CS 281 Final Project

Peize Song and Zihao Miao
Professor: Matt Kretchmar

May 7, 2018

1. **Overview**

   In this project, we designed an "animated eye" that follows people moving in front of it. If the person is too close, it will show it is friendly and give the person a "big smile face". The eye is presented by a 8x8 LED panel and we detect people by using ultrasonic range finder. We added a feature to it that allows emotion detection. If a person is presented near the range finder, you would have an option to detect human emotion through a camera connected to a raspberry Pi.The Pi would be able to take a picture using the connected camera and send it back via a python socket. The laptop will send the received image to the google cloud vision api for facial expression detection. The report is divided into two parts, where the first part is the animated eye part, and the second is the raspberry Pi and emotion recognition part.

2. **Tools**
   1. 2 Arduino
   2. 3 Ultrasonic Ranger Finder
   3. 1 8x8 LED panel (with chip)
   4. wires
   5. 3 motherboards
   6. 1 raspberry Pi
   7. 1 camera module
   8. 1 laptop

3. **Part 1: the animated eye**

   (a) **8x8 LED Panel**

   First, we learned to use 8x8 LED Panel. When we knew there are too any thing to implement with a normal 16 pins 8x8 LED panel to show the eye, we decides to choose the RGB Matrix from SUNFOUNDER. This RGB matrix works with a chip, which is also provided by SUNFOUNDER. To make it work, we have to use arduino with the Colorduino library. We can simply mount the matrix on the chip and mount the chip on the arduino, inserting each pin into corresponding socket.

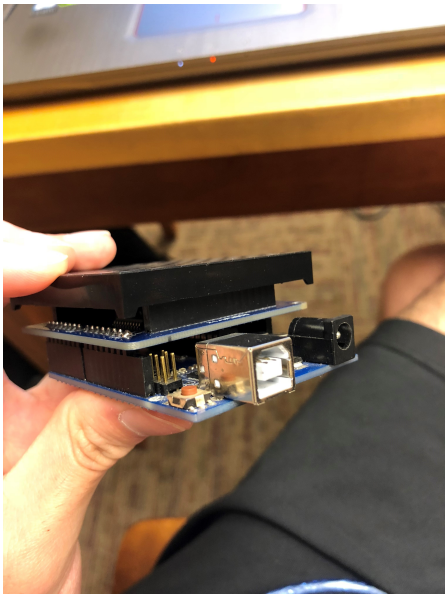

Figure 1: RGB Matrix

To understand how it print things on the panel, we tried some sample codes provided by hobbycomponents.com. There are 2 ways to show things on the panel, we decided to use the plot() function because print() function can only print existing characters (strings) on the panel. But with plot() function, we can design the content by ourselves. We only should provide the color and coordinates of the LEDs to light. Here is the basic idea of our eye:

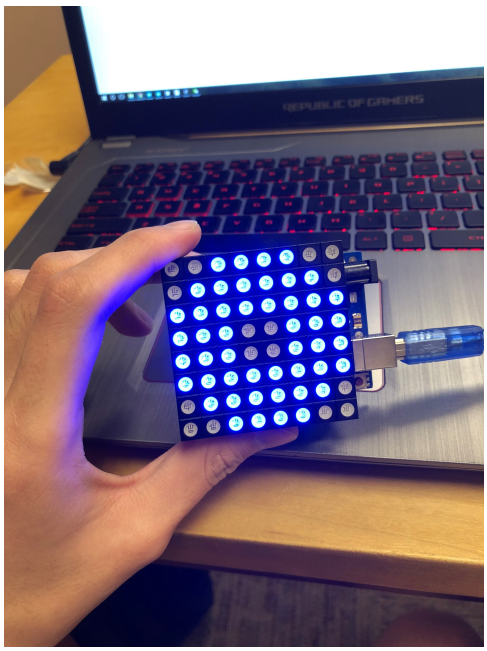

Figure 2: LED Eye

The blue part is the eye and the blank part in the middle is the eye ball. The eye ball will move when people move. To make it move, we only have to change coordinates of lighting LEDs.

(b) **Ultrasonic Range Finder**

To prepare this part, we just reviewed what we learned in Lab 4. We used another Arduino to gather data from 3 ultrasonic finders. To do this, we just connected Vcc and Gnd of Arduino with those of ultrasonic range finder and connected Trig pins with pin 2, 4, 6 of Arduino and Echo pins with pin 3,5,7 on Arduino. With following code, we can convert the input of ultrasonic range finders to a reasonable distance (in centimeter).

```
#define TRIG_0 2
#define ECHO_0 3
#define TRIG_2 4
#define ECHO_2 5
#define TRIG_1 6
#define ECHO_1 7

long duration, distance, distance_1, distance_2, distance_3;

void setup()
{
    Serial.begin(9600);
    pinMode(TRIG_0,OUTPUT);
    pinMode(ECHO_0,INPUT);
    pinMode(TRIG_1,OUTPUT);
    pinMode(ECHO_1,INPUT);
    pinMode(TRIG_2,OUTPUT);
    pinMode(ECHO_2,INPUT);
}

void loop()
{
    temp(TRIG_0, ECHO_0);
    if(distance <= 200)
        distance_1 = distance;
    else if(distance > 600)
     distance_1 = distance_1;
    else
        distance_1 = 300;
    temp(TRIG_1, ECHO_1);
    if(distance <= 200)
        distance_2 = distance;
    else if(distance > 600)
        distance_2 = distance_2;
    else
        distance_2 = 300;
    temp(TRIG_2, ECHO_2);
    if(distance <= 200)
        distance_3 = distance;
    else if(distance > 600)
        distance_3 = distance_3;
    else
        distance_3 = 300;
    Serial.write(distance_1);
    Serial.write(distance_2);
    Serial.write(distance_3);

    delay(500);
}

void temp(int trig, int echo)
{
    digitalWrite(trig, LOW);
    delayMicroseconds(2);
```

```
        digitalWrite(trig, HIGH);
        delayMicroseconds(10);
        digitalWrite(trig, LOW);
        duration = pulseIn(echo, HIGH);
        distance = (duration / 2) / 29.1;
    }

```

To detect the position of people, we used 3 ultrasonic range finder. If one finder returns a value lower than the other two, it means the person is in the range of this ultrasonic finder. And to make sure they can detect a wide range, we used wires to connect 3 motherboards and there is a ultrasonic range finder on each of them. In order to arrange these 3 ultrasonic finders' positions more flexibly, we wired the device as following.

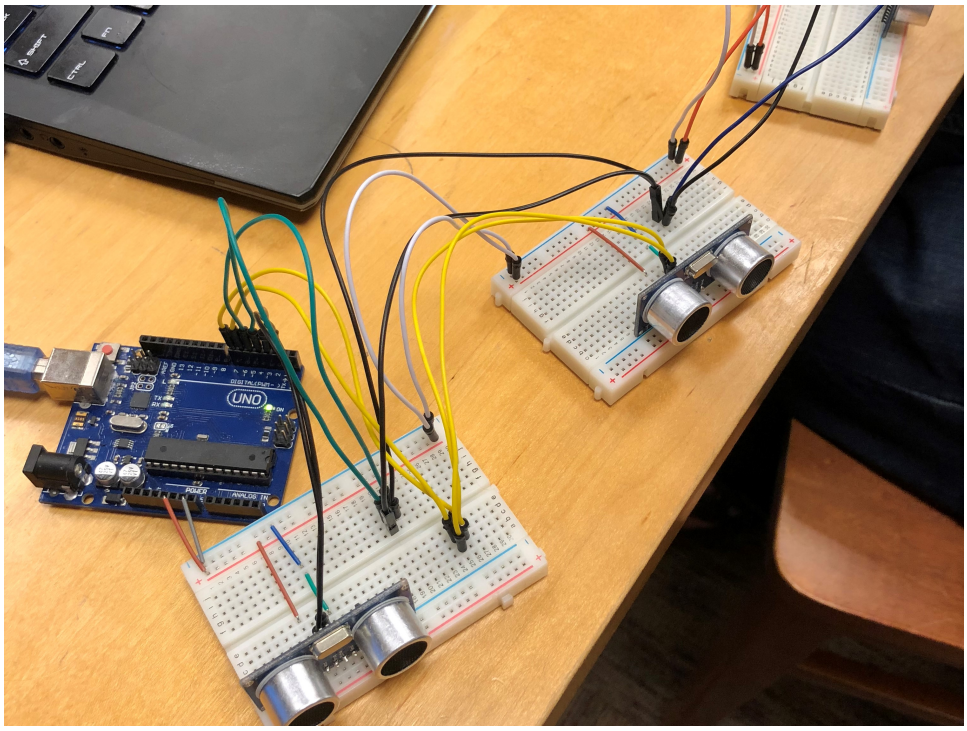

Figure 3: Ultrasonic Range Finder

It looks complex but it actually does the same thing as directly connecting them to the Arduino.

(c) **Communication between Arduino**

It took us a long time to figure this problem. Because our RBG matrix uses almost all pins of that Arduino, we can't let 2 Arduino's communicate by connecting them together. So, we have to find a way to let them communicate via Serial (the USB port). Arduino doesn't offer a explicit way for multiple Arduino's talking with each other. But python does. There is a library called pyserial can help us read and write values from and to arduino's with python. We created 2 serial objects. Ser1 is for the RGB matrix, ser2 is for the ultrasonic range finders. We read datas from ser2 to compute the signal sent to ser1, then send the signal back to ser1. Here is the python code to achieve this.

```
import serial

def eye(val1,val2,val3):
```

```
    rough = min(val1,val2,val3)
    if rough == val1:
        if val2 < val3:
            return 'k'
        else:
            return 'j'
    elif rough == val2:
        if val1 < val3:
            return 'k'
        elif val1 > val3:
            return 'm'
        else:
            return 'l'
    else:
        if val1 > val2:
            return 'n'
        else:
            return 'm'


def face(val2):
    if(val2 <= 10):
        return True

def main():
    ser1 = serial.Serial("COM4", 9600)
    ser2 = serial.Serial("COM3", 9600)
    while(1):
        val1 = ord(ser2.read(1))
        val2 = ord(ser2.read(1))
        val3 = ord(ser2.read(1))
        signal = eye(val1,val2,val3)

        if(val2 <= 20):
            signal = 'f'

        ser1.write(signal.encode())

main()
```

Because datas are tranfered in bytes through wire, we used encode() and ord() functions here.

(d) **Combined**

Finally, we combined them together. First, using the python program, we read the three values our ultrasonic range finders detected and used these values to compute the rough position of the person. Because our LED panel is only a 8x8 one. So, we only have 5 different status of out eye (left, mid-left, mid, mid-right, right). As you can see in the above codes, we have a eye() function to compute the status. This function finds the smallest value of the 3 values and gives a rough range (left, right, or mid). Then, it uses the other 2 values to compute the exact status the eye should be. For example, if the smallest value of these 3 is val2, that means the person is standing roughly in the middle. Then, we compare val1 and val3, if val1 is smaller, the eye should look a little to the left or a little to the right if val3 is smaller. Otherwise, the eye will directly to the middle. Also, if the person is really close to the middle finder, the simling face will show up. After getting the status, we send the signal to the RGB matrix. ('j': left, 'k': mid-left, 'l': mid, 'm': mid-right, 'n': right, 'f': smiling face) Then, our LED arduino

(ser1) will read the signal and present the correct pattern on the panel. Here is the code for it.

```
/* Include the HCTimer2 library */
#include <HCTimer2.h>

/* Include the HCColorDuino library */
#include "HCColorDuino.h"

/* Create an instance of the library */
HCColorDuino HCColorDuino;

int turn = 0;
int times = 0;

int y[48] =
    {0,0,0,0,1,1,1,1,1,1,2,2,2,2,2,2,2,2,3,3,3,3,3,3,4,4,4,4,4,4,5,5,5,5,5,5,5,5,6,6,6,6,

int y1[26] = {0,0,0,0,1,1,2,2,2,3,3,3,3,4,4,4,4,5,5,5,6,6,7,7,7,7};
int x_6[26] = {2,3,4,5,1,6,0,3,7,0,2,5,7,0,2,5,7,0,3,7,1,6,2,3,4,5};
int x_1[48] =
    {2,3,4,5,1,2,3,4,5,6,0,1,2,3,4,5,6,7,0,3,4,5,6,7,0,3,4,5,6,7,0,1,2,3,4,5,6,7,1,2,3,4,

int x_2[48] =
    {2,3,4,5,1,2,3,4,5,6,0,1,2,3,4,5,6,7,0,1,4,5,6,7,0,1,4,5,6,7,0,1,2,3,4,5,6,7,1,2,3,4,

int x_3[48] =
    {2,3,4,5,1,2,3,4,5,6,0,1,2,3,4,5,6,7,0,1,2,5,6,7,0,1,2,5,6,7,0,1,2,3,4,5,6,7,1,2,3,4,

int x_4[48] =
    {2,3,4,5,1,2,3,4,5,6,0,1,2,3,4,5,6,7,0,1,2,3,6,7,0,1,2,3,6,7,0,1,2,3,4,5,6,7,1,2,3,4,

int x_5[48] =
    {2,3,4,5,1,2,3,4,5,6,0,1,2,3,4,5,6,7,0,1,2,3,4,7,0,1,2,3,4,7,0,1,2,3,4,5,6,7,1,2,3,4,

int* x[7] = {x_1,x_2,x_3,x_4,x_5};

int current;
int target;
int last;
byte R, G, B;

void setup()
{
  Serial.begin(9600);
  /* Initialise the HCTimer2 library with a 2.4mS interval */
  HCTimer2Init(T2_CLK_DIV_256, 150);

  /* Calibrate the LED matrix display for differences in RGB luminance */
  HCColorDuino.RGBColourCorrection(25, 63, 63);

  R = 0;
  G = 0;
  B = 255;

  current = 2;
  target = 10;
  last = 0;
}

/* Main program */
void loop()
```

```
{
  switch(Serial.read())
  {
    case 'j': target = 0;
              break;
    case 'k': target = 1;
              break;
    case 'l': target = 2;
              break;
    case 'm': target = 3;
              break;
    case 'n': target = 4;
              break;
    case 'f': target = 10;
              break;
    default: break;
  }

  if(target == 10)
  {
    if(last != 10)
    {
      HCColorDuino.ClearMatrix();
      last = 10;
    }
    for(int i = 0; i < 26; i++)
      HCColorDuino.Plot(255,255,0,x_6[i],y1[i]);
  }
  else
  {
    if(last == 10)
    {
      HCColorDuino.ClearMatrix();
      target = 2;

      last = 0;
    }
    for(int i = 0; i < 48; i++)
      HCColorDuino.Plot(R,G,B,y[i],x[current][i]);

    if(current < target)
    {
      if(times == 20)
      {
        current += 1;
        HCColorDuino.ClearMatrix();
        times = 0;
      }
      times += 1;
    }
    else if(current > target)
    {
      if(times == 20)
      {
        current -= 1;
        HCColorDuino.ClearMatrix();
        times = 0;
      }
      times += 1;
    }
    else
      times = 0;
```

```
110   }
111
112   delay(10);
113 }
114
115 /* Refresh the matrix display in the background */
116 void HCTimer2()
117 {
118     HCColorDuino.Refresh();
119 }
120
```

The program will select the correct coordinates of lighting LED's by checking the signal and then plot the correct pattern.

(e) **Result**

Till now, we can make our eye coordinate with the range detector. The eye followed me when I move around in front of it. But the problem is ultrasonic range finders are not accurate enough to get the distance. Sometimes, weird values may come in and interfere the performance. But I think this actually makes the eye more mimic. And don't forget the simling face, if you are close to the sensor, the LED panel will show you a smiling face!

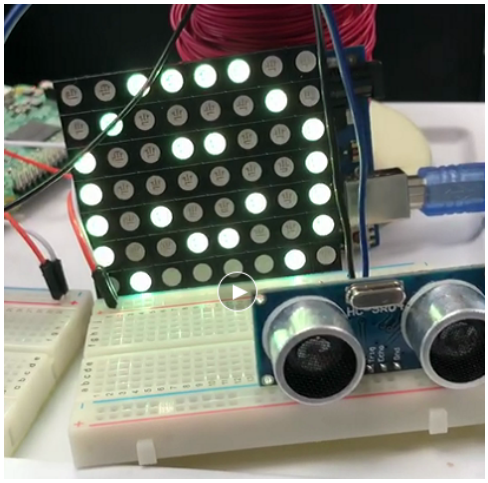

Figure 4: smiling face

4. Part 2: Raspberry Pi and emotion detection

(a) Raspberry Pi and Picamera

Figure 5: the Raspberry Pi with the camera

The raspberry Pi3 runs a linux system that is connected to the same Wifi as my laptop. A camera module is connected to the raspberry Pi. The module is called "CAMERA MODULE V2". In this project, the raspberry pi is a server that waits for a signal from my laptop that tells it to take a picture. After that, it would send the image file back to my laptop. The raspberry Pi can control the camera using a python script after installing the module "PiCamera". Specifically, the script would first pre-load the camera, and whenever the signal arrives, it would take the picture. The server part of the code is shown below.

```
1    import time
2    import socket
3    import sys,os
4    from picamera import PiCamera
5    camera = PiCamera()
6
7    port = 6457
8    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
10
11   s.bind(('',port))
12   print('listening on port', port)
13   s.listen(5)
14   print('about to accept')
15   peer, address = s.accept()
16   print('accept', address)
17
18
19   def take():
20     f = 'image.jpg'
21     camera.capture(f)
22     s = os.stat(f).st_size
23     print(str(s))
24     peer.send(str(s))
25     file = open(f,"rb")
26     while 1:
27       pkt = file.read(1024)
28       if not pkt:
```

```
            break
          peer.send(pkt)
        file.close()
        print('finished')
    camera.start_preview()
    time.sleep(2)
    while True:
      cmd = peer.recv(1024)
      print(cmd)
      if cmd == b"now":
        take()
    peer.close()
    s.close()

```

(b) **Communication with Raspberry Pi**

In this project, the laptop is a client that request for new pictures when the condition is satisfied. Specifically, if the middle range detector reads a signal of less than 30, my laptop would send the signal for new images. The raspberry Pi takes on the role of the server.The connection between the client and server is reset after each image transfer. To prepare for this part, we reviewed the simple python server and client module used in our networking class.

The image transfer starts with the server sending the length of the file. Afterwards, the server begins sending the file, 1024 bytes a time. The client keeps receiving 1024 bytes of data and write it to a local file until the incoming data is less than 1024 bytes, which implies that it reaches the end of file. The client would just collect the residuals and call it a file. The next step of the client would be to do facial expression. This file transferring part of the project actually took most of my debugging time. The file sending part of the file is show below.

```
port = 6457
host = "140.141.154.52"
file = "image.jpg"
print("connecting server",host,port)
cmd = ''
while True:
  val1 = ord(ser2.read(1))
  val2 = ord(ser2.read(1))
  val3 = ord(ser2.read(1))
  sig = eye(val1,val2,val3)
  if val2 < 30 or val1 < 30 or val3 < 30:
    sig = 'f'
  ser1.write(sig.encode())
  if val2 < 30:
    cmd = "Y"
  if cmd == "Y":
    s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    server = (host,port)
    print("connecting server",host,port)
    s.connect(server)
    s.send(b'now')
    f = open(file,'wb')
    size = int(s.recv(1024))
    data = s.recv(1024)
    f.write(data)
    size -= len(data)
    while(size>0 ):
      if size > 1024:
        data = s.recv(1024)
        size -= len(data)
```

```
      else:
        data = s.recv(size)
        size = 0
      f.write(data)
    f.close()
    s.close()
    print(face_expr(file))
    sys.stdout.flush()
    time.sleep(0.1)
    cmd = ''

```

(c) **coordination with the two Arduino**

Since we are going to keep the both two parts of the project working at the same time, we need to put the codes that controls the two Arduino in the correct places. Since the two Arduino boards are connected to the laptop, the code would be inserted to the client. As shown in the code above, the code starts with reading from the range detector. Same as the last part, we determine whether to display the tracking eyes or the green smilly face. If the any of the range detectors find someone in distance of 30, we would display the green face. If the middle range detector finds someone in distance of 30, we would send the signal for a new picture. This design make sense since the camera is placed near the middle range detector.

(d) **facial expression detection**

We are not able to design our own facial expression algorithm sadly. Instead, we used the Google cloud vision api for facial detection. The code for utilizing two functions mainly,detect face and face expr.

```
def detect_face(face_file, max_results=4):
  client = vision.ImageAnnotatorClient()
  content = face_file.read()
  image = vision.types.Image(content=content)
  return client.face_detection(image=image).face_annotations

def face_expr(file):
  with open(file, 'rb') as image:
    result = detect_face(image,1)
    if not result:
      return 0
    expr = 'joy'
    maxi = result[0].joy_likelihood
    if maxi < result[0].sorrow_likelihood:
      maxi = result[0].sorrow_likelihood
      expr = 'sad'
    if maxi < result[0].anger_likelihood:
      maxi = result[0].anger_likelihood
      expr = 'angry'
    if maxi < result[0].surprise_likelihood:
      maxi = result[0].surprise_likelihood
      expr = 'surprise'
    if max(maxi,result[0].under_exposed_likelihood,
    result[0].blurred_likelihood,
    result[0].headwear_likelihood) != maxi:
      expr = 'not sure'
    return expr

```

The main function calls the face expr function to start the facial detection process with a filename passed in. This is the file that the client received from the Raspberry Pi. The function then calls the detect face function, which does the heavy lifting of the

facial detection for us: connecting to the vision api, uploading, and fetching results. The return object look like this:

```
[bounding_poly {
  vertices {
    x: 156
  }
  vertices {
    x: 630
  }
  vertices {
    x: 630
    y: 542
  }
  vertices {
    x: 156
    y: 542
  }
}
...
landmarks {
  type: LEFT_EYE
  position {
    x: 351.2039794921875
    y: 266.00262451171875
    z: 0.0023154355585575104
  }
}
landmarks {
  type: RIGHT_EYE
  position {
    x: 467.5223388671875
    y: 262.3701477050781
    z: 15.99450397491455
  }
}
landmarks {
...
roll_angle: 0.7406389713287354
pan_angle: 7.649404525756836
tilt_angle: -9.674330711364746
detection_confidence: 0.9624384045600891
landmarking_confidence: 0.7042537927627563
joy_likelihood: VERY_UNLIKELY
sorrow_likelihood: VERY_UNLIKELY
anger_likelihood: POSSIBLE
surprise_likelihood: VERY_LIKELY
under_exposed_likelihood: VERY_UNLIKELY
blurred_likelihood: VERY_UNLIKELY
headwear_likelihood: LIKELY
]

```

Since we are doing facial detection, the information that we are interested in is at the bottom. However, the likelihood parameters are actually just integer names, so we have to parse it one by one, which is what the rest of the face expr function does. Finally, it would return the results as strings like "angry" or "sad".

(e) **the client code**

The combined code for the client is shown below:

```
import socket
```

```
import sys
import time
from google.cloud import vision
import serial
ser1 = serial.Serial("COM9", 9600)
ser2 = serial.Serial("COM6", 9600)

def eye(val1,val2,val3):
  rough = min(val1,val2,val3)
  if rough == val1:
    if val2 < val3:
      return 'k'
    else:
      return 'j'
  elif rough == val2:
    if val1 < val3:
      return 'k'
    elif val1 > val3:
      return 'm'
    else:
      return 'l'
  else:
    if val1 > val2:
      return 'n'
    else:
      return 'm'
def detect_face(face_file, max_results=4):
  client = vision.ImageAnnotatorClient()
  content = face_file.read()
  image = vision.types.Image(content=content)
  return client.face_detection(image=image).face_annotations

def face_expr(file):
  with open(file, 'rb') as image:
    result = detect_face(image,1)
    if not result:
      return 0
    expr = 'joy'
    maxi = result[0].joy_likelihood
    if maxi < result[0].sorrow_likelihood:
      maxi = result[0].sorrow_likelihood
      expr = 'sad'
    if maxi < result[0].anger_likelihood:
      maxi = result[0].anger_likelihood
      expr = 'angry'
    if maxi < result[0].surprise_likelihood:
      maxi = result[0].surprise_likelihood
      expr = 'surprise'
    if max(maxi,result[0].under_exposed_likelihood,
    result[0].blurred_likelihood,
    result[0].headwear_likelihood) != maxi:
      expr = 'not sure'
    return expr

port = 6457
host = "140.141.154.52"
file = "image.jpg"
print("connecting server",host,port)


cmd = ''
while True:
```

```
  val1 = ord(ser2.read(1))
  val2 = ord(ser2.read(1))
  val3 = ord(ser2.read(1))
  sig = eye(val1,val2,val3)
  if val2 < 30 or val1 < 30 or val3 < 30:
    sig = 'f'
  ser1.write(sig.encode())
  if val2 < 30:
    cmd = "Y"
  if cmd == "Y":
    s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    server = (host,port)
    print("connecting server",host,port)
    s.connect(server)
    s.send(b'now')
    f = open(file,'wb')
    size = int(s.recv(1024))
    data = s.recv(1024)
    f.write(data)
    size -= len(data)
    while(size>0 ):
      if size > 1024:
        data = s.recv(1024)
        size -= len(data)
      else:
        data = s.recv(size)
        size = 0
      f.write(data)
    f.close()
    s.close()
    print(face_expr(file))
    sys.stdout.flush()
    time.sleep(0.1)
    cmd = ''
```

(f) **Result**

The final picture of our project look like this:

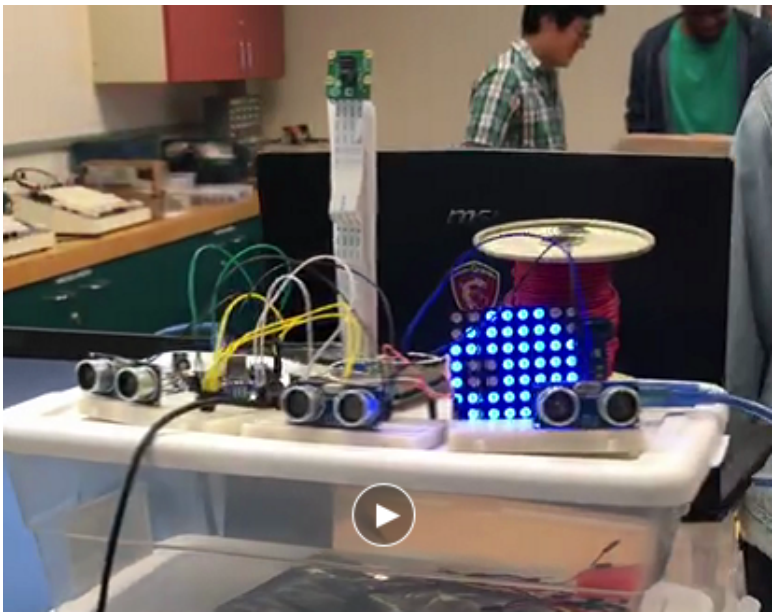

Figure 6: the whole picture

In addition to the last part, we are able to tell the Raspberry pi to take a picture when someone is in the 30 range of the range detector. We put a screen right next to (and connected with) the Raspberry pi for the user to see what the picture would be like. The file transmission from the Raspberry pi is a little bit lagging than when I tested at my dorm, by is able to do the job finally. After the file arrives, the client is able to upload it to the google cloud vision api and fetch the emotion of the image that google tells us. However, the facial expression recognition from google is not as accurate as I imagined.

5. **Conclusion**

To conclude with, this project accomplished most of the expectation written in our project proposal. Through this final project, we learned various new skills and made some of our old skills more robust. For example, we learned how to use the 8x8 LED, and how to do facial expression detection with Google vision API. At the same time we reviewed how to use ultrasonic range detector, the pi camera and raspberry pi, and the client server file transformation from our networking class. If we had more time, we would add some memory to the Arduino board, and implement more expression, so that we can pass the emotion information to the Arduino, and show it up on the LED. Another plan of mine was to add a voice system that say different things according to the emotion detected. It was a lot of pain and fun trying to fix all the bugs that appeared, and we were pretty proud of the final results. This is the last picture it took before we tore it apart.

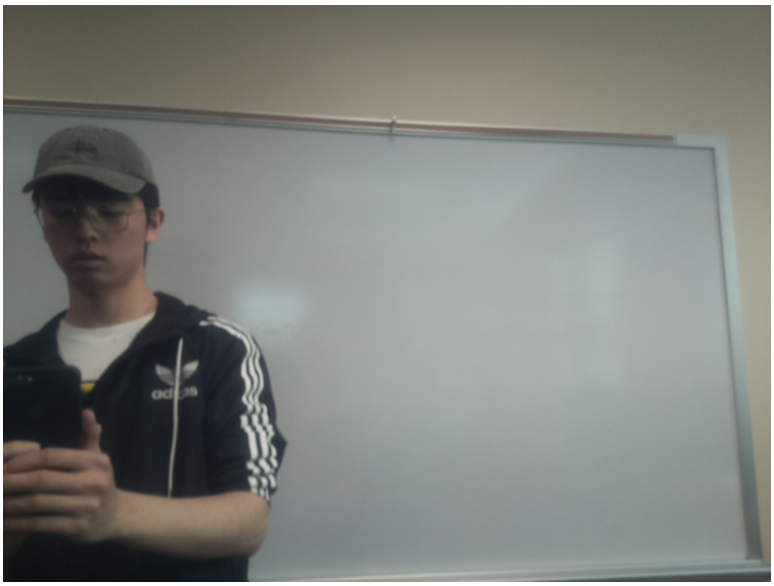

Figure 7: the last one