attribute the level of my Masters degree to his encouragement and effort and without him this thesis, too, would not have been completed or written. One simply could not wish for a better or friendlier advisor.

I would also like to thank (Professors) Dr. Keith E.Green and Dr. Richard E. Groff for accepting to be my committee members and providing me with helpful suggestions for improving the thesis.

I would also like to thank Mr. Scheen Thurmond from the Bioengineering lab for helping me get acquainted with the laser cutter system and, Ms. Esther Kaufmann and Ms. Lillian Burns for keeping up with the large number of purchases that were made during the last two years.

I would like to thank my parents and brother for instilling in me the importance of hard work, honesty, integrity and free thinking through the way they have lived their lives. Finally, I thank my girlfriend Vandita, for all her love and support during the most hectic part of my life and keeping me company even while being on the other side of the world.

Table of Contents

Title Page i

Abstract ii

Dedication iii

Acknowledgments iv

List of Tables vii

List of Figures viii

1 Introduction **1**
- 1.1 Background 4

2 Research Design and Methods **8**
- 2.1 Background on Linear Actuators 8
- 2.2 First Prototype 9
- 2.3 Second Prototype 11
- 2.4 Construction of the final prototype 11
- 2.5 Servo motor assembly 21
- 2.6 Pressure sensor 22
- 2.7 System block diagram 28
- 2.8 Assembly 35
- 2.9 Software 41
- 2.10 Materials used 42

3 Results **45**
- 3.1 Linear actuator characteristics 45
- 3.2 Pressure sensor characteristics 48
- 3.3 Power consumption 48
- 3.4 Data Visualization 49
- 3.5 Object Transfer and Manipulation 52
- 3.6 Gesture control 53
- 3.7 Cost comparison 54

4 Conclusions and Suggestions for Future Research **55**

Appendices **59**
- A Calibration data 60
- B MATLAB, Arduino and Processing Code for the system 61

References **86**

List of Tables

2.1 Comparison of properties of different mechanisms of linear actuation 8
2.2 Comparison of Springs 18
2.3 Servo motor specifications 21
2.4 Material list 44

3.1 Measurements of maximum force 46
3.2 Absolute regression coefficients for position feedback 46
3.3 Upward and downward speeds of actuators 47
3.4 Average characteristics of the linear actuator 47

1 Servo motor position values for calibration positions 60

List of Figures

1.1 MIT's shape changing display . . . 2
1.2 Low cost shape changing display . . . 3
1.3 Second generation of FEELEX [15] . . . 4
1.4 Lumen device [21] . . . 4
1.5 Relief device [16] . . . 5
1.6 inFORM by MIT Media Labs [11] . . . 5
1.7 Terrain table by Northrop Grumman [6] . . . 6
1.8 ShapeClips by Lancaster University, UK [13] . . . 6
1.9 EMERGE by Lancaster University, UK [22] . . . 7

2.1 Example of a commercially available linear actuator . . . 9
2.2 3D rendering of the first prototype . . . 10
2.3 Prototype modified with extra servo motors . . . 10
2.4 Second prototype . . . 11
2.5 Details of the actuator container wall . . . 12
2.6 Actuator container after gluing . . . 12
2.7 Details of the container cap . . . 13
2.8 Actuator container with the end cap . . . 13
2.9 Dimensional details of the actuator shaft . . . 13
2.10 Schematic of a linear potentiometer . . . 14
2.11 Actuator shaft with conductive ink painted on one side . . . 14
2.12 Actuator shaft with copper tape . . . 15
2.13 Details of the electrical joints . . . 15
2.14 Photograph of the acrylic pieces used to hold the contacts . . . 16
2.15 Sliding contact soldered to copper tape . . . 16
2.16 Sliding contacts mounted on the actuator container . . . 17
2.17 Actuator container with the contacts fully mounted . . . 17
2.18 Photograph of the spring . . . 17
2.19 Close up photograph of the fishing line attached to the actuator shaft . . . 18
2.20 Dimensional details of the pulley . . . 19
2.21 Pulley mounted to the servo motor . . . 19
2.22 Block diagram of a servo motor . . . 20
2.23 Servo motor with the potentiometer removed . . . 21
2.24 Gearbox of the servo motor . . . 22
2.25 3D rendering of the servo assembly . . . 23
2.26 Servo motor blocks . . . 24
2.27 Dimensional details of the top plate of the servo assembly . . . 25
2.28 Dimensional details of the bottom plate of the servo assembly . . . 26
2.29 Conductive foam used to protect ICs . . . 26
2.30 Dimensional details of the pressure sensor holder . . . 27
2.31 Pressure sensor inserted into the 3D printed piece . . . 27

2.32 Pressure sensor with touch plate mounted on the actuator shaft . . . 28
2.33 Block diagram of the system . . . 29
2.34 Schematics of the DAQ board . . . 30
2.35 PCB Layout of the DAQ board . . . 31
2.36 Schematics of the actuator and sensor board . . . 32
2.37 PCB layout of the actuator and sensor board . . . 33
2.38 16 channel servo controller . . . 34
2.39 Power supply . . . 34
2.40 Arduino . . . 35
2.41 Arduino communication chain . . . 35
2.42 Dimensional details of the group of four blocks . . . 36
2.43 Servo group during assembly . . . 37
2.44 Servo group after full assembly . . . 38
2.45 3D rendering of the contacts separator . . . 39
2.46 Contact separator attached to the actuators . . . 39
2.47 Pixel display with the projection system . . . 40
2.48 Microsoft kinect used to detect gestures . . . 41
2.49 Screenshot of the GUI . . . 42
2.50 Gesture detection from Kinect data . . . 43
2.51 Materials before assembly . . . 43
2.52 Materials before assembly . . . 44

3.1 Setup for measuring speed of actuators . . . 47
3.2 Plot of raw pressure sensor output with varying weights . . . 48
3.3 Plot of raw and thresholded sensor output . . . 49
3.4 Visualization of a sine wave . . . 50
3.5 Visualization of a bar graph . . . 50
3.6 Visualization of a sinc function . . . 51
3.7 Visualization of a pyramid . . . 51
3.8 Transfer of object . . . 52
3.9 Manipulation of heavy object . . . 53
3.10 Example of gesture control for quadrant 2 . . . 53
3.11 Example of gesture control for quadrant 3 . . . 54
3.12 Details of the cost comparison . . . 54

4.1 Example of 3D projection mapping [7] . . . 57
4.2 Example of NURBS modeling . . . 58
4.3 Rendering of remote diagnosis scenario . . . 58

Chapter 1

Introduction

The two dimensional screen has been ubiquitous for many years. Its found almost everywhere from the smartphones we carry in our pockets to large wall mounted displays. The whole computing world is designed based on this two dimensional visual representation of data, and it has worked wonderfully and will continue to do so for a long time. Stereo displays have also become quite common, especially in the movie industry as they simulate the sense of depth in images and videos. Other forms of data visualization such as Augmented and Virtual Reality glasses have moved out of the research labs and turned into commercial products such as the Oculus Rift [19] and Microsoft Hololens [17]. However we are still yet to fully utilize the third dimension for such purposes, at least commercially.

One of the ways that the current display technology could be extended to the third dimension is by developing dynamic shape changing displays. In recent years there has been a sharp increase in the research on such technology, the most relevant to the work in this Thesis being the inFORM [11] project by Massachusetts Institute of Technology (MIT) which is shown in Figure 1.1. It consists of an array of 30x30 linear actuators which actuate tiles up and down creating a 2.5 dimensional surface. An overhead projector is used to display images on the surface, thereby creating a dynamic shape changing display. This type of hardware opens up a lot of areas in Human-Computer interaction which were not possible before, such as being able to visualize data in 2.5D, actuating physical objects, and interacting with a tangible platform. Some of the immediate applications for interactive shape changing displays are terrain viewing, city and building planning, 3D modelling and animation.

However one of the issues with the design of such displays is the use of commercial linear

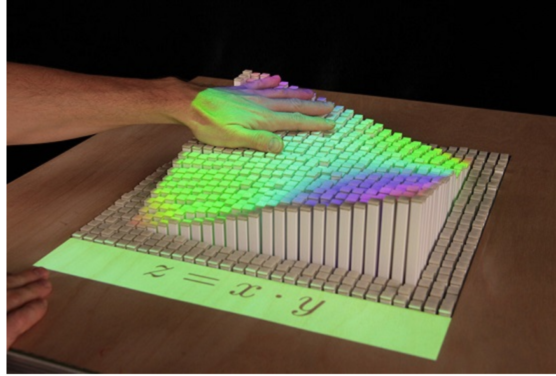


Figure 1.1: MIT's shape changing display

actuators. These actuators are generally built to produce a lot of force which goes unused in the 2.5D display application and also take up a lot of space. They are also extremely expensive, ranging anywhere from $30 to $200 for just the actuators which makes scaling the display commercially infeasible. This is one of the reasons why such displays are in limited use, even in research labs. By building custom linear actuators which are designed specifically for the purpose of actuating pixels, the cost of the whole system can be brought down significantly.

This thesis presents a novel design for a shape changing display with a focus on keeping the cost of the system low using custom built linear actuators and sensors while maintaining scalability of the design. A 16x16 shape changing pixel display was built to test the effectiveness of the actuator and sensor system and the design constraints that have to be considered while building it. The system also features custom pressure sensors to make the display tangible and an overhead projector to display images and videos on the surface. Figure 1.2 shows the system and its main components. The top of the pressure sensors also double as the display surface which is 16 inches X 16 inches with each pixel being an inch apart. There are a total of 256 pixels which are actuated by 256 servo motors through tendons. The motor assembly and the pixel array is divided into 16 blocks, each of which contains 16 motors and controls 4x4 pixels. This was done to improve the ease of assembly/disassembly and improve scalability, as more actuator blocks can be easily added to increase the size of the display. The servo motor assembly is slightly wider than the display surface at 24 inches X 24 inches and the total height of the system is 40 inches. This was due to restrictions on the maximum size of the PCB which could be fabricated at a reasonable price. If it were to be manufactured commercially, the servo motor blocks and the pixel array could be the same size.

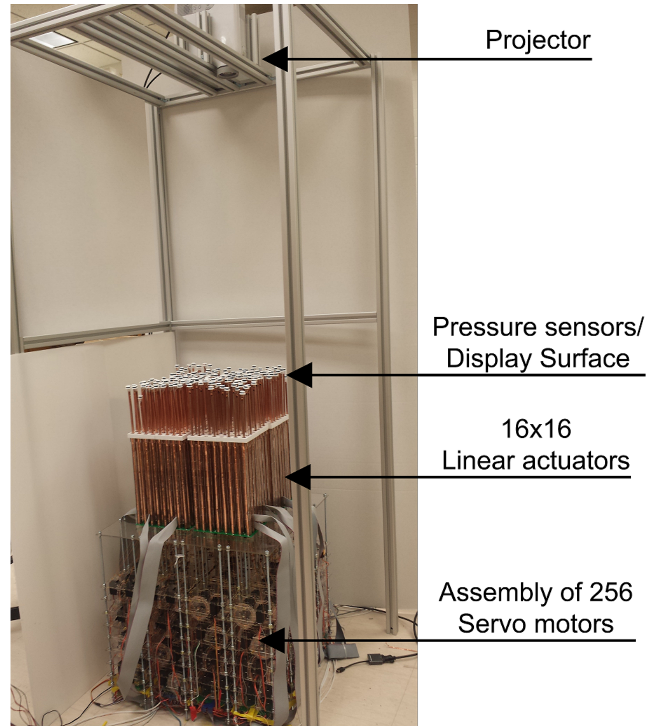


Figure 1.2: Low cost shape changing display

Over the years, different actuation mechanisms have been implemented in shape changing displays. These mechanisms were reviewed and is available in Section 1.2. Our system is inspired by some of these designs, but the designs of the linear actuators and pressure sensors are novel. The design considerations and the constraints for the new linear actuator are described in Chapter 2. This Chapter also provides the construction details of the actuator as well as the whole system describing every aspect in detail including CAD models, fabrication techniques and a list of materials that were used.

Several applications for the system were developed such as using the display for data visualization, object manipulation, interaction through a gesture interface and a basic touch input. The results of these applications are discussed in Chapter 3. The Chapter also includes details about some of the preliminary experiments done to assess the linear actuators and the pressure sensors. Chapter 4 presents the conclusions and the summary of the Thesis and also provides suggestions for future work based on the results of the system.

1.1 Background

One of the earliest dynamic shape changing display was the FEELEX developed by Iwata et al. [15], which involves 36 motorized pins that actuate the shape of a soft surface. The second generation of the device is shown in Figure 1.3. While it improved on the resolution compared to the first generation device, it involved complex mechanical linkages which could not be easily scaled. Its main application was to be used as a medical haptic device and as such is not suitable for general purpose data visualization and interaction.

Figure 1.3: Second generation of FEELEX [15]

Figure 1.4 shows the Lumen device created by Poupyrev et al. [21] It consists a 5x5 array of moveable light guides actuated through shape memory alloy (SMA) wires. SMA wires contract when current passes through them and return to the original length when the current stops flowing. Although the actuation was fast, the amount of actuation (travel) which was produced was very low which is an inherent limitation of using SMA wires for actuation.

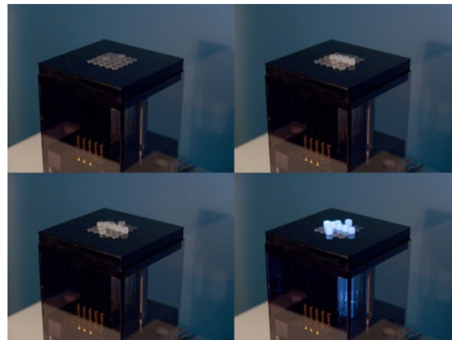


Figure 1.4: Lumen device [21]

The relief system [16] which was developed at MIT Media Labs is shown in Figure 1.5. It is a tabletop surface which is actuated by an array of 120 motorized pins built upon open-source hardware and software. This is probably the most effective shape-changing display to date in terms of cost and scalability. However, it still uses commercially available linear actuators which tend to be expensive as mentioned earlier. Since it also used DC motors, this system required motor drivers which again adds to the cost and complexity.

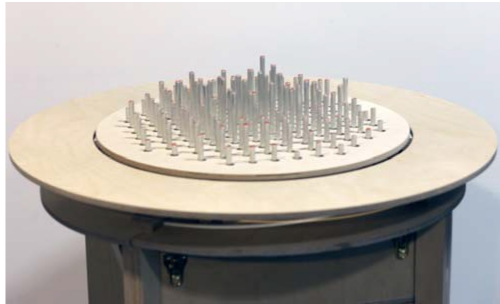


Figure 1.5: Relief device [16]

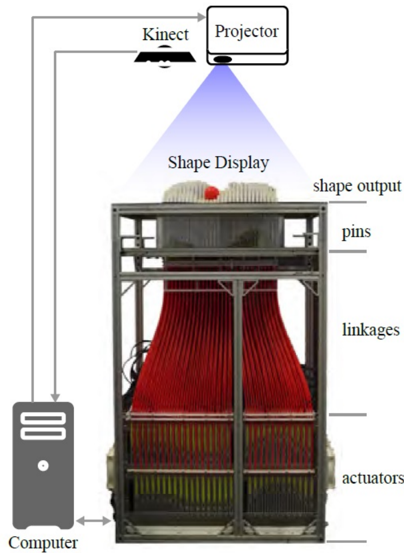


Figure 1.6: inFORM by MIT Media Labs [11]

The inFORM [11] shape changing display built by MIT Media labs is shown in Figure 1.6. It is an improvement of the relief display but uses the same actuator mechanism, the only difference

being that the pins are connected to the actuators through push-pull rods. This increases the ability to build displays with higher resolution but increases the cost and complexity per pixel.

One of the few commercially available shape changing displays is manufactured by Northrop Grumman [6] and is used in terrain viewing for military applications (see Figure 1.7). It is a fairly high resolution display created by 4600 pins with a spacing of 0.72 inches between pins. It also has a silicone skin for image projection. However, as it was built for terrain viewing, it is not a dynamic display and takes around 15 seconds to change the terrain.

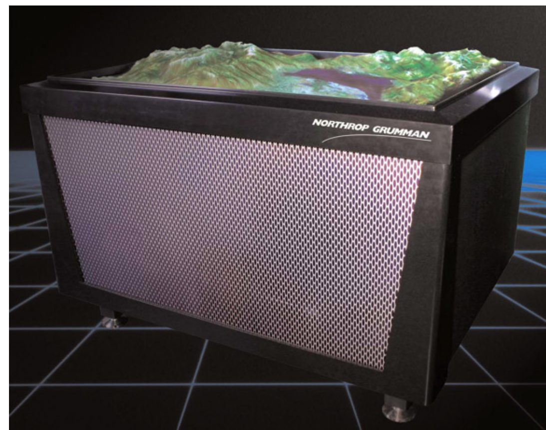


Figure 1.7: Terrain table by Northrop Grumman [6]

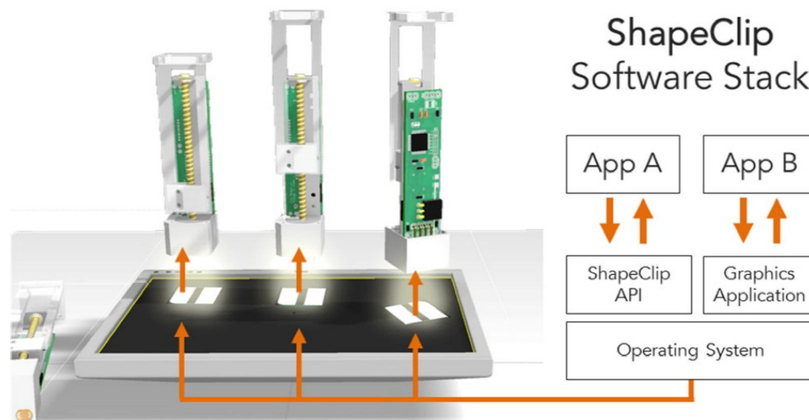


Figure 1.8: ShapeClips by Lancaster University, UK [13]

ShapeClips [13] is a very innovative prototype developed at the Lancaster Unversity, UK and is shown in Figure 1.8. It consists of modular clips which can move up and down based on

the input from an LDR (Light Dependent Resistor) at the bottom. This allows the shapeclip to be used without any programming and can be used in conjunction with any device which is capable of displaying colors. However, because of the modular and independent nature of the shapeclips, they involve a large number of components with each module which increases the cost.

Another shape changing display produced by the Lancaster University, UK is the EMERGE [22] which is shown in figure 1.9. It is very similar to inFORM [11] in terms of the actuation mechanism as well as the use of push-pull rod to control the pins. There are however, RGB LEDs embedded within the rods to illuminate them and help visualize data in a different way which does not use projection. Since the actuation mechanism is the same as inFORM [11], this display also has similar limitations on cost and scalability.

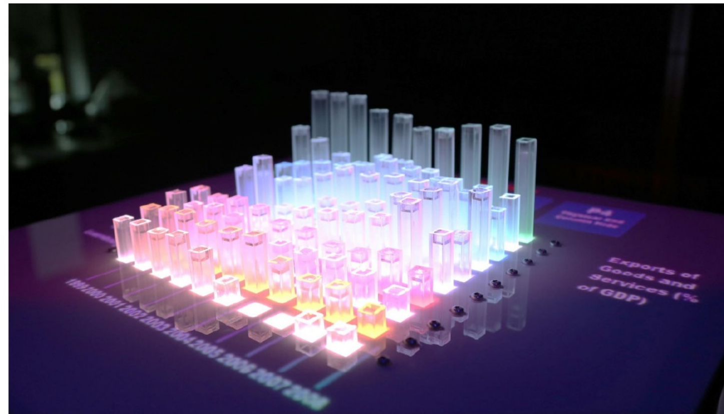


Figure 1.9: EMERGE by Lancaster University, UK [22]

The above review suggests that in order to make the shape-changing display more commercially feasible and scalable, we need a better design for the actuator mechanism. The use of commercial linear actuators in previously developed displays help in keeping the prototype complexity low but add to the cost of the systems. By designing and building custom linear actuators which meet the requirements of the system, the cost of the system can be brought down significantly while increasing scalability. The design of such a linear actuation mechanism and the actuator assembly is a key contribution of this Thesis, and is explained in detail in the next Chapter. To validate the effectiveness of the new mechanism, a 16x16 shape-changing display was built. The demonstration of the effectivenes of the system in a seriees of novel application scenarios is another key contribution of this Thesis and the results of these experiments are provided in Chapter 3.

Chapter 2

Research Design and Methods

2.1 Background on Linear Actuators

A linear actuator is an actuator that creates motion in a straight line, in contrast to the rotary motion of a conventional electric motor. Linear actuation can be achieved through several different mechanisms and energy sources. However, because of reasons which will be explained in the next section, the focus will be on electromechanical linear actuators, an example of which is shown in Figure 2.1. Even with this subset there are several different methods by which the actuation can be achieved. They can be broadly divided into three categories which are Screw type, Wheel and axle, and Cam. The screw type linear actuator operates on the principle of a simple screw. By rotating the actuator's nut, the screw shaft moves in a linear motion. The wheel and axle type produce linear motion by rotating a wheel which moves a cable, rack, chain or belt. Cam actuators work on a principle similar to a wedge, they produce linear actuation by rotating an eccentric shaped wheel which moves a shaft up and down. Table 2.1 shows the comparison of some properties of these mechanisms [10].

Mechanism	***Speed***	***Force***	***Travel***
Screw	Slow	High	Medium
Wheel and Axle	Fast	Medium	High
Cam	Medium	Medium	Low

Table 2.1: Comparison of properties of different mechanisms of linear actuation

Figure 2.1: Example of a commercially available linear actuator

Since the purpose of the linear actuator in the project is to represent the height of a pixel, the force required is considerably low. The important parameters would be speed and travel as it is desirable to have the pixel quickly change heights and also have sufficient play in the linear motion. Keeping this in mind it can be easily seen that the screw type linear actuator can be eliminated as an option. The other two options however, required more careful consideration.

2.2 First Prototype

The first option which was explored was the Cam mechanism. Even though the cam mechanism did not offer fast speeds with a single actuator, the idea was to control several shafts with a single actuator. The prototype was built out of laser cut acrylic sheet and a 3D rendering of it is shown in Figure 2.2.

The prototype consists of a shaft which has teeth and is able to move vertically up and down in a slot. An acrylic piece of 2cm width and 10cm length was connected to a servo motor to act as the Cam. The solenoid was to be used to lock the shaft in place once it was in the correct position. Although this design would have worked, it was quickly realized that it would not allow for tight spacing between the shafts. Therefore the Cam was modified with two extra servo motors to attach multiple arms as shown in Figure 2.3.

The modified design worked and would allow for tighter spacing between the shafts. The problem however was that it would not allow for fast actuation of multiple shafts.

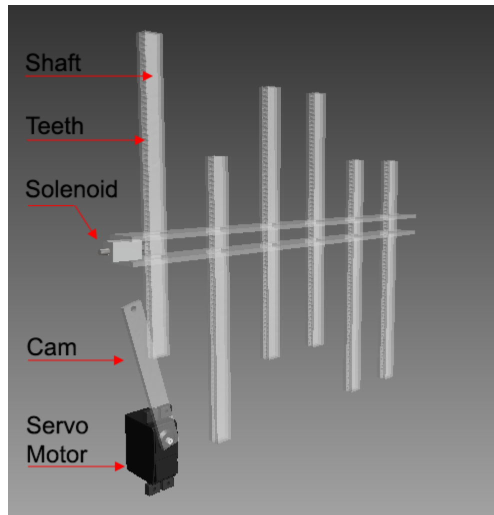


Figure 2.2: 3D rendering of the first prototype

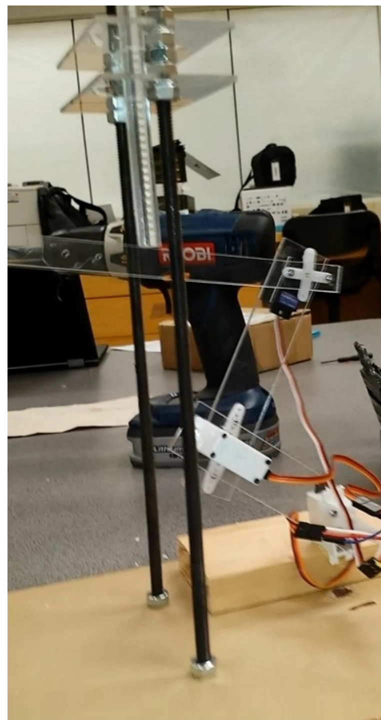


Figure 2.3: Prototype modified with extra servo motors

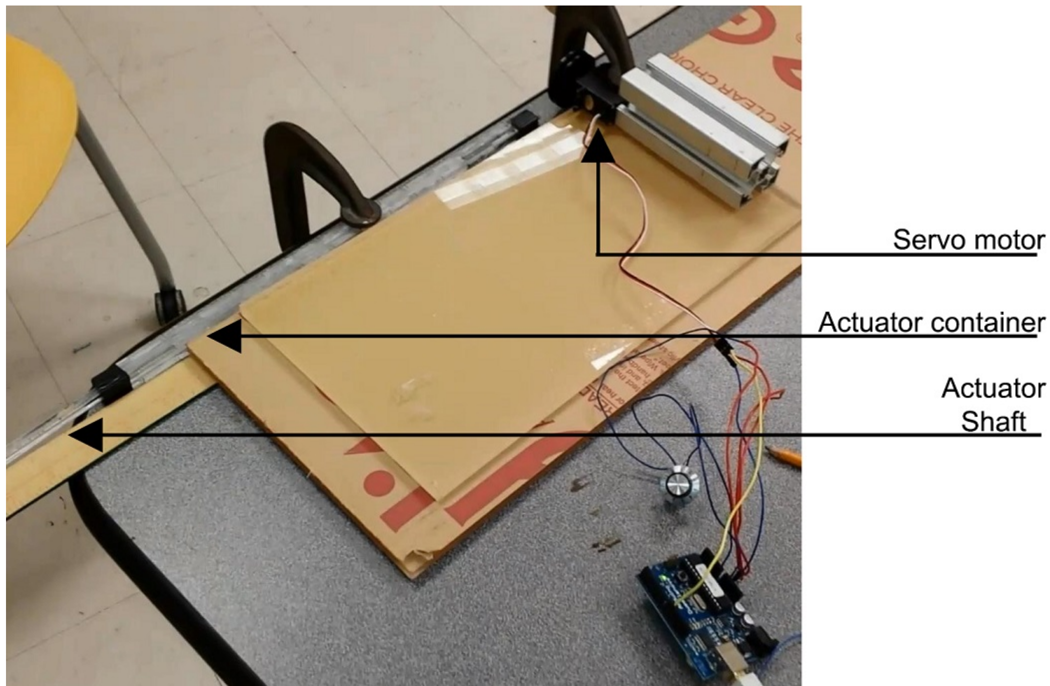


Figure 2.4: Second prototype

2.3 Second Prototype

The next mechanism which was tried was the wheel and axle. The basic idea was to use a servo motor as a winch and pull the shaft. But this would only allow for movement in one direction as the shaft would not move up again. Therefore the shaft had to be spring loaded to allow for bidirectional movement. To test this, a square tube was built out of laser cut acrylic sheet to be used as the container for the actuator. The shaft was also a piece of acrylic sheet which was cut to fit inside the container. The prototype is shown in Figure 2.4. Although the prototype worked well, there was no way to control the position of the shaft. The next section describes the final prototype, which has position feedback, in full detail.

2.4 Construction of the final prototype

The above prototype was further refined and the constructional details are provided in this section. All 3D renderings and CAD models were created using Autodesk Inventor [2] and all dimensions in CAD models are in mm.

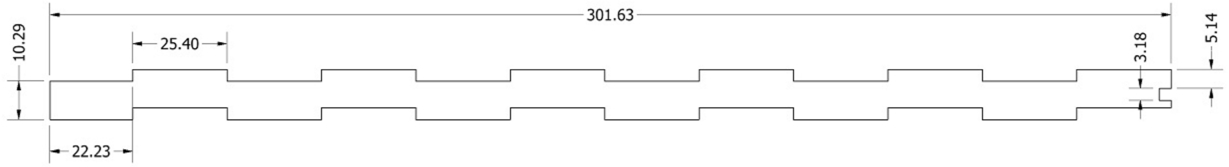


Figure 2.5: Details of the actuator container wall

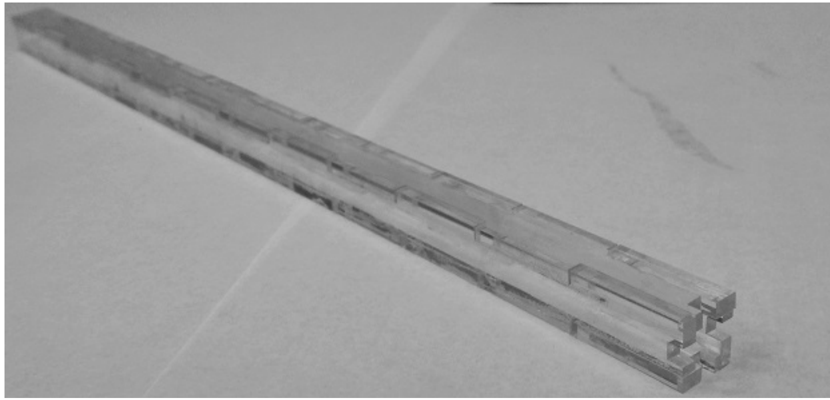


Figure 2.6: Actuator container after gluing

2.4.1 Actuator container

The actuator container was constructed from four interlocking pieces of acrylic which were cut as shown in Figure 2.5. The interlocking pattern was created because it would strengthen the container. The four pieces were then glued together using super glue as shown in Figure 2.6. An additional piece was cut to close the bottom of the container using the slots created. This piece served two purposes, one was to hold the spring in place and the other was to guide a cable through the container.

2.4.2 Actuator shaft

The actuator shaft was a simple rectangular piece of acrylic whose dimensions are given in Figure 2.9. The area in the red was not cut but scored to a depth of 0.5mm. This purpose of this is to keep the cable flush with the surface of the shaft and reduce friction between the actuator shaft and container.

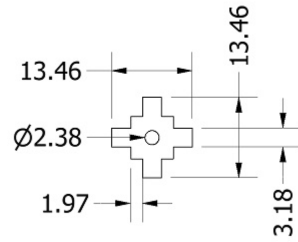


Figure 2.7: Details of the container cap

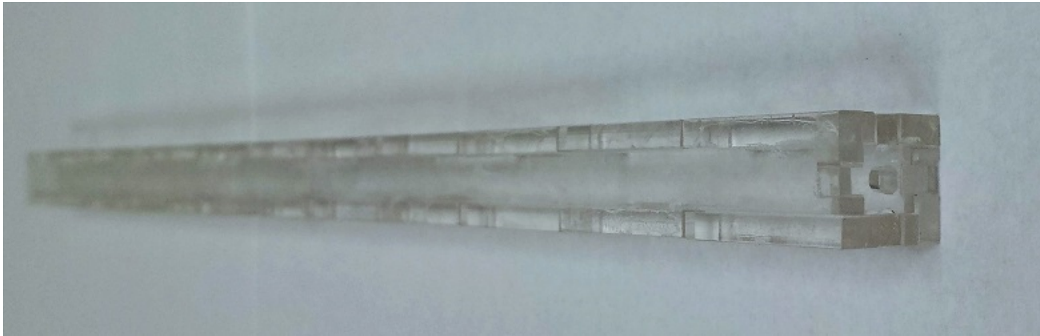


Figure 2.8: Actuator container with the end cap

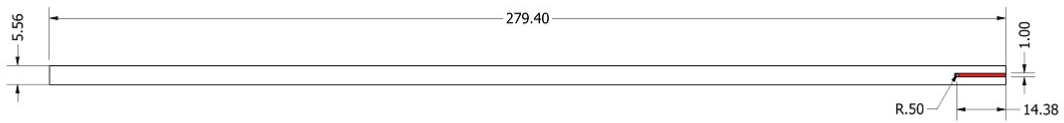


Figure 2.9: Dimensional details of the actuator shaft

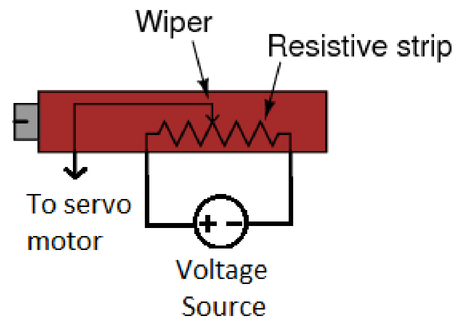


Figure 2.10: Schematic of a linear potentiometer

Figure 2.11: Actuator shaft with conductive ink painted on one side

2.4.3 Linear potentiometer

Since it was essential to detect the position of the shaft with respect to the container, some sort of a sensor had to be implemented. Since space and cost was a constraint, external sensors were avoided. The most obvious way to sense the position of the shaft was to convert it into a linear potentiometer. A linear potentiometer is a device which consists of a resistive strip and sliding contact which touches the resistive strip at one point. If an electric current is passed through the resistive element then the sliding contact acts as a voltage divider. By measuring this voltage, the position of the slider can be determined (see Figure 2.10). Therefore, if there was a resistive strip on the shaft and the point of contact was constant, the position of the shaft could be measured.

To develop the resistive strip different materials were tested. The first was to use a commercially available conductive ink and coat one side of the shaft with it. The ink had a resistivity of 150 Ω / inch for a width of 0.25 inches. Even though the ink worked as intended, it would have been difficult to apply the ink to all the actuators the same way without automation. Figure 2.11 shows the actuator shaft with the conductive ink painted on one of the sides.

The second material tested was a plastic film which was coated with Indium Tin Oxide(ITO).

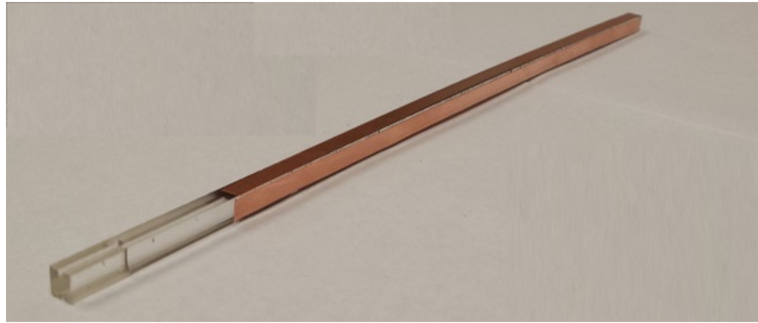


Figure 2.12: Actuator shaft with copper tape

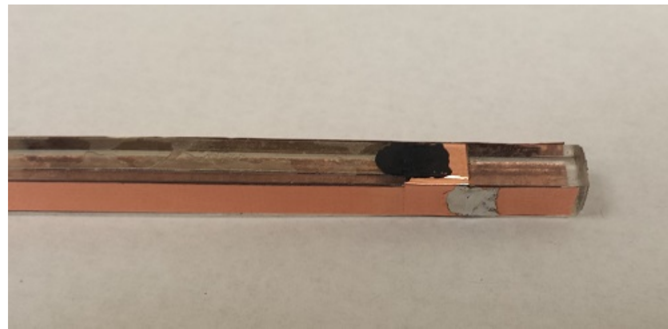


Figure 2.13: Details of the electrical joints

The film was flexible and could easily be cut and it had a resistivity of 50 Ω / square inch. The film came in sheets of 100mmX200mm and it was cut to strips of 5mmX200mm using a laser cutter. This strip was then glued to the side of the shaft. To supply the voltage required for the resistive strip to function, copper tape was used to form continuous contact on the sides perpendicular to the resistive strip.

A smaller piece of copper tape was used to make contact with the two ends of the resistive strip. To ensure good contact, the joint between the copper tapes was soldered and conductive ink was spread on the joint between the copper tape and the resistive strip.

To establish constant electrical contact between the actuator shaft and the container, it was essential to find a contact which would not destroy the resistive strip when the shaft was in motion. The best option was to use a sliding spring contact [14]. To mount the contacts four pieces of acrylic of size 25.4mm X 7.6mm were cut and glued to the top end of the container as shown in Figure 2.14.

The sliding contacts were then soldered to the ends of copper tapes and taped as shown. The copper tapes were then extended to the outside of the actuator container. This would serve as

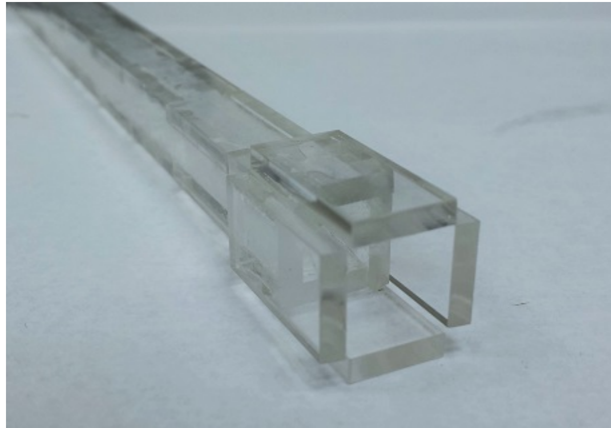


Figure 2.14: Photograph of the acrylic pieces used to hold the contacts

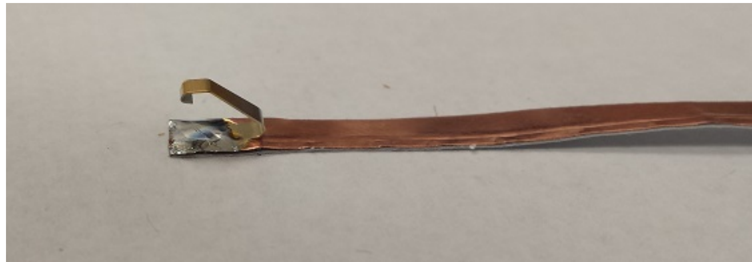


Figure 2.15: Sliding contact soldered to copper tape

external contacts for the actuator is shown in Figure **??** along with all the various components.

2.4.4 Spring selection

As described earlier, the actuator can only move in one direction with a cable. Therefore a spring has to be inserted into the actuator container to allow it to come back to its original position. Several different springs were tested and their information is shown in Table 2.2. The criteria for selection was that the spring was to be able to compress at least 150mm and the diameter was to be less than or equal to 6.35mm. It was also desired that the force required to compress the spring was to be less than 30N as this was the maximum force capable of being produced by the servo motor actuator after taking into account the diameter of the pulley attached (see section 2.4.6).

The second spring was found to best satisfy the specifications and Figure 2.18 shows the spring which was selected.

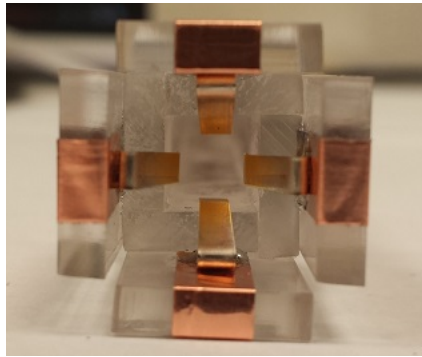


Figure 2.16: Sliding contacts mounted on the actuator container

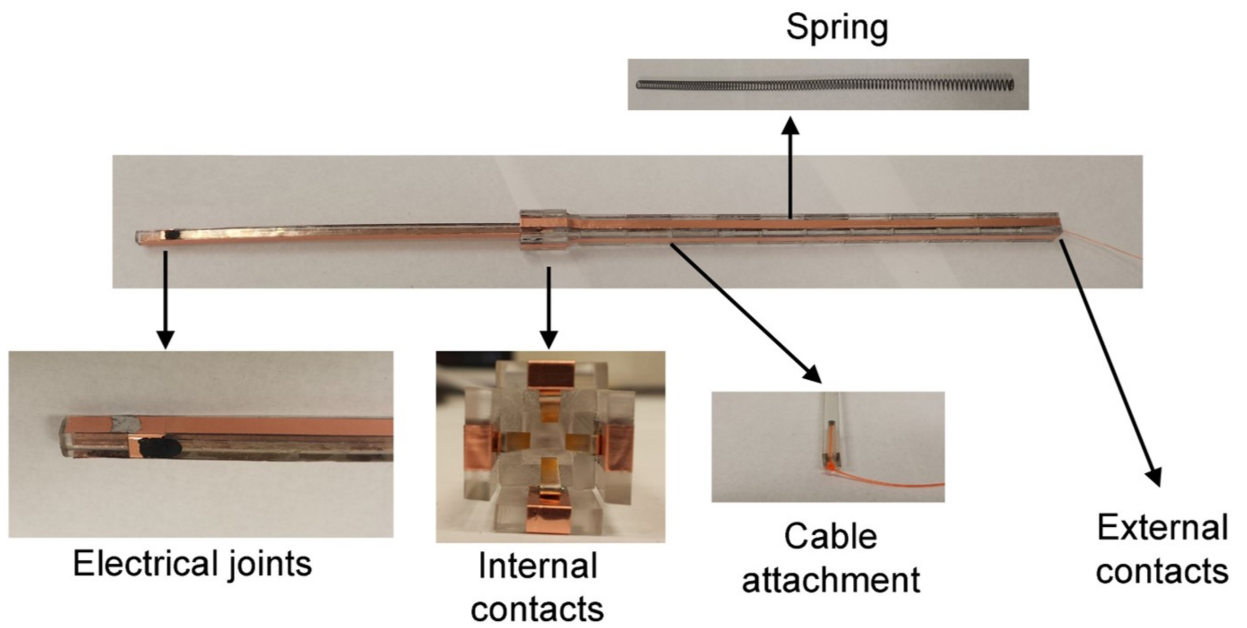


Figure 2.17: Actuator container with the contacts fully mounted

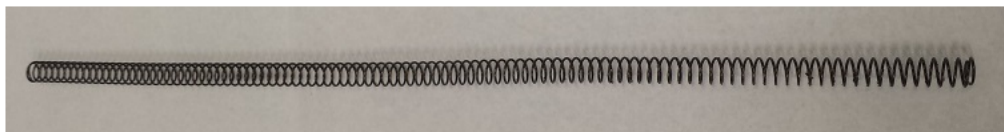


Figure 2.18: Photograph of the spring

Outer Diameter(mm)	***Wire Diameter (mm)***	***Force required to compress 10mm(N)***	***Maximum Compression(mm)***
6	0.71	1.96	100
6.35	0.55	0.65	160
3	0.7	4.1	50

Table 2.2: Comparison of Springs

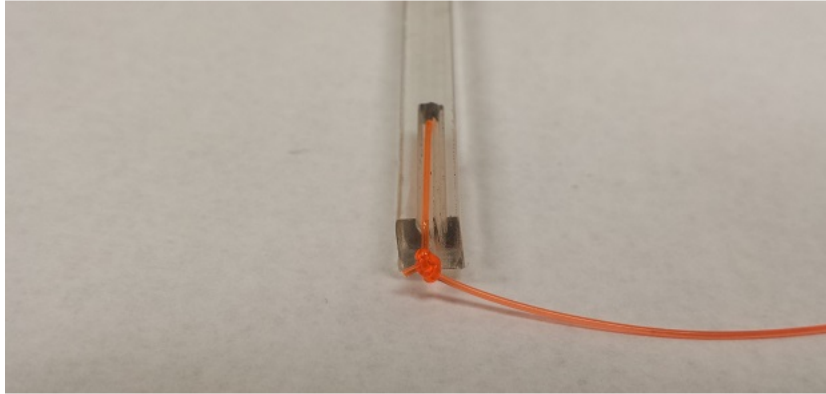


Figure 2.19: Close up photograph of the fishing line attached to the actuator shaft

2.4.5 Cable selection

To pull the shaft of the actuator down, a cable had to be connected to the base and routed through the spring to the outside. Since the force required to compress the spring was known, any cable with a breaking strength of at least twice that amount could be considered. Since fishing lines are easily available and are very thin, a fishing line with a breaking strength of 30lb was selected and is shown in Figure 2.19.

2.4.6 Pulley

To connect the fishing line to the servo motor, a pulley had to be fabricated to reel it in and keep it in place. The diameter of the pulley was chosen to be such that the actuation would be fast enough while maintaining enough torque. The details of the pulley are shown in Figure 2.20. It consists of a small disc sandwiched between two bigger discs. This was then mounted to the servo motor using one of the accessories with screws as shown in Figure 2.21.

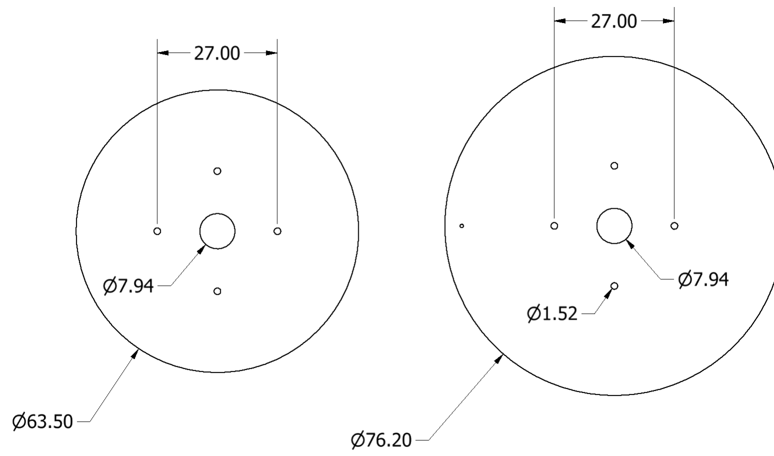


Figure 2.20: Dimensional details of the pulley

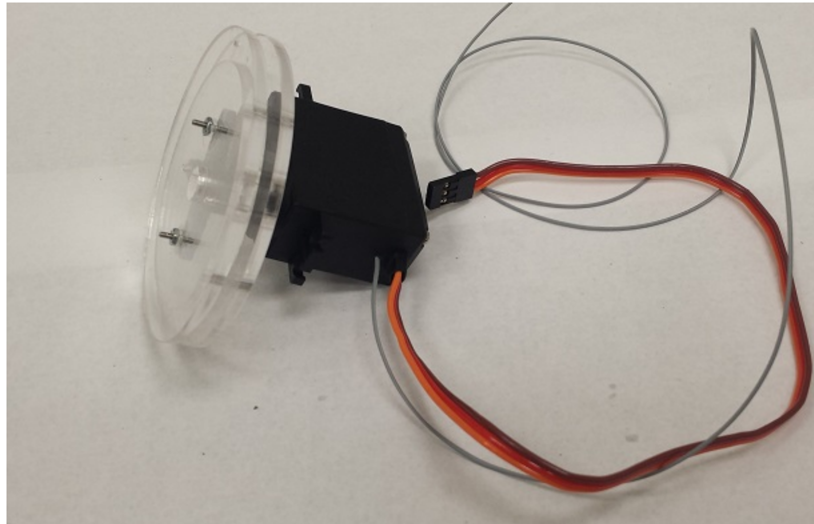


Figure 2.21: Pulley mounted to the servo motor

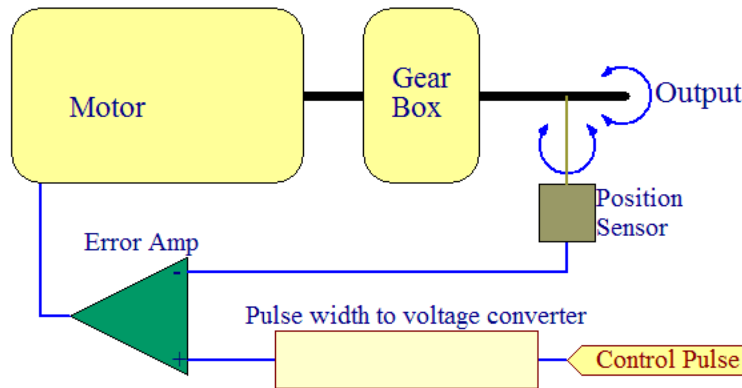


Figure 2.22: Block diagram of a servo motor

2.4.7 Actuation mechanism

The winding action needed for the linear actuation can be achieved through rotary mechanisms such as stepper motors, geared DC motors and servo motors. Several factors influenced the choice of servo motors over the others. Unlike stepper motors and geared DC motors, servo motors have built in motor drivers which eliminates the need to use external drivers making it cost effective (see Figure 2.22). They also have a feedback control system required for position control which makes it easier to control through simple PWM (Pulse Width Modulation) signal.

The details of the particular servo motor used in the project is given in Table 2.3. This motor was chosen based on the size, torque, availability and cost requirements. It is capable of rotating 180 degrees and has a rotary potentiometer to sense the position. If this potentiometer is replaced with the output from the linear actuator, then it is possible to control the linear position of the actuator with regular PWM servo control. To achieve this, the servo motor was disassembled and the potentiometer (see Figure 2.23) was removed and the leads were desoldered. An external wire was soldered to the middle pin of the control board which is the input for position feedback. The motor also has a safety mechanism for protecting the potentiometer by limiting the travel of the shaft to 180 degrees. This is achieved through a metallic pin inserted in the gear connected directly to the shaft (see Figure 2.24). Since the servo motor is being modified to rotate continuously, this pin was removed.

Servo model	***Size (mm)*(mm)*(mm)***	***Weight (grams)***	***Stall Torque (kg-cm)***	***Working Voltage (V)***	***Stall Current (A)***	***Speed (s/60°)***
TowerPro MG-995 [25]	40.7 * 19.7 * 42.9	55	13	5 to 7.2	2	0.2

Table 2.3: Servo motor specifications

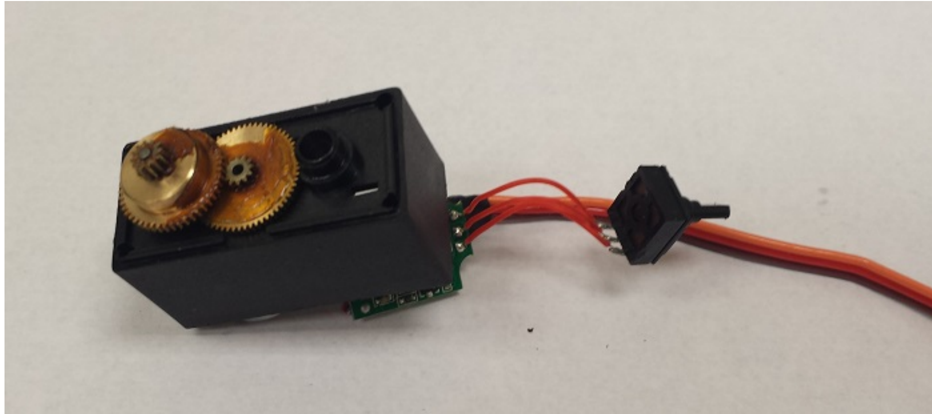


Figure 2.23: Servo motor with the potentiometer removed

2.5 Servo motor assembly

Since the display was to have a resolution of 16x16, it needed 256 of the linear actuators that were described in the previous section. Since space was a constraint, it was essential that the servo motors were packed as tightly as possible. It was also necessary that the packing would allow for easy assembly and disassembly. After considerations of various alternatives, it was decided that the best design choice would be to divide the assembly into 16 blocks each of which contained 16 motors. Each block was then further divided into four levels which contained four motors each. The motors were arranged in a staggered configuration both in plane and between levels as shown in Figure 2.25 to increase packing density. The structure is supported by four threaded steel rods which are two feet in length and a quarter inch in diameter. Figure 2.26 shows all the servo motor blocks placed next to each other.

The external and internal dimensions of the top and bottom plate are shown in Figures 2.27 and 2.28. Figure 2.28 also shows some of the key features of the design. The guides for the fishing line as well as the cables were slotted towards the inside so that the blocks could be placed flush

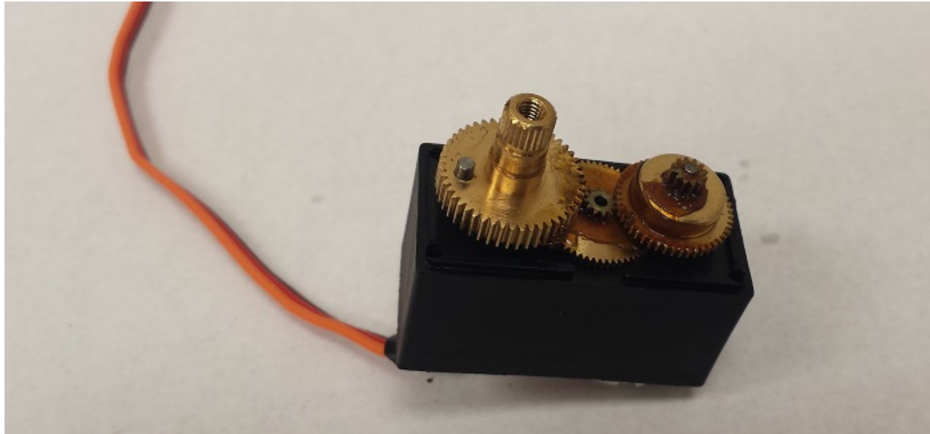


Figure 2.24: Gearbox of the servo motor

with one another thereby reducing the space required.

2.6 Pressure sensor

To make the display tangible it was necessary to have pressure sensors embedded in the actuators. Since cost was a factor, commercially available pressure sensors were out of reach. Several different materials were tested for their effectiveness in sensing pressure and the most effective was found to be a conductive foam which is used to protect ICs [20].

This low density foam is made of electrically conductive polyurethane fibers and as pressure is applied to the material, the fibers move closer together and the electrical conductivity increases. By measuring the resistance between two opposite surfaces of the material (along which pressure is applied), the applied pressure can be determined.

The material was cut into a square of 10mmx10mm and a 3D printed plastic piece was used to hold the foam in place. The details of the 3D printed piece are shown in Figure 2.30. To establish electrical contact with the foam, copper tape was attached to the top and bottom surface. The copper tapes were then routed outside the plastic piece through the thin slits at the bottom. This was then attached to the top of the actuator shaft with super glue. A plastic piece was printed to act as the sensor surface and to enable the application of pressure evenly to the foam. This also acted as the display surface for the projected image. The complete assembly is shown in Figure 2.32.

Figure 2.25: 3D rendering of the servo assembly

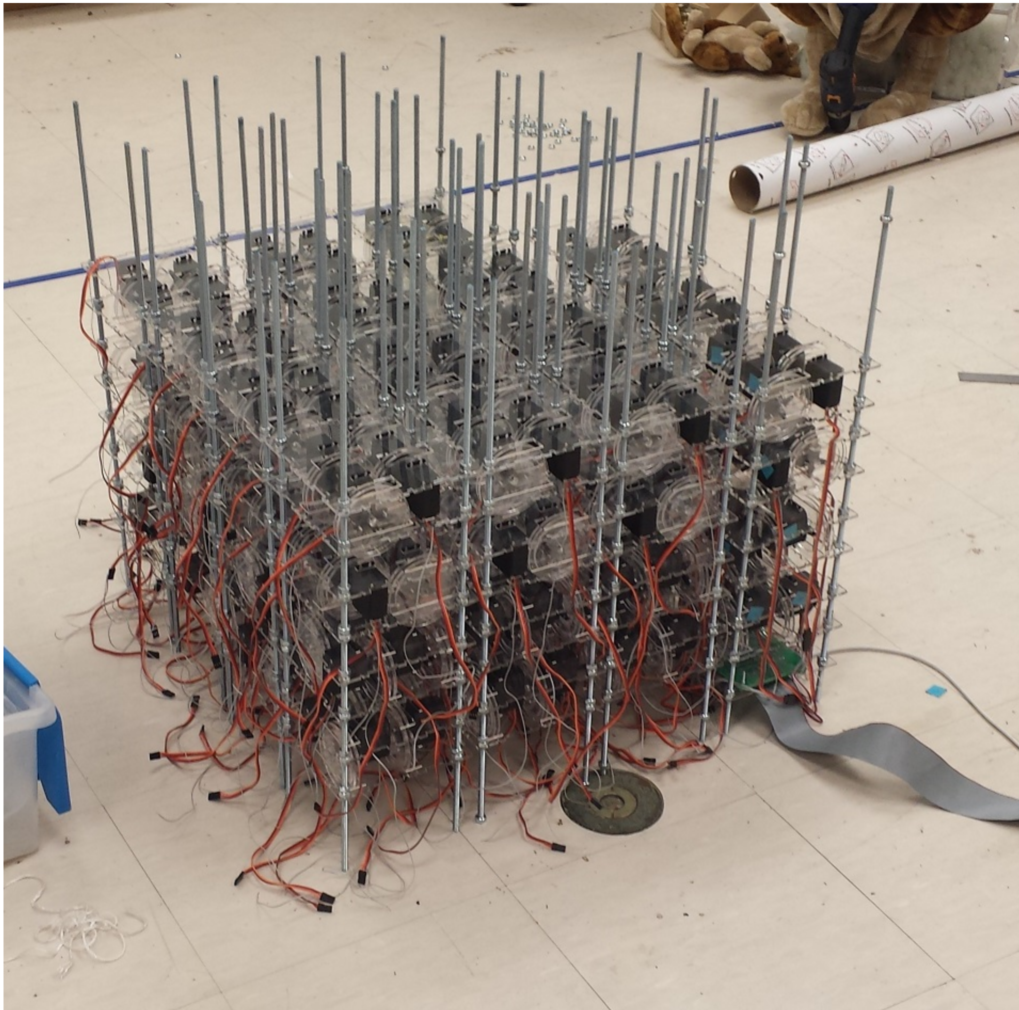


Figure 2.26: Servo motor blocks

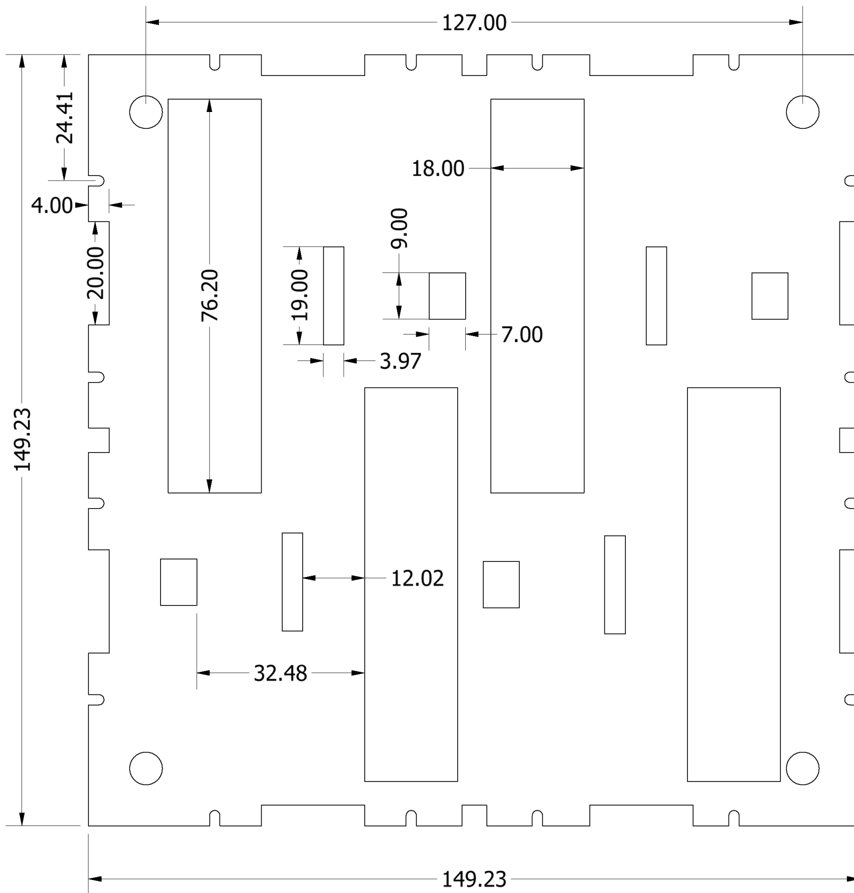


Figure 2.27: Dimensional details of the top plate of the servo assembly

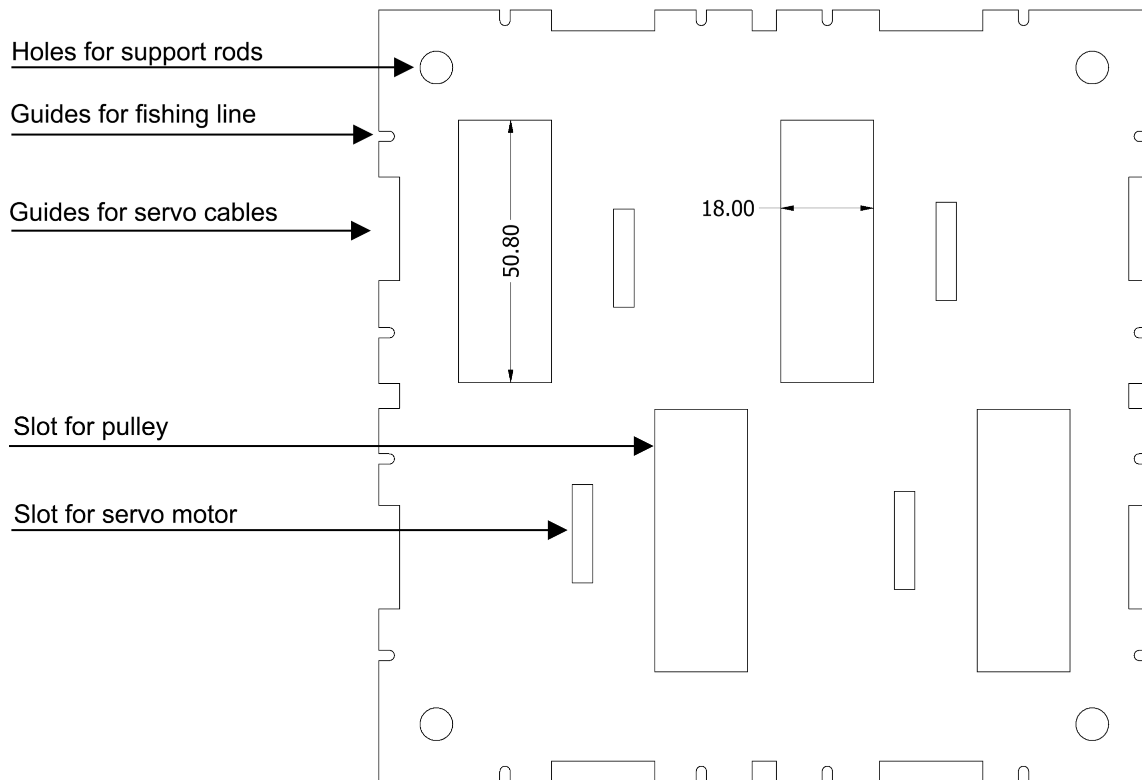


Figure 2.28: Dimensional details of the bottom plate of the servo assembly

Figure 2.29: Conductive foam used to protect ICs

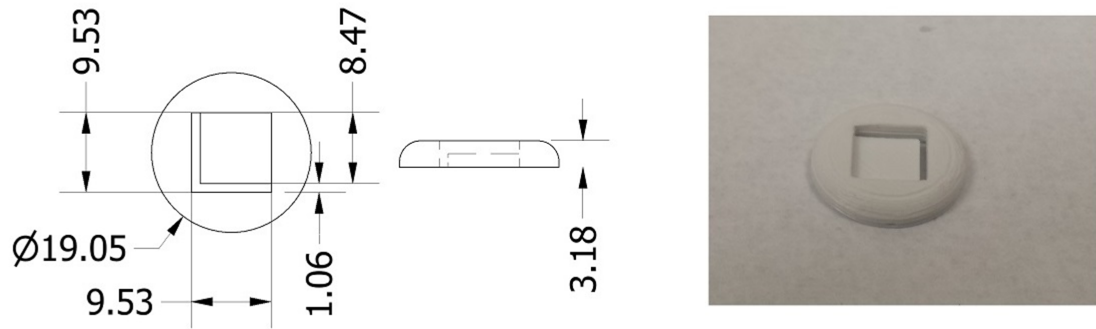


Figure 2.30: Dimensional details of the pressure sensor holder

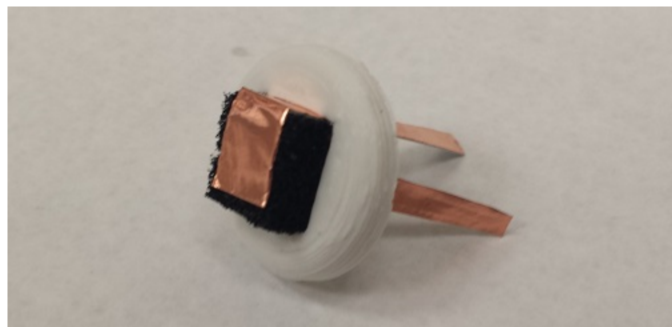


Figure 2.31: Pressure sensor inserted into the 3D printed piece

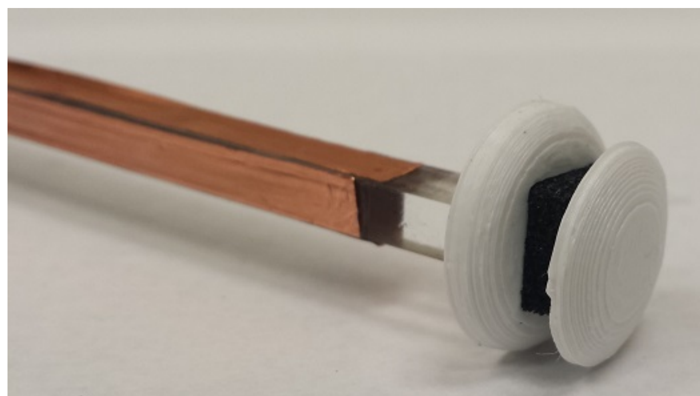


Figure 2.32: Pressure sensor with touch plate mounted on the actuator shaft

2.7 System block diagram

Figure 2.33 shows the overall block diagram of the whole system. Each block will be explained in detail in the following sub-sections. The blocks which are contained within the red outline are for one block and there are 16 such blocks. The Arduino [1] is an open source Hardware and Software platform for using AVR microcontrollers. It is used as a communication interface between the PC and the servo controllers and sensor boards. The PC generates the required servo commands based on the user input which are then transmitted to the Arduino via serial communication. The Arduino then transmits the servo commands to the specified servo controller via I2C to control the servos. The position feedback of the actuators is received through the actuator and sensor board which is connected to the Data acquisition board (DAQ) board through a 40 pin data cable. The individual servo motors then receive this input from the DAQ board. Power to the servo motors in each block is provided by a computer power supply. The pressure sensor values are also received at the DAQ board which converts the analog singal to digital values and transmits them via I2C to the Arduino.

The Microsoft Kinect sensor [5] is a horizontal bar connected to a small base with a motorized pivot and is designed to be positioned lengthwise above or below the video display. The device features an RGB camera, depth sensor and multi-array microphone running proprietary software, which provide full-body 3D motion capture, facial recognition and voice recognition capabilities. It is used to track the user and detect gestures which are then used to control the pixel display. An overhead projector is mounted on a metal frame, vertically above the system and pointing down on

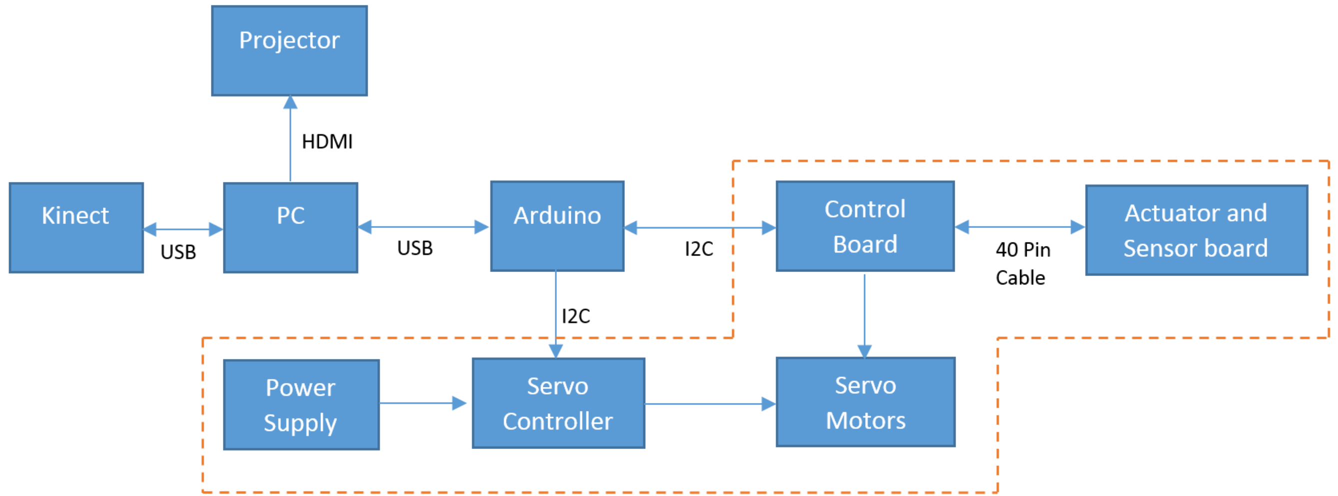


Figure 2.33: Block diagram of the system

it, to enable images to be displayed on the pixel display and make it easier for complex shapes to be visualized.

2.7.1 Data acquisition board(DAQ)

The schematic of the DAQ board is shown in Figure 2.34. The pressure sensor values and the position feedback from the linear actuators are multiplexed using CD4067 [24] which is a 16 channel Multiplexer. Since there are 16 actuators and 16 pressure sensors per block, two Multiplexer ICs are required. The multiplexed output is received by ATMEGA8 [9] which is an 8 bit microcontroller. The microcontroller is bootloaded with Arduino for ease of programming. The microcontroller (slave) performs analog to digital conversions of the inputs and transmits them via I2C when the Arduino (master) requests the data.

Since the servo motor's feedback works at a voltage of 2.5V, a voltage regulator IC, LM317 [23], is used to generate the required voltage. The analog reference of the microcontroller is also given the same voltage to ensure maximum resolution. Figure 2.35 shows the board layout which was used for the PCB. The board size was 10cmX10cm and was designed using EAGLE Schematic and PCB design software [3].

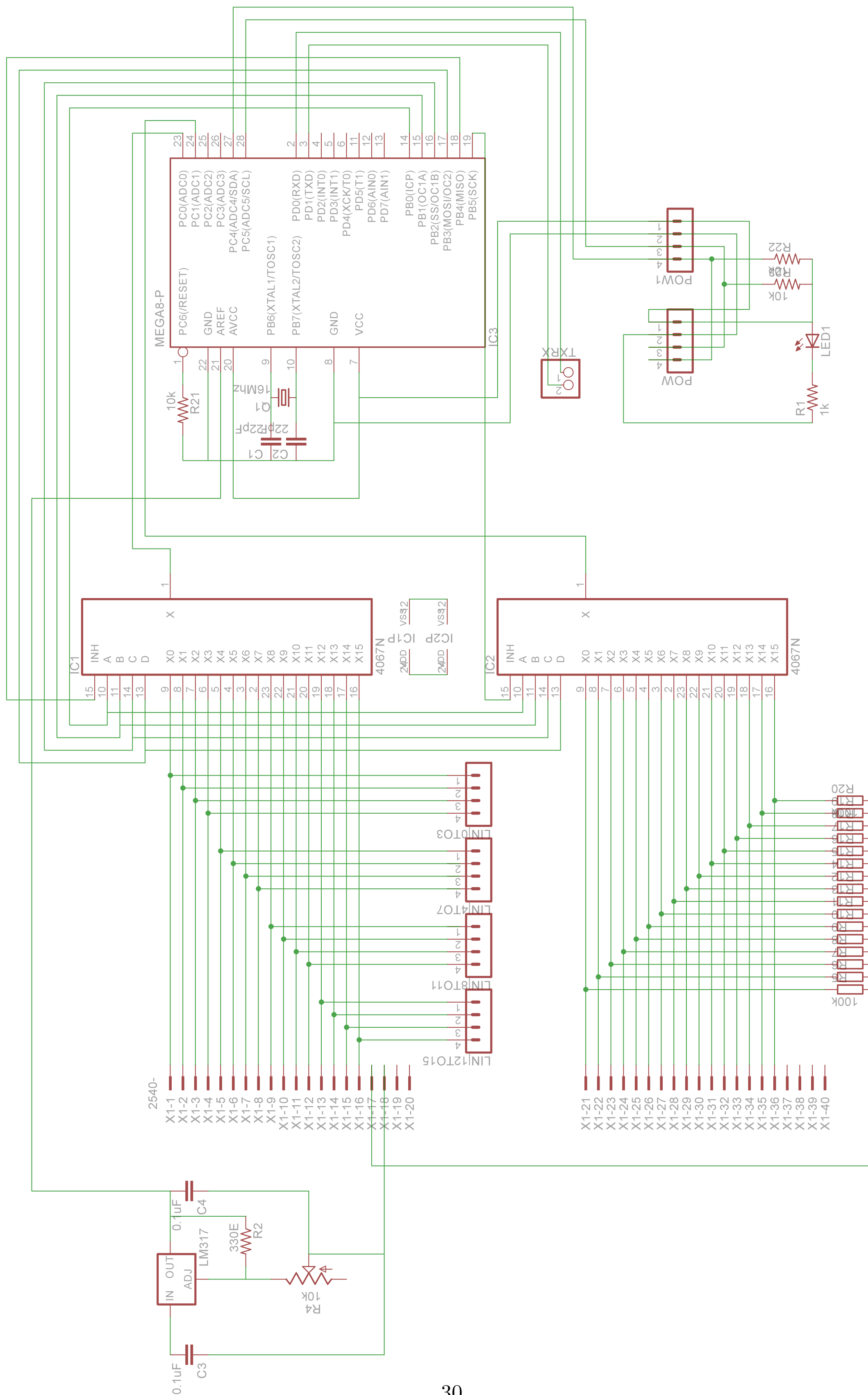


Figure 2.34: Schematics of the DAQ board

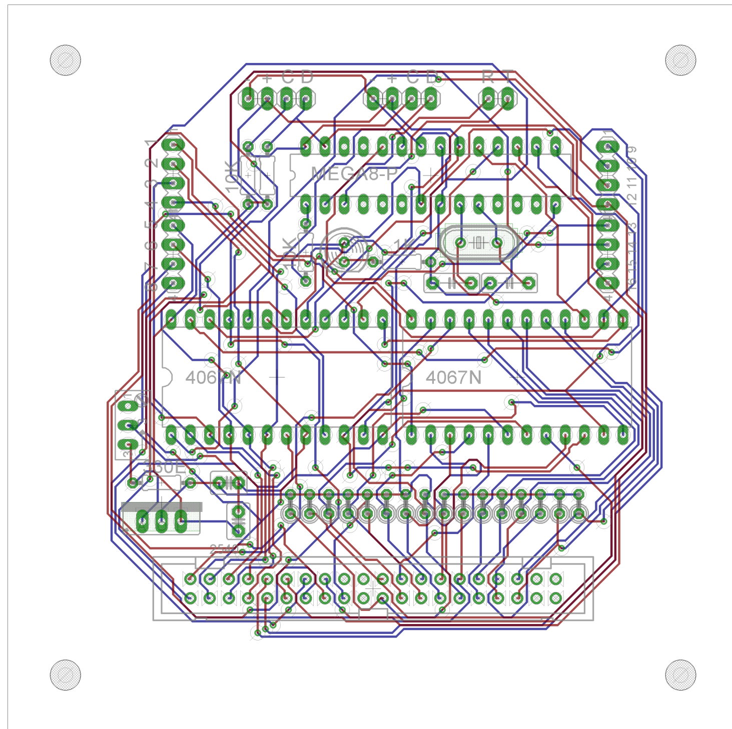


Figure 2.35: PCB Layout of the DAQ board

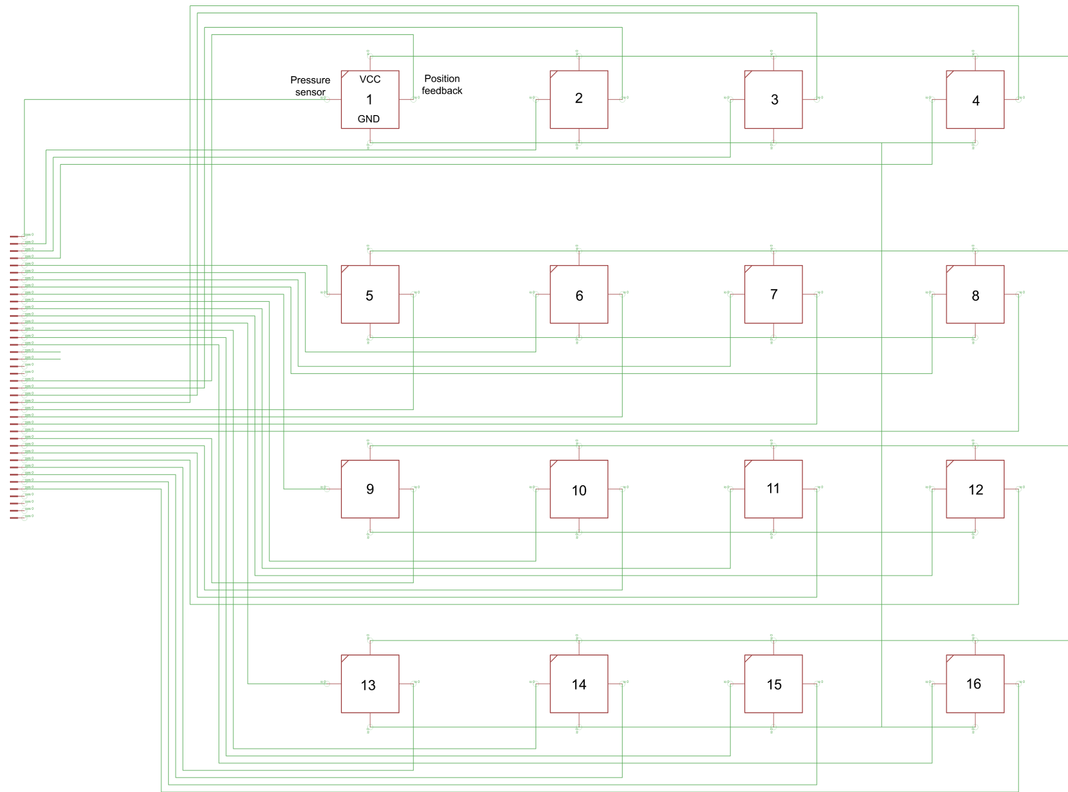


Figure 2.36: Schematics of the actuator and sensor board

2.7.2 Actuator and sensor board

The schematic of the actuator and sensor board is shown in Figure 2.36. This board provides a structural platform for the actuators and establishes electrical contacts. Each actuator has 4 contacts - VCC, Ground, Position feedback output and pressure sensor output. All the contacts from the 16 actuators are routed to the 40 pin cable connector for easy connection. Figure 2.37 shows the PCB layout of the board. The actuators are spaced an inch apart from each other and the connector is placed in between two rows to keep the board size below 10cmX10cm.

2.7.3 Servo controller board

To control the servo motors in each block, a commercially available 16 channel servo motor controller [18] from Adafruit was used and is shown in Figure 2.38. The servo motor controller also worked on I2C and 16 such boards were used to control 256 actuators. Each board was given a different I2C address and were daisy chained along with the control boards which is explained in

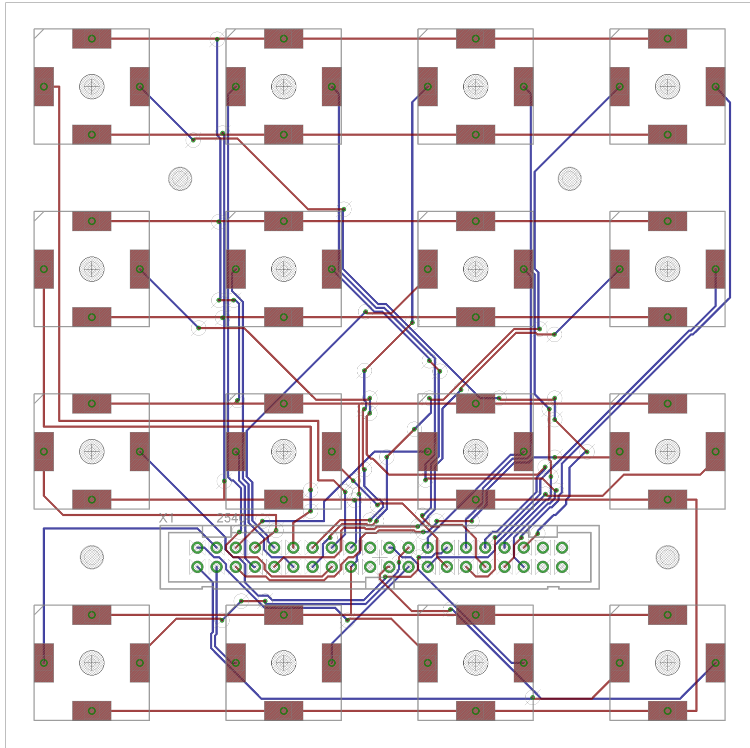


Figure 2.37: PCB layout of the actuator and sensor board

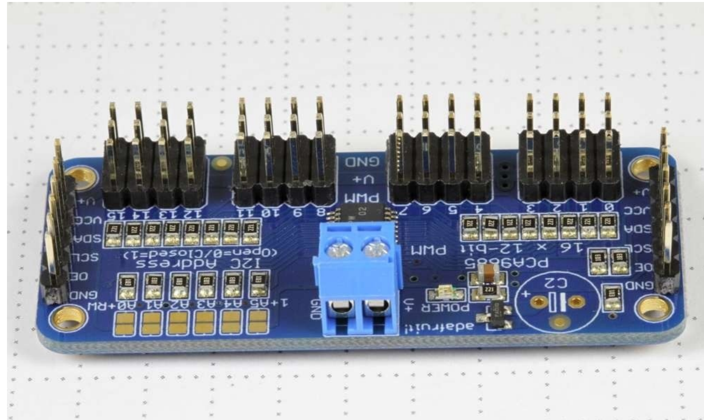


Figure 2.38: 16 channel servo controller

subsection 2.7.5.

2.7.4 Power supply

The stall current of the servo motors is rated at 2A at 5V. Therefore the power supply needs to handle a maximum of 32A as there are 16 motors per block. The most cost effective way to handle such a large amount of power was to use a computer power supply. The specific power supply used in the project was rated at 36A at 5V and 16 such power supplies were used. The power supply is in Figure 2.39.

Figure 2.39: Power supply

2.7.5 Arduino

An Arduino Uno [8] was the main interface between the PC and all the control boards (see Figure 2.40). It was responsible for receiving the servo commands from the PC and transmitting them

to the right servo control boards as well as requesting sensor data from the DAQ boards and sending them to the PC. The Arduino communicated through I2C which is a multi-master, multi-slave, single-ended, serial computer bus. It requires only two data lines for establishing communication, a clock line (SCL) and a data line (SDA). Figure 2.41 shows how the Arduino was connected to the DAQ and Servo control boards through I2C. The area inside the dotted rectangle makes up one block and the communication lines are connected serially from one block to the next to all 16 blocks.

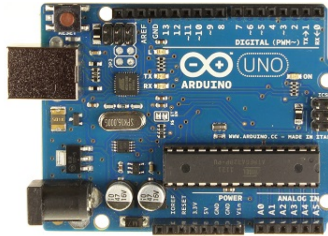


Figure 2.40: Arduino

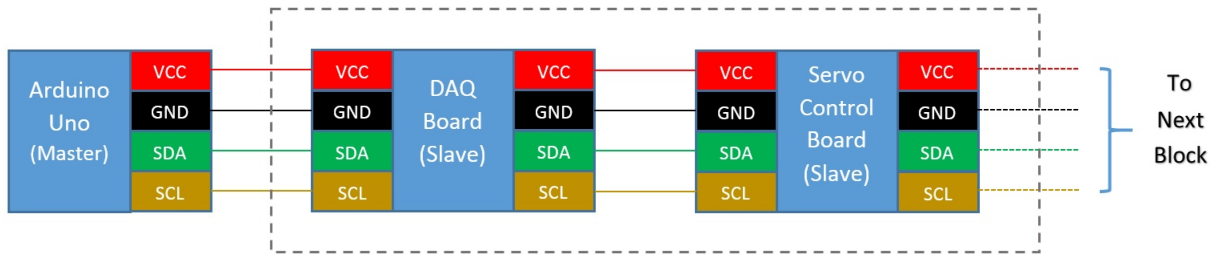


Figure 2.41: Arduino communication chain

2.8 Assembly

To make assembly easier, the 16 blocks were divided further into groups of 4. The details of one such group are shown in Figure 2.42. The PCBs were arranged on one of the corners of an acrylic plate of dimensions 304.8mm x 304.8mm. The holes which are highlighted in blue (dotted) are for the steel support rods and the ones highlighted in red (thicker) are for mounting the PCBs. All other holes are for the fishing lines which are connected to the actuators. Figure 2.43 show the groups during Assembly and Figure 2.44 shows the all the groups after complete assembly.

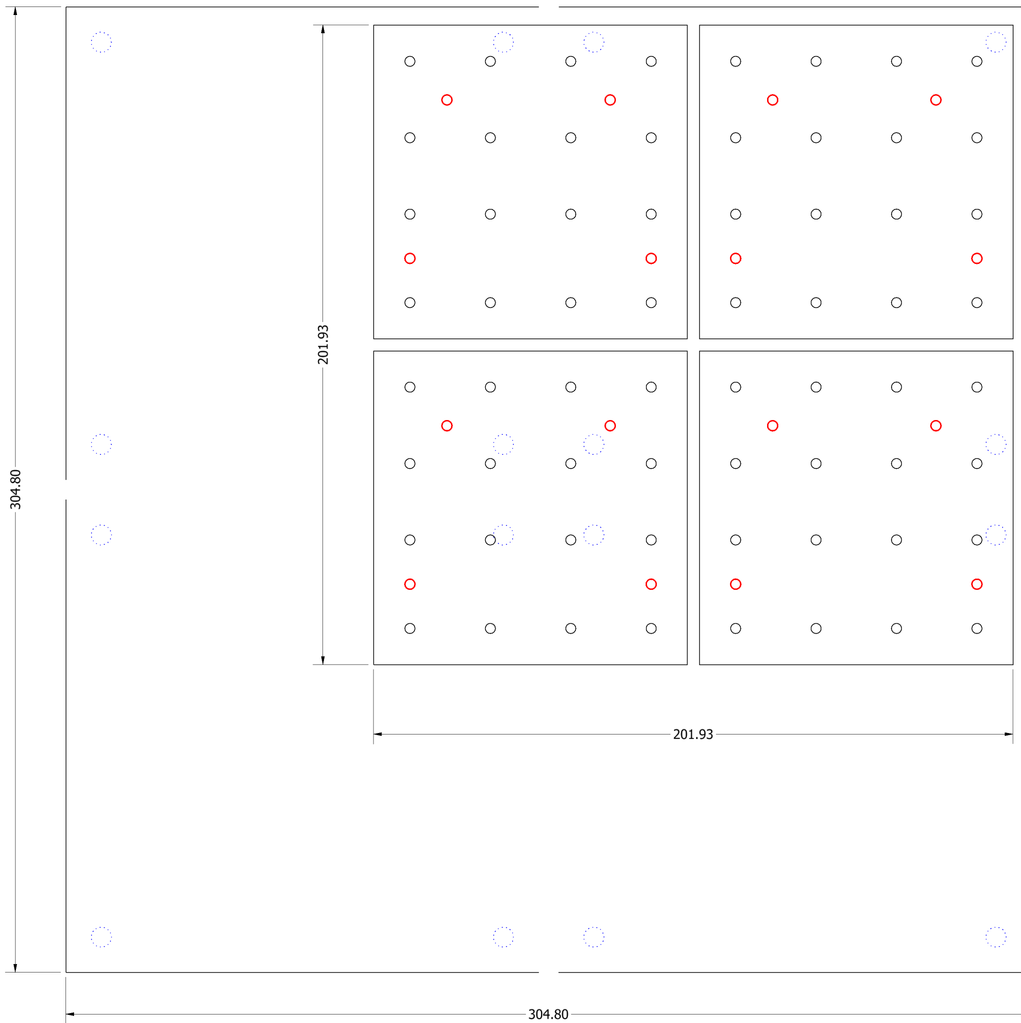
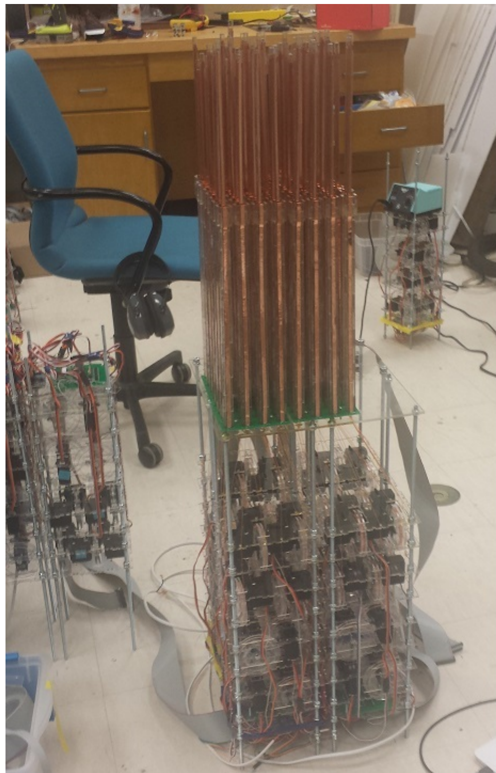


Figure 2.42: Dimensional details of the group of four blocks

2.8.1 Actuator contact separator

Figure 2.44 shows the actuators after they were mounted and soldered to the PCBs. To keep the copper tapes of the actuators from touching each other and to keep them at the right distance, a plastic separator was 3D printed which is shown in Figure 2.45. For easy insertion, the ridges on the piece were tapered. Figure 2.46 shows the actuator system with the separator.



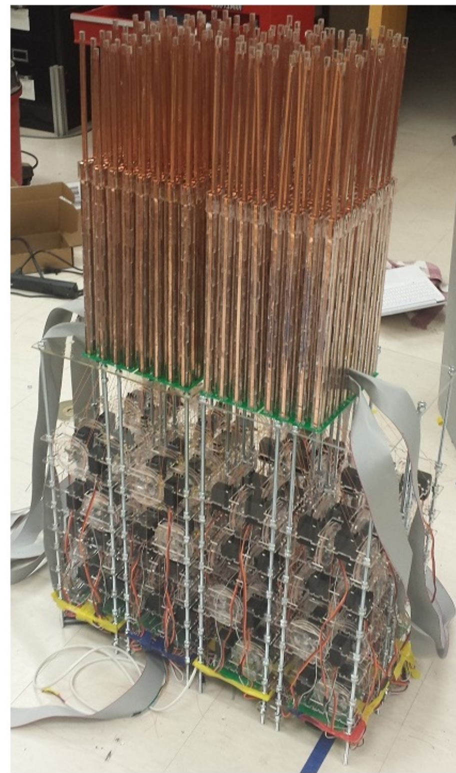


Figure 2.43: Servo group during assembly

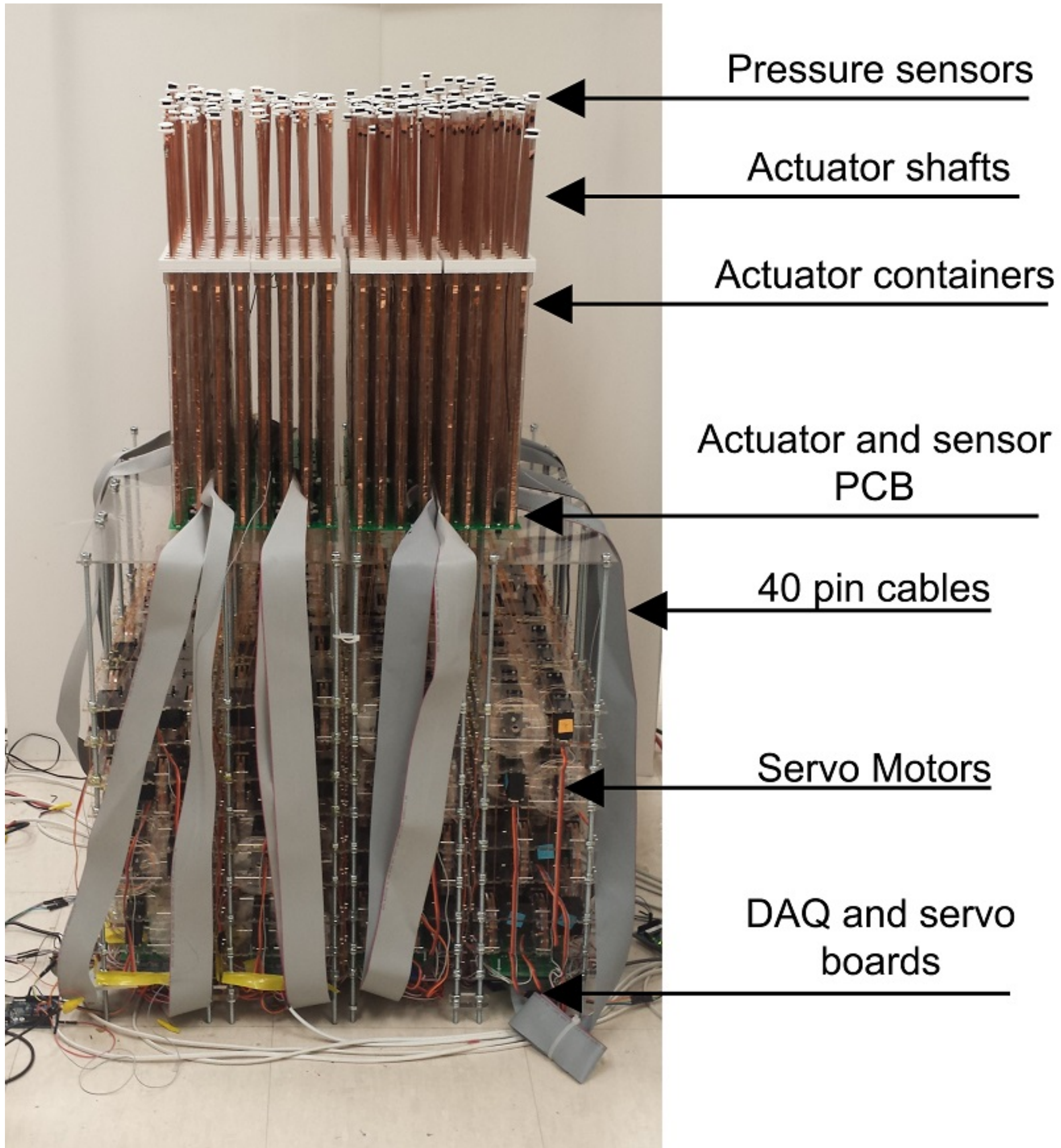


Figure 2.44: Servo group after full assembly

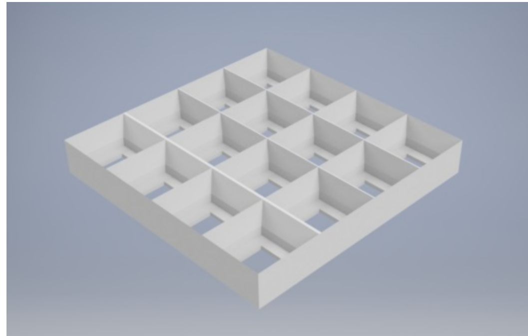


Figure 2.45: 3D rendering of the contacts separator

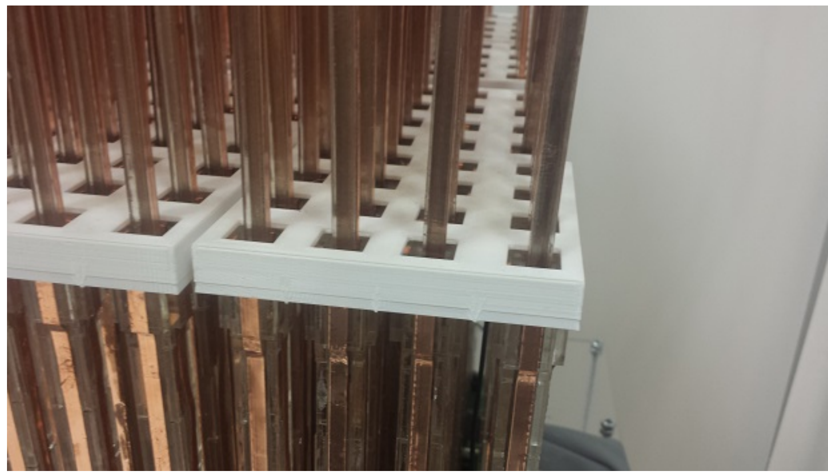


Figure 2.46: Contact separator attached to the actuators

2.8.2 Kinect and projection system

To enable images to be displayed on the surface, a projection system was constructed. A metallic structure was built out of Bosch frames to support the projector as shown in Figure 2.47. The projector was connected to the PC through an HDMI cable and the location of the projector was adjusted to get a full image on the display.

A Microsoft Kinect was used to make the display interactive by detecting gestures made by the user and using it to move the pixels on the display. The software used to detect the gestures as well as the gesture mapping will be explained in subsequent sections.

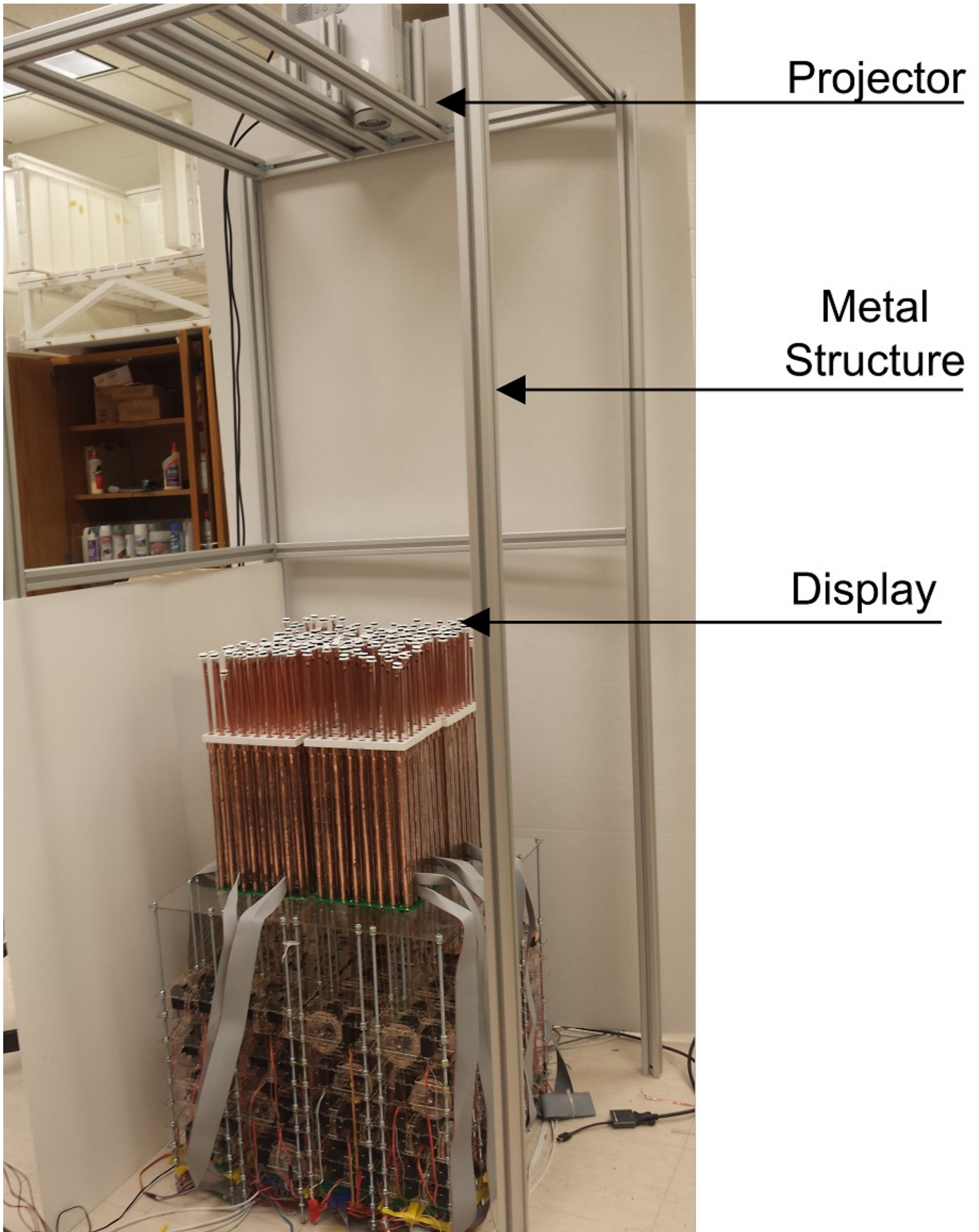


Figure 2.47: Pixel display with the projection system

Figure 2.48: Microsoft kinect used to detect gestures

2.9 Software

2.9.1 Calibration

Since the actuators were put together by hand, there were slight differences in the way they behaved. Specifically, the conductive ink which was used to join the copper tape to the resistive strip exhibited different resistance values. Because of this, each linear actuator had slightly different range of values for the position feedback. To overcome that, the servo commands which were sent to the actuators had to be calibrated for each individual actuator. The actuators were calibrated for 8 positions in increments of half an inch. To simplify the process, a graphical user interface (GUI) was built using ”Processing” [12] which is an integrated development environment for Java. A screenshot of the GUI is shown in Figure 2.49.

To calibrate, a specific block had to be selected using the 4x4 matrix after which the specific linear actuator was selected using the left and right arrow keys. Then, the top most and the bottom most positions were calibrated by moving the actuator up/down using the up/down arrow keys. Since the position feedback was linear, the servo commands for the rest of the positions were interpolated using these values and stored in a data file. This process was repeated for all of the 256 actuators and the calibration file was created.

2.9.2 Gesture detection

Gesture detection was implemented by using a software called FLEXIBLE ACTION AND ARTICULATED SKELETON TOOLKIT (FAAST) [4] which was developed at the University of Southern California. This program enabled the use of Kinect to detect gestures and map them to key combinations which were then sent to the GUI to be processed. A square region was defined using the length of the users arm around the left side of the body (see Figure 2.50 and note that the

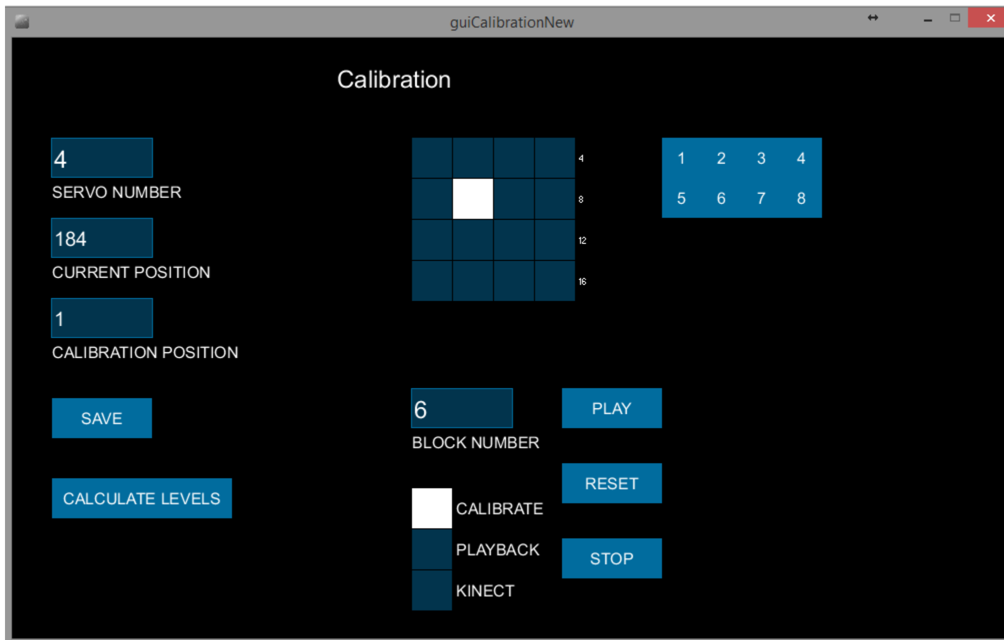


Figure 2.49: Screenshot of the GUI

image is flipped). By moving the wrist to different quadrants of this square, the specific quadrant of the pixel display could be selected. Four levels were defined towards the user's right side of the body and by moving the right wrist up and down, the height of the pixels could be controller. By using a combination of both hands, the user could select the specific quadrant to be controlled and use the right hand to control the height.

2.10 Materials used

The list of materials used to build the pixel display, their quantity and where they were used in detailed in Table 2.4. The quantity indicated is to build all the 256 actuators. Figures 2.51 and 2.52 shows some of the components of the system before assembly.

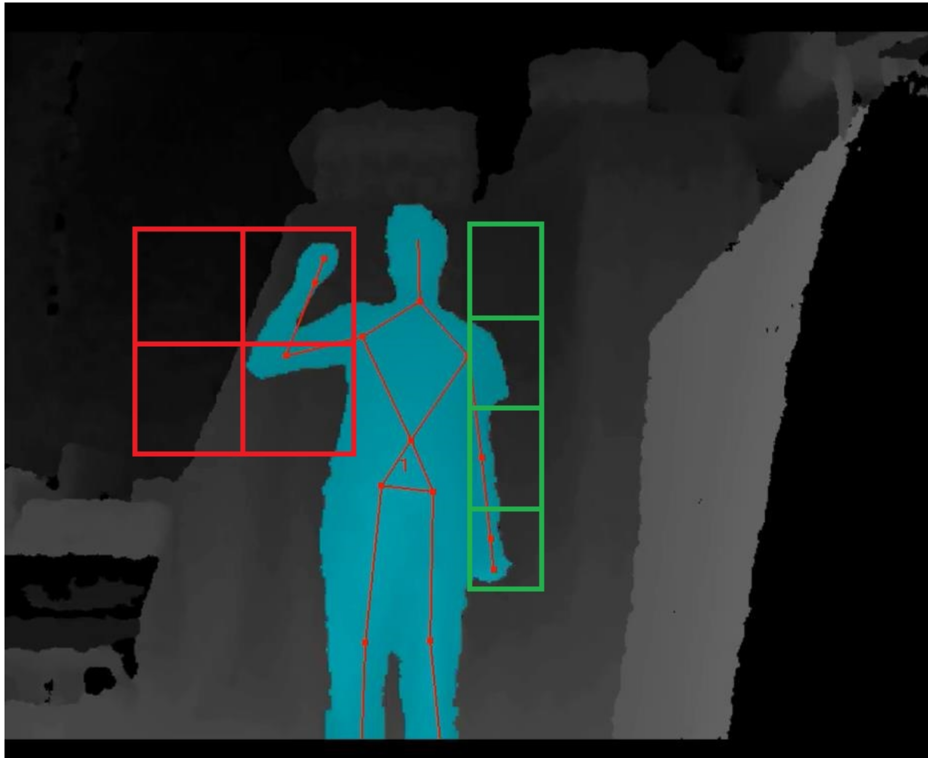


Figure 2.50: Gesture detection from Kinect data

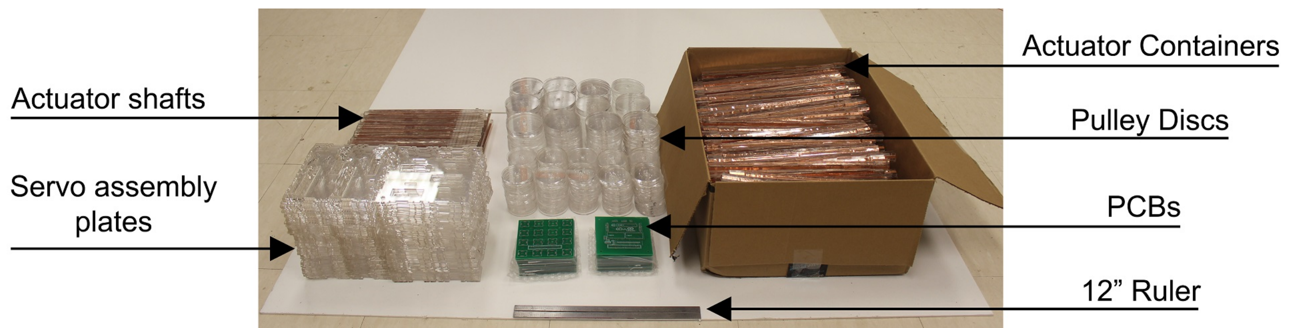


Figure 2.51: Materials before assembly

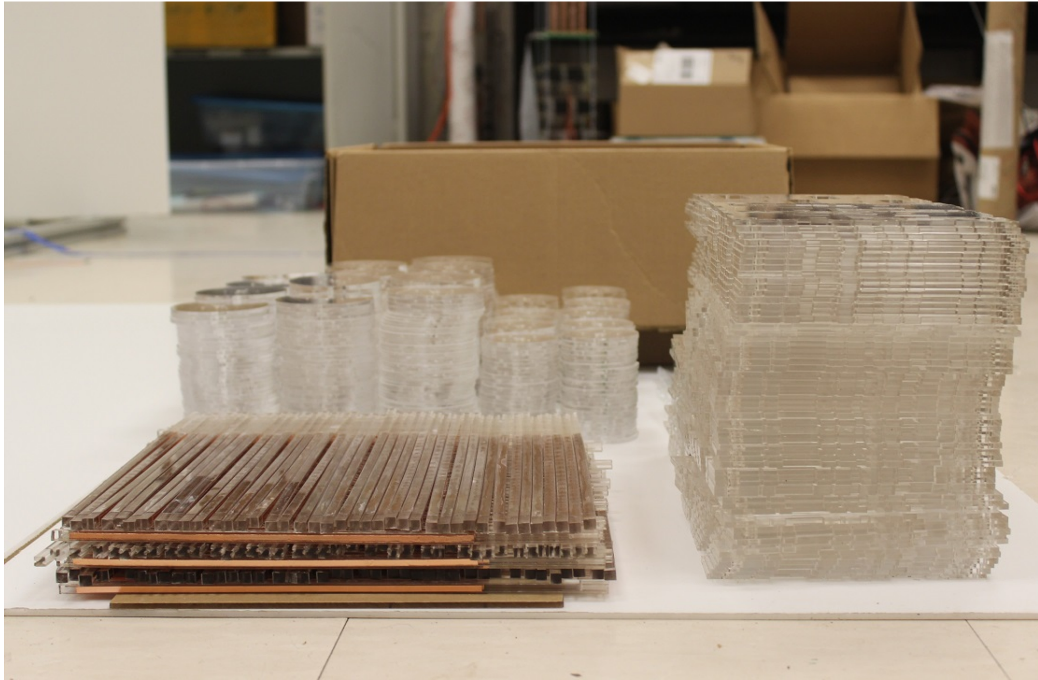


Figure 2.52: Materials before assembly

Material	***Size/Specs***	***Quantity***	***Use***
Acrylic sheet	30.48cmX30.48cmX3.175	37	Actuator container
Acrylic sheet	30.48cmX30.48cmX6.35	6	Actuator shaft
Acrylic sheet	30.48cmX30.48cmX3.175	32	Servo assembly
Acrylic sheet	30.48cmX30.48cmX3.175	45	Pulley assembly
Acrylic sheet	30.48cmX30.48cmX3.175	8	PCB Support
ITO Coated Plastic	10cmX20cm	13	Position feedback
Copper tape	3mmX50m	3	Electrical contacts
Sliding contacts	3mm	1024	Electrical contacts
Fishing line	0.5mmX250m	1	Actuation cable
Servo motors	13kg-cm Torque	1	Actuation
Threaded Steel rods	6.35mmX60.9cm	64	Support
PLA Spool	3mm,1kg	1	Pressure sensors

Table 2.4: Material list

Chapter 3

Results

This Chapter presents the results of experiments which were conducted to validate the effectiveness of the new actuator design, the sensor used and the overall system. The next section discusses experiments performed to assess some of the physical characteristics of the linear actuator, such as the maximum force which can be produced, the linearity of the position feedback and the speed of the actuation. The characteristics of the custom built pressure sensor were also determined and the results are provided section 3.2. The second part of the Chapter(sections 3.4 to 3.6) deals with developing novel applications for the shape-changing display such as using it for data visualization, object transfer and manipulation and controlling it through gestures.

3.1 Linear actuator characteristics

Since the linear actuator is spring loaded, there is no force being produced by the servo motor to push the shaft up. Therefore the force for the shaft to resist the change in position when it is pushed down comes from the spring and the friction between the shaft and the sliding contacts. To determine the force required to get the shaft to move, the top of the actuator was loaded with blocks of increasing weight until it started to move and the total weight was recorded. The experiment was performed on 10 randomly selected actuators and the results were recorded and are tabulated in Table 3.1. The average weight was calculated to be 308g, which is 3 times more than the commercial actuators which are used in the relief [16] and inFORM [11] shape displays.

As mentioned in the previous Chapter in section 2.9.1, the position feedback of the linear

Actuator	***Total weight(g)***
1	300
2	300
3	320
4	270
5	300
6	320
7	300
8	350
9	320
10	300

Table 3.1: Measurements of maximum force

Actuator	***Absolute regression coefficient($\|r\|$)***
1	0.9992
2	0.9999
3	0.9993
4	0.9992
5	0.9997
6	0.9998
7	0.9869
8	0.9999
9	0.9984
10	0.9986

Table 3.2: Absolute regression coefficients for position feedback

actuators needed to be calibrated. Before they were calibrated, the position feedback was tested for linearity. This was performed by manually adjusting the height of the shaft in steps of half an inch. The experiment was performed for 10 randomly selected actuators and the results were recorded. The coefficient of regression for the recorded data was calculated and is shown in Table 3.2 and the data set used to calculate them are given in Appendix A for reference. It can be seen that the position feedback mechanism is linear to a good extent and the rest of the actuators can be calibrated on this assumption.

To determine the speed of the actuator, a camera was set up perpendicular to the actuator array. A sheet of paper with the calibrated positions was placed as shown in Figure 3.1. The actuator was made to move from the top most position to the bottom and the motion was recorded at 60fps (frames per second). The recorded video was then analyzed for the time taken for the actuator to cover the specified distance which was used to calculate the speed. The experiment was performed for 5 different actuators and the results are tabulated in Table 3.3 and it was found that the speed

Figure 3.1: Setup for measuring speed of actuators

Actuator	***Upward Speed(cm/s)***	***Downward Speed(cm/s)***
1	14.22	11.85
2	12.19	10.16
3	17.78	11.85
4	12.9	12.7
5	13.2	12.1

Table 3.3: Upward and downward speeds of actuators

of the actuation was slower than similar actuators. This is due to the fact that the actuator can withstand more weight than them. However, the speed could be easily increased through two ways. One way is to make the diameter of the pulley which is attached to the servo motor larger, this would make the winding of the cable faster thereby increasing the speed of the actuator. However, by doing this the size of the overall system would increase as well which is not desirable. Another way to increase the speed of the actuation is to use a spring with lower spring constant that could handle lesser force and using a faster servo motor. The summary of the characteristics of the linear actuator is given in Table 3.4.

Maximum force(g)	***Upward Speed(cm/s)***	***Downward Speed(cm/s)***	***linearity(r)***
308	14.05	11.73	0.9981

Table 3.4: Average characteristics of the linear actuator

3.2 Pressure sensor characteristics

The response of the pressure sensor was tested by loading the top with varying weight blocks and recording the output. The graph of one such measurement is shown in Figure 3.2 where the weight loaded is shown by the red markers. As it can be seen, there is a significant amount of high frequency noise present in the measurement and therefore it cannot be used directly. Since the system only needed a touch sensor in our experiments, the output of the pressure sensor was thresholded at particular value which corresponded to a light touch. This graph and the response are shown in Figure 3.3. However, if needed, higher resolution in sensing can be obtained by filtering the measurement through various filtering techniques.

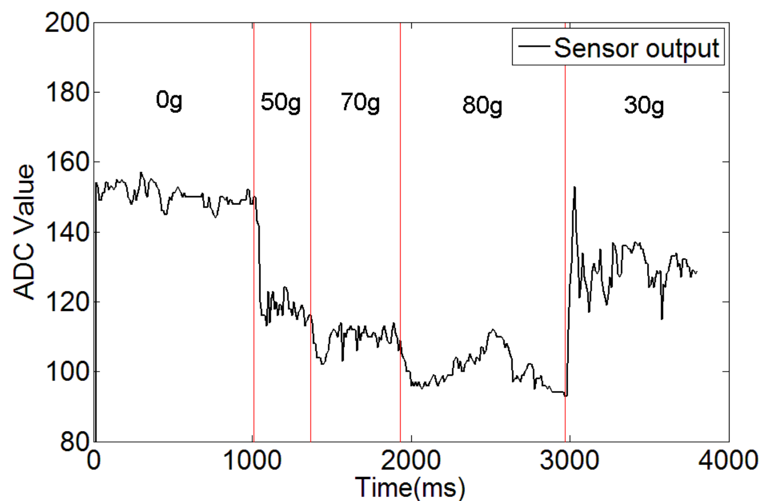


Figure 3.2: Plot of raw pressure sensor output with varying weights

3.3 Power consumption

Each actuator is controlled through a servo motor capable of drawing 2A of current at maximum load. However, the current drawn by each motor was found to be somewhere around 600mA to 1.5A in practice. Therefore for a total of 256 motors the current draw would be in the range of 150A to 380A and at a voltage of 5V this would equate to a maximum power consumption of 1920W. With such a huge power draw heating is a major concern. However, since each of the servo assembly blocks have their own power supply with cooling, this did not prove to be a significant

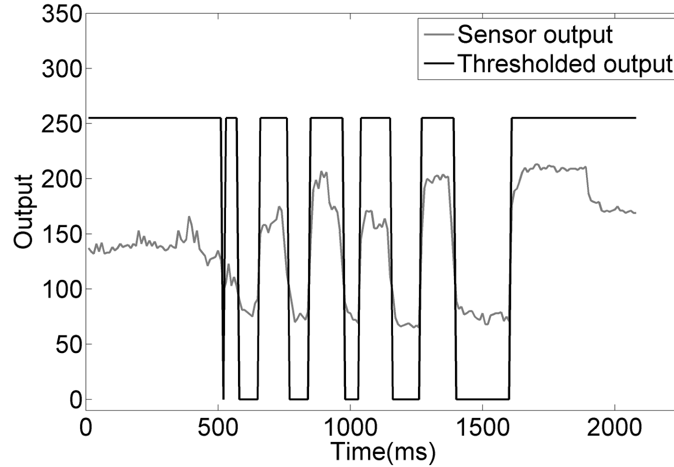


Figure 3.3: Plot of raw and thresholded sensor output

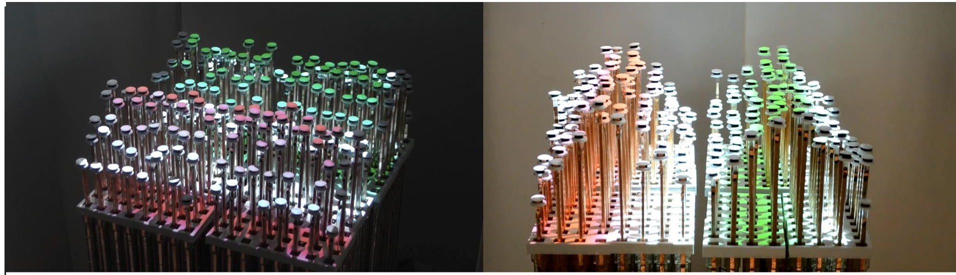
issue in our use of the system.

3.4 Data Visualization

The first application for the shape changing display which was developed was to use it as a tool for data visualization. As mentioned in the introduction, one of the obvious applications is to utilize the dynamic shape-changing ability of the system to visualize an extra dimension of data. Specifically, the system was used to visualize 2.5D data. The 3D point cloud data required for creating the shapes was produced in MATLAB for a variety of mathematical functions. These were then scaled to match the output of the system and stored in data files. The same functions were also used to create depth map images to be displayed using the projector on the surface for better visualization.

To capture the output, two cameras were setup. One was oriented to be pointing horizontally towards the surface and another to obtain a perspective view. The captured images along with the plots and depth maps generated through MATLAB are shown in Figures 3.4 through 3.7. As it can be seen, the system works very well as a tool for 2.5 dimension data visualization. The following link contains the video showing the system being used for data visualization:

Video link: `https://youtu.be/WN4-IgyEf-k`



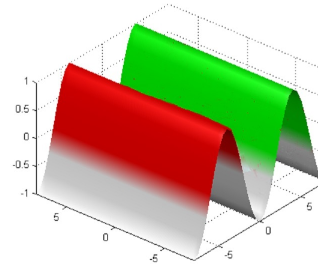
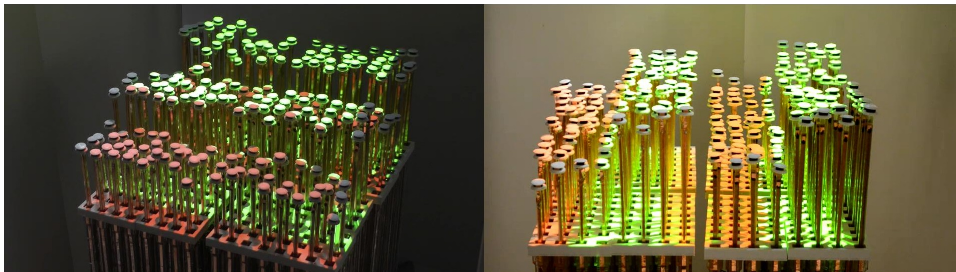
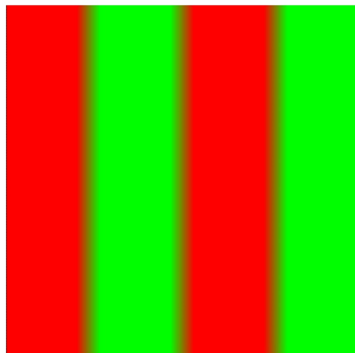


Figure 3.4: Visualization of a sine wave





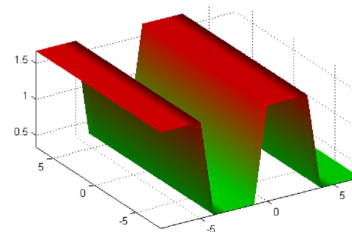
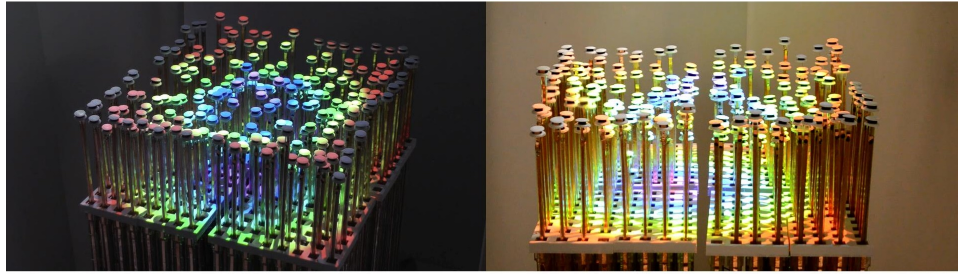
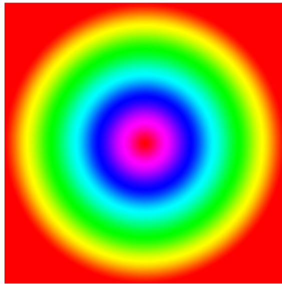


Figure 3.5: Visualization of a bar graph





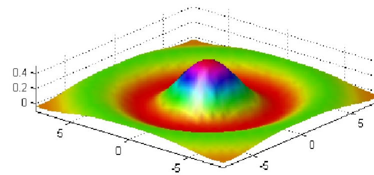
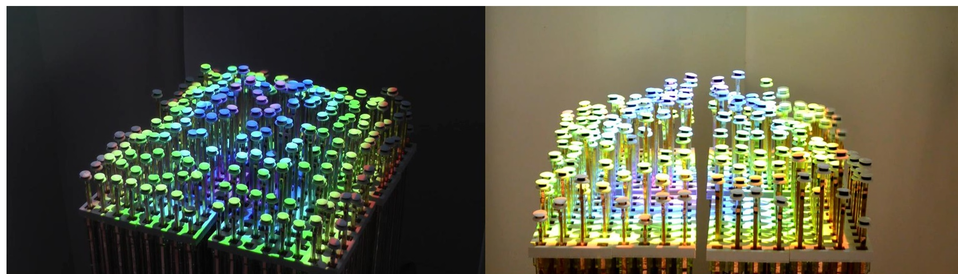
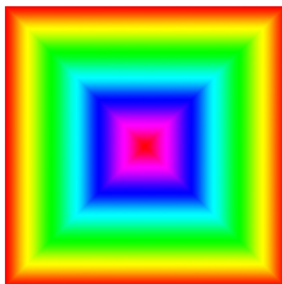


Figure 3.6: Visualization of a sinc function





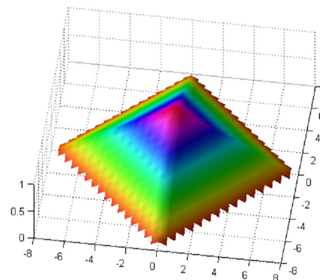


Figure 3.7: Visualization of a pyramid

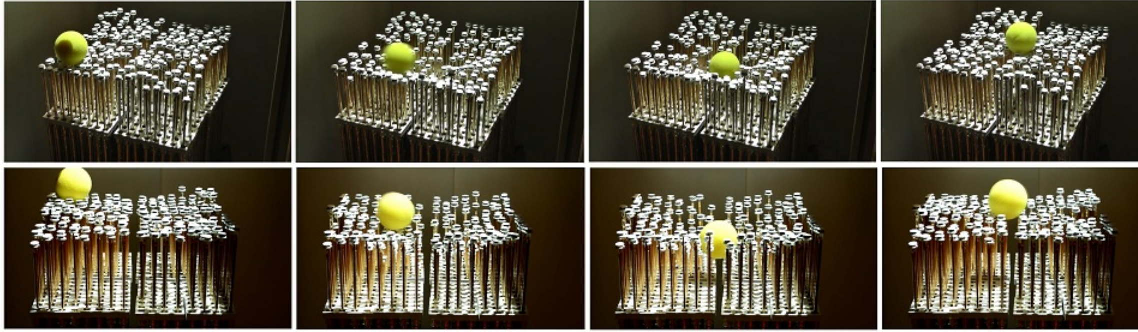


Figure 3.8: Transfer of object

3.5 Object Transfer and Manipulation

One of the ways to utilize the dynamic and tangible aspect of the shape-changing display is to use it to transfer and manipulate objects. To demonstrate this, the system was programmed to move a ball from one corner of the array to the center. Figure 3.8 shows snapshots taken when the ball was being moved. The top row of the figure is from the perspective camera and the bottom row is from the horizontal camera. This application can be extended (through cameras and Computer Vision algorithms) and the system can be used to sort objects based on color, size or any other characteristics on a platform. Because of the enhanced capability to withstand more weight than similar commercial actuators the system can also be used to manipulate relatively heavy objects. This is demonstrated in Figure 3.9 where the system was programmed to manipulate a heavy box. The box was filled with metal pieces and the total weight of the box was 5kg (11 pounds). The system was then programmed to tilt it towards the four corners as shown in Figure 3.9. As it can be seen the system was able to withstand a significant amount of weight. This could be used in cases where there is need to manipulate heavy objects either through automation or human control. Some examples include rocking of containers used for chemical processes and using it as a tilt bed for engraving and scoring at different angles. The following link contains the video showing the system being used for object transfer and manipulation:

Video link: `https://youtu.be/WN4-IgyEf-k?t=27s`

Figure 3.9: Manipulation of heavy object

3.6 Gesture control

To make the system interactive, a gesture based interface was developed as described in the previous Chapter. Figures 3.10 and 3.11 show the system responding to the gesture commands as well as the output from the Kinect sensor. Even though the interface is basic, it demonstrates the use of gestures to control the array. The gestures can be made more intuitive by mounting the kinect overhead the display looking down on it as shown in the inFORM [11] project. This would allow for tracking hands of the user and be able to control the display in a more comfortable manner. The following link contains the video showing the system being controlled through gestures:
Video link: `https://youtu.be/WN4-IgyEf-k?t=53s`

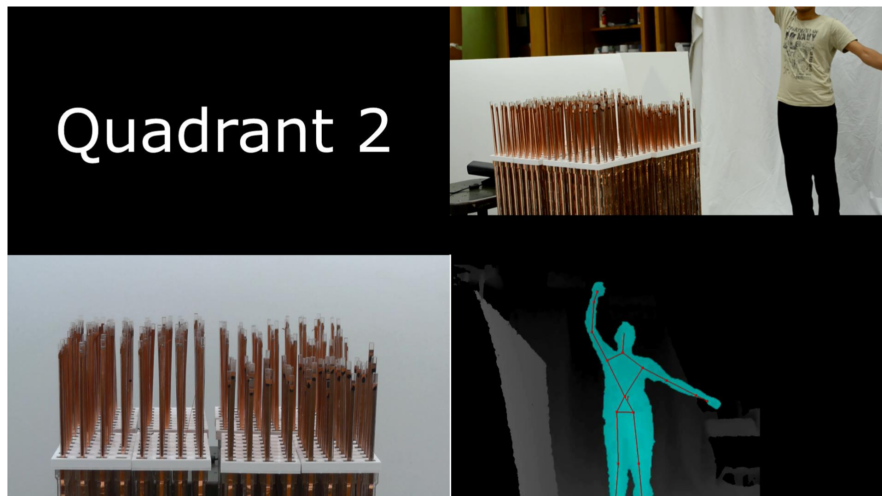


Figure 3.10: Example of gesture control for quadrant 2

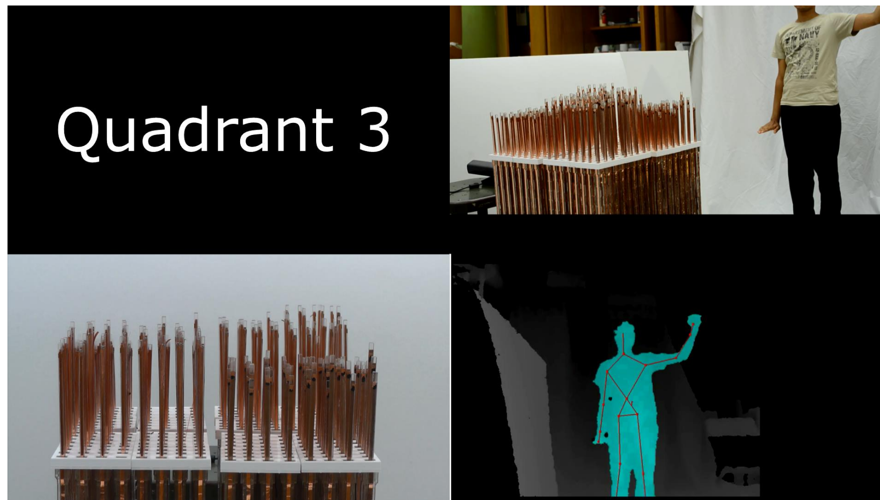


Figure 3.11: Example of gesture control for quadrant 3

3.7 Cost comparison

To analyse the cost effectiveness of the system, a cost comparison was performed with other similar systems.[1] The comparison includes only the cost associated with the actuators as other hardware is either similar of the cost reduction is negligible. Figure 3.12 shows the details of the comparison. The main component of the cost reduction comes from not using commercial linear actuators as well as not using push-pull rods for linkages. The cost per actuator was reduced by a factor of four which could be further improved if these actuators were manufactured in bulk.

	Others*	This display
Actuator	ALPS RSA0N11M9A07 Linear actuator Bulk price - **$29.95**	Servo motor - $6 Fabricated actuator - $4
Actuation link	Gold-N-Rods Nylon rods Price - $7 to $10	30lb Monofilament Fishing line Price – 0.01
Total cost per actuator	$37 to $40	$10

Figure 3.12: Details of the cost comparison

[1] Actuators used in inFORM[11], Relief [16] and EMERGE [22]

Chapter 4

Conclusions and Suggestions for Future Research

In this Thesis, a new design for shape-changing displays which focused on scalability and commercial feasibility was introduced, constructed, and evaluated. This was achieved through novel actuator and sensor designs and packages and by making use of rapid prototyping tools. The resulting built system performed satisfactorily as detailed in the previous Chapter. The new design, which is unique in being scalable and commercially feasible, can allow researchers in various labs to quickly build and develop new applications for shape-changing displays in a way which has not been feasible till now.

During the process of designing new actuators, various mechanisms were considered and tested for their feasibility. This was discussed in Chapter 2 and the design for the final prototype was explained in detail. The new actuator design was used to build a 16x16 shape-changing display and the construction details were also presented in Chapter 2. The choice of using the servo motor as part of the actuation mechanism significantly reduced the complexity and cost. This was because the servo motors have their own control system as well as a motor driver which eliminated several external components. The choice of using the ITO plastic as the resistive strip helped in making sure that the actuator prototypes that were produced were uniform and robust. The modular design helped in the assembly of the servo blocks and improved their scalability as well. The system can be easily expanded by adding more servo assembly blocks to increase the display area. The use of

I2C as a communication protocol allows for the electronics to be scaled up to 64 blocks (each block requires two I2C addresses and 7-bit I2C supports 128 addresses) and if more blocks are needed then only an extra Arduino board has to be added to the system. The resolution of the display can also be increased easily because of the cable mechanism which is used to control the actuators. This means that the actuator area can be shrunk in size while keeping the same servo assembly.

Various characteristics of the actuator and the pressure sensor were determined through experiments described in Chapter 3. The actuator's speed, maximum force which it can withstand and the linearity of its position feedback were calculated and presented. While the maximum force was found to be three times that of similar commercial actuators, the speed was found to be less than originally hoped. However, the speed can be easily increased by using springs which can handle lesser force. This is a result of the inevitable trade-off that has to be made between speed and force of the actuator. The pressure sensors were found to have high frequency noise associated with the signal and they were used as a binary touch sensor. However the resolution of the sensor can be increased by filtering the output.

Characteristics of the actuators and sensors was followed by a series of application oriented experiments which were developed to show the capabilities of the shape-changing display and how they could be used. The applications included data visualization, object manipulation and controlling the system through a gesture interface. These experiments demonstrate the versatility of the system, and highlight its potential for supporting future research in multiple directions.

The system could be improved in several different areas to increase performance. One of the problems encountered during the construction of the actuator was intermittent electrical contact with the resistive strip. This was solved by using conductive ink to form electrical joints, which resulted in uneven resistance. By using other conductive inks which are metal based or by completely replacing the resistive strip better response can be achieved. As mentioned above, the speed of the actuators was fairly low. One way to increase this is by using a spring of lower spring constant which would result in lesser force. Another way is to increase the diameter of the pulley so the speed of the motor is increased, but this would also increase the size of the overall system. A servo motor with a slightly lower torque and higher speed could also be used, as the current motors do not run at their full power.

Currently the DAQ boards read the position of the actuators along with the pressure sensor values, but these values are not utilized. By incorporating the position feedback into the control

system a more tangible way to interact with the system can be developed. The top of the display can also be covered, for example by an elastic silicone sheet to allow for better projection of images and thereby providing better visualization of data. A design change which could be made to provide better visualization is to 3D print the pressure sensor holder with black material so that it blends into the background and the touch plate becomes more visible.

The current shape changing display could be modified for specific applications in future iterations. The most immediate application could be to use it as a 3D screen by covering the display with an elastic material and using 3D projection mapping technology. An example of this is shown in Figure 4.1. It could be also be used for NURBS (Non-Uniform Rational Basis Spline) modeling where the user could create a shape by carefully pushing and pulling on the display surface as shown in Figure 4.2. This could be used to decrease the time required for modeling 3D characters and models where natural curves are involved. A system can also be developed for remote communication between people by having pairs of such screens and using the Kinect to relay human faces for more natural interaction.

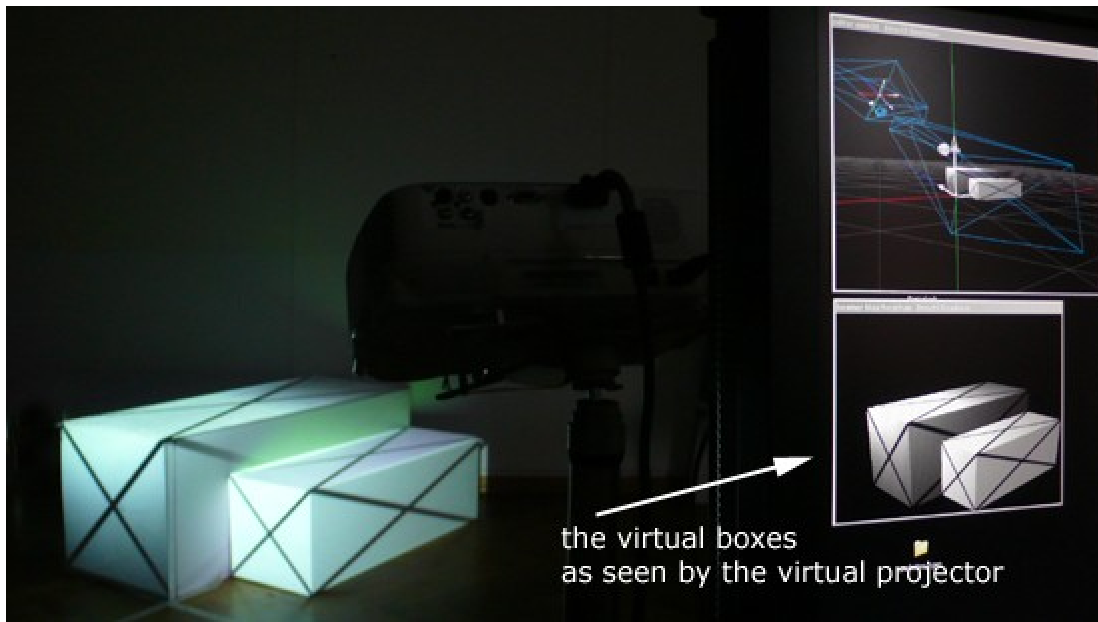


Figure 4.1: Example of 3D projection mapping [7]

The system can also be used in remote medical diagnosing by constructing two human-sized displays. The patient could just walk into a center having one of these units and lie down on it and

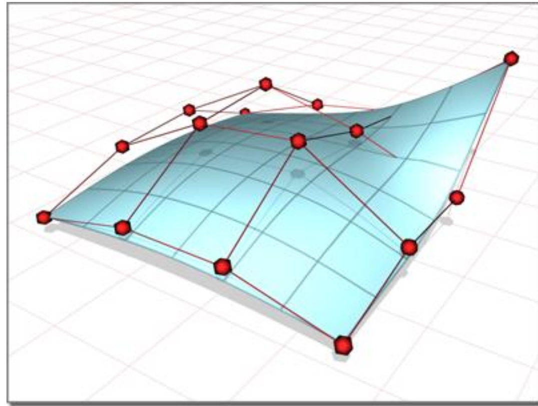


Figure 4.2: Example of NURBS modeling

the physician could diagnose tumors, abnormal growth or any other palpable medical conditions even while being on the other side of the world. A rendering of such a scenario is shown in Figure 4.3.

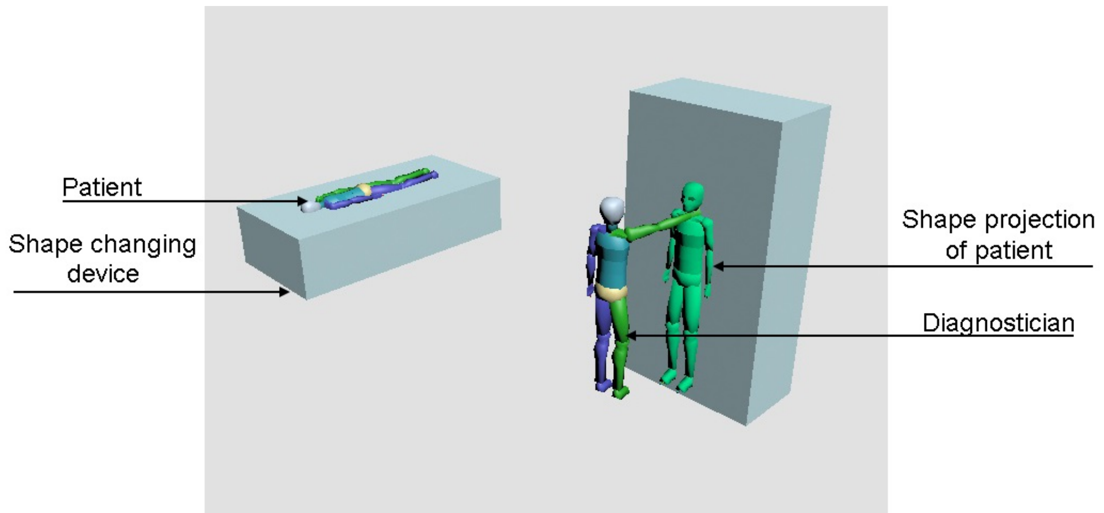


Figure 4.3: Rendering of remote diagnosis scenario

Appendices

Appendix A Calibration data

	Position	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***
Actuator									
1		514	474	438	404	370	336	298	273
2		501	468	433	400	365	330	293	257
3		156	187	228	269	300	335	368	411
4		492	457	424	391	362	335	306	275
5		515	480	441	408	371	332	293	253
6		485	441	399	357	317	275	229	183
7		458	420	382	348	320	278	242	251
8		457	424	391	358	325	292	259	223
9		159	182	205	230	255	282	305	341
10		490	453	420	385	360	329	288	249

Table 1: Servo motor position values for calibration positions

Appendix B MATLAB, Arduino and Processing Code for the system

B.1 MATLAB Code for generating shapes and depth maps

```
1
2  %Code for generating sinc function
3  [X,Y] = meshgrid(-8:.5:8);
4  R = sqrt(X.^2 + Y.^2) + eps;
5  Z = (sin(R)./(R.*2));
6  figure
7  colormap hsv
8  surf(X,Y,Z,'FaceColor','interp',...
9    'EdgeColor','none',...
10   'FaceLighting','gouraud')
11 daspect([5 5 1])
12 axis tight
13 view(-50,30)
14 camlight left
15
16  %Code for generating sine wave
17
18 [X,Y] = meshgrid(-8:.5:8);
19 Z = (sin((X*pi/4)- pi/2));
20 figure
21 map = [1,1,1
22          0,1,0
23         1,0,0 ];
24 colormap(map)
25 surf(X,Y,Z,'FaceColor','interp',...
26   'EdgeColor','none',...
27   'FaceLighting','gouraud')
28 daspect([5 5 1])
29 axis tight
30 view(-50,30)
31 camlight left
32
33  %Code for generating pyramid
34
35 [X,Y] = meshgrid(-8:0.5:8);
36 Z = ((4-abs(X)) + (4-abs(Y)))/8;
37 Z(Z < 0) = NaN;
38 figure
39 colormap hsv
40 surf(X,Y,Z,'FaceColor','interp',...
41     'EdgeColor','none',...
42     'FaceLighting','gouraud')
43 daspect([5 5 1])
44 axis tight
45 view(-50,30)
46 camlight left
47
48  %Code for generating bar chart
49
50 A = importdata('pillar.txt');
51
52 [X,Y] = meshgrid(-8:1:7);
53 Z = A./3;
54
55 figure
```

```
56 map = [0,1,0
57          1,0,0];
58 colormap(map)
59 surf(X,Y,Z,'FaceColor','interp',...
60     'EdgeColor','none',...
61     'FaceLighting','gouraud')
62 daspect([5 5 1])
63 axis tight
64 view(-50,30)
```

B.2 Arduino Code(Slave) for reading sensor values

```
1  #include <Wire.h>
2
3
4  const int sensorPin = A0;  // Analog input pins the multiplexer is attached
5  const int servoPin = A1; // Analog input pins the multiplexer is attached
6  int sensorval[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
7  int servoval[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
8
9  byte sensorvalout[32] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
10 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
11 byte servovalout[16] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
12 byte tempout[32] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
13 16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32};
14
15 int sensorValue = 0;
16 int servoValue = 0;
17 int outputValue = 0;
18 int state = LOW;
19
20 void setup() {
21   // initialize serial communications at 9600 bps:
22   Serial.begin(9600);
23   analogReference(EXTERNAL);
24
25   DDRB = B00111111;
26   PORTB= B00000000;
27    Wire.begin(2);                 // join i2c bus with address #2,
28    //different address for each block
29   Wire.onRequest(requestEvent); // register event
30
31 }
32
33 void loop() {
34   // read the analog in value:
35
36   for(byte i=0;i<16;i++)
37   {
38   PORTB= i;
39   sensorValue = analogRead(sensorPin);
40   servoValue = analogRead(servoPin);
41   sensorval[i] = sensorValue;
42   servoval[i] = servoValue;
43
44   sensorvalout[i] = map(sensorval[i], 0, 1023, 0, 255);
45   sensorvalout[i+16] = map(servoval[i], 0, 1023, 0, 255);
46
47   delay(2);
48
49   }
50
51   // map it to the range of the analog out:
52    // print the results to the serial monitor:
```

```
53
54    for(int i=0;i<32;i++)
55    {
56    Serial.print("sensor ");
57    Serial.print(i);
58    Serial.print(" = ");
59    Serial.println(sensorvalout[i]);
60    }
61    Serial.println();
62
63    // wait 2 milliseconds before the next loop
64    // for the analog-to-digital converter to settle
65    // after the last reading:
66
67  }
68  void requestEvent()
69  {
70    Wire.write(sensorvalout,32);
71  }
```

B.3 Arduino Code(Slave) for reading sensor values

```
1  #include <Wire.h>
2  #include <Adafruit_PWMServoDriver.h>
3
4  byte sensorval[32] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
5  ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
6  byte boardAddress[] = {0x49,0x48,0x47,0x45,0x4b,0x4a,0x46,
7  0x44,0x43,0x42,0x4d,0x4f,0x41,0x40,0x4c,0x4e};
8
9  int servoArray[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
10 int sp=0;
11 int dir[] = {1,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1};
12 Adafruit_PWMServoDriver pwmArray[16];
13
14 uint8_t servonum = 0;
15
16
17
18 int servoNum=0;
19 int pos=0;
20 int sync1=0;
21 int sync2=0;
22 int reset = 0;
23 int boardNum=0;
24 int overshoot=0;
25 void setup() {
26   Serial.begin(9600);
27   Serial.println("16 channel Servo test!");
28
29   for(int i=0;i<16;i++)
30   {
31   pwmArray[i] = Adafruit_PWMServoDriver(boardAddress[i]);
32   pwmArray[i].begin();
33   pwmArray[i].setPWMFreq(60);
34   }
35
36 }
37
38 void loop() {
39   // Drive each servo one at a time
40
41   for(int i=0;i<16;i++)
42   {
```

```
43
44     Wire.requestFrom(i, 32);     // request 6 bytes from slave device #2
45  int i =0;
46    while(Wire.available())     // slave may send l than requested
47    {
48      if(i<32)
49      {
50      sensorval[i] = Wire.read(); // receive a byte as character
51      }
52      i++;
53    }
54
55
56     for(i=0;i<32;i++)
57    {
58    Serial.print("sensor ");
59    Serial.print(i);
60    Serial.print(" = ");
61    Serial.println(sensorval[i]);
62    }
63
64    }
65
66    while (Serial.available() > 0) {
67
68      sync1 = Serial.parseInt();
69      if(sync1 == 97)
70      {
71
72      sync2 = Serial.parseInt();
73      if(sync2 == 76)
74      {
75
76      sp = Serial.parseInt();
77
78
79      if(sp==1)
80      {
81      boardNum =  Serial.parseInt();
82      servoNum = Serial.parseInt();
83      pos = Serial.parseInt();
84
85      }
86
87      else if(sp==2)
88      {
89      boardNum =  Serial.parseInt();
90
91      for(int i=0;i<16;i++)
92      servoArray[i] = Serial.parseInt();
93
94      }
95      }
96      }
97
98      if (Serial.read() == '\n') {
99         break;
100     }
101
102   }
103
104     if((boardNum>0)&&(boardNum<17))
105     {
106
```

```
107           if(sp==1)
108           pwmArray[boardNum-1].setPWM(servoNum, 0, pos );
109           else if(sp==2)
110           {
111           for(int i=0;i<16;i++)
112           pwmArray[boardNum-1].setPWM(i, 0, servoArray[i]);
113           }
114           else
115           {
116             for(int i=0;i<16;i++)
117           pwmArray[boardNum-1].setPWM(i, 0, 0);
118           }
119       }
120
121   }
```

B.4 Processing code for GUI and control

```
1
2   import controlP5.*;
3
4   ControlP5 cp5;
5
6
7   int nx = 4;
8   int ny = 4;
9
10  int servoStep = 2;
11
12  int defServoVal=275;
13
14  int servoMin = 100;
15  int servoMax = 600;
16
17  int servoNum = 0;
18  int servoPos = 0;
19  int calPos   = 0;
20  int blockNum = 0;
21  int calPlayselect  = 0;
22  int overshoot = 0;
23
24  int kinectlevel=0;
25
26  int kinectquad = 0;
27
28
29  int maxCalPositions = 8;
30
31  int[][] calPosMatrix = new int[16][8];
32
33  int[][] playMatrix = new int[16][16];
34  int[][] kinectplayMatrix = new int[16][16];
35  int[][] gridMatrix = new int[16][16];
36  int[][] newPlayMatrix = new int[16][16];
37  int[][] outputMatrix = new int[16][16];
38  int[][][] calfullMatrix = new int[16][16][8];
39
40  int[] dir = {1,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,1,1,1};
41
42  RadioButton r;
43
44  RadioButton calPlay;
45
46  String textValue = "";
```

```
47
48 void setup() {
49   size(1000,600);
50
51   initialize();
52   serialInit();
53
54   PFont font = createFont("arial",24);
55
56   cp5 = new ControlP5(this);
57
58   cp5.addTextfield("Servo Number")
59       .setPosition(40,100)
60       .setSize(100,40)
61       .setFont(font)
62       .setFocus(false)
63       .setColor(color(255,255,255))
64       .getCaptionLabel().setFont(createFont("arial",16))
65       ;
66
67   cp5.addTextfield("Current Position")
68       .setPosition(40,180)
69       .setSize(100,40)
70       .setFont(createFont("arial",20))
71       .setAutoClear(false)
72       .getCaptionLabel().setFont(createFont("arial",16))
73       ;
74
75
76   cp5.addTextfield("Calibration Position")
77       .setPosition(40,260)
78       .setSize(100,40)
79       .setAutoClear(false)
80       .setFont(createFont("arial",20))
81       .getCaptionLabel().setFont(createFont("arial",16))
82       ;
83
84     cp5.addTextfield("Block Number")
85       .setPosition(400,350)
86       .setSize(100,40)
87       .setFont(font)
88       .setAutoClear(false)
89       .getCaptionLabel().setFont(createFont("arial",16))
90       ;
91
92   cp5.addBang("save")
93       .setPosition(40,360)
94       .setSize(100,40)
95       .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
96       ;
97
98         cp5.addBang("play")
99       .setPosition(550,350)
100      .setSize(100,40)
101       .setLabel("Play")
102      .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
103      ;
104
105          cp5.addBang("stop")
106      .setPosition(550,500)
107      .setSize(100,40)
108       .setLabel("Stop")
109      .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
110      ;
```

```
111
112
113         cp5.addBang("reset")
114        .setPosition(550,425)
115        .setSize(100,40)
116         .setLabel("Reset")
117        .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
118         ;
119
120         cp5.addBang("calculatelevels")
121        .setPosition(40,440)
122        .setSize(180,40)
123        .setLabel("Calculate Levels")
124        .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
125         ;
126
127
128        cp5.addBang("shape1")
129        .setPosition(650,100)
130        .setSize(40,40)
131        .setLabel("1")
132        .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
133        ;
134
135       cp5.addBang("shape2")
136        .setPosition(690,100)
137        .setSize(40,40)
138        .setLabel("2")
139        .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
140        ;
141
142         cp5.addBang("shape3")
143        .setPosition(730,100)
144        .setSize(40,40)
145        .setLabel("3")
146        .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
147        ;
148
149         cp5.addBang("shape4")
150        .setPosition(770,100)
151        .setSize(40,40)
152        .setLabel("4")
153        .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
154        ;
155
156         cp5.addBang("shape5")
157        .setPosition(650,140)
158        .setSize(40,40)
159        .setLabel("5")
160        .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
161        ;
162
163        cp5.addBang("shape6")
164        .setPosition(690,140)
165        .setSize(40,40)
166        .setLabel("6")
167        .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
168          ;
169
170        cp5.addBang("shape7")
171        .setPosition(730,140)
172        .setSize(40,40)
173        .setLabel("7")
174        .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
```

```
175     ;
176
177     cp5.addBang("shape8")
178     .setPosition(770,140)
179     .setSize(40,40)
180     .setLabel("8")
181     .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER).setFont(createFont("arial",16))
182
183     ;
184
185      r = cp5.addRadioButton("radioButton")
186         .setPosition(400,100)
187         .setSize(40,40)
188         .setColorForeground(color(120))
189         .setColorActive(color(255))
190         .setColorLabel(color(255))
191         .setItemsPerRow(4)
192
193         .setSpacingColumn(1)
194         .addItem("1",1)
195         .addItem("2",2)
196         .addItem("3",3)
197         .addItem("4",4)
198         .addItem("5",5)
199         .addItem("6",6)
200         .addItem("7",7)
201         .addItem("8",8)
202         .addItem("9",9)
203         .addItem("10",10)
204         .addItem("11",11)
205         .addItem("12",12)
206         .addItem("13",13)
207         .addItem("14",14)
208         .addItem("15",15)
209         .addItem("16",16)
210
211         ;
212
213         calPlay = cp5.addRadioButton("CalSelect")
214         .setPosition(400,450)
215         .setSize(40,40)
216         .setColorForeground(color(120))
217         .setColorActive(color(255))
218         .setColorLabel(color(255))
219         .setItemsPerRow(1)
220
221         .setSpacingColumn(1)
222         .addItem("Calibrate",1)
223         .addItem("Playback",2)
224         .addItem("Kinect",3)
225
226
227         ;
228
229     for(Toggle t:r.getItems()) {
230       t.captionLabel().setColorBackground(color(0,0));
231       t.captionLabel().style().moveMargin(-7,0,0,-3);
232       t.captionLabel().style().movePadding(7,0,0,3);
233       t.captionLabel().style().backgroundWidth = 1;
234       t.captionLabel().style().backgroundHeight = 1;
235     }
236
237     for(Toggle t:calPlay.getItems()) {
238       t.captionLabel().setFont(createFont("arial",16));
```

```
239
240         }
241
242
243     textFont(font);
244   }
245
246   void draw() {
247     background(0);
248     fill(255);
249     //text(cp5.get(Textfield.class,"Servo Number").getText(), 360,130);
250     text("Calibration", 325,50);
251
252     cp5.get(Textfield.class,"Servo Number").setValue(str(servoNum+1));
253     cp5.get(Textfield.class,"Current Position").setValue(str(servoPos));
254     cp5.get(Textfield.class,"Calibration Position").setValue(str(calPos+1));
255
256   }
257
258   public void save() {
259     writecalMatrix();
260   }
261
262
263   public void play() {
264     loadPlayMatrix1();
265     playPMatrix();
266
267     //delay(3000);
268
269
270     //reset();
271
272   }
273   public void shape1() {
274     loadshape1();
275     playPMatrix();
276
277   }
278
279   public void shape2() {
280     loadshape2();
281     playPMatrix();
282
283   }
284
285   public void shape3() {
286     loadshape3();
287     playPMatrix();
288
289   }
290
291   public void shape4() {
292     loadshape4();
293     playPMatrix();
294
295   }
296
297   public void shape5() {
298     loadshape5();
299     playPMatrix();
300
301   }
302
```

```
303 public void shape6() {
304   loadshape6();
305   playPMatrix();
306
307 }
308
309 public void shape7() {
310   loadshape7();
311   playPMatrix();
312
313 }
314
315 public void shape8() {
316   loadshape8();
317   playPMatrix();
318
319 }
320 public void reset() {
321   loadResetMatrix();
322   playPMatrix();
323   delay(500);
324
325   stop();
326
327
328 }
329
330 public void stop() {
331  sendstopall();
332  // sendstoplast();
333 }
334
335 public void calculatelevels() {
336 //writecalMatrix();
337 calcLevels();
338 }
339
340 void controlEvent(ControlEvent theEvent) {
341   if(theEvent.isAssignableFrom(Textfield.class)) {
342     println("controlEvent: accessing a string from controller '"
343             +theEvent.getName()+"': "
344             +theEvent.getStringValue()
345             );
346   }
347
348     if(theEvent.isFrom(r)) {
349
350     //myColorBackground = color(int(theEvent.group().value()*50),0,0);
351     blockNum = (int)theEvent.group().value();
352     cp5.get(Textfield.class,"Block Number").setValue(str(blockNum));
353     loadcalMatrix();
354   }
355
356    if(theEvent.isFrom(calPlay)) {
357
358     //myColorBackground = color(int(theEvent.group().value()*50),0,0);
359     calPlayselect = (int)theEvent.group().value();
360   }
361
362 }
363
364
365 public void input(String theText) {
366   // automatically receives results from controller input
```

```
367    println("a␣textfield␣event␣for␣controller␣'input'␣:␣"+theText);
368  }
369
370  void keyPressed() {
371
372
373     int keyIndex = 0;
374    if (key >= '1' && key <= '8') {
375      keyIndex = key - '1';
376       calPos = keyIndex;
377       servoPos = calPosMatrix[servoNum][calPos];
378
379       if(calPlayselect==2)
380       {
381       sendServoPos();
382       }
383    }
384
385
386    if(calPlayselect==3)
387    {
388
389    if(key == 'h')
390    {
391    kinectlevel = 1;
392    updatekinectMatrix();
393    }
394
395    else if(key == 'j')
396    {
397    kinectlevel = 2;
398    updatekinectMatrix();
399    }
400
401    else if(key == 'k')
402    {
403    kinectlevel = 3;
404    updatekinectMatrix();
405    }
406
407    else if(key == 'l')
408    {
409    kinectlevel = 4;
410    updatekinectMatrix();
411    }
412
413    else if(key == 'a')
414    {
415    kinectquad = 1;
416    updatekinectMatrix();
417    }
418
419    else if(key == 'b')
420   {
421    kinectquad = 2;
422    updatekinectMatrix();
423    }
424
425    else if(key == 'c')
426    {
427    kinectquad = 3;
428    updatekinectMatrix();
429    }
430
```

```
431     else if(key == 'd')
432     {
433     kinectquad = 4;
434     updatekinectMatrix();
435     }
436
437     }
438
439     else
440     {
441
442       kinectlevel = 0;
443       kinectquad = 0;
444
445     updatekinectMatrix();
446
447
448     }
449
450     if (key == CODED)
451     {
452
453       if (keyCode == UP)
454       {
455
456       servoPos+=dir[servoNum]*servoStep;
457
458         if(servoPos<servoMin)
459        servoPos=servoMin;
460        if(servoPos>servoMax)
461        servoPos=servoMax;
462
463        calPosMatrix[servoNum][calPos] = servoPos;
464
465        sendServoPos();
466
467       }
468       else if (keyCode == DOWN)
469       {
470
471        servoPos-=dir[servoNum]*servoStep;
472
473         if(servoPos<servoMin)
474        servoPos=servoMin;
475        if(servoPos>servoMax)
476        servoPos=servoMax;
477
478       calPosMatrix[servoNum][calPos] = servoPos;
479       sendServoPos();
480
481       }
482
483       else if (keyCode == LEFT)
484       {
485         if(servoNum>0)
486       servoNum-=1;
487
488       servoPos = calPosMatrix[servoNum][calPos];
489
490
491       }
492
493       else if (keyCode == RIGHT)
494       {
```

```
495           if(servoNum<15)
496         servoNum+=1;
497
498         servoPos = calPosMatrix[servoNum][calPos];
499
500         }
501
502
503         else if (keyCode == 16) {
504
505           sendResetAll();
506         }
507
508         }
509
510
511   }
512   void initialize()
513
514   {
515   for(int i=0;i<16;i++)
516   for(int j=0;j<maxCalPositions;j++)
517   calPosMatrix[i][j]=defServoVal;
518
519   for(int i=0;i<16;i++)
520   for(int j=0;j<16;j++)
521   {
522   kinectplayMatrix[i][j]=0;
523   playMatrix[i][j]=0;
524   gridMatrix[i][j]=0;
525   newPlayMatrix[i][j]=0;
526   outputMatrix[i][j]=0;
527   for(int k=0;k<8;k++)
528   calfullMatrix[i][j][k]=0;
529   }
530   loadgridMatrix();
531   loadfullcalMatrix();
532
533
534   }
535   void writecalMatrix()
536   {
537
538     String[] lines = new String[16];
539     for (int i = 0; i < 16; i++) {
540       lines[i] = str(i);
541       for(int j = 0;j<8;j++)
542       lines[i] += "\t" + calPosMatrix[i][j];
543     }
544       String filename = str(blockNum) + ".txt";
545     saveStrings(filename, lines);
546
547
548   }
549   void writenewPlayMatrix()
550   {
551
552     String[] lines = new String[16];
553     for (int i = 0; i < 16; i++) {
554       lines[i] = ""+newPlayMatrix[i][0];
555       for(int j = 1;j<16;j++)
556       lines[i] += "\t" + newPlayMatrix[i][j];
557     }
558       String filename = "playout.txt";
```

```
559   saveStrings(filename, lines);
560
561
562 }
563 void writeoutputMatrix()
564 {
565
566   String[] lines = new String[16];
567   for (int i = 0; i < 16; i++) {
568     lines[i] = ""+outputMatrix[i][0];
569     for(int j = 1;j<16;j++)
570     lines[i] += "\t" + outputMatrix[i][j];
571   }
572     String filename = "output.txt";
573   saveStrings(filename, lines);
574
575
576 }
577 void writefullcalMatrix()
578 {
579
580   String[] lines = new String[16];
581   for (int i = 0; i < 16; i++) {
582     lines[i] = ""+calfullMatrix[i][0][0];
583     for(int k = 1;k<8;k++)
584     lines[i] += "\t" + calfullMatrix[i][0][k];
585   }
586     String filename = "fullcal.txt";
587   saveStrings(filename, lines);
588
589
590 }
591
592 void loadcalMatrix()
593 {
594   String[] input;
595   String filename = str(blockNum) + ".txt";
596   input = loadStrings(filename);
597
598   if(input!=null)
599   {
600
601
602   for(int i=0;i<16;i++)
603   {
604   String[] pieces = split(input[i], '\t');
605     print(pieces.length);
606
607     for(int j=0;j<8;j++)
608     {
609        calPosMatrix[i][j] = int(pieces[j+1]);
610     }
611
612   }
613 }
614
615 servoPos = calPosMatrix[servoNum][calPos];
616
617 }
618 void loadfullcalMatrix()
619 {
620   int newi=0;
621   int newj=0;
622   for(int i=0;i<16;i++)
```

```
623     {
624     blockNum=i+1;
625     loadcalMatrix();
626
627
628     for(int j=0;j<16;j++)
629     {
630       newi = 4*(i/4) + j/4;
631       newj = 4*(i%4) + (j%4);
632
633       for(int k=0;k<8;k++)
634       {
635
636           calfullMatrix[newi][newj][k] = calPosMatrix[j][k];
637       }
638
639     }
640     }
641
642
643
644 }
645 void loadPlayMatrix1()
646 {
647     String[] input;
648     String filename = "sin1.txt";
649     input = loadStrings(filename);
650
651     if(input!=null)
652     {
653
654
655     for(int i=0;i<16;i++)
656     {
657     String[] pieces = split(input[i], '\t');
658       //print(pieces.length);
659
660       for(int j=0;j<16;j++)
661       {
662           playMatrix[i][j] = int(pieces[j]);
663       }
664
665     }
666 }
667
668 //servoPos = calPosMatrix[servoNum][calPos];
669
670 }
671 void loadPlayMatrix2()
672 {
673     String[] input;
674     String filename = "bowl.txt";
675     input = loadStrings(filename);
676
677     if(input!=null)
678     {
679
680     for(int i=0;i<16;i++)
681     {
682     String[] pieces = split(input[i], '\t');
683       //print(pieces.length);
684
685       for(int j=0;j<16;j++)
686       {
```

```
687         playMatrix[i][j] = int(pieces[j]);
688       }
689
690     }
691   }
692
693   //servoPos = calPosMatrix[servoNum][calPos];
694
695   }
696   void loadshape1()
697   {
698     String[] input;
699     String filename = "sin1.txt";
700     input = loadStrings(filename);
701
702     if(input!=null)
703     {
704
705
706     for(int i=0;i<16;i++)
707     {
708     String[] pieces = split(input[i], '\t');
709       //print(pieces.length);
710
711       for(int j=0;j<16;j++)
712       {
713         playMatrix[i][j] = int(pieces[j]);
714       }
715
716     }
717   }
718
719   //servoPos = calPosMatrix[servoNum][calPos];
720
721   }
722   void loadshape2()
723   {
724     String[] input;
725     String filename = "sin2.txt";
726     input = loadStrings(filename);
727
728     if(input!=null)
729     {
730
731
732     for(int i=0;i<16;i++)
733     {
734     String[] pieces = split(input[i], '\t');
735       //print(pieces.length);
736
737       for(int j=0;j<16;j++)
738       {
739         playMatrix[i][j] = int(pieces[j]);
740       }
741
742     }
743   }
744
745   //servoPos = calPosMatrix[servoNum][calPos];
746
747   }
748   void loadshape3()
749   {
750     String[] input;
```

```
751     String filename =  "pyramid.txt";
752     input = loadStrings(filename);
753
754     if(input!=null)
755     {
756
757
758     for(int i=0;i<16;i++)
759     {
760     String[] pieces = split(input[i], '\t');
761       //print(pieces.length);
762
763       for(int j=0;j<16;j++)
764       {
765          playMatrix[i][j] = int(pieces[j]);
766       }
767
768     }
769   }
770
771   //servoPos = calPosMatrix[servoNum][calPos];
772
773   }
774
775   void loadshape4()
776   {
777     String[] input;
778     String filename =  "bowl.txt";
779     input = loadStrings(filename);
780
781     if(input!=null)
782     {
783
784
785     for(int i=0;i<16;i++)
786     {
787     String[] pieces = split(input[i], '\t');
788       //print(pieces.length);
789
790       for(int j=0;j<16;j++)
791       {
792          playMatrix[i][j] = int(pieces[j]);
793       }
794
795     }
796   }
797
798   //servoPos = calPosMatrix[servoNum][calPos];
799
800   }
801   void loadshape5()
802   {
803     String[] input;
804     String filename =  "ball1.txt";
805     input = loadStrings(filename);
806
807     if(input!=null)
808     {
809
810
811     for(int i=0;i<16;i++)
812     {
813     String[] pieces = split(input[i], '\t');
814       //print(pieces.length);
```

```
815
816       for(int j=0;j<16;j++)
817       {
818          playMatrix[i][j] = int(pieces[j]);
819       }
820
821    }
822 }
823
824 //servoPos = calPosMatrix[servoNum][calPos];
825
826 }
827 void loadshape6()
828 {
829    String[] input;
830    String filename = "ball2.txt";
831    input = loadStrings(filename);
832
833    if(input!=null)
834    {
835
836
837    for(int i=0;i<16;i++)
838    {
839    String[] pieces = split(input[i], '\t');
840       //print(pieces.length);
841
842       for(int j=0;j<16;j++)
843       {
844          playMatrix[i][j] = int(pieces[j]);
845       }
846
847    }
848 }
849
850 //servoPos = calPosMatrix[servoNum][calPos];
851
852 }
853
854
855 void loadshape7()
856 {
857    String[] input;
858    String filename = "ball3.txt";
859    input = loadStrings(filename);
860
861    if(input!=null)
862    {
863
864
865    for(int i=0;i<16;i++)
866    {
867    String[] pieces = split(input[i], '\t');
868       //print(pieces.length);
869
870       for(int j=0;j<16;j++)
871       {
872          playMatrix[i][j] = int(pieces[j]);
873       }
874
875    }
876 }
877
878 //servoPos = calPosMatrix[servoNum][calPos];
```

```
879
880 }
881
882 void loadshape8()
883 {
884   String[] input;
885   String filename = "ball4.txt";
886   input = loadStrings(filename);
887
888   if(input!=null)
889   {
890
891
892   for(int i=0;i<16;i++)
893   {
894   String[] pieces = split(input[i], '\t');
895     //print(pieces.length);
896
897     for(int j=0;j<16;j++)
898     {
899        playMatrix[i][j] = int(pieces[j]);
900     }
901
902   }
903 }
904
905
906 }
907
908
909 void loadgridMatrix()
910 {
911   String[] input;
912   String filename = "grid.txt";
913   input = loadStrings(filename);
914
915   if(input!=null)
916   {
917
918
919   for(int i=0;i<16;i++)
920   {
921   String[] pieces = split(input[i], '\t');
922     //print(pieces.length);
923
924     for(int j=0;j<16;j++)
925     {
926        gridMatrix[i][j] = int(pieces[j]);
927     }
928
929   }
930 }
931
932 //servoPos = calPosMatrix[servoNum][calPos];
933
934 }
935
936 void convertMatrix()
937 {
938   int newi = 0;
939   int newj = 0;
940
941   for(int i=0;i<16;i++)
942   {
```

```
943
944       for(int j=0;j<16;j++)
945       {
946         newi = 4*(i/4) + (gridMatrix[i][j] - 1)/4;
947         newj = 4*(j/4) + (gridMatrix[i][j] - 1)%4;
948
949         if(playMatrix[i][j]>0)
950         {
951         newPlayMatrix[newi][newj] = playMatrix[i][j];
952         }
953         else
954         {
955         newPlayMatrix[newi][newj] = 0;
956         }
957
958       }
959
960    }
961
962 }
963
964 void loadResetMatrix()
965 {
966
967    String[] input;
968    String filename = "reset.txt";
969    input = loadStrings(filename);
970
971    if(input!=null)
972    {
973
974
975    for(int i=0;i<16;i++)
976    {
977    String[] pieces = split(input[i], '\t');
978       //print(pieces.length);
979
980       for(int j=0;j<16;j++)
981       {
982          playMatrix[i][j] = int(pieces[j]);
983       }
984
985    }
986 }
987
988 }
989
990 void loadOutputMatrix()
991 {
992
993
994    for(int i=0;i<16;i++)
995    {
996
997       for(int j=0;j<16;j++)
998       {
999          if(newPlayMatrix[i][j]>0)
1000          outputMatrix[i][j] = calfullMatrix[i][j][newPlayMatrix[i][j]-1];
1001          else
1002          outputMatrix[i][j]=0;
1003       }
1004
1005    }
1006
```

```
1007  }
1008
1009  void calcLevels()
1010  {
1011  int base = 0;
1012  int step = 0;
1013  int top = 0;
1014
1015  for(int i=0;i<16;i++)
1016  {
1017
1018    base = calPosMatrix[i][0];
1019    top  = calPosMatrix[i][7];
1020
1021    step = (top-base)/7;
1022
1023    for(int j=1;j<7;j++)
1024    calPosMatrix[i][j] = base + j*step;
1025
1026  }
1027
1028  }
1029
1030  void updatekinectMatrix()
1031
1032  {
1033
1034  if(kinectquad > 0)
1035
1036  {
1037
1038    int iOff = 8*((kinectquad-1) % 2);
1039    int jOff = 8*((kinectquad-1) / 2);
1040
1041    for(int i=0;i<8;i++)
1042    for(int j=0;j<8;j++)
1043    {
1044
1045    kinectplayMatrix[iOff+i][jOff+j] = kinectlevel;
1046
1047    }
1048
1049    playMatrix = kinectplayMatrix;
1050    playPMatrix();
1051
1052  }
1053
1054    String[] lines = new String[16];
1055    for (int i = 0; i < 16; i++) {
1056      lines[i] = ""+kinectplayMatrix[i][0];
1057      for(int j = 1;j<16;j++)
1058      lines[i] += "\t" + kinectplayMatrix[i][j];
1059    }
1060      String filename = "kinectplayMatrix.txt";
1061    saveStrings(filename, lines);
1062
1063  }
```

B.5 XML Data for gesture mapping in FAAST


```
1
2
3
4   <sensor>
5           <tracker>Microsoft</tracker>
6           <mode>Full Body</mode>
7           <mirrormode>true</mirrormode>
8           <smoothing>true</smoothing>
9           <smoothingfactor>0.5</smoothingfactor>
10          <correction>0.5</correction>
11          <prediction>0.5</prediction>
12          <jitter>0.05</jitter>
13          <deviation>0.04</deviation>
14  </sensor>
15  <server>
16          <transformations>Global coordinates</transformations>
17          <automaticassignment>true</automaticassignment>
18  </server>
19  <display>
20          <showviewerwindow>true</showviewerwindow>
21          <background>RGB</background>
22          <foreground>Depth</foreground>
23          <showconsolewindow>true</showconsolewindow>
24          <movechildwindows>true</movechildwindows>
25          <savewindowlayout>false</savewindowlayout>
26  </display>
27  <gestures>
28          <gesture name="Block2" timeout="0" enabled="true" outputloop="false" outputtimeout="0">
29                  <input type="1">
30                          <descriptor>left wrist</descriptor>
31                          <descriptor>to the left of</descriptor>
32                          <descriptor>head</descriptor>
33                          <descriptor>at most</descriptor>
34                          <descriptor>40</descriptor>
35                          <descriptor>centimeters</descriptor>
36                  </input>
37                  <input type="1">
38                          <descriptor>left wrist</descriptor>
39                          <descriptor>above</descriptor>
40                          <descriptor>left shoulder</descriptor>
41                          <descriptor>at least</descriptor>
42                          <descriptor>10</descriptor>
43                          <descriptor>centimeters</descriptor>
44                  </input>
45                  <output type="0">
46                          <descriptor>press</descriptor>
47                          <descriptor>b</descriptor>
48                          <descriptor>for</descriptor>
49                          <descriptor>0</descriptor>
50                  </output>
51          </gesture>
52          <gesture name="Block4" timeout="0" enabled="true" outputloop="false" outputtimeout="0">
53                  <input type="1">
54                          <descriptor>left wrist</descriptor>
55                          <descriptor>to the left of</descriptor>
56                          <descriptor>head</descriptor>
57                          <descriptor>at most</descriptor>
58                          <descriptor>40</descriptor>
59                          <descriptor>centimeters</descriptor>
60                  </input>
61                  <input type="1">
62                          <descriptor>left wrist</descriptor>
```

```
63                  <descriptor>below</descriptor>
64                  <descriptor>left shoulder</descriptor>
65                  <descriptor>at least</descriptor>
66                  <descriptor>10</descriptor>
67                  <descriptor>centimeters</descriptor>
68              </input>
69              <output type="0">
70                  <descriptor>press</descriptor>
71                  <descriptor>d</descriptor>
72                  <descriptor>for</descriptor>
73                  <descriptor>0</descriptor>
74              </output>
75          </gesture>
76          <gesture name="Block1" timeout="0" enabled="true" outputloop="false" outputtimeout="0">
77              <input type="1">
78                  <descriptor>left wrist</descriptor>
79                  <descriptor>to the left of</descriptor>
80                  <descriptor>head</descriptor>
81                  <descriptor>at least</descriptor>
82                  <descriptor>40</descriptor>
83                  <descriptor>centimeters</descriptor>
84              </input>
85              <input type="1">
86                  <descriptor>left wrist</descriptor>
87                  <descriptor>above</descriptor>
88                  <descriptor>left shoulder</descriptor>
89                  <descriptor>at least</descriptor>
90                  <descriptor>10</descriptor>
91                  <descriptor>centimeters</descriptor>
92              </input>
93              <output type="0">
94                  <descriptor>press</descriptor>
95                  <descriptor>a</descriptor>
96                  <descriptor>for</descriptor>
97                  <descriptor>0</descriptor>
98              </output>
99          </gesture>
100         <gesture name="Block3" timeout="0" enabled="true" outputloop="false" outputtimeout="0">
101             <input type="1">
102                 <descriptor>left wrist</descriptor>
103                 <descriptor>to the left of</descriptor>
104                 <descriptor>head</descriptor>
105                 <descriptor>at least</descriptor>
106                 <descriptor>40</descriptor>
107                 <descriptor>centimeters</descriptor>
108             </input>
109             <input type="1">
110                 <descriptor>left wrist</descriptor>
111                 <descriptor>below</descriptor>
112                 <descriptor>left shoulder</descriptor>
113                 <descriptor>at least</descriptor>
114                 <descriptor>10</descriptor>
115                 <descriptor>centimeters</descriptor>
116             </input>
117             <output type="0">
118                 <descriptor>press</descriptor>
119                 <descriptor>c</descriptor>
120                 <descriptor>for</descriptor>
121                 <descriptor>0</descriptor>
122             </output>
123         </gesture>
124         <gesture name="Level1" timeout="0" enabled="true" outputloop="false" outputtimeout="0">
125             <input type="1">
126                 <descriptor>right wrist</descriptor>
```

```
127                         <descriptor>below</descriptor>
128                         <descriptor>right hip</descriptor>
129                         <descriptor>at least</descriptor>
130                         <descriptor>1</descriptor>
131                         <descriptor>centimeters</descriptor>
132                 </input>
133                 <input type="1">
134                         <descriptor>right wrist</descriptor>
135                         <descriptor>to the right of</descriptor>
136                         <descriptor>head</descriptor>
137                         <descriptor>at least</descriptor>
138                         <descriptor>10</descriptor>
139                         <descriptor>centimeters</descriptor>
140                 </input>
141                 <output type="0">
142                         <descriptor>press</descriptor>
143                         <descriptor>l</descriptor>
144                         <descriptor>for</descriptor>
145                         <descriptor>0</descriptor>
146                 </output>
147         </gesture>
148         <gesture name="Level2" timeout="0" enabled="true" outputloop="false" outputtimeout="0">
149                 <input type="1">
150                         <descriptor>right wrist</descriptor>
151                         <descriptor>above</descriptor>
152                         <descriptor>right hip</descriptor>
153                         <descriptor>at least</descriptor>
154                         <descriptor>10</descriptor>
155                         <descriptor>centimeters</descriptor>
156                 </input>
157                 <input type="1">
158                         <descriptor>right wrist</descriptor>
159                         <descriptor>to the right of</descriptor>
160                         <descriptor>head</descriptor>
161                         <descriptor>at least</descriptor>
162                         <descriptor>10</descriptor>
163                         <descriptor>centimeters</descriptor>
164                 </input>
165                 <input type="1">
166                         <descriptor>right wrist</descriptor>
167                         <descriptor>below</descriptor>
168                         <descriptor>right shoulder</descriptor>
169                         <descriptor>at least</descriptor>
170                         <descriptor>20</descriptor>
171                         <descriptor>centimeters</descriptor>
172                 </input>
173                 <output type="0">
174                         <descriptor>press</descriptor>
175                         <descriptor>k</descriptor>
176                         <descriptor>for</descriptor>
177                         <descriptor>0</descriptor>
178                 </output>
179         </gesture>
180         <gesture name="Level3" timeout="0" enabled="true" outputloop="false" outputtimeout="0">
181                 <input type="1">
182                         <descriptor>right wrist</descriptor>
183                         <descriptor>above</descriptor>
184                         <descriptor>right hip</descriptor>
185                         <descriptor>at least</descriptor>
186                         <descriptor>40</descriptor>
187                         <descriptor>centimeters</descriptor>
188                 </input>
189                 <input type="1">
190                         <descriptor>right wrist</descriptor>
```

```
191                         <descriptor>to the right of</descriptor>
192                         <descriptor>head</descriptor>
193                         <descriptor>at least</descriptor>
194                         <descriptor>10</descriptor>
195                         <descriptor>centimeters</descriptor>
196                 </input>
197                 <input type="1">
198                         <descriptor>right wrist</descriptor>
199                         <descriptor>below</descriptor>
200                         <descriptor>head</descriptor>
201                         <descriptor>at least</descriptor>
202                         <descriptor>10</descriptor>
203                         <descriptor>centimeters</descriptor>
204                 </input>
205                 <output type="0">
206                         <descriptor>press</descriptor>
207                         <descriptor>j</descriptor>
208                         <descriptor>for</descriptor>
209                         <descriptor>0</descriptor>
210                 </output>
211         </gesture>
212         <gesture name="Level4" timeout="0" enabled="true" outputloop="false" outputtimeout="0">
213                 <input type="1">
214                         <descriptor>right wrist</descriptor>
215                         <descriptor>above</descriptor>
216                         <descriptor>head</descriptor>
217                         <descriptor>at least</descriptor>
218                         <descriptor>10</descriptor>
219                         <descriptor>centimeters</descriptor>
220                 </input>
221                 <input type="1">
222                         <descriptor>right wrist</descriptor>
223                         <descriptor>to the right of</descriptor>
224                         <descriptor>head</descriptor>
225                         <descriptor>at least</descriptor>
226                         <descriptor>10</descriptor>
227                         <descriptor>centimeters</descriptor>
228                 </input>
229                 <output type="0">
230                         <descriptor>press</descriptor>
231                         <descriptor>h</descriptor>
232                         <descriptor>for</descriptor>
233                         <descriptor>0</descriptor>
234                 </output>
235         </gesture>
236 </gestures>
237 <plugins/>
```

References

[1] Arduino. `https://www.arduino.cc/`. Accessed: 2013-09-15.

[2] Autodesk Inventor. `http://www.autodesk.com/products/inventor/overview`. Accessed: 2013-09-15.

[3] EAGLE. `http://www.cadsoftusa.com/eagle-pcb-design-software/about-eagle/`. Accessed: 2014-06-12.

[4] FAAST. `http://projects.ict.usc.edu/mxr/faast/`. Accessed: 2015-04-10.

[5] Microsoft Kinect. `https://www.microsoft.com/en-us/kinectforwindows/`. Accessed: 2015-04-10.

[6] Northrop Grumman Terrain Table. `http://www.is.northropgrumman.com/products/terraintable/index.html`. Accessed: 2014-09-10.

[7] vvvv - a multipurpose toolkit. `http://vvvv.org/`. Accessed: 2015-06-15.

[8] Arduino CC. *Arduino Uno Reference Design*, 2010. Rev. 3.

[9] ATMEL Corporation. *ATMEGA8 8-bit AVR Microcontroller*, 2013. Revision AA.

[10] Bosch Rexroth AG. Linear motion technology handbook. http://www.aapautomation.com/wp-content/uploads/2014/12/LM-Handbook.pdf, 2000.

[11] Sean Follmer, Daniel Leithinger, Alex Olwal, Akimitsu Hogge, and Hiroshi Ishii. inform: Dynamic physical affordances and constraints through shape and object actuation. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 417–426, New York, NY, USA, 2013. ACM.

[12] Ben Fry. *A Processing: Programming Handbook for Visual Designers and Artists.* MIT Press, 2007.

[13] John Hardy, Christian Weichel, Faisal Taher, John Vidler, and Jason Alexander. Shapeclip: Towards rapid prototyping with shape-changing displays for designers. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 19–28, New York, NY, USA, 2015. ACM.

[14] Harwin Inc. *Technical Drawing/Datasheet for S1791-42R Spring Contact*, 2011. Rev. 3.

[15] Hiroo Iwata, Hiroaki Yano, Fumitaka Nakaizumi, and Ryo Kawamura. Project feelex: Adding haptic surface to graphics. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 469–476, New York, NY, USA, 2001. ACM.

[16] Daniel Leithinger and Hiroshi Ishii. Relief: A scalable actuated shape display. In *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '10, pages 221–222, New York, NY, USA, 2010. ACM.

[17] Microsoft. Hololens.
https://www.microsoft.com/microsoft-hololens/en-us, 2015.

[18] NXP Semiconductors. *PCA9685 16-Channel,12-bit PWM Fm+ I2C-bus LED Controller*, 2015. Rev. 4.

[19] Oculus VR. Rift.
https://www.oculus.com/en-us/, 2015.

[20] Hannah Perner-Wilson, Leah Buechley, and Mika Satomi. Handcrafting textile interfaces from a kit-of-no-parts. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '11, pages 61–68, New York, NY, USA, 2011. ACM.

[21] Ivan Poupyrev, Tatsushi Nashida, Shigeaki Maruyama, Jun Rekimoto, and Yasufumi Yamaji. Lumen: Interactive visual and shape display for calm computing. In *ACM SIGGRAPH 2004 Emerging Technologies*, SIGGRAPH '04, pages 17–, New York, NY, USA, 2004. ACM.

[22] Faisal Taher, John Hardy, Abhijit Karnik, Christian Weichel, Yvonne Jansen, Kasper Hornbæk, and Jason Alexander. Exploring interactions with physically dynamic bar charts. In *Proceedings*

of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI '15, pages 3237–3246, New York, NY, USA, 2015. ACM.

[23] Texas Instruments. *LM317 3-Terminal Adjustable Regulator*, 1997. Rev. SLVS044W.

[24] Texas Instruments. *CMOS Analog Multiplexers/Demultiplexers CD4067B, CD4097B Types*, 2003. Datasheet acquired from Harris Semiconductor.

[25] TowerPro. *MG995 High Speed Metal Gear Dual Ball Bearing Servo*.