A Sound Ceiling

Created by:

Frank Scarola

Jordan Parsons

Dan Rapoport

Created for:

Carnegie Mellon University

Making Things Interactive Fall 2009

Instructed by Professor Mark Gross

Abstract:

An acoustically responsive ceiling would adapt by changing shape and providing visual cues to the users it interacts with. This idea arose from the observation that acoustical considerations in architecture tend to be static or at best requires much forethought and planning in order to be adapted to accommodate varied situations. Instead a surface with the ability to adapt in real time to the acoustical needs of a space would be of great benefit to any space with multiple changing uses. The purpose of this project is to demonstrate the possibility of such a surface.

A 4'x8' frame supporting a similarly dimensioned surface consisting of 21 balloons was constructed. Each balloon is suspended 3 feet from the frame and contains a tricolor LED that is used to visually demonstrate the real time acoustic sensitivity of the surface. The balloons themselves can be individually raised and lowered using stepper motors. The actuation of the balloons and color and brightness control of the LEDs are in response to the data collected by a microphone and fed into an arduino. The arduino is used to evaluate different frequency ranges detected and adjust the surface according to the parameters set in the program running on it. For demonstration uses we programmed the lights to send a wave down the surface when high end and low end frequencies were detected. The motors were not programmed to respond instantly; instead the frequencies were recorded over a given interval and then averaged to determine the heights of the balloons.

While balloons and LEDs do little to affect the acoustics of a space they demonstrate the two factors necessary to do so in real time; instant collection of data and a physical change in the surface. The LEDs are an instant manifestation of the data being collected and change in shape and surface area are the only two non material factors that can impact the acoustic properties of a surface.

Parts List

Tools:

Drill

Electrical Tape

Hot Glue Gun

Laptop

Laser Cutter

Needle Nose Pliers

Saw

Scissors

Screw Driver

Solder Remover

Soldering Gun

Wire Snips

Wire Strippers

Zap a Gap

Frame & Mechanics:

½” plywood

20 lb Fish line

2x3 x8’ lumber (5)

50 lb Fish line

Balloons (white, 21)

Bic pens (10 pack)

Rubber Bands

Zip Ties (21+)

Pizza (6 boxes)

1/8” Acrylic (1 sheet)

1/16” Diameter Brass Tubing (1’)

2.5” Drywall Screws

3.5” Drywall Screws

Circuits:

Arduino

Braided Speaker Wire (400’)

H-Bridges (21)

TLC9540 NT (LED Driver) (2)

LM741 op amp ic

NE567 IC tone decoder chip

Perf Board (~50 in^2)

Solder

Tip 120 Transistors (42)

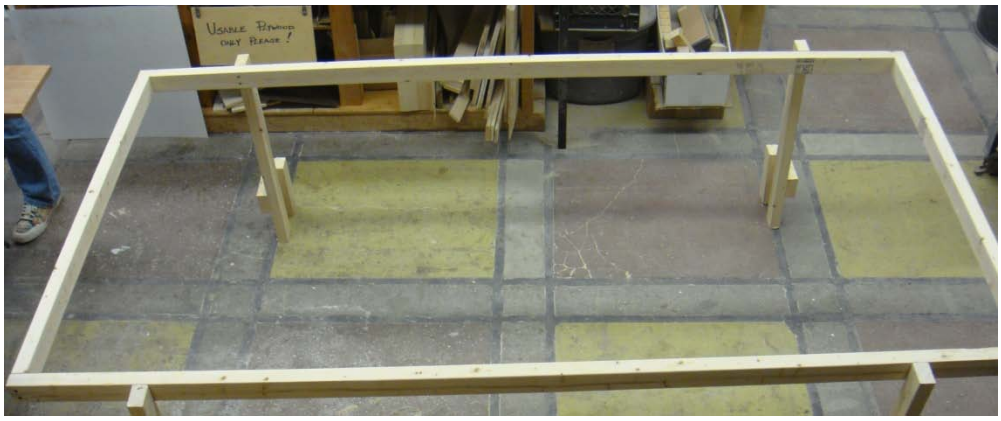
Tricolor LEDS (21)

How to Make It

Frame:

- Measure and Cut Square the 2x3
  - 8’ (x2)
  - 4’ (x2)
  - 2.5’ (x4)
  - 9” (x6)

- Measure and cut the 2x3 at 45°
  - 14” (x4)

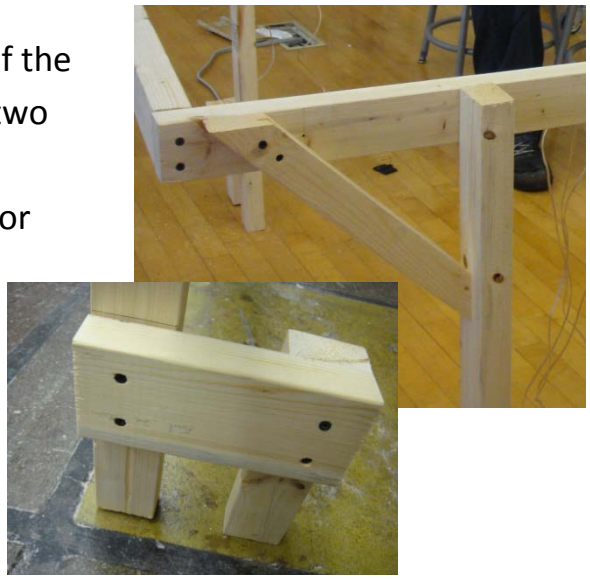


- Pre-drill and screw upper frame together using eight 3.5” drywall screws
  We put the 4’ lengths on the inside of the 8’ since the fish line will create tension along the width.

- Legs- *Our installation involved propping our basic 4 piece frame up on top of 8’ partition panels, this required the use of 2.5’ legs to achieve the desired height. Please note that longer legs or a method of suspension could substitute for the following set of instructions.*
  - For each leg create a basic “h” form using on 2.5’ length and two 9’ lengths. Secure with 2.5” screws

- Attach the legs on the outside of the frame 12” from the ends using two 3.5” screws each.
- Attach to each leg a 14” kicker for stability.

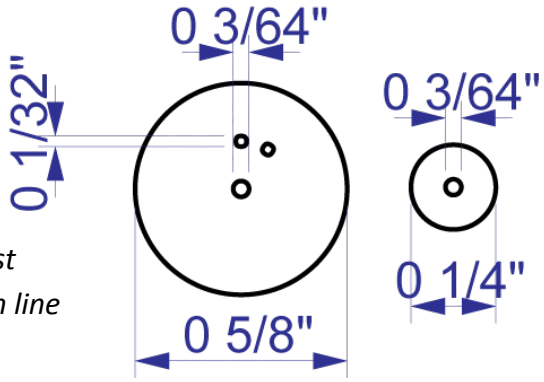
  3.5” screws may be necessary for the leg end while 2.5” are best for the frame.



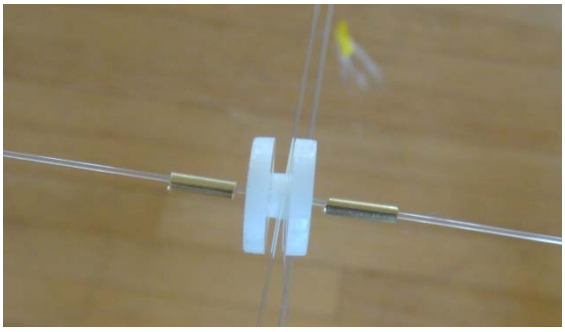
Mechanics:

- Laser-cut and assemble (Zap-A-Gap) 21 pulleys and 21 motor spools from 1/8” acrylic.

  Each pulley and spool consists of two outer, larger wheels and one small inner. The two smallest holes in the large wheels are for attaching 20lb fish line to the spools.



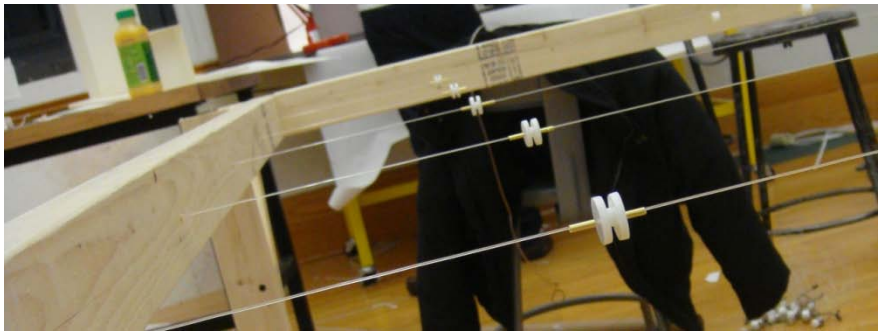
- Cut the 1/16” brass tube into 42 ¼” segments being careful not to crush or close the hole.

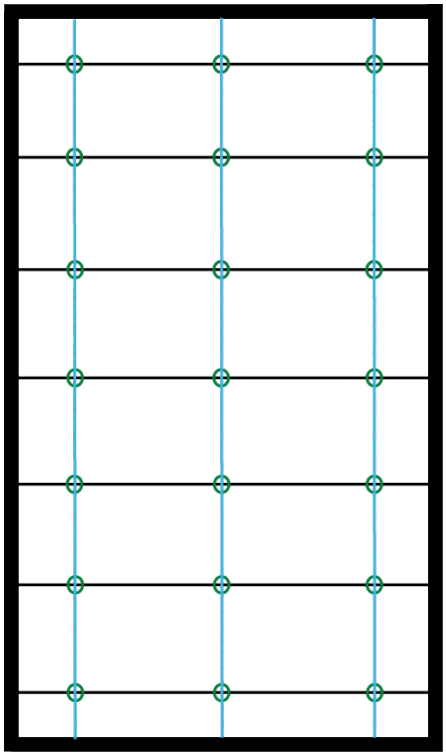


- Drill holes on center along the length of the frame every 14” - starting 5” from one side.

- Anchor one end at the first hole and begin to string a continuous length of 50 lb fishing line through each of them creating 7 strands parallel to the 4’ width. But wait! ...

- As you string thread on to each of the 7 parallel lines three sets of pulleys and brass tubing. Each set contains brass, pulley, brass so that each of the 7 lines contains 3 pulleys and 6 pieces of brass tubing.

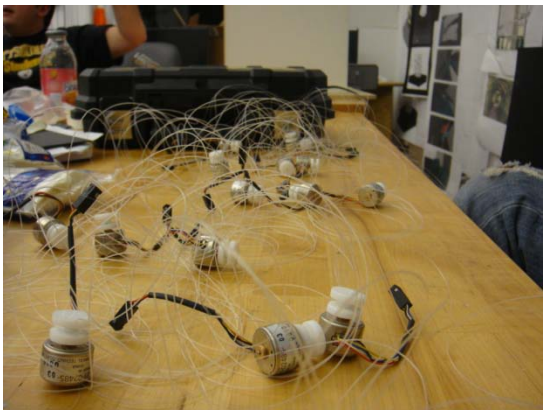




- Next drill three holes in the 4' widths one inch below center. The first hole should be centered on the width and the second two 15 inches to each side.

- Next run three parallel lengths of 50lb fish line. These will serve as supports for wiring. You should now have something that resembles this diagram, 21 pulleys strung up on the 7 lines hovering just above three lower perpendicular lines.

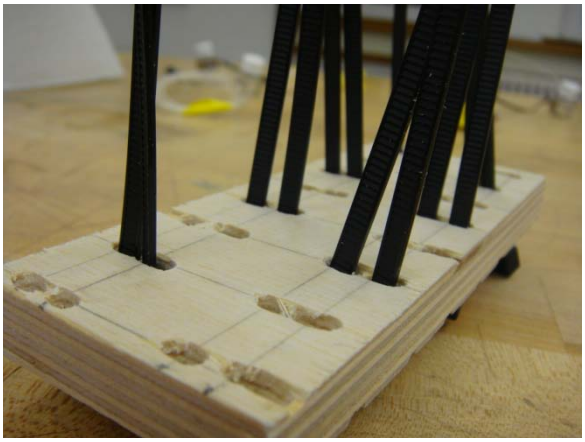
Motor Setup:



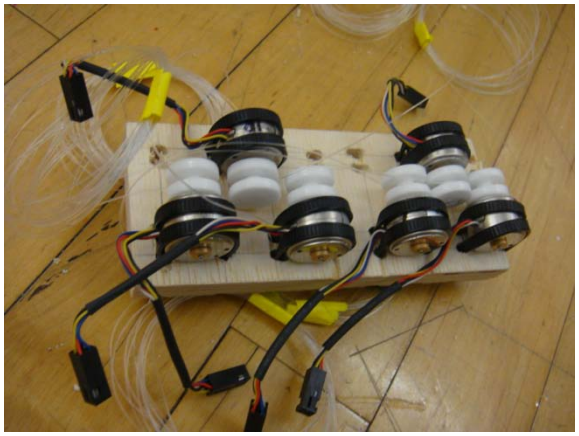
- Affix the spools to the motors. They should wedge tightly.

- Attach a length of 20 lb fishing line to each motor
    - (3x 46", 3x 60", 3x 74", 3x 88", 3x 102", 3x 116", 3x 130").
    - Label each motor by the length.

- Cut 3 pieces of 1/2” ply to 2” x 4.5”

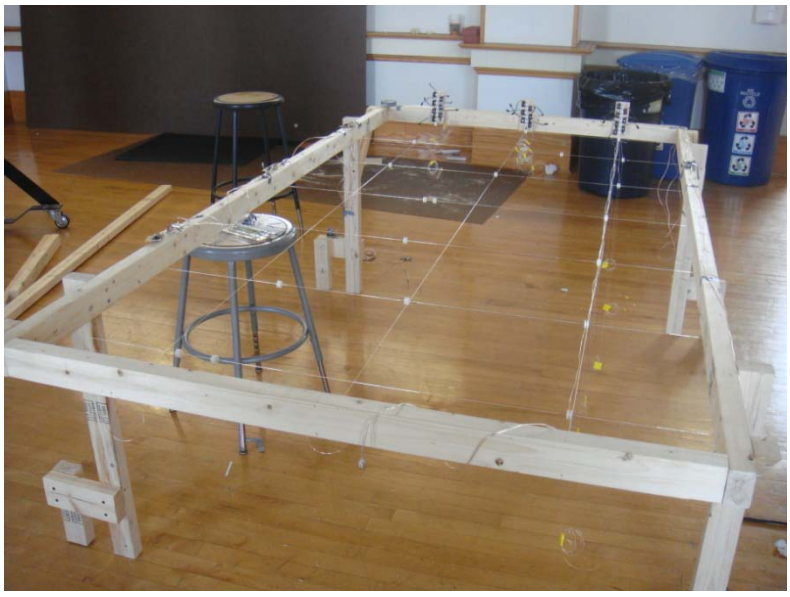


- Drill 4 holes spaced just inwards of the outer corners of each motor.
    - Motors should alternate as shown so that the spools are all in line and centered.
    - Ensure that each motor mount board contains a full set of lengths from bottom to top, short to long.



- Run Zip ties through the holes and secure them on either side of the motor wires. Trim any excess. Do this for all three mounts.

- Use two 3.5” drywall screws to mount each motor to the short side of the frame. The mounts should be in line with the three longer 50lb line strands and the lowest, shortest motor should be level with the spools of the 7 short lengths of 50 lb line.



- Line everything up and crimp the brass tubing to ensure the pulleys will spin but remain in place.

- Drape the 20lb spooled line form each motor over the pulleys to be secured to the balloon

LEDs and Balloons:

- Solder the 4 pins of the tricolor LEDs to 2 lengths of 2 strand braided speaker wire. The length of this wire can be either the full distance from each of the pulleys to the frame plus 3 feet of drop, or can be three feet and spliced to any type of wire to be carried to the frame.

We recommend using the two copper strands of speaker wire for power and red

  - Be sure to insulate each pin of the LED with electrical tape before proceeding

- Gut the bic pens and cut the outer tubing into 21 1.25” lengths

- Slip the tubing over the LEDs and secure with hot glue and electrical tape around the base.

- Blow up a balloon and twist the “top” of the neck of the balloon to keep the air from leaking out. This should leave the majority of the neck open.

- Insert the LED/tube/wires into the neck. Ensure the neck engulfs the tube and some of the wire then twist the mouth of the balloon around the wire

You should now have something that resembles a latex candy wrapper (tootsie roll style)

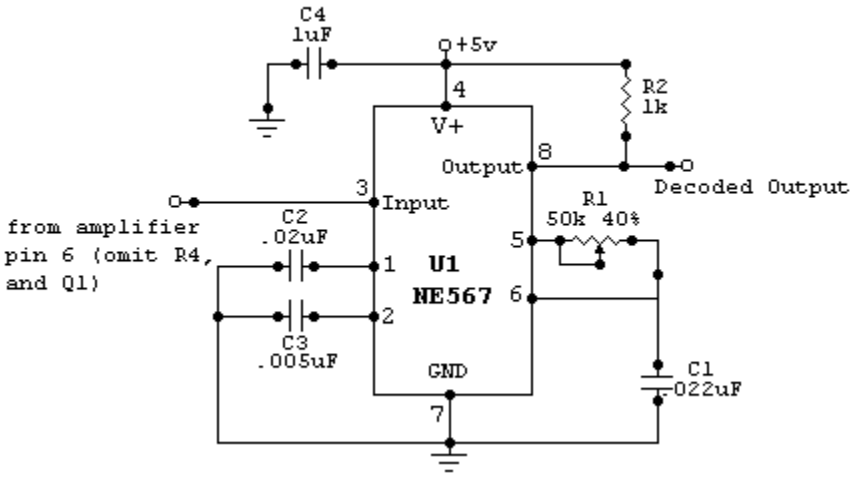
- Now tightly rubber band both around the middle of the tube, and the mouth of the balloon. Add hot glue or silicon sealant if necessary.

- Release the inner most twist. The LED should now be wired within the balloon.

- Now wire the colored pins of the LEDs to the outputs on the driver. Each row of LEDs should be on the same pins.

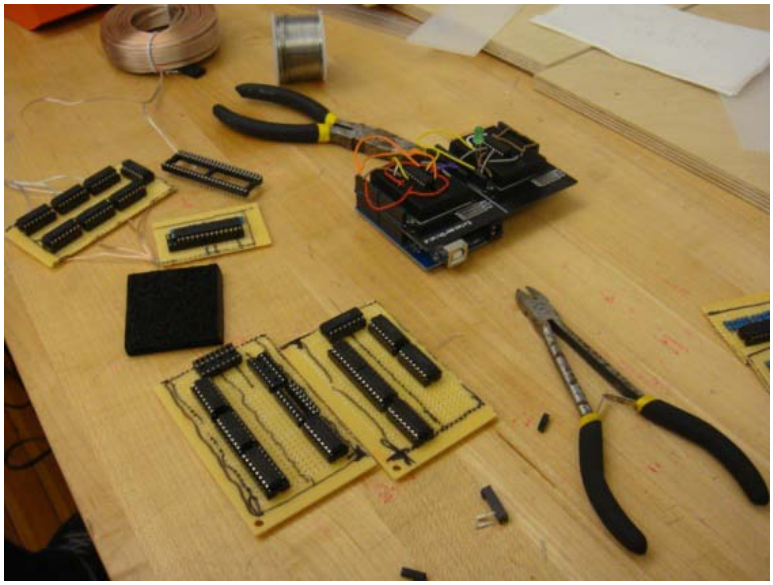
- Wire the longest leg of the LED to the common power on the LED driver.

Input:



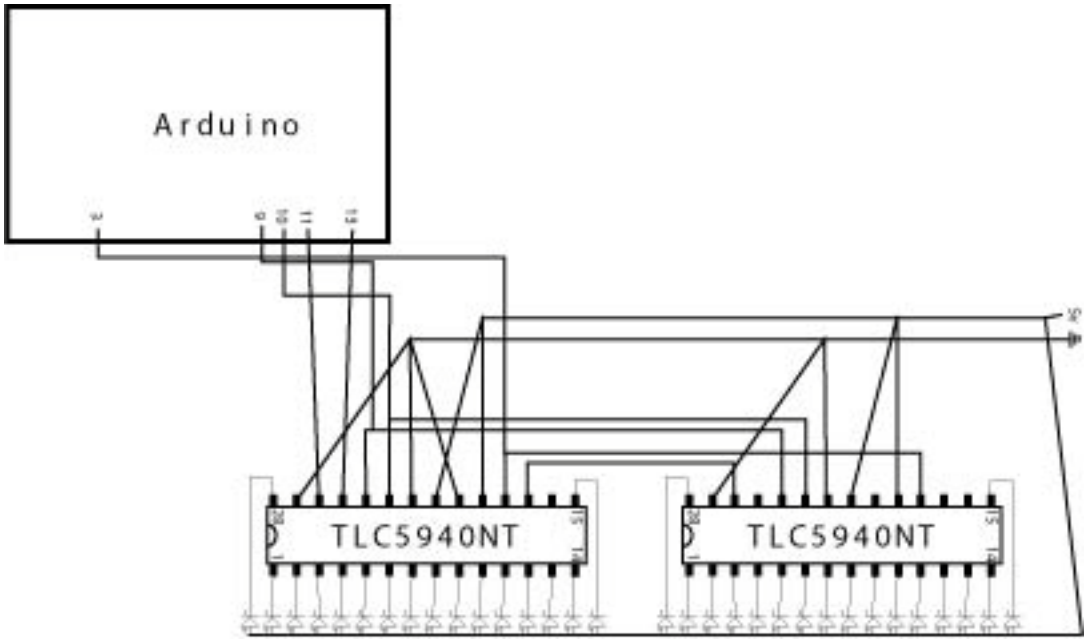
The sound sensing circuit is wired according to this diagram and the potentiometer at pins 5 and 6 should be set to around 20k, this puts the sensor in a range that responds to ambient room noise.

Position the sensor along the top of the project and drape the mic down the side so it hangs free about 7 feet from the ground.

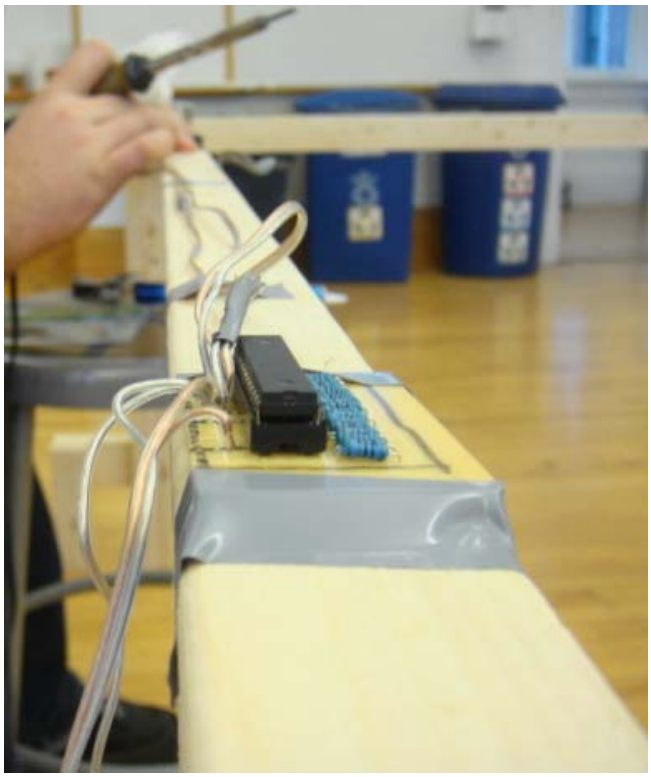


Output:

LEDs:



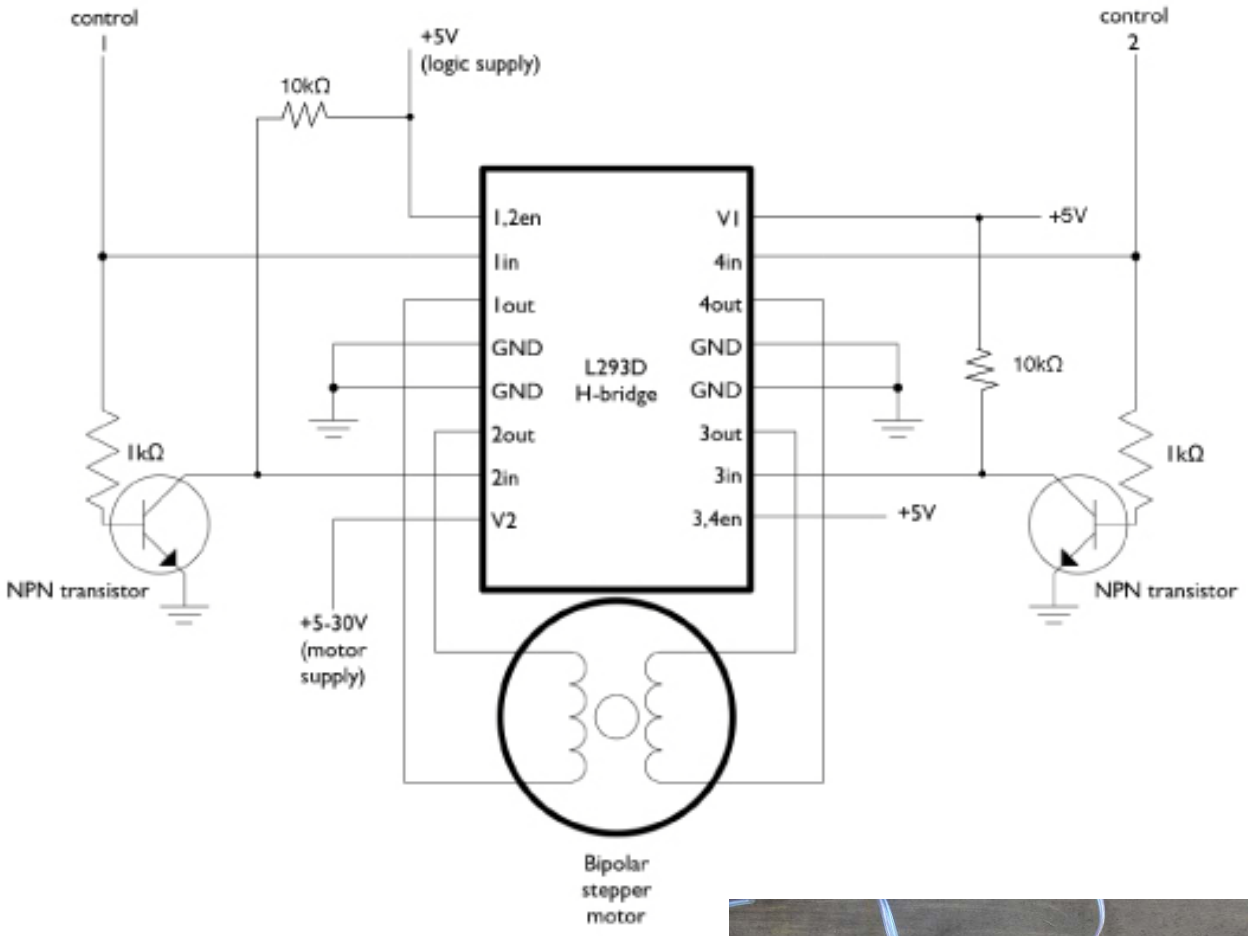
This code for the LED's is a very simple yet effective response. It creates a wave leading back from the front of the project which increases in brightness in response to the input.



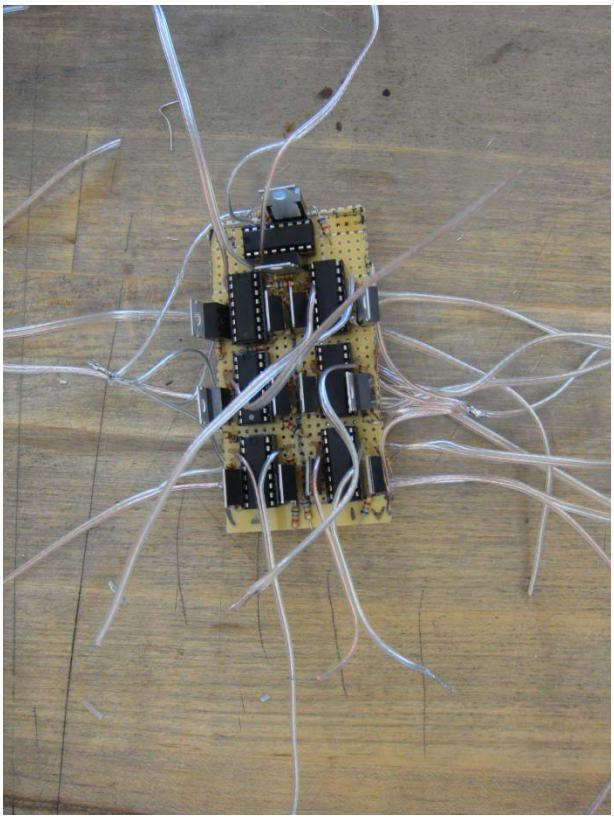
LED Code

```
/*
Code runs on a seperate arduino and provides live feeback
from the sound input.In the form of a wave.


*/
#include "Tlc5940.h"
int data[10]={9,3,2,4,3,2,7,8,9,11};

int dataStore,dataTemp,input;
void setup()
{
  Serial.begin(9600);

  /* Call Tlc.init() to setup the tlc.
   You can optionally pass an initial PWM value (0 - 4000) for all channels.*/
  Tlc.init();



}

void loop()
{
  for(int n=1; n>0;n--){
    dataTemp =  analogRead(1);
    if(dataTemp>15&&dataTemp<26){
     dataTemp=26;
    }
    Serial.println(dataTemp);
    dataStore = dataStore+dataTemp;
    delay(10);
  }
   input = dataStore/1; //raw
   data[0] = map(input,10,35,0,4095); //mapped

  for(int i=6;i>0;i--){
    data[i]=data[i-1];
  }
   data[0] = map(input,10,35,0,4095); //mapped


  Tlc.set(10,data[0]/1);
  Tlc.set(11,data[0]/1);
  Tlc.set(9,data[0]*10);

  Tlc.set(12,data[1]*10);
  Tlc.set(13,data[1]/1);
  Tlc.set(14,data[1]/1);

  Tlc.set(8,data[2]/1);
  Tlc.set(7,data[2]/1);
  Tlc.set(6,data[2]/1); /// fix

  Tlc.set(28,data[3]*10);
  Tlc.set(29,data[3]/1);
  Tlc.set(27,data[3]/1);

   Tlc.set(26,data[4]/1);
  Tlc.set(25,data[4]*10);
  Tlc.set(24,data[4]/1);

 Tlc.set(20,data[5]/1); //fix
  Tlc.set(19,data[5]/1);
  Tlc.set(18,data[5]/1);

 Tlc.set(23,data[6]/1);
  Tlc.set(22,data[6]/1);
  Tlc.set(21,data[6]/1);

  //29 row 4 green
  //28 row 4 red
  //27 row 4 blue
  //26 row 5 green
  //25 row 5 red
  //24 row 5 blue
  //23 row 7 green 1

  //14 row 2 green
  //13 row 2 blue
  //12 row 2 red
  //11 row 1 green
  //10 row 1 blue
  //9 row 1 red


  Tlc.update();
  delay(50);
  //Serial.println(analogRead(0));
  //Serial.println(analogRead(2));


}
```

Motors:



The motors are wired according to the diagram above. This allows the motors to be controlled by two pins on the arduino. Using the stepper library the arduino can generate the signal needed to turn the motors one step at at time. This method works, but it was very brute force. There are far better methods of controlling steppers, including chips designed to do so.



Motor Code

```
/*
Code uses input data to send a pre planned motor response to
the arduino where it is run.
*/

import processing.serial.*;
PrintWriter raw;
PrintWriter average;
Serial myPort;        // The serial port
int xPos = 1;         // horizontal position of the graph
int val;
long tStart;
PFont timer;
int nFiles=0;
int n0, n1, n2, n3, n4, n5, storeData;
int topRange = 400;
boolean inputStarted = false;
String planLet;
int mode = 0; //0 = pre planned; 1 = pre planned set 2; 2 = adaptiv

void setup () {
  // set the window size:
  size(1500, 400);
  raw = createWriter("raw_data_"+nFiles+".xls");
  average=createWriter("averaged_data_"+nFiles+".xls");

  // List all the available serial ports
  println(Serial.list());

  myPort = new Serial(this, Serial.list()[0], 9600);
  // don't generate a serialEvent() unless you get a newline charac
  myPort.bufferUntil('\n');

  background(0);
}
void draw () {
  // everything happens in the serialEvent()
  if((millis()-tStart)>1000*60*.2){
    nFiles++;
    tStart=millis();
    average.println("0"+TAB+n0);
    average.println("1"+TAB+n1);
    average.println("2"+TAB+n2);
    average.println("3"+TAB+n3);
    average.println("4"+TAB+n4);
    average.println("5"+TAB+n5);
    average.flush();
    average.close();
    raw.flush(); // Writes the remaining data to the file
    raw.close(); // Finishes the file
    raw = createWriter("raw_data_"+nFiles+".xls");
    average = createWriter("averaged_data_"+nFiles+".xls");

    //exit(); // Stops the program
    //run prePlanMover code
    if(mode==0){
      //Use mover 1
      prePlanMover1("averaged_data_"+(nFiles-1)+".xls");
    }
    else if(mode==1){
      //Use Mover 2
      prePlanMover2("averaged_data_"+(nFiles-1)+".xls");
    }

  }
}

void serialEvent (Serial myPort) {
  rectMode(CORNERS);
  inputStarted = true;
  String inString = myPort.readStringUntil('\n');

  if (inString != null) {
    // trim off any whitespace:
    inString = trim(inString);
    // convert to an int and map to the screen height:
    //Set 1---------------------------------
    if(inString.equals("a") == true){
      System.err.println("Plan 0: Done");
    }
    else if(inString.equals("b") == true){
      System.err.println("Plan 1: Done");
    }
    else if(inString.equals("c") == true){
      System.err.println("Plan 2: Done");
    }
    else if(inString.equals("d") == true){
      System.err.println("Plan 3: Done");
    }
    else if(inString.equals("e") == true){
      System.err.println("Plan 4: Done");
    }//Set 2---------------------------------
    else if(inString.equals("f") == true){
      System.err.println("Plan 5: Done");
    }
    else if(inString.equals("g") == true){
      System.err.println("Plan 6: Done");
    }
    else if(inString.equals("h") == true){
      System.err.println("Plan 7: Done");
    }
    else if(inString.equals("i") == true){
      System.err.println("Plan 8: Done");
    }
    else if(inString.equals("j") == true){
      System.err.println("Plan 9: Done");
    }//Else---------------------------------
    else{
```

```
      float inByte = float(inString);
      storeData = int(inString);
      inByte = map(inByte, 0, topRange, 0, height);
      //println(inByte);
      //raw.println(inByte);
      val = (int) storeData;
      // draw the line:
      //color(#ff7700);
      // stroke(#ff7700);

      rect(xPos,height,xPos+3,height-inByte);
      // line(xPos, height, xPos, height - inByte);

      // at the edge of the screen, go back to the beginning:
      if (xPos >= width) {
        xPos = 0;
        background(0);
      }
      else {
        // increment the horizontal position:
        xPos=xPos+3;
      }

      raw.println(storeData);

      averageData(storeData);
    }
  }
}

void keyPressed() {
  average.println("0"+TAB+n0);
  average.println("1"+TAB+n1);
  average.println("2"+TAB+n2);
  average.println("3"+TAB+n3);
  average.println("4"+TAB+n4);
  average.println("5"+TAB+n5);
  average.flush();
  average.close();
  raw.flush(); // Writes the remaining data to the file
  raw.close(); // Finishes the file
  exit(); // Stops the program
}

void averageData(int val){
  if(val<5){
    n0++; // 1-24
  }
  if(val>=5&&val<100){
    n1++; // 45-99
  }
  if(val<175&&val>=100){
    n2++; // 100-174
  }
  if(val<300&&val>=175){
    n3++; // 175 - 299
  }
  if(val>=300){
    n4++; // 300+
  }
  n5++; // count

}
//----------------
//Pre Plan Set 1
//Reacts to Most Hits
//--------------
void prePlanMover1(String fileName){
  int pNum=0;
  int[] vals = new int[6];
  String line;
  boolean breaker=false;
  BufferedReader data;
  data=createReader(fileName);
  int i =0;

  while(!breaker){
    try {
      line = data.readLine();
    }
    catch (IOException e) {
      e.printStackTrace();
      line = null;

    }

    if (line == null) {
      // Stop reading because of an error or file is empty
      breaker=true;
    }
    else {
      String[] pieces = split(line, TAB);
      vals[i] = int(pieces[1]);

      i++;
    }
  }

  int maxVal=max(shorten(vals)); //Get the max after dropping off the total

  boolean n = true;
  int ii=0;
  int id=0;
  while(n){
    if(vals[ii]==maxVal){
      id = ii;
      n=false;
    }
    else{
      ii++;
```

```
    }
  }

  System.err.println("Reading: "+fileName);
  delay(10);
  println(vals);
  println("Most Hits:" + id);

  switch(id){
  case 0:
    //Plan 0
    pNum=id;
    planLet = "a"; //motor letter
    myPort.write(planLet);
    //plan settings
    break;
  case 1:
    //Plan 1
    pNum=id;
    planLet = "b";
    myPort.write(planLet);
    //plan settings
    break;
  case 2:
    //Plan 2
    pNum=id;
    planLet = "c";
    myPort.write(planLet);
    //plan settings
    break;
  case 3:
    //Plan 3
    pNum=id;
    planLet = "d";
    myPort.write(planLet);
    //plan settings
    break;
  case 4:
    //Plan 4
   /* pNum=id;
    planLet = "e";
    myPort.write(planLet);*/
    planLet = "a"; //motor letter
    myPort.write(planLet);
    myPort.write(400);
    //plan settings
    break;
  }
  System.err.println("Running Plan: "+pNum);

}
//----------------
//Pre Plan Set 2
//Reacts to Fewest Hits
//--------------

void prePlanMover2(String fileName){

  int pNum=0;
  int[] vals = new int[6];
  String line;
  boolean breaker=false;
  BufferedReader data;
  data=createReader(fileName);
  int i =0;

  while(!breaker){
    try {
      line = data.readLine();
    }
    catch (IOException e) {
      e.printStackTrace();
      line = null;

    }

    if (line == null) {
      // Stop reading because of an error or file is empty
      breaker=true;
    }
    else {
      String[] pieces = split(line, TAB);
      vals[i] = int(pieces[1]);

      i++;
    }
  }

  int minVal=min(shorten(vals)); //Get the min after dropping off the total

  boolean n = true;
  int ii=0;
  int id=0;
  while(n){
    if(vals[ii]==minVal){
      id = ii;
      n=false;
    }
    else{
      ii++;
    }
  }

  System.err.println("Reading: "+fileName);
  delay(10);
  println(vals);
  println("Most Hits:" + id);

  switch(id){
  case 0:
    //Plan 5
```

```
    pNum=id;
    planLet = "f";
    myPort.write(planLet);
    //plan settings
    break;
  case 1:
    //Plan 6
    pNum=id;
    planLet = "g";
    myPort.write(planLet);
    //plan settings
    break;
  case 2:
    //Plan 7
    pNum=id;
    planLet = "h";
    myPort.write(planLet);
    //plan settings
    break;
  case 3:
    //Plan 8
    pNum=id;
    planLet = "i";
    myPort.write(planLet);
    //plan settings
    break;
  case 4:
    //Plan 9
    pNum=id;
    planLet = "j";
    myPort.write(planLet);
    //plan settings
    break;
  }
  System.err.println("Running Plan: "+pNum);

}
```

Considerations and Alternatives:

Given the opportunity to redo this project there are of course many things we would have done differently. The issues we had with our motors could have easily been avoided with either better quality stepper motors or stepper motor drivers. While higher end steppers would be the more expensive alternative it would provide us with the opportunity to use a more substantial surface material and diffuser than balloons. Something else we considered was acquiring more LED drivers to individually control each tricolor led. The effect of this however would not have been very evident unless we executed a few other alternatives. The first of those would be to expand the surface from a hallway setup to a more omnidirectional room type ceiling surface. This would be done in conjunction with the addition of multiple microphones placed throughout. What this would achieve is the ability to identify and create source specific interactions. Multiple microphones could also be used broaden our range of frequency detection. Being able to identify with full range detection the location, frequency, and intensity of a sound would infinitely expand the range of programming possibilities. All this of course is the outer reaches of what we wished to demonstrate. Given enough time, money and resources there is no doubt we could achieve our ideal levels of sensitivity, resolution, and interaction.

Concept and Conclusion:

With adjustable ceilings in many high end performance halls a truly intelligent ceiling surface is only the next step. Such a surface would allow a room to be tuned to very specific needs. It could adjust for specific reverberation times or sound pressure level distributions in real time, or using presets with customizable resolutions. A lecture hall could know when one person is presenting, or when there is a dialogue with the audience. A banquet hall could know how full it is, when the band is playing, when people are eating, or when someone stands up to make a toast. A music hall could tune itself to the type of performance, or simulate occupancy during a practice session. The implications are boundless and the means are available.