December 5, 2019

OurBuddy

CS 207 Project

Brandon Huzil & Lukas Hoffman

Table of Contents

1. Introduction & Background

Our project we have worked on for this semester is named OurBuddy. OurBuddy is a “robot” that searches for objects in its surroundings. If an object is detected, it reacts to the object, and also remembers it’s location. An LED is also included, which lights up when a new object is detected. OurBuddy searches random areas in the space in front of it roughly between 0 and 180 degrees on its “yaw” axis, and about 0 to 40 degrees on its “pitch” axis. The location where it looks depends on a random number that is generated. The maximum distance it reacts to an object is 60cm, and any object past this distance will be neglected. OurBuddy has a curiosity and boredom aspect to it. Once it finds a new object, its curiosity is increased and boredom is decreased. As its reacting to the same object, it will become more bored, and look for something else as he explores it more. We managed to include two different audio recordings of Dr. Tomesh that he made during lectures, They are “so cool!” and “woooah”. These recordings may play through OurBuddys audio jack and are picked randomly when OurBuddy detects a new object.

The Arduino Uno Rev3 and breadboard component, the SunFounder model PCA9685, 16-channel Servo driver, and the dedicated 5 x AA battery power source for the servo driver are the main components for OurBuddy. The Servo driver is used to power the three servos that are used for OurBuddy. One MG996R metal gear servo is used to rotate the neck portion on the yaw axis, another additional one for the pitch and one HXT900 plastic gear micro servo for the roll axis for the head. The head itself is made up of the plastic gear servo, and an Elegoo HC-SR04 ultrasonic distance measuring sensor.

Comparing OurBuddy to the original “Buddy” project based in the proposal, some functionalities and components were unfortunately cut. Our original proposal was to 3d print the body,

neck, and head parts of Buddy. We also wanted to include the audio using an SD card which would store Dr. Tomesh's voice recordings he used from lectures through the semester. The audio was to be used to express different emotions that Buddy was feeling at any given time while sensing an object in its area. Instead, we just put the audio recordings onto the Arduino using the method from lab 9 [10] without any mood attatched to it. The reasoning for cut functionalities/components was due to time constraints, as well as difficulties actually building OurBuddy. The original project we were to build off of is called LittleBot's Social Buddy, which was found on the *create.arduino.cc [1]* website. When the time came to create OurBuddy, the period to help support the Kickstarter program [2] had ended. One of the benefits to supporting was to attain the parts to build Buddy. Since it was no longer an option, we attempted to model our own parts to 3d print. The problem we encountered was the neck part of Buddy, which we eventually opted out of doing entirely. After conferring our dilemma with Dr. Tomesh, we decided to model our project using popsicle sticks, hot glue, and styrofoam instead.

2. Design & Building Phase

2.1 Assembling Parts

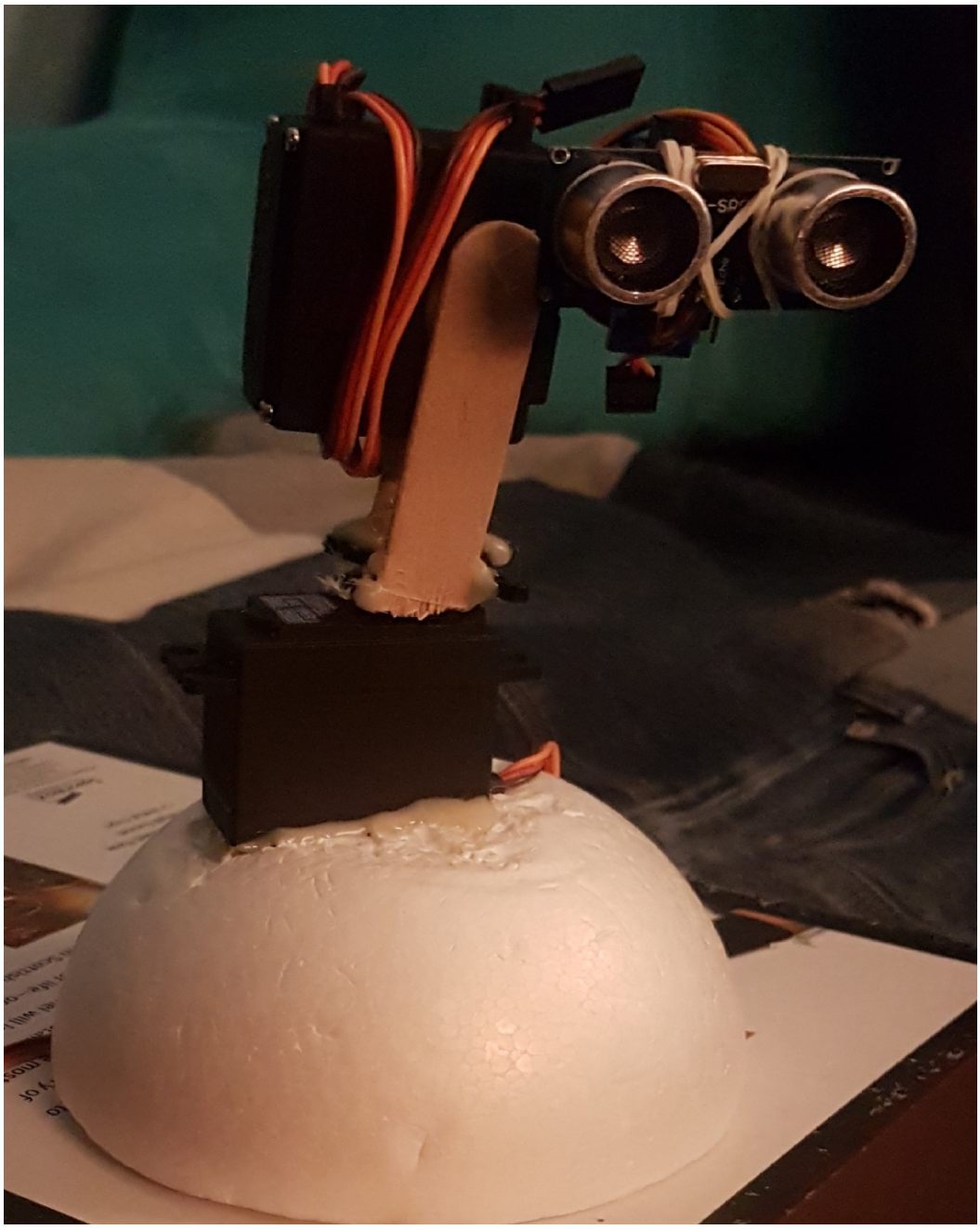
The main physical components to create OurBuddy consist of:

- 1x Elegoo HC-SR04 ultrasonic sensor.
- 1x elastic band
- 1x HXT900 plastic gear micro servo
- 2x MG996R metal gear servo
- 1x 111m x 55m styrofoam solid dome base (base for head & neck)
- 1x Arduino Uno Rev3
- 1x LED
- 1x Breadboard
- 1x SunFounder PCA9685 16-channel servo driver
- 1x AA Battery holder with barrel jack
- 5x AA 1.5v battery
- Jumper wires
- Glue
- Popsicle sticks
- Electrical tape

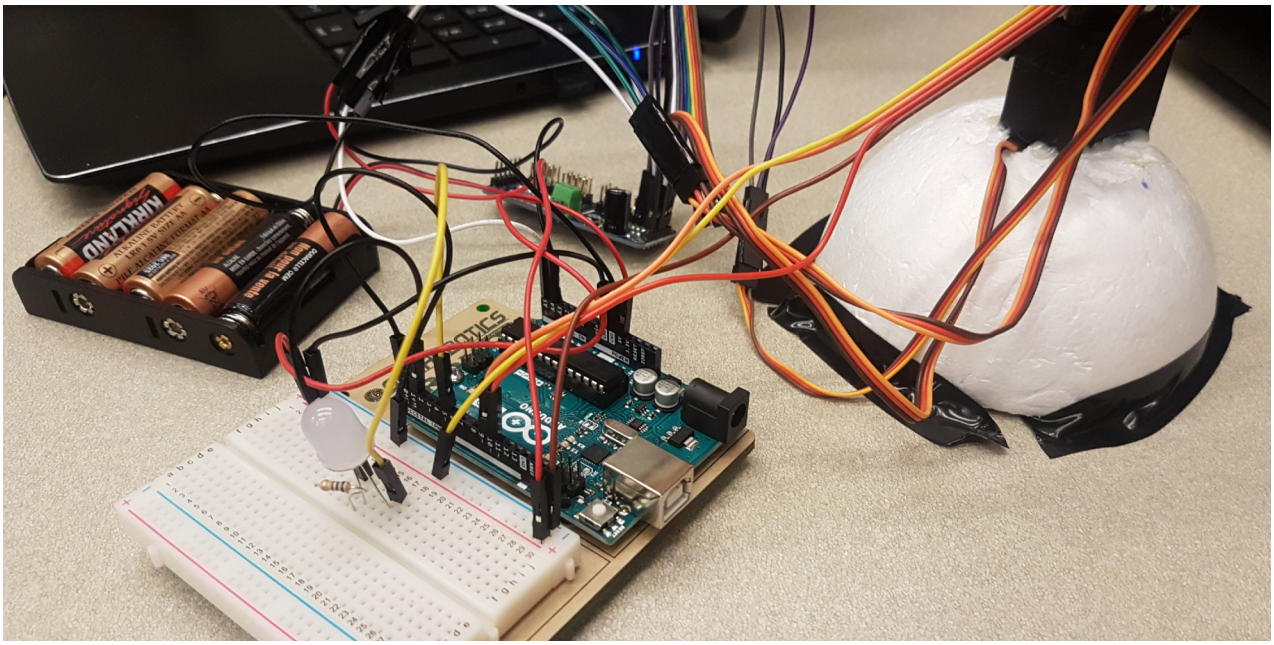
To assemble OurBuddy, we first had to order parts from Amazon: the ultrasonic sensor, the metal gear servos, and the servo driver. We then gathered the styrofoam body and neck/head base, and popsicle sticks from Michaels. We used our own hot glue gun to combine the parts together. The neck/head base had the rounded end shaved off roughly 5mm, to have a flat base to rest the first metal gear servo on. We then used a hot glue gun to glue the servo to the solid base. This first servo acts as our yaw axis to rotate the neck. With the other metal gear servo, we glued two halves of a popsicle stick to both sides of it. After it dried, we glued the other end of the popsicle sticks to the first servo. The third plastic gear servo was glued to the second servo, and then the ultrasonic sensor was attached to it

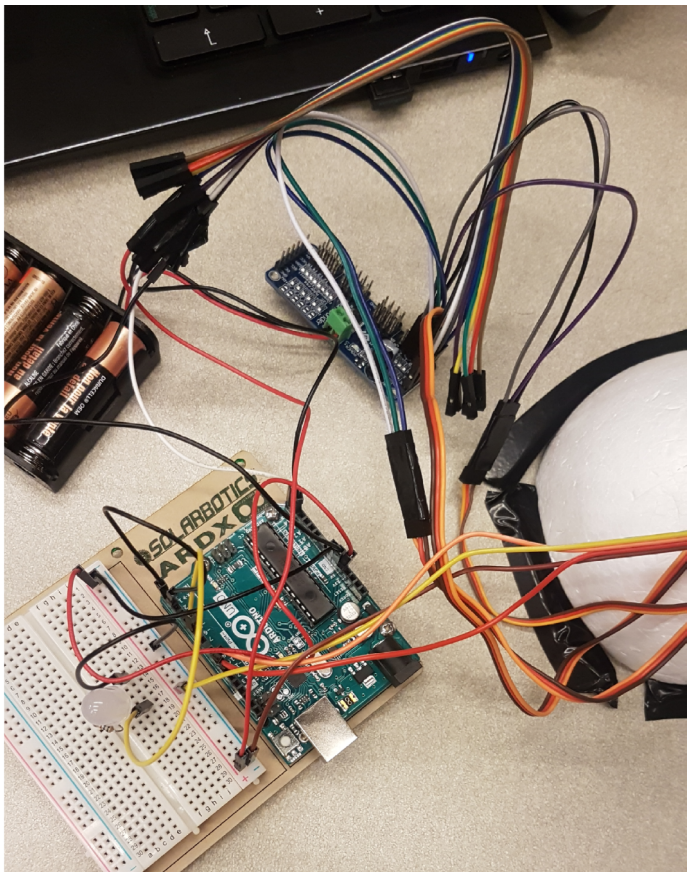
with an elastic band, as well as electrical tape. Below in section 2.3, are pictures of the progress thus far. The Arduino, servo driver, and power source are placed alongside. The ultrasonic sensor is connected to the Arduino, while the servos are connected to the servo driver. The servo driver is connected to the Arduino via I2C communication, and also powered with the battery pack with barrel jack connector. The barrel jack was stripped off of the wires since the servo driver's power block was used in conjunction with individual wire 5v and ground pole connectors, and not the 2.1 mm jack port.

2.2 Image: OurBuddy Assembled



2.3 Images: The Mess





***Note: Circuit diagrams can be viewed in section 3.1.2, and section 3.2.2.**

3. Setting Up OurBuddy

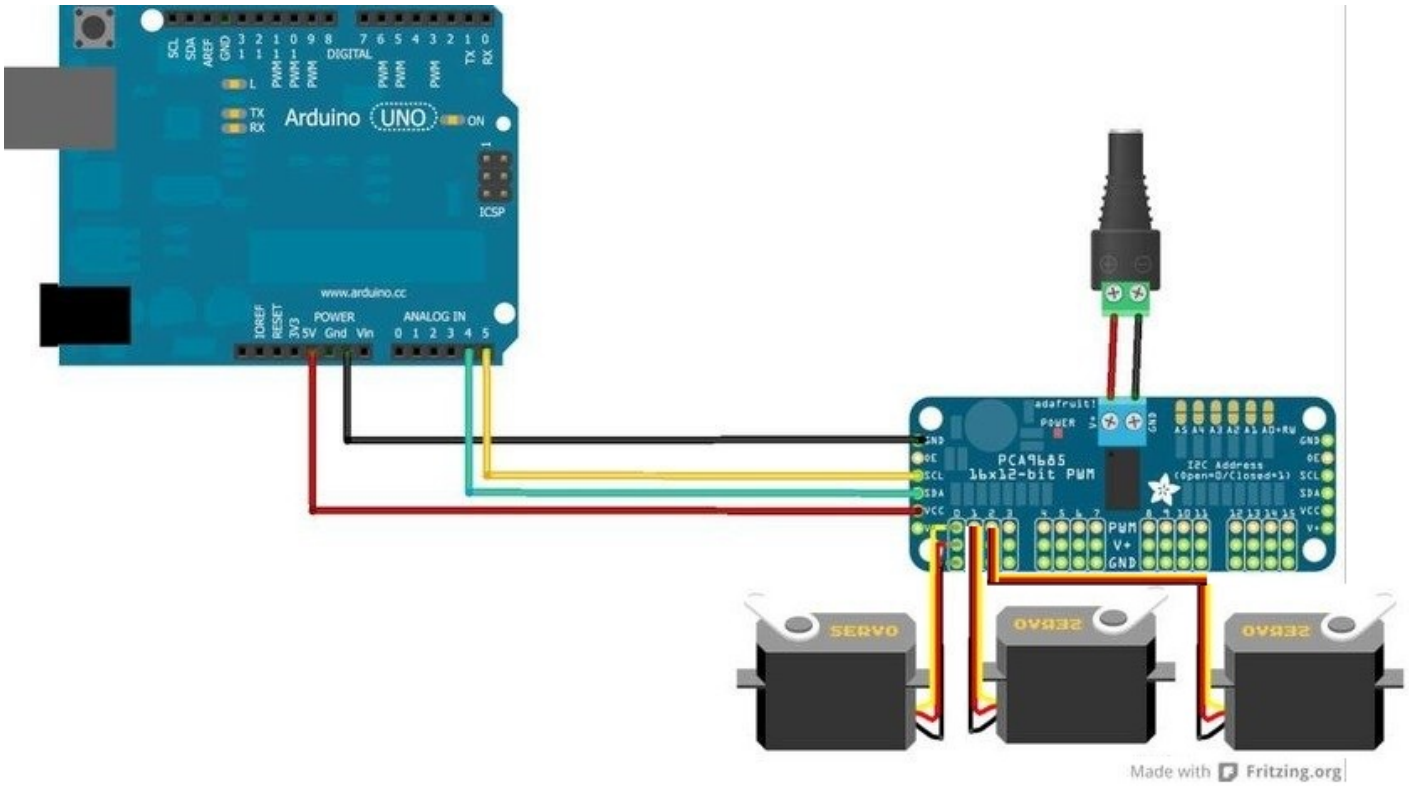
3.1 SunFounder 16-channel Servo Driver

Setting up OurBuddy has been briefly discussed throughout the report. Assuming all the parts are gathered and prepared, ie: styrofoam body base has been cut, setup is fairly straightforward. All that needs to be done is connecting the jumper wires to the correct interfaces and components. The I2C connection between the servo driver [5] and Arduino is potentially the most difficult aspect. 4 female to male jumper wires are needed to connect both components together. Starting with the female and servo driver, and mapping to the Arduino, connect the wires from:

Servo Driver	Arduino Uno
GND	GND
SCL	A5
SDA	A4
VCC	5V

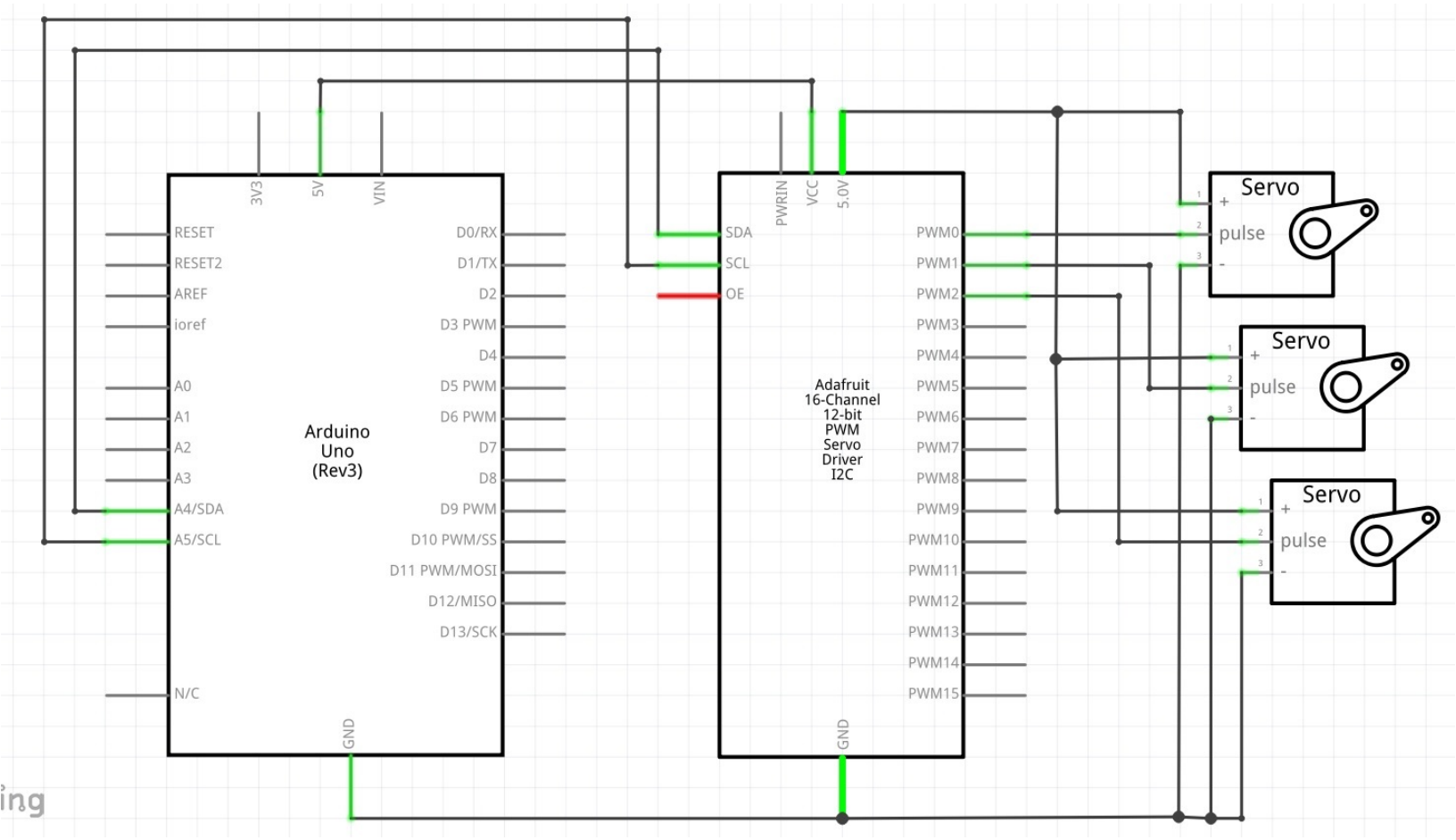
To connect the servos, we see that there are 16 channels on the driver labeled from 0 to 15. Connect the 3 pin servo to the 3 pin channels accordingly at the 0, 1, and 2 positions. We can then connect the dedicated power supply to the servo driver, to power the servos. A diagram example can be viewed at section below here in section 3.1.1, and circuit diagram in section 3.1.2. To use the servo driver, a library from Adafruit [3] was downloaded and used to control each servo. Normal function usage with the servo library would look similar to servo.write(). With the Adafruit library, we would use servo.setPWM(channelNum, startPulse, endPulse). This uses pulse width modulation.

3.1.1 Image: I2C for servo driver & Arduino Hardware



[4]

3.1.2 Image: I2C for servo driver & Arduino Circuit



[7] *Note: Found a library for the servo driver. Previous hardware diagram would look the same though!

3.1.3 Image: Code Example – makeMovement()

```
//function moves the pitch and yaw servos so that they both reach the desired angle at the same time smoothly
int makeMovement(int furtherDistance, int lesserDistance, int * furtherPosition, int * lesserPosition, int furtherDirection, int lesserDirection,
Adafruit_PWMServoDriver * furtherTurningServo, uint8_t furtherTurningServoNum, Adafruit_PWMServoDriver * lesserTurningServo,
uint8_t lesserTurningServoNum, int lesserFromHigh, int furtherFromHigh, int lesserToHigh, int furtherToHigh)
{
  int totalTicks = furtherDistance + lesserDistance;
  float lesserQuotient = (float)totalTicks / (float)lesserDistance;

    //for loop ensures that movement in both axis is done as evenly as possible
    int scaler = 1;
    for(int tick = 1; tick <= totalTicks; tick++)
    {
      if(tick == ceil(scaler * lesserQuotient))
      {
        (*lesserPosition) += lesserDirection;
        lesserTurningServo->setPWM(lesserTurningServoNum, 0, SERVOMIN + map((*lesserPosition), 0, lesserFromHigh, 0, lesserToHigh));
        //lesserTurningServo->write((*lesserPosition));
        scaler++;
        delay(turnSpeed*4);
      }
      else
      {
        (*furtherPosition) += furtherDirection;
        furtherTurningServo->setPWM(furtherTurningServoNum, 0, SERVOMIN + map((*furtherPosition), 0, furtherFromHigh, 0, furtherToHigh));
        //furtherTurningServo->write((*furtherPosition));
        delay(turnSpeed*4);
      }
    }
    delay(500);
    return makePing();
}
```

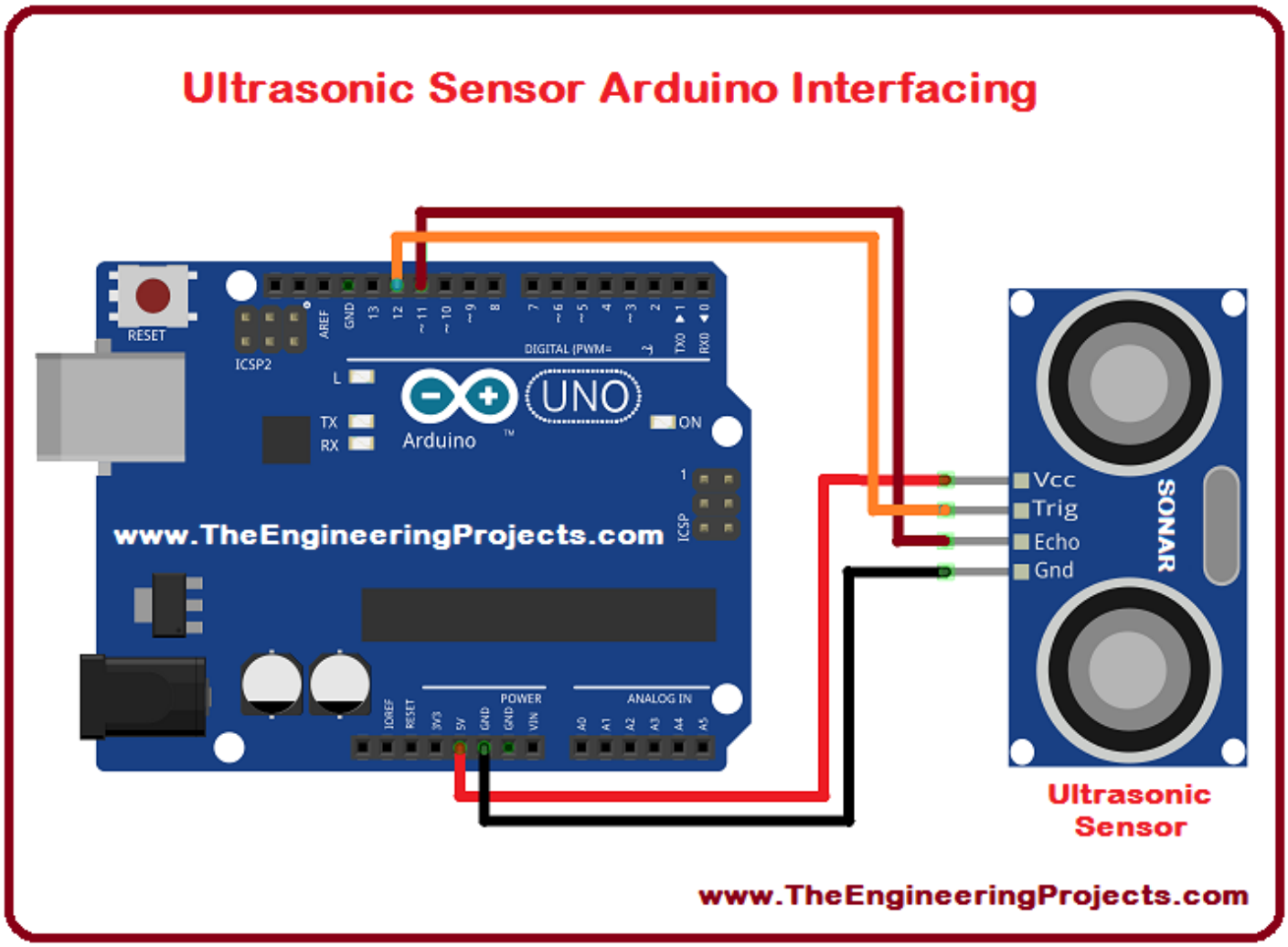
3.2 Elegoo HC-SR04 Ultrasonic Sensor

The ultrasonic sensor [8] is rated to have a minimum range of 4 cm, and a max range of ~450 cm. Using it in practice however, we found anything after 150 cm to be a little unreliable at times. The HC-SR04 has 4 pins to attach to the Arduino. Similarly to the servo driver, there are 4 female end wires to be attached to the sensor, and the male end to the Arduino. An example would be:

Ultrasonic sensor	Arduino Uno
GND	GND
ECHO	11
TRIG	12
VCC	5V

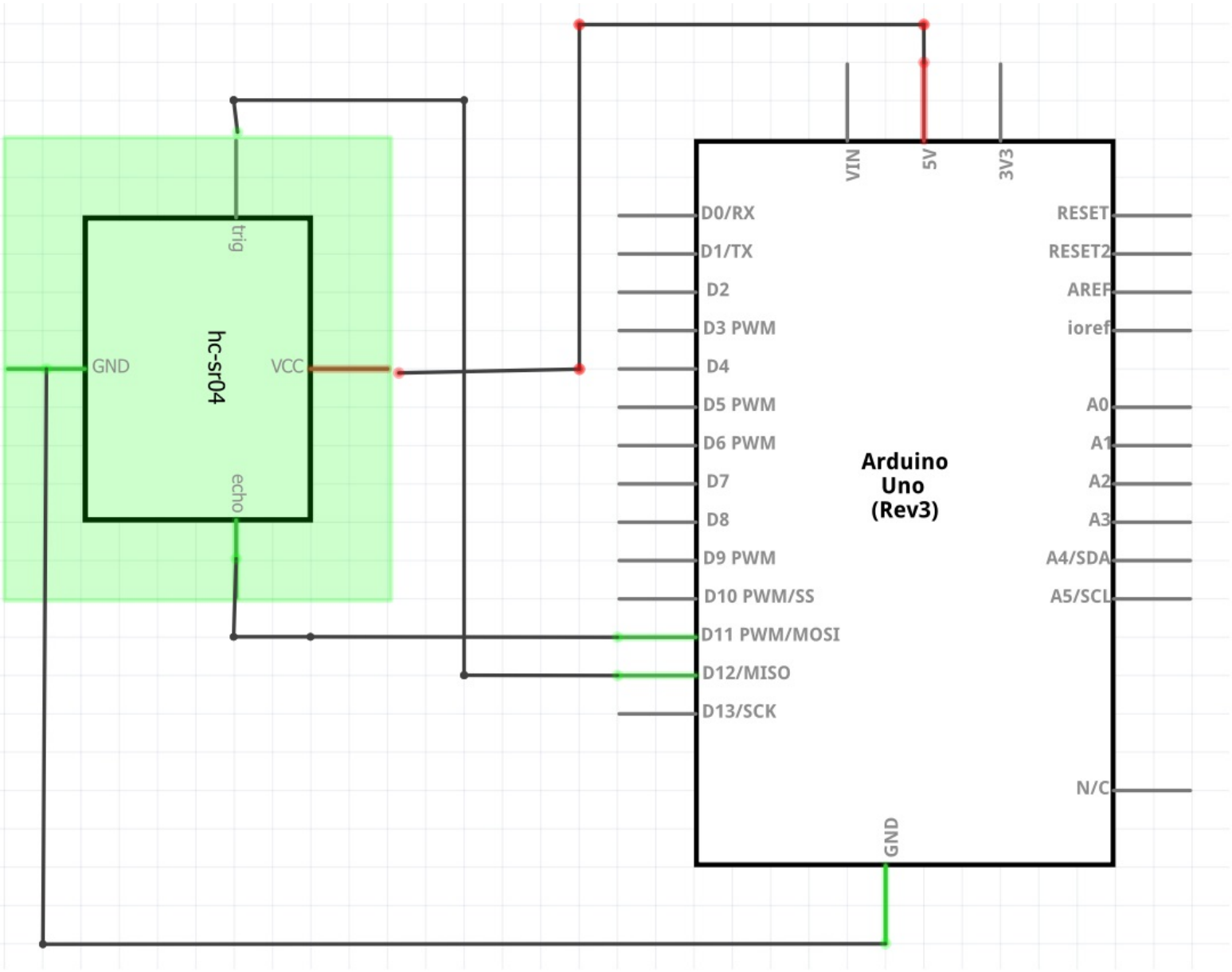
The ultrasonic sensor [6] works by creating a short 10uS or greater burst of soundwaves at a frequency of ~40hz when its trigger pin is sent a signal. At this time, a timer is started and then stopped when the soundwaves sent out are bounced back to the receiver. To take into account of the soundwave being sent out, and then back to the sensor, the timing value is divided by 2. A hardware diagram is provided below in section 3.2.1, and circuit diagram in section 3.2.2.

3.2.1 Image: 4-pin connection between SR-04 sensor and Arduino Uno Hardware



[9]

3.2.2 Image: 4-pin connection between SR-04 sensor and Arduino Uno Circuit



[7] *Note: Adafruit library contained model for ultrasonic sensor.

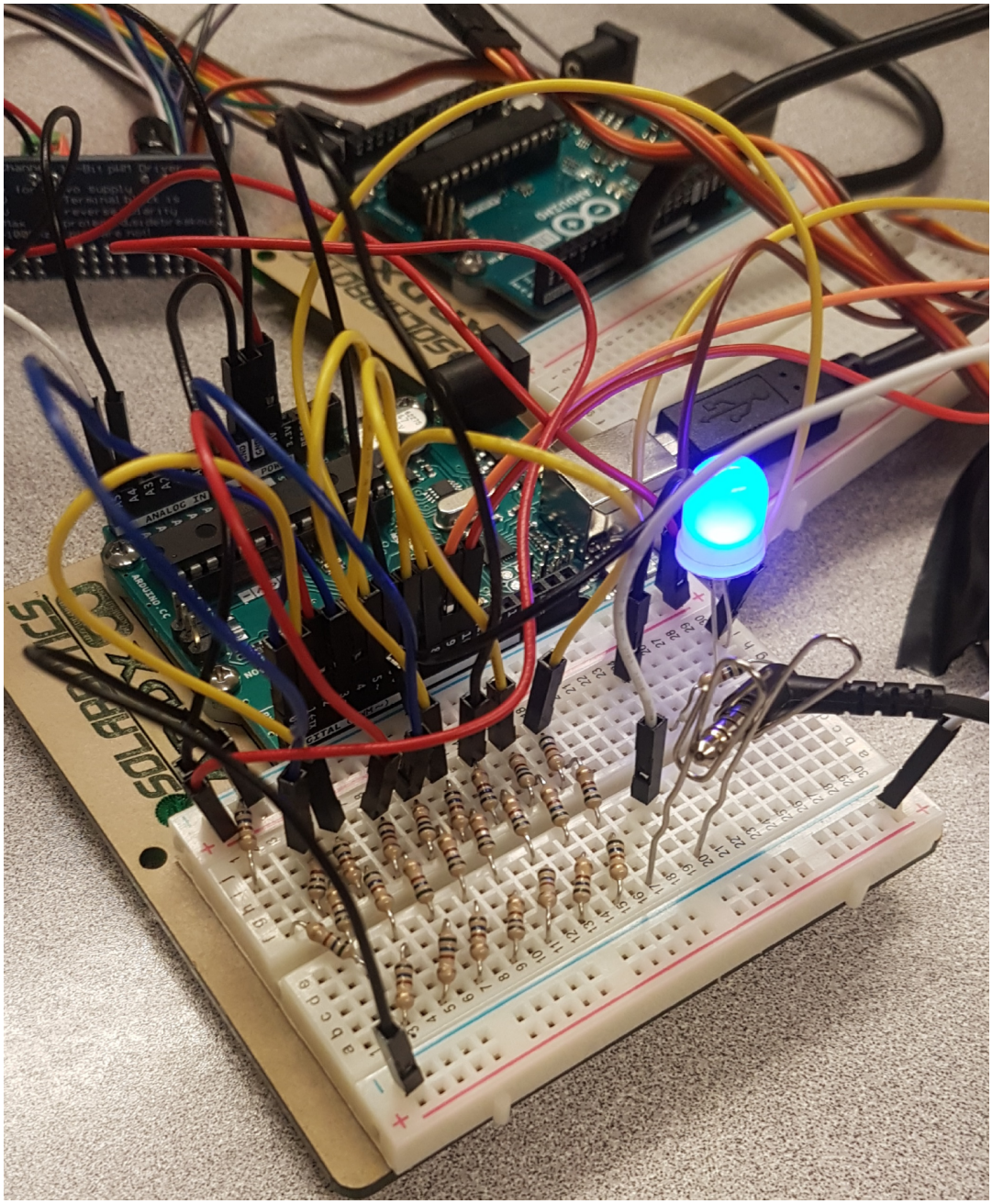
3.2.3 Image: Code Example – makePing()

```
// triggers the ultrasonic sensor to measure the distance to the point in front of it
// it then stores this value in the pointmap
// returns 1, 2, or 3 for a new distance, a distance within 2cm of previous(old), or over 60cm (too far to care) respectivly
// also contains serial.prints so a programmer can see the values and what our buddy considers then, new, old, or far
int makePing()
{
  // Set low to ensure trigpin is clear
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);

  // Emit pulse for 10 microseconds and turn off
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Start pulseIn timer
  duration = pulseIn(echoPin, HIGH);

  // Calculate by multiplaying by speed of sound and dividing by 2
  // since the sound wave emits to object, and then bounces back to sensor
  distance = duration *= 0.034 / 2;
  if(distance >250 || distance <4){delay(10);}

  Serial.print(distance);
  Serial.print(" cm  ");

  if (distance <= 60)
  {
    // Determine if object sensed is "new"
    if (distance < (pointMap.getPoint(yawPosition, pitchPosition) - 2) ||
    distance > (pointMap.getPoint(yawPosition, pitchPosition) + 2))
    {
      //Senses something new!
      pointMap.setPoint(yawPosition, pitchPosition, distance);
      Serial.println("NEW");
      Serial.println();
      digitalWrite(5, HIGH);
      delay(1000);
      digitalWrite(5, LOW);
      return 1;
    }
    // Senses something, but is not new
    pointMap.setPoint(yawPosition, pitchPosition, distance);
    Serial.println("OLD");
    Serial.println();
    return 2;
  }
  // Too far away to care
  pointMap.setPoint(yawPosition, pitchPosition, distance);
  Serial.println("FAR");
  Serial.println();
  return 3;
}
```

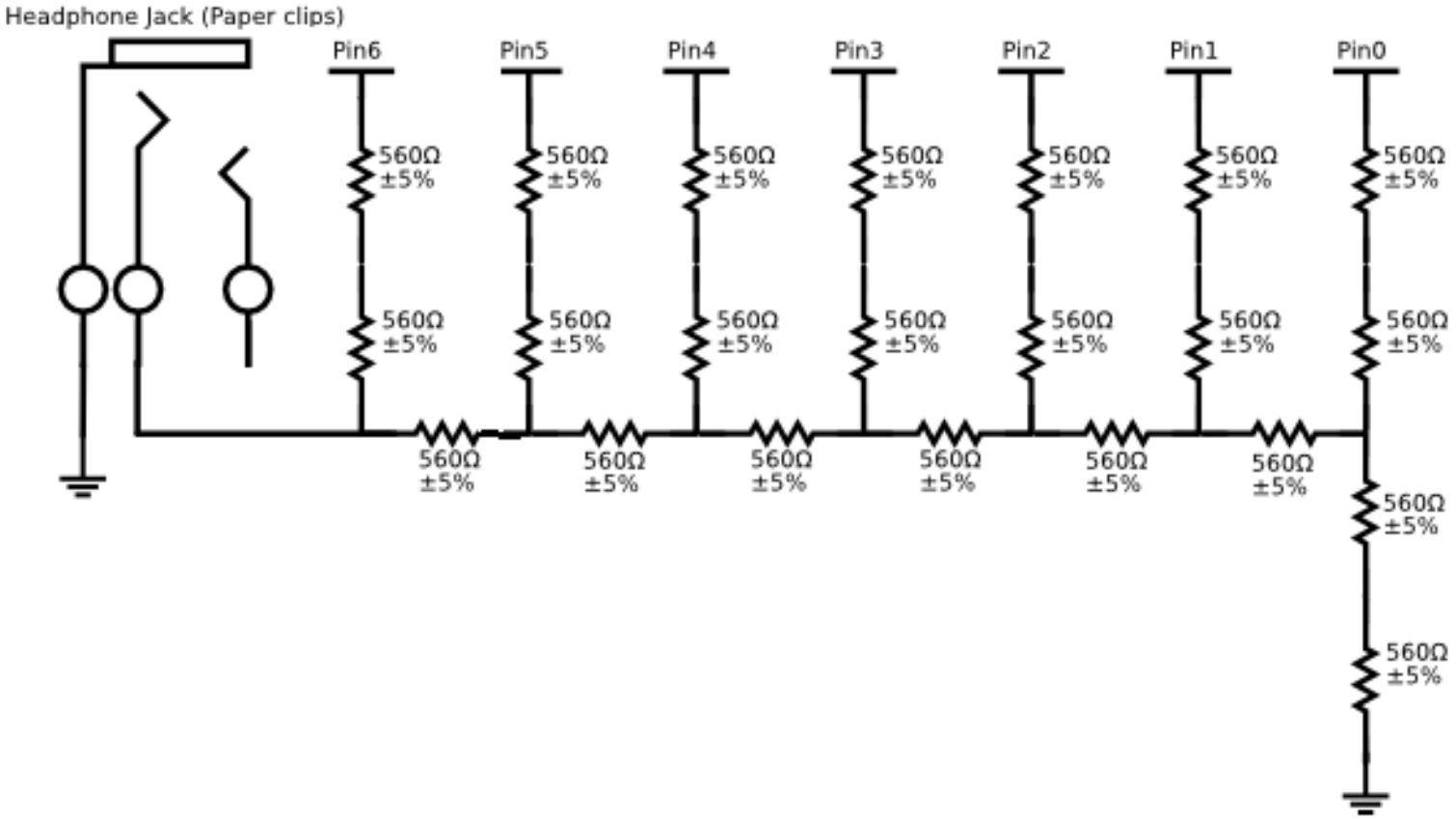
3.3 Audio with Headphone Jack

To include our audio portion of the project, we didn't have to do very much. We followed the notes and code from lab 9 [10]. The same circuit that was provided in the notes was used and shown below in section 3.3.2.

3.3.1 Image: Audio with Headphone Jack Hardware



3.3.2 Image: Audio with Headphone Jack Circuit



[10] *Note: Reused circuit diagram from lab

4. Milestones

Our main goal to create an “anthropomorphized” robot was more or less achieved. OurBuddy searches its environment and reacts to objects on detection. This being the main purpose intention, was executed well. Many things could still be improved on if we were to continue developing OurBuddy, especially looks! To further improve, we could add different components, such as the Arduino Nano, instead of the Arduino Uno and breadboard combo we worked with here. If we were to provide a better and more detailed model, we could include all the components in a neater fashion, and get OurBuddy to look like a real detailed robot. With the code, we could organize and separate it in a more clear manner to improve efficiency.

The biggest drawback of the project would be the content we thought would be provided by the original project we were referring to [1][2]. As mentioned previously, we had missed the window to support the Kickstarter project to acquire the materials and models needed to create a 3D printed version of OurBuddy. We attempted to model our own version, but found the neck portion to be problematic in terms of servo incorporation. In the end, we have what we have now. Turned out to work fairly well to build, with wire management and the super gluing process to be the most difficult to look neat. Another setback was a previous build state contained a much larger array to hold point distance then we have now, however it pushed our memory usage over 1200% of what was available, oops. We also originally had object distances saved in EEPROM for a more efficient boot up of OurBuddy in a familiar environment, but accidentally killed it with too many writes and lost a $100 Arduino Uno in the process… May have saw that coming.

4.1 Original Milestones

Milestone 1	**NOV 1st**	Materials gathered and printed.
Milestone 2	**NOV 7th**	Research completed on C++ mapping and assembly completion.
Milestone 3	**NOV 21st**	Basic movement without environmental interaction.
Milestone 4	**DEC 2nd**	Movement with environment interaction and audio features.
Milestone 5	**DEC 6th**	Coloured LED's to match Buddy's reactions. Presentation complete.

Our actual milestones were achieved in a different order than originally intended. We ended up doing different bits and pieces as development progressed. First we ordered the parts such as the ultrasonic sensor and servos online. The parts came in fine, but we were unable to print the original project. As we were trying to decide how to solve this problem, we started on the coding aspect with the parts we had ordered online. The code to create the "environment" of OurBuddy was also not included in the code sample that was provided online. The original project had little functionality that did not include much. As a result, we had to come up with our own method for OurBuddy to remember its surrounding area. We wanted to originally create an actual map in front of OurBuddy, but proved to be a little more complicated to accomplish in the time we had left. During the mapping process, we worked on basic movement of the servos. It originally scanned the area from left to right, and repeated. The easiest functionality was getting the ultrasonic sensor to work correctly on its own. We worked on all of the other aspects in parallel to the map, so we would at least have something that could detect and react to objects in front of it. At this point we almost had ruled out providing any audio component

entirely. We managed to incorporate audio recordings from the lectures at the last minute. The last original milestone to have an LED was met in a timely matter and did not have too much difficulty with it.

5. Contributions

Brandon Huzil's contributions include most of the coding aspect, creating the "mapped" area that OurBuddy stores object distances into, servo movement, the program structure, integrating all functionality together, and the behavior of OurBuddy, such as his curiosity for new objects.

Lukas Hoffman's contributions include setting up the audio circuit on the breadboard, and making the audio functions in the code. He also set up the code for the proximity sensor to work. He wrote most of the documentation as well as the circuit and hardware images. He was responsible for format of the documentation and references used.

Both Brandon and Lukas contributed to the construction and design of OurBuddy and worked on using the Adafruit_PWM library functions with servos. It took both of us to notice each other's silly or tedious mistakes along the way, whether it was pin connections, bad variable initial values, or even missing semi-colons.

6. Code Listings

6.1 Libraries

#include <Adafruit_PWMServoDriver.h>

Used to move the servos with the use of pulse width modulation.
Installing this library included example sketchs, including a servo example that works with I2C. Default values, initializing, and setup from this sketch was used as reference to ours. [3] Code can also be seen here [5]

#include <Wire.h>

Used for the I2C connection between the Arduino and servo Driver.

#include <avr/pgmspace.h>

Used for flash memory to store the audio and to read from the array

6.2 GitHub

Our code can be viewed here on GitHub:

https://github.com/BA-Huz/CS-207-project

7. References

[1] LittleBots. Buddy, the 3D-Printed Arduino Social Robot. Retrieved December 2, 2019 from:

https://create.arduino.cc/projecthub/slantconcepts/buddy-the-3d-printed-arduino-social-robot-ec3dca

[2] Slant Robotics. Buddy. An Arduino Social Robot. Retrieved December 2, 2019 from:

https://www.kickstarter.com/projects/slantrobotics/littlebot-buddy?ref=creator_nav

[3] Adafruit. Adafruit PWM Servo Driver Library. Retrieved December 4, 2019 from:

https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library

[4]Adafruit. Hooking it Up. Retrieved December 3, 2019 from:

https://learn.adafruit.com/16-channel-pwm-servo-driver/hooking-it-up

[5] SunFounder. PCA9685 16 Channel 12 Bit PWM Servo Driver. Retrieved December 4, 2019 from:

http://wiki.sunfounder.cc/index.php?title=PCA9685_16_Channel_12_Bit_PWM_Servo_Driver

[6] Components101. HC-SR04 Ultrasonic Sensor. Retrieved December 3, 2019 from:

https://components101.com/ultrasonic-sensor-working-pinout-datasheet

[7] Adafruit. Adafruit parts, components, breakouts, etc...in Fritzable format! Retrieved December 4, 2019 from:

https://github.com/adafruit/Fritzing-Library

[8] ElecFreaks. Ultrasonic Ranging Module HC - SR04 . Retrieved December 3, 2019 from:

https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[9] TheEngineeringProjects. Ultrasonic Sensor Arduino Interfacing. Retrieved December 4, 2019 from:

https://www.theengineeringprojects.com/wp-content/uploads/2017/08/Ultrasonic-Sensor-Arduino-Interfacing_1.png

[10] University of Regina. Lab 9: RGB LED and Digital Music. Retrieved December 6, 2019 from:

http://www.cs.uregina.ca/Links/class-info/207/Lab9/