COMP 444 Project

By: Michael G. Oranski

Date: March 2, 2019

User ID: 2743708

Course ID: COMP444

Index

Project Proposal

As someone with a severe hearing loss, I would like to design a robot or series of robots that would alert me to certain sounds in the most efficient manner possible. I would also like to further explore the different type of feedback control discussed in chapter 10.4 of the textbook. My proposal:

"A robot that seeks out the source of certain sounds while avoiding collisions with objects in it's way. Once the robot thinks it found the source of the sound, it sends an alert (ex. by sound, light, picture, etc.) to indicate that it has found it. I would like to analyze the efficiency of the feedback control by timing how long it wold take the robot to find the source of the noise under each control type and how close it got to the source."

Background Information

PID Feedback Control

The heart of feedback control is control theory, a subfield of mathematics that deals with the control of continuously operating dynamical systems. Feedback control is a means of getting systems to achieve and maintain a desired state. This control is done by continuously comparing the current state with the desired state.

A controller monitors a controlled process variable and compares it with a set point. The difference between the current and desired value of the process variable is called the error signal. This error signal is then applied as feedback to generate a controlled action to bring the process variable to the same value as the set point.

A proportional-integral-derivative (PID) controller is a type of control loop feedback mechanism. This controller continuously calculates an error value which is the difference between the desired and measured process variable. The controller then applies a correction based on proportional, integral and derivative term.

Proportional control occurs when the system responds in proportion to an error. Derivative control gives the controller additional control action when the error changes consistently. Integral control occurs when the system keeps track of its own errors and adds the accumulation to the feedback. PD control is the combination of proportional and derivative control while PID control is the combination of proportional, integral and derivative control.

For this project I will explore the efficiency of the following feedback controls: proportional, integral, derivative, PD and PID control.

Hearing loss

Anakusis (Hearing loss or hearing impairment) is the partial or total inability to hear and may occur in one or both ears. The measurement of hearing loss in an individual is conducted over several frequencies: 500 Hz, 1000 Hz, 2000 Hz, and 4000 Hz. The hearing loss of the individual is the average of hearing loss values over different frequency. My hearing begins to degrade around 1000 Hz in the right ear and 500 Hz on the left ear. Taking the average between them, the robot should only detect sound frequency no lower than 750 Hz.

Checkpoint One: Movement and Object Avoidance

Feb 5, 2019 to Feb. 8, 2019

I have ordered the parts from Amazon for detecting sound and frequency. As I am unfamiliar with these components, I bought a variety of them and will be testing them out. Until they get here I will be creating the movement and object Avoidance part of the robot. Substituting the sound sensor with the photoresistor for this section. When the sound sensor comes in, I can hopefully just swap the light sensor for the sound sensor without too much difficultly.

Part 1

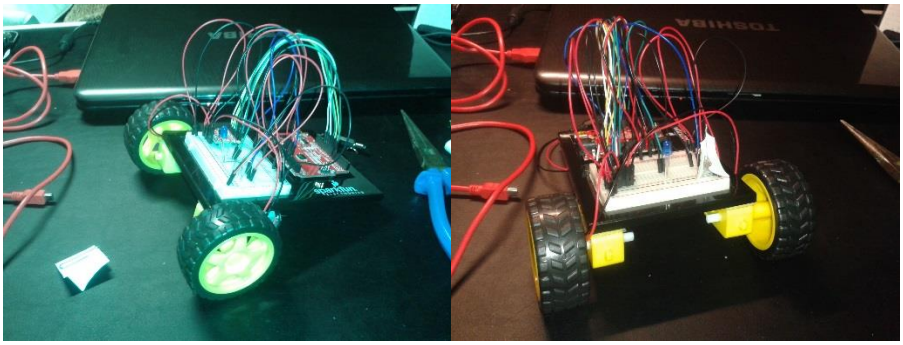
Testing the photoresistor and recording the light levels. My goal is to use a flash light in order to trigger the robot to move towards the light. I also wanted the robot to face towards the light source and still be able to detect it from all around. For this I bent the photoresistor to face forward in the robot's direction of travel. Next, I took a small piece of paper (smaller then those sticky note pads), bent it in half and held in place by a paper clip. I placed the bent piece of paper behind the photoresistor so that the only opening is in front of the robot.

I tested this set up using serial output, checking for light levels. In an unlit room, the light level is around 400. With the flash light coming from behind the paper, the light level is around 800. When the light is shined directly on the photoresistor, it is over 900. I tried testing without the paper, and the light level is around 900 regardless the direction the flashlight shine on it.

Part 2

Next step is to have the robot shift itself to face the direction of the light source when detected. For this I added the two gear motors, wheels and one LED light. The LED light is to provide a visual indicator to the human that it has finished shifting towards the light. For software, I am reusing the motor functions from circuit 5C. Upon its first iteration, the robot calibrates itself to the room natural light level.

Now the robot has two ranges of light to detect: first range is detecting spike in light level compared to normal and a second range is detecting when light level passing a pre-determined threshold. If the first range is detected, the robot begins to spin around towards the source of the light. On the second range, the light source has been found and will flash the LED.



(**video**) Movement and Object Avoidance; face the light.mp4

Part 3

Next is to have the robot go towards the source of the light after turning to face it. For this section, I encapsulated the motor's movement in functions that determine its direction (forward, backward, right, left). The functions parameters are motor speed and distance.

I created two other functions called adjust() and zeroIn(). Adjust() is suppose to make the robot do some minute adjustment to it's position so that it can better center itself on to the source of the light. zeroIn() is suppose to move the robot towards the light as much as possible. Testing these functions resulted in a failed attempt:

(**video**) Movement and Obect Avoidance; move towards the light fail.mp4

After toying with the functions and the sensitivity of the robot, I found that the problem was the adjust() function. It was causing it to over adjust and spinning out of control, like in the video. I removed the function and did some minimal adjustment to the other functions. Re-testing the robot resulted in a successful run:

(**video**) Movement and Obect Avoidance; move towards the light success.mp4

Part 4

The final part is to add in the ultrasonic sensor to enable the robot to avoid objects. From past experience (Circuit 5C), I thought there was a limit to the number of PWM pins that can be active at the same time. Even still, I believe that I could hook up both the motors and the ultrasonic sensor to the PWM pins. I hooked up the ultrasonic sensor while making note of the senor's pin location. Next I copied the getDistance() function from circuit 3B and added some serial print to check for proper output. Testing this showed that the ultrasonic distance sensor was properly working.

(**video**) Movement and Obect Avoidance; Distance sensor 1.mp4

The last step to this part is to integrate the new measurement into the zeroIn() function. With this new measurement, the zeroIn function works in the following way:

1. If there is an object blocking the way, turn to the right
2. If no object but not close to the light, move forward
3. If no object and close to the light, flash LED.

This part required a few tests in order to get the sensitivity right. Here are a few failed attempts:

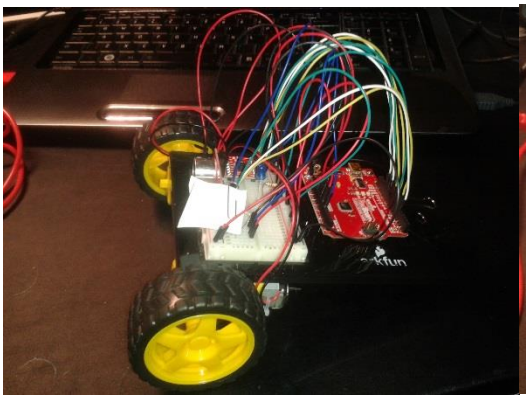
(**video**) Movement and Obect Avoidance; Object Avoid fail 1.mp4

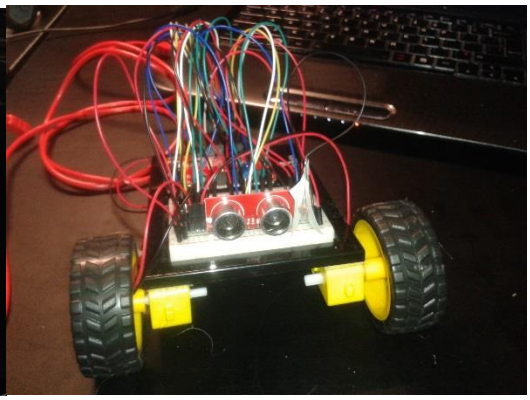
(**video**) Movement and Obect Avoidance; Sensitivity adjustment needed.mp4

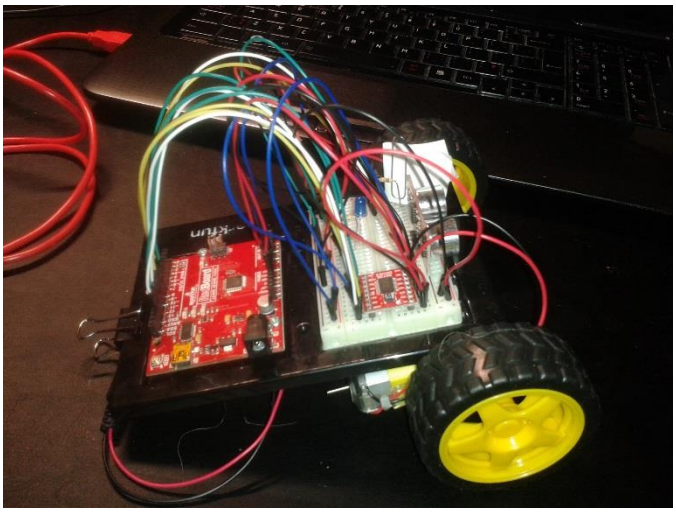
Eventually I was able to get the sensitivity correct:

(**video**) Movement and Obect Avoidance; object avoidance success.mp4

As you can see from the video, it moved towards the light before detecting the object. Then it started to move away from the light, at which point I stopped it by shining the light on the photoresistor. My reason for this is that I didn't add in the logic for how the robot should reorient itself afterwards. This is easily fix by having the robot move forward once in the new direction, reorient itself and resume.







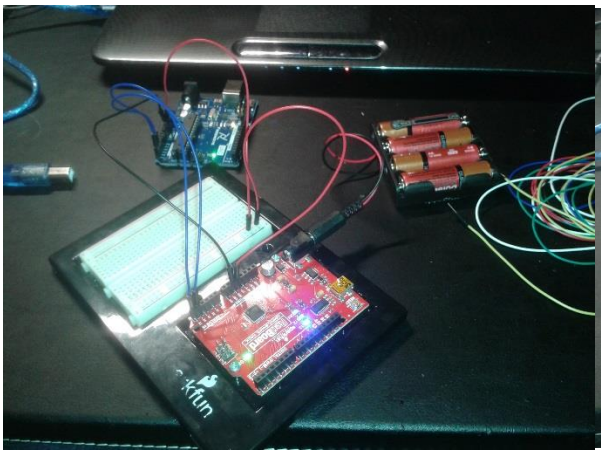
Checkpoint Two: Arduino to Arduino Communication

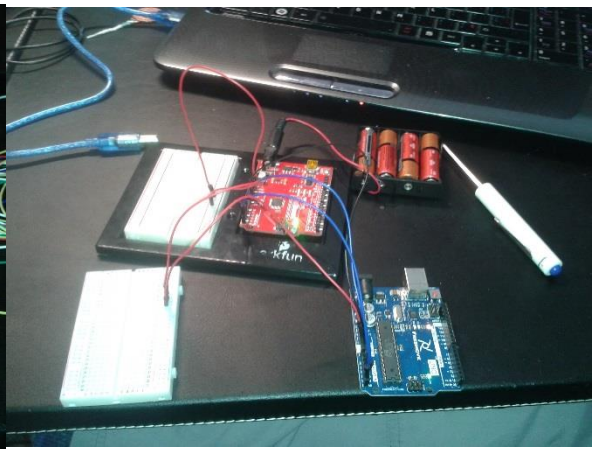
Feb. 9 to Feb. 10, 2019

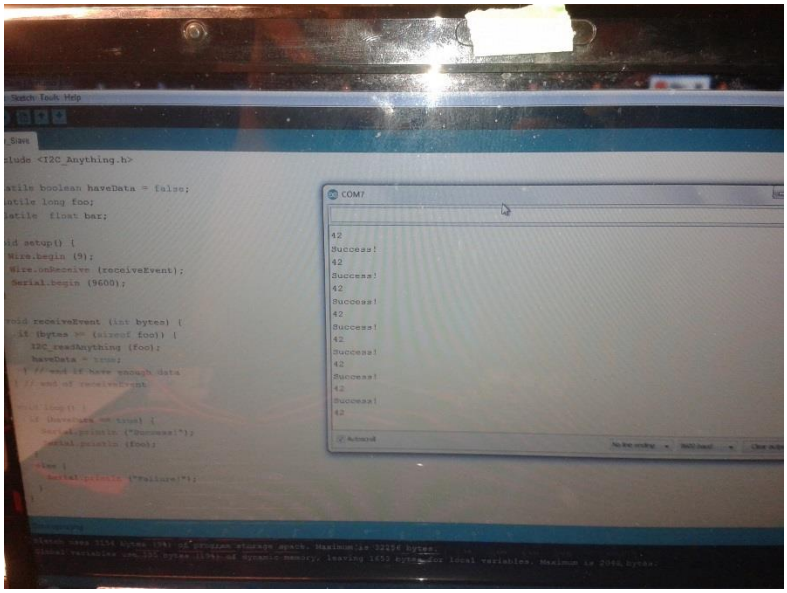
Part 1

One of the problems with the current circuit setup from the previous checkpoint is the increasing lack of space on the breadboard and microcontroller. A solution for this problem is to add a second set of microcontroller and breadboard and link them together. To achieve this solution I will be using the library Wire.h. This library allows for the communication with I2C / TWI (2 Wire Interface) devices. Using this library I will set up the two devices in a master and slave relationship with the master device collecting the sensory information and the slave device controlling the motors.

My first attempt at this solution was to connect the second microcontroller to the first one. The goal is to have the 2nd Arduino board control the blinking LED on the 1st Arduino board. The data being sent over are integers ranging from 0 to 5 inclusive. On 6 the first Arduino board resets to 0 and on 0 or 3 changes the blinking speed on the second Arduino increases.







This attempt at device to device communication was a success but it brought up some shortcomings. The main problem was that the data was being read one byte at a time. This solution was a problem because it limited the type of data that I could send.

There are at least three solutions:

1. I work with the problem by sending only char values over the line. My original plan would have to change to accommodate this issue by having the master device do all the calculations and then sending over movement commands in the form of ‘f’, ‘b’, ‘l’ and ‘r’.
2. I try to reconstruct the other variable types from the bytes. This reconstruction has its own problem due to the fact that the Arduino language is created from C/C++. C and C++ compilers do not necessarily execute function calls in left to right order in certain statements such as:

   int received = Wire.read() << 8 | Wire.read();

   This is because ‘|’ does not define a sequence point. A solution to this is to break it into separate statements:

   int received = Wire.read () << 8;

received |= Wire.read();

For some data types this execution would not be a problem until we start going into 4 bytes or more, then this can be more complex to deal with.

3. I could find a different library that solves this problem. In this case there is a library called I2C_anything.h available that would allow me to send any data types and reconstruct them on the receiving side.

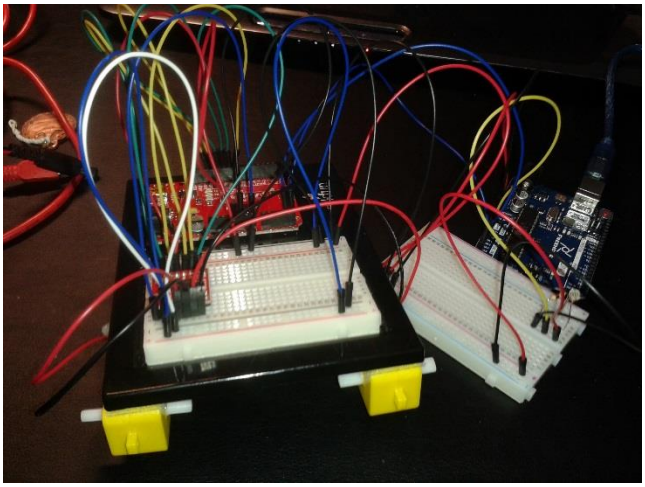
Of these three solutions I chose the third one since it required the least amount of modifications. The library was downloaded, unzipped and stored with the rest of the Arduino libraries. I created new basic programs to test the library and uploaded them onto the two devices. I connected them together through pins GND, A4 and A5 and open serial for testing: success!

Part 2

When I had the two Arduino's connected, the next step was to add all the components from the previous checkpoint while checking the battery requirement. The role of the master Arduino would be to collect all sound measurement, sending measurement to the slave Arduino and creating a visual indicator that it had found the source of noise. The role of the slave Arduino was to control the motors and avoid obstacles. As a result of using two microcontrollers, two motors and three sensors, there was a slight battery concern.

Since I still do not have the sound sensing components, I will once again use a photoresistor in its place. I added the photoresistor to the master Arduino and uploaded a new program that takes light measurement and sends it to the slave Arduino. A quick test showed no concerns on the master Arduino side.

On the slave Arduino side I re-created the circuit setup from part 4 of the first checkpoint minus the LED and photoresistor. The parts included two motors, two wheels, and one motor driver. I uploaded the modified program to the slave Arduino and disconnected it from the host computer. Testing the setup with the battery attached showed that there were no complications with the motor's response. Testing through a serial connection indicated that I might want to include a delay for the slave Arduino in order to get the two Arduino's to synch better.



Next I added in the ultrasonic distance sensor to the slave Arduino. This addition was done to ensure that the slave Arduino could quickly detect and react to incoming objects without having to wait for the master Arduino. Testing the new setup with only the battery attached showed that the motor's responses were becoming weaker. This reaction probably means that I am beginning to reach the max that my circuits can handle with the current battery pack so I may need to upgrade the battery pack.

(**video**) Uno to Uno (complete run).mp4

Part 3

No major modifications are completed for this part.

My goal here is simple: send data from slave Arduino to master Arduino and use the received data to get the master Arduino turn on an LED.

For this part I will be making use of the request functions from the Wire.h library.

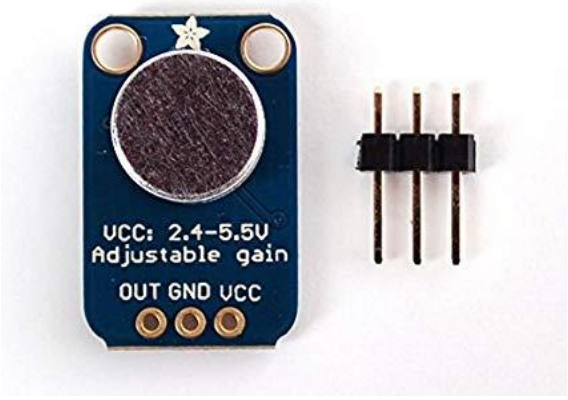
These functions are: Wire.requestFrom(), Wire.onRequest(), and Wire.write(). Wire.requestFrom() is used by the master to request bytes from a slave device. The Wire.onRequest() tells the slave device to register a function that is called whenever the master requests data. This registered function will be called as an interrupt function. Finally Wire.write() simply writes the bytes to the link. Testing while checking through a serial connection showed that the data is being properly sent from slave to master device and the LED light is successfully turning on when the light source is found.

(**video**) Uno to Uno (Complete run plus LED).mp4

Experiment: MAX4466 with Adjustable Gain

Feb. 16, 2019

For this experiment, I will be experimenting with a new component called the Adafruit Electret Microphone Amplifier. I will be using this component to check the frequency of any sounds detected. Part of my project proposal is to have the robot detect sounds that are 800 Hz or higher. This small little microphone is able to detect sounds between 20 Hz to 20 KHz (or 20000 Hz).



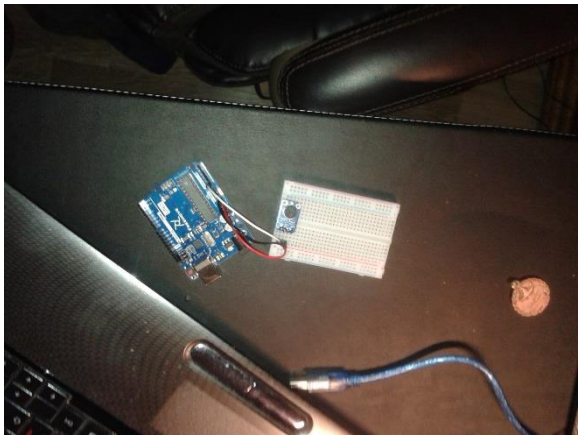
Part 1

Before I could do any testing with this component, I needed to assemble it. The only assembly required for this component was to solder the pins onto the circuit. Soldering is the process in which metal items are joined together by melting a filler metal into the joint. This process is used to form permanent connections between electronic components. Despite some light burns on the circuit the soldering was completed without much difficulty.

Part 2

With the successful assembly of the component the next part was to figure out how to use it. Following the tutorial provided by adafruit.com I recreated the Sketch provided under the label “Measuring Sound Levels”. The Sketch measures sound by taking in multiple measurements to find the minimum and maximum extent of a signal.

During every iteration the program spent 50 millisecond sampling the audio input. From these samples the highest rating and lowest rating samples were recorded. To obtain the peak to peak amplitude the absolute difference from the highest and lowest samples was calculated. This amplitude was then converted into volts to allow for any further processing and/or analysis.



(**Sketch**) Test_microphone_1.ino

From the Sketch I was able to use the component but I did found out how to get the frequency. I decided to check to see if there were any libraries available that would covert the input into the desired value.

Part 3

Generally when I search for a library I search for a library that would make my job easier. In this case I would have liked to have a library that could take the raw input from the component and convert it to frequencies. The first library I found was called the Audio Frequency Meter library which did everything I wanted as efficiently as possible. There was only one problem which was that it could only be used with the Arduino Zero boards. Since I was using the Uno version of the microcontroller I could not use this library. All searches for an Uno version of that library turned up null.

I searched for alternatives to the Audio Frequency Meter. There were a few libraries such as the ArduinoFFT library or the ffft library that fit some of my requirements but found they increased in complexity which was a deterrent to using them. These libraries allowed for greater control in the analysis and processing of the audio samples which was too complex for what I needed.

Part 4

The next idea I had was to use a series of Serial print to get some raw data from the audio output and perform some analysis on it. I found a tone generator online that allowed me to create tones at various frequencies. I tested and recorded the microphone on the following frequencies: 500 Hz, 600 Hz, 700 Hz, 800 Hz, and 1000 Hz.

(**text files**) High pitch test 1/New 800 Hz with Samples.txt

High pitch test 1/ 800 Hz with Samples.txt

High pitch test 1/ 700 Hz with Samples.txt

High pitch test 1/ 600 Hz with Samples.txt

High pitch test 1/ 500 Hz.txt

My plan was to make use of the sine wave equation by solving for frequencies. I already had the values for amplitude and time so this plan was feasible. In my search online I found a period to frequency calculator that showed me what length of time I should sample the audio to get the desired frequency range. For 1000 Hz the sample period would be 1 millisecond. I adjusted the sampling period on the tutorial program and tested the program with the tone generator. This test showed that as the frequency approaches 1000 Hz, the volt reading approaches 1. Anything less than 1000 Hz did not reach pass 0.5 on the volt reading. Success.

Part 5

The final step was to check to see how the microphone dealt with distance. Up to now I had the circuit next to the computer where the tone was being generated. This time I used a separate computer to generate the same tone from across the room while recording the reading on the original computer. Testing unfortunately showed that the microphone did not pick up on the reading as well as the previous test barely rising pass 0.1.

This result meant I would have to modify my original plan for the robot. Originally I wanted the robot to detect sound at a particular frequency and move towards the source. As this plan might now be possible with the hardware instead I will have the robot detect and find the source of a sound. Once it reaches the source it will detect the frequency and alert me if the frequency is in the right range.

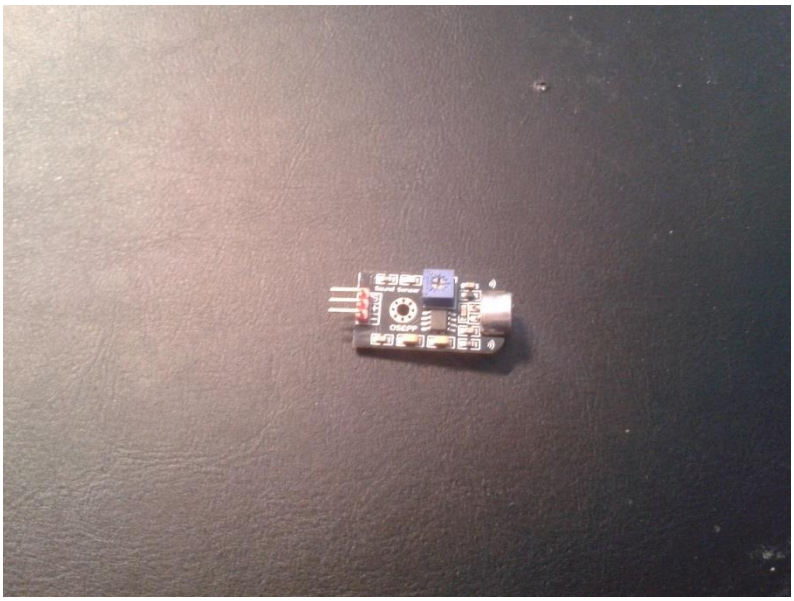
Checkpoint Three: Putting it all Together

Feb. 20 - 24

The purpose of this checkpoint is to report on the assembly of the robot which combines all previous checkpoints and experiments.

Part 1

There was one step that needed to be completed before the assembly of the robot and that was the analysis of the LM 386 sound sensor. The sound sensor converts sound waves into an electrical signal. Sound is a physical force in a form of a wave and the sound sensor detects the changes in the pressure caused by the sound.



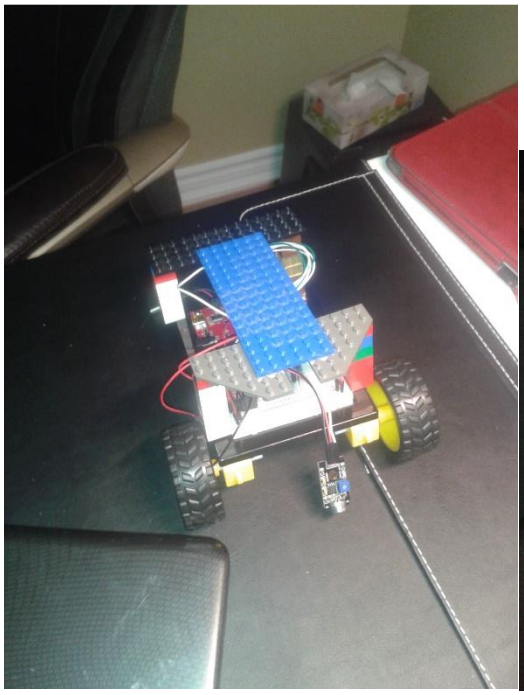
The setup for the analysis consisted of the sound sensor being attached to an Arduino Uno. Music consisting of various high frequencies was played from various distances as well as from various directions. The output from the sound sensor was printed in the serial link for recording purposes and the sum and average was calculated. My goal for conducting this analysis was to find out how reliable the sound sensor was.

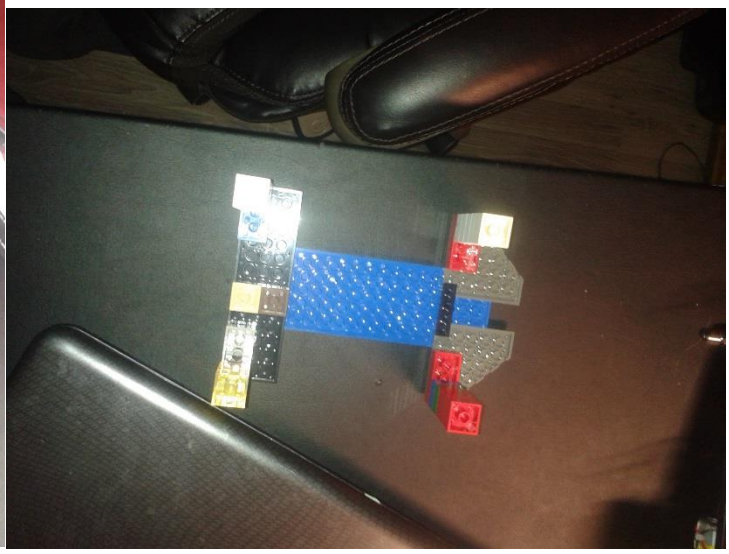
Sounds Levels from Various Sides									
	Facing Away from Music			Facing Towards the Music			From the Side		
	6 feet	3 feet	1.5 feet	6 feet	3 feet	1.5 feet	6 feet	3 feet	1.5 feet
	0.4	0.4	1	0.6	1.4	0	0	1.4	0
	0	3.4	1.2	3.2	4.2	5.6	0.2	1	0
	0	0	3	1.4	0	0.4	0.2	8	4.4
	0	1	0	1.2	0.4	3	1.2	3.2	3.6
	0.2	1.4	2	0	1.8	5	0.6	0.6	1
	1.4	2.6	0	0.6	1.4	1.4	0.8	0.2	0.6
	2.6	2.2	1.2	2	0.4	3.4	0.2	2.8	3
	0.6	0.6	1.8	1	4	1.4	0	0	0
	3.6	1.4	1.2	0.8	1.8	0.2	1.4	0.4	1
	0.4	5.2	1	3.4	2.8	1.6	0	0.2	1
	0.8	0.2	2	0	0.8	3	0.4	1.6	0.4
	0.2	0	1	2.8	0.4	5	0.2	0.8	0.2
	0	1.4	4.6	1.6	2.6	6.6	0	1	2
	0.2	0	3.8	0.4	0.2	3.8	0.8	1	0
	0	0	0	0		6.4	0.8	0	4.4
	1	1.6	0	0.8	0	0	0.4	0.8	0.6
	0	0.2	0	1	0.6	0.6	0.8	0	0
	1	0.2	2.2	0	0	7.8	1.6	0.2	0.6
	0.8	0.6	1	0.6	2.4	0.6	1	0.8	2
	1.2	1.6	0.6	0.8	2.2	4	0.2	0.6	2.8
	0	4.2	0	1.4	0.2	9.8	2.2	0.8	3
	0	5.2	1.8	2.4	0.6	9.8	0.4	0.2	5.8
	1.2	3.2	1.8	1.2	0.2	0.4	0.4	0.6	1.4
	1.8	0	0.8	0	0.6	15.6	0.4	0.6	4.6
	0	0.6	4	1	0.6	0.2	0.4	2	3.2
	2.6	1.6	0.6	3.2	2	6.2	0.4	0.8	2.2
Sum:	20	38.8	36.6	31.4	31.6	101.8	15	29.6	47.8
Average:	0.769231	1.492308	1.407692	1.207692	1.264	3.915385	0.576923	1.138462	1.838462

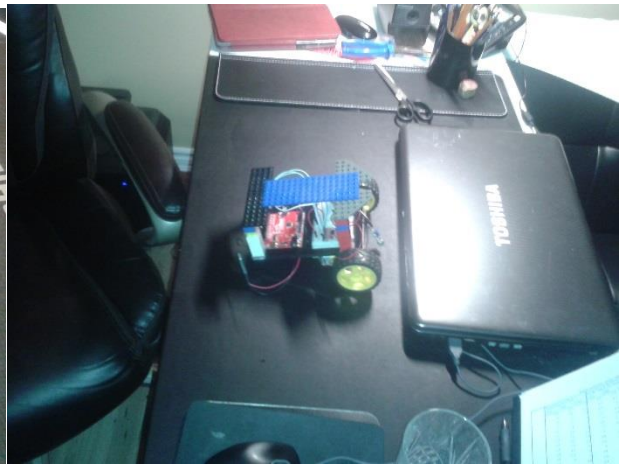
Based on the result from the analysis the sound sensor could detect sounds better from the front than from the rear or side. On the other hand the sound sensor did not show a stable output as the sound levels tended to fluctuate from high to low regardless of distance or direction. This fact meant that while the robot would behave as desired but there might be cases of odd behaviour where an if-statement gets triggered prematurely or not triggered at all.

Part 2

The next step was the creation of the second level on top of the Sparkfun board. To accomplish this, I made use of some old Lego pieces. Lego is very light and durable which makes it ideal for projects that have weight limitations as in this case. The Lego was attached to the Sparkfun board by use of M3 Dual Lock tape attached to each corner of the board. To help with the stability Gorilla glue was applied to the Lego pieces holding up the second level. Finally rails were added to the top layer to help keep the second Arduino Uno and breadboard from sliding off.





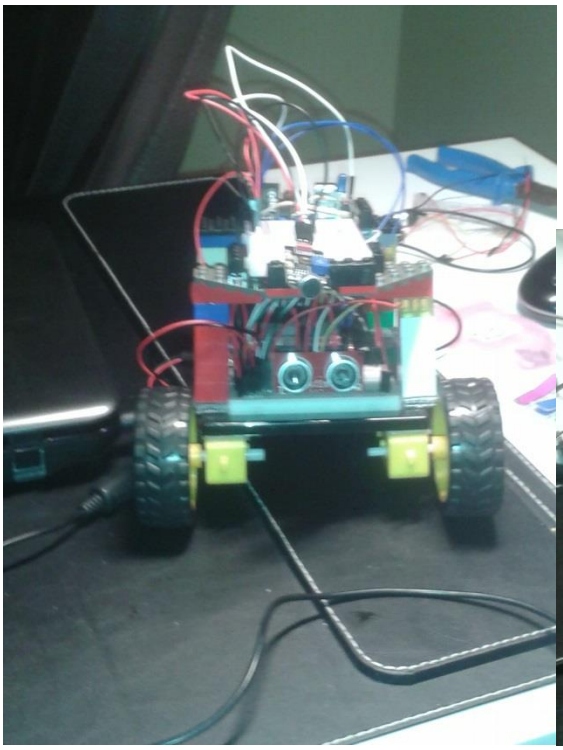


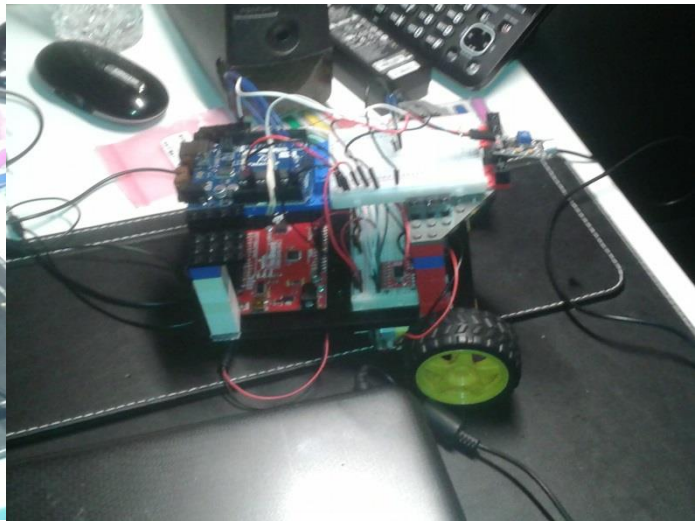
Part 3

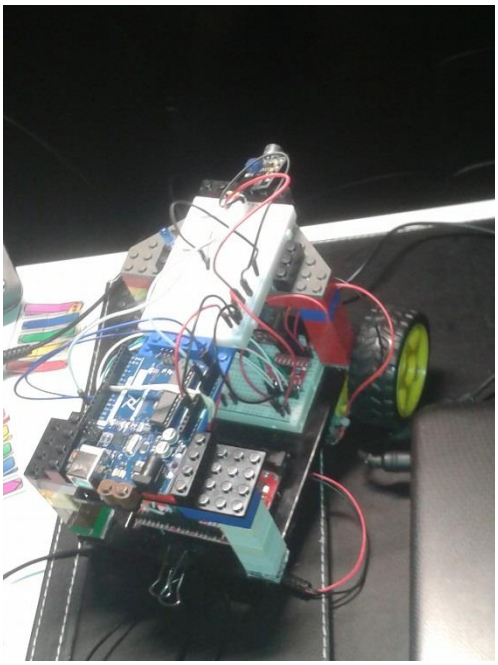
The incremental assembly and testing of the robot's hardware and software using the results from previous checkpoints and experiments was the next step. First I set up the master and slave device setup. The Uno on top of the robot would be the master device in charge of sensor input while the bottom Uno would be the slave device in charge of movement and object detection.

With the master/slave devices setup, the next components added were the motors, motor driver and wheels. These were added to the bottom layer of the robot which would be controlled by the slave device. The software was copied over from checkpoint one with minimal changes to the code. The behaviour of the sound sensitive robot should be the same as the light sensitive robot. This behaviour is as follows: turn your body around to face the direction of the sound and move towards it.

The last component to be added was the sound sensor. This addition was probably the most annoying of the components to add since the threshold calibration required consistent uploading and testing to find the right setting. The Sound Level table made earlier did help with finding a starting point with the calibration but still required trial and error. This step was also important for feedback control testing since this threshold would be used as the goal state.





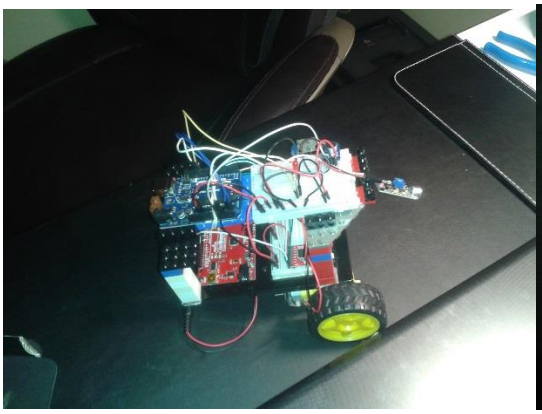


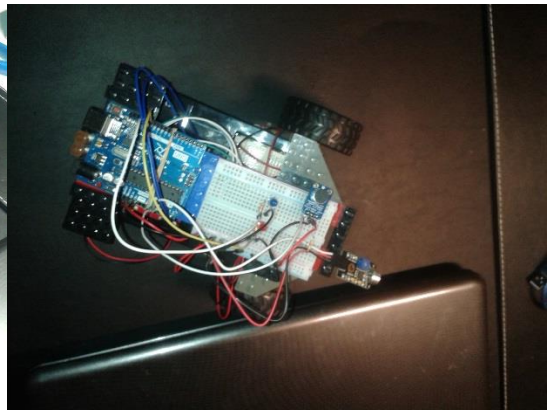
Part 4

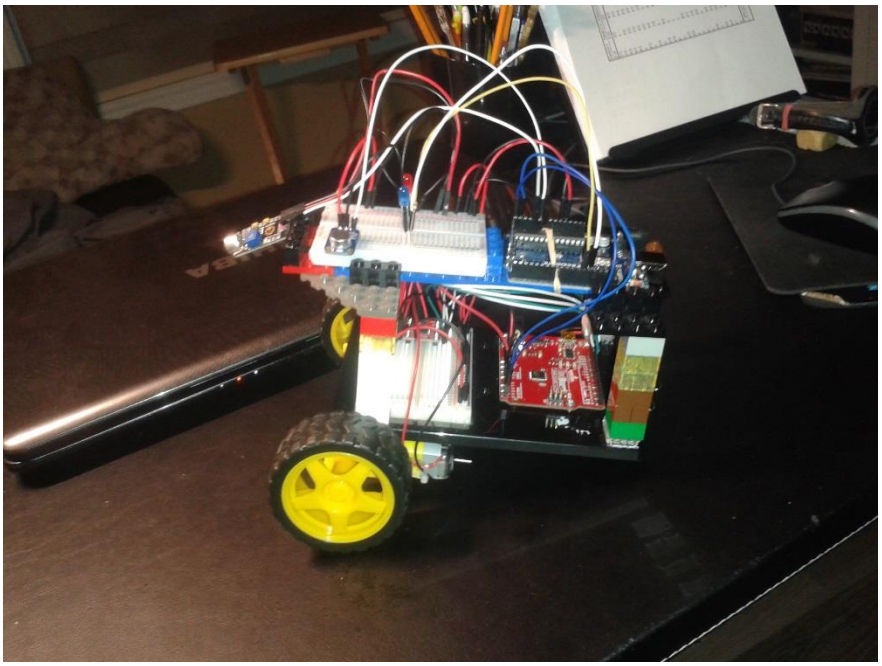
The last step involved testing the robot to make sure all parts were working correctly and were performing minor tweaks to the software. To pass the robot had to correctly point in the direction of the sound (ie. my phone) and move towards it. Up to now all tests conducted were passed with minimal errors. Unfortunately this time there were some complications.

When moving forward the robot was supposed to use the ultrasonic sensor to detect incoming objects and move to avoid them. Since the robot would not be using the sound sensor and the ultrasonic sensor together at the same time I thought that there would be no cross interference. The problem came from the sound source itself (ie. The music from my phone). It caused the ultrasonic sensor to generate false readings causing the robot to move to avoid even when there was nothing there.

There were only two solutions that I could think of. One solution was to swap out the ultrasonic sensor for an inferred sensor. The other solution was to simply remove the offending sensor and make do without object avoidance. Since I did not have the proper inferred sensor I chose the second solution.







Final Checkpoint: Feedback Controls

Feb. 25 – Mar.1

In this final checkpoint the feedback controls discussed earlier were implemented to control the robot's movement. The feedback controls that were examined are proportional control, integral control, derivative control, proportional-derivative (aka PD) control and proportional-integral-derivative (aka PID) control. Once each feedback control was implemented they were tested to see how well they performed in a real-world environment.

Part 1

Before I began implementing the feedback controls I needed to re-calibrate the threshold variables for the sound sensors. A threshold variable is used to determine when a sensor's output has reached a certain point. Since I had decided to use these threshold variables as goal states for the feedback controls I had to ensure that these variables were properly set.

Over the course of this project I noticed that the output from the sound sensors tended to fluctuate. This fluctuation was due to environmental factors, choice in power supply, type of wire used and which microcontroller was used, etc.. As a result the threshold variables needed to be readjusted at least once per day before any experiment could be done with it.

The recalibration process was the most tedious part of this project. This process involved constantly hooking up the robot to the computer, making the necessary adjustments, testing with the serial cable attached and following by testing on the ground when the robot was not connected by a serial cable. The reason for the last test was that the sound sensor tended to react differently when it relied on battery power alone than when it was powered by the serial cable. When the robot was running on battery power alone the threshold needed to be lower than when it was connected by serial cable.

Halfway through the calibration process I installed a switch on the top microcontroller which enabled/disabled output to the bottom microcontroller. Up to that point I was disconnecting the wire between the microcontrollers to keep the robot from trying to escape while I was updating it. The bottom microcontroller only moved the robot if it had received any data from the top microcontroller. By controlling when the top microcontroller sends data I effectively immobilized the robot.

(**video**) Calibration 1.mp4

After about two hours I finally had the threshold values at the right setting.

Part 2

With the threshold variables properly set I could finally get started on the feedback control algorithms. As mentioned earlier in the report feedback controls are a means of getting a system to achieve and maintain a desired state (ie. a goal state). There are two desired states for this robot which are the two threshold variables: turnTheshold and sourceThreshold.

The error signal is calculated by calculating the difference between a threshold variable and the current sample. This error signal is used to calculate the control variable (ie. distance) which would bring the robot closer to the desired state. The process to calculate the control variable will vary depending on the algorithm used. For this robot the control variable was the distance travelled in any direction.

The feedback controls examined in this project are the ones discussed in the textbook. They are: proportional control, integral control, derivative control, PD control and PID control.

Proportional control has the system respond in proportion to an error. This response is achieved by multiplying the error with a constant. In integral control the system keeps track of all the error signals by summing them together. The robot moves based on the sum times a constant. In derivative control a derivative value is calculated from the rate of change from the current error signal from the last error signal. How much the robot moves is based on the value of the derivative times a constant.

It should be noted that the constants applied in proportional, integral and derivative controls are separate constants and are not related to each other. The PD and PID feedback controls are just combinations of the other three controls and therefore do not introduce any new constants. The value of each constant had to be determined through trial and error.

Each one of these controllers are implemented in a function that takes in a float variable and returns a float. This little fact allowed me to create an array of function pointers that allows for quick access to each feedback control without having to search through the program.

Part 3

With the hardware and software set up was completed it was time to begin testing the robot on each feedback control functions. The feedback controls functions were examined in the following order: proportional, integral, derivative, PD and PID. The robot was placed at various distances from my laptop and timed on how long the robot took to reach my laptop. These distances included 3’ 8’’, 2’8’’ and 1’8’’. Eight inches was added on to factor in the distance from the speaker on the laptop. I also set up a time limit of eight minutes where in the event the robot exceeded this limit then it was assumed that it would never reach the sound source.

The first control to be examined was proportional control. On 1’8’’ the robot was able to correctly identify the source of the noise and accurately head towards it. When the robot was placed further back it began to lose in accuracy. At 2’8” it still identified the source of the sound but aimed at a nearby pillow rather than the laptop. At 3’8’’ the robot was able to detect the source of the sound but was not able to pinpoint the location and reach the end of the time limit. This last result was due to the robot over- adjusting itself when turning towards the sound which resulted in the robot constantly spinning around.

(**video**) Proportional Control Test Run.mp4

Integral control was the next feedback control to be examined. On all three distances the robot was able to successfully find and locate the source of the sound. One of the drawbacks to this function was the length of time needed to complete the task. On the shortest distance it took the robot an average of 2 minutes to find the source of the sound. On the longest distance it took the robot an average of 4 minutes. This performance was due to the fact that the feedback control started with small adjustments which grew over time.

(**video**) Integral Control Test Run.mp4

Next on the list to be examined was derivative control. Derivative control experienced the same problem as proportional control by tending to over compensate when turning around and going over the time limit. Unlike proportional control, derivative control tended to be more accurate when the robot finally got the turn correction right. At 1’8” the average time for the robot to turn around and find the source was 2 minutes and 7 seconds. At 2’8’’ the average time for the robot to turn around and find the source was 4 minutes and 50 seconds. At 3’8’’ the robot on average went over the time limit due to the over compensation problem.

(**video**) Derivative control Test run.mp4

The fourth control to be examined was the proportional-derivative feedback control. This feedback tended to experience the most difficulty from over adjusting when turning around. At 1’8” the robot on average took 2 minutes to turn around and locate the source. At 2’8” and 3’8” the robot on average went over the time limit due to the over adjusting problem.

The last control to be examined was the proportional-integral-derivative feedback control. This feedback control had the most consistent results across all distances. At 1’8” it took the robot on average 19 seconds to locate the sound. At 2’8” it took the robot on average 2 minutes and 3 seconds to locate the sound. At 2’8” it took the robot an average 2 minutes and 30 seconds to locate the sound. There were no observed cases where the robot went over the time limit.

Feedback Control Performance Table

	1 ft 8 in	**2 ft 8 in**	**3 ft 8 in**
Proportional Control	12 seconds	17 seconds	∞
Integral Control	56 seconds	2 minutes 13 seconds	4 minutes
Derivative Control	2 minutes 7 seconds	4 minutes 56 seconds	∞
PD Control	2 minutes	∞	∞
PID Control	19 seconds	2 minutes 3 seconds	2 minutes 31 seconds

∞ = went over the time limit

Final Thoughts

This robot unintentionally follows the subsumption architecture by design. The robot has been built incrementally by designing the simple components separately and then combining them to create more complex components. Each layer on the robot represents a module that worked to achieve their own goals.

The modules could work independently of each other but the robot only worked when the modules were communicating with each other. This meant that there was low coupling between the modules because the only thing the modules responded to was the data sent/received between each other. So as long as the communication stayed the same the hardware and software of a module can be swapped out as desired.

The bottom most layer was subservient to the top most layer since the bottom layer only moved in response to the data from the top layer. By suppressing the sensory input in the top layer the bottom module receives no sensory input and computes no reaction.

This robot is audiophiliac since it is moves towards all sound detected which implies an attraction. Working with sound in this regard has proven to be a nuisance. Due to the nature of sound the project requires a controlled environment to limit the amount of specular reflection occurring.

My annoyance was further amplified by the difficulty in trying to achieve part of the project goal which was to detect only high frequency noises. Due to some complications with getting the necessary components I had to make accommodations with some backup components. Unfortunately these components proved to be inefficient in managing the task and therefore required modifications to the overall goals.

Another difficultly encountered was the implementation of object avoidance. My original idea was to use the ultrasonic sensor to detect objects while the robot was moving forward. This idea did not work out since in the event of a continuous sound detection the ultrasonic sensor would continuously detect objects and try to avoid them. The only solutions in this case was to either seek alternative sensors (ie. laser, camera, inferred) or scrap the object avoidance. I chose to scrap the object avoidance based on the limitations of available components, complications with acquiring said components and time constraints.

The final aspect of the project that needed to be addressed were the feedback controls. By implementing and testing each feedback control I was able to gain a better understanding on how each feedback control affected the robot as well as the performance of each control. Integral and PID controls showed the most consistent result in detecting and locating the sound source due to the fine adjustment the robot made. Proportional control showed the quickest result but the robot tended to get stuck turning around due to over adjusting on each turn.

Derivative and PD control showed the least efficiency in consistency and time since the robot tended to get stuck on spinning around. This inefficiency was due to the fact that the sound sensor's output tended to fluctuate from high to low regardless of the robot's direction. Derivative and PD control relied on the absolute difference between the last error signal and current error signal to make the necessary adjustment. The fluctuation caused this difference to erroneously spike or shrink which caused the robot to overcompensate on some movements. This meant that derivative and PD controls might be a poor choice of feedback control for robots such as in this case.

Hardware List

Bottom Layer	**Top Layer**	**To Connect the Two Devices**
1x Arduino/Genuino Uno 1x Sparkfun baseplate 1x Motor driver 2x Gear motors 2x Wheels 14x jumper wires 1x battery holder 4x AA batteries Dual lock tape	1x Arduino/Genuino Uno 1x Switch 2x LED 3x 330 Ohm resistor 1x LM 386 Sound Sensor 1x Adafruit Electret Microphone Amplifier - MAX4466 with Adjustable Gain 12x Jumper Wires 3x Male/Female Jumper Wires	4x Jumper Wires
To Build Second Level Platform (Lego Pieces)		
20x Brick 2x2 4x Brick 2x3 3x Brick 1x4 2x Brick 1x3 1x Brick 1x6 1x Brick 1x2 2x Plate 2x2 1x Plate 1x4 3x Plate 2x3 2x Plate 2x4 2x Plate 6x6 without Corner 1x Plate 4x8 1x Plate 4x6 1x Plate 6x16		

Bottom Layer Assembly

Jumper Wires (* = Uno pin slot**)**

5V* to 5V	GND* to GND	VIN* to A1	D8* to J7	D9* to J6	D10* to J5
D11* to J6	D12* to J2	D13* to J1	D7* to I27	5V to 5V	GND to GND
A2 to 5V	A3 to GND	J4 to 5V	I26 to GND		

Motor Driver

C1 – C8 to G1 to G8

Motor Right

A4 (Red) A5 (Black)

Motor Left

A6 (Red) A7 (Black)

Connecting the Two Unos

^{1a} = Bottom Uno ^{1b} = Bottom Breadboard ^{2a} = Top Uno ^{2b} = Bottom Uno

A4^{1a} to A4^{2a} A5^{1a} to A5^{2a} 5V^{1b} to 5V^{2b} GND^{1b} to GND^{2b}

Top Layer Assembly

Jumper Wires (* = Uno pin slot) (^() = require male/female wire)

5V* to 5V GND* to GND A0* to B7 ^A1* to S pin on LM 386 D13* to E17

D12* to J17 D7* to C28 C27 to GND J16 to GND B6 to GND B5 to 5V

E16 to GND J16 to GND

Switch

A26 to A28

LED

A16 to A17 F16 to F17

330 Ohm Resistors

B17 to D17 G17 to I17

Adafruit Electret Microphone Amplifier

A4 to A6

LM 386 Sound Sensor (all wires are male to female connectors) (* = Uno pin slot)

S to A1* + to 5V - to GND

Note: to hold the sensor in place an extra resistor was used to hold down the wire

References

(2019). *Wire.h not able to send Int data type between two arduinos · Issue #2282 · arduino/Arduino*. [online] Available at: https://github.com/arduino/Arduino/issues/2282 [Accessed 10 Feb. 2019].

Gammon.com.au. (2019). *Gammon Forum : Electronics : Microprocessors : I2C - Two-Wire Peripheral Interface - for Arduino*. [online] Available at: http://www.gammon.com.au/forum/?id=10896 [Accessed 10 Feb. 2019].

Arduino.cc. (2019). *Arduino - WireRequestFrom*. [online] Available at: https://www.arduino.cc/en/Reference/WireRequestFrom [Accessed 10 Feb. 2019].

Forum.arduino.cc. (2019). *i2c questions - sending multiple bytes to and from arduinos*. [online] Available at: http://forum.arduino.cc/index.php?topic=288713.0 [Accessed 10 Feb. 2019].

Forum.arduino.cc. (2019). *I2C slave send data from multiple sensor to the Master Problem!*. [online] Available at: https://forum.arduino.cc/index.php?topic=167162.0 [Accessed 10 Feb. 2019].

Instructables.com. (2019). *I2C Between Arduinos*. [online] Available at: https://www.instructables.com/id/I2C-between-Arduinos/ [Accessed 9 Feb. 2019].

En.wikipedia.org. (2019). *Control theory*. [online] Available at: https://en.wikipedia.org/wiki/Control_theory#PID_feedback_control [Accessed 11 Feb. 2019].

En.wikipedia.org. (2019). Hearing loss. [online] Available at: https://en.wikipedia.org/wiki/Hearing_loss [Accessed 11 Feb. 2019].

Wai Weng, K. (2010). *PID for Embedded Design*. [online] Tutorial by Cytron. Available at: https://tutorial.cytron.io/2012/06/22/pid-for-embedded-design/ [Accessed 11 Feb. 2019].

Electronoobs.com. (2019). *Whistle detect circuit arduino control switch*. [online] Available at: http://www.electronoobs.com/eng_arduino_tut19.php [Accessed 12 Feb. 2019].

Sensorsone.com. (2019). *Period to Frequency Calculator*. [online] Available at: https://www.sensorsone.com/period-to-frequency-calculator/ [Accessed 17 Feb. 2019].

Cdn-learn.adafruit.com. (2019). [online] Available at: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-microphone-amplifier-breakout.pdf [Accessed 17 Feb. 2019].

Szynalski, T. (2019). *Online Tone Generator - generate pure tones of any frequency*. [online] Szynalski.com. Available at: http://www.szynalski.com/tone-generator/ [Accessed 16 Feb. 2019].