Angelo L. Ramirez

CSC 400 – 01

Fall 2016

3D Room Mapping Via Ultrasonic Sensor

1. Introduction:

This project will explore the feasibility of three-dimensional room mapping utilizing only ultrasonic waves. This system will generate a three-dimensional map of the room/area that it is used in. Mapping will be achieved through an ultrasonic sensor that will send a pinging signal that will travel and bounce off a surface and return to the receiver. The time taken for the signal to send and return is used to calculate the distance of an object in front of the sensor within a specific range. The system will comprise of an Arduino Uno microcontroller board connected to an ultrasonic sensor and Bluetooth communication module all mounted atop a treaded RC car. The cart will be capable of rotating and perform sweeps around an area collecting the distances of objects and be used to generate a three-dimensional map of the area using the sensor as the reference point. Additional sensors could be implemented in a later date to try to increase the accuracy of these readings. The goal of this system is to find an efficient and cost effective method of environment mapping using solely sound waves, testing the feasibility of this method and providing alternatives to more expensive equipment that use multiple paired sensors. Advantages of this system over others will be minimizing materials/equipment and costs to achieve the same effect of other similar methods. Potential users could be room planners and architects that want exact measurements of rooms while saving time and minimizing equipment needed.

Motivation for this project comes from growing interests in autonomous machines capable of navigating freely. I'd like to have a better understanding of what kind of spatial analysis problems there are for computers and what effective methods go into a problem similar to this. This also presents an opportunity to expand my experience with electronics and circuit design as well as learning new coding languages that are effective for these specific tasks.

2. Architecture:

1) Languages:
    i. Python
    ii. C/C++
2) Technology:
    i. Arduino Uno Microcontroller (**Figure 1**)
    ii. L298N Motor Driver (**Figure 2**)
    iii. HC-06 Bluetooth Module
    iv. HC-SR04 Range Finder
    v. DC Motors (2)
    vi. Tamiya Tracked Vehicle Chassis Kit

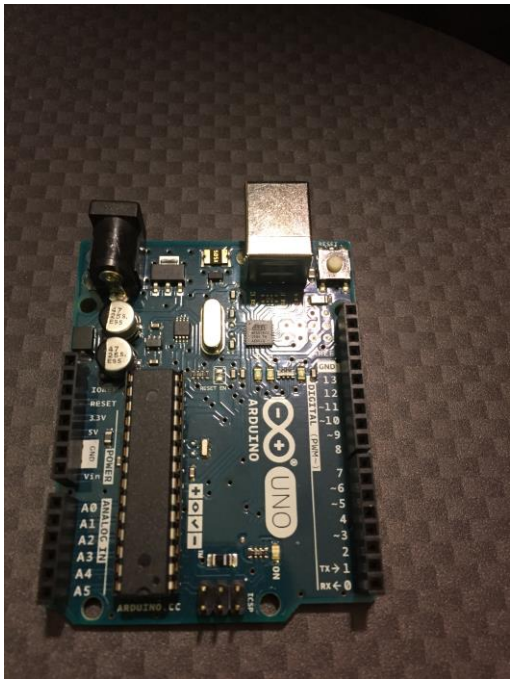


Figure 1: Arduino Uno Microcontroller

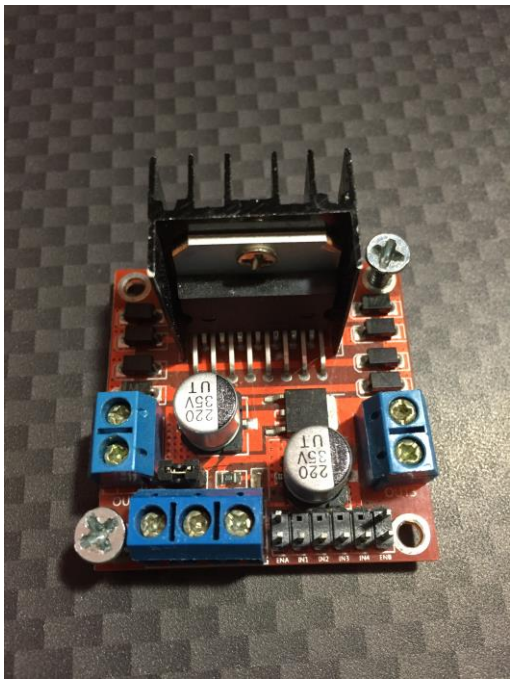


Figure 2: L298N Motor Driver

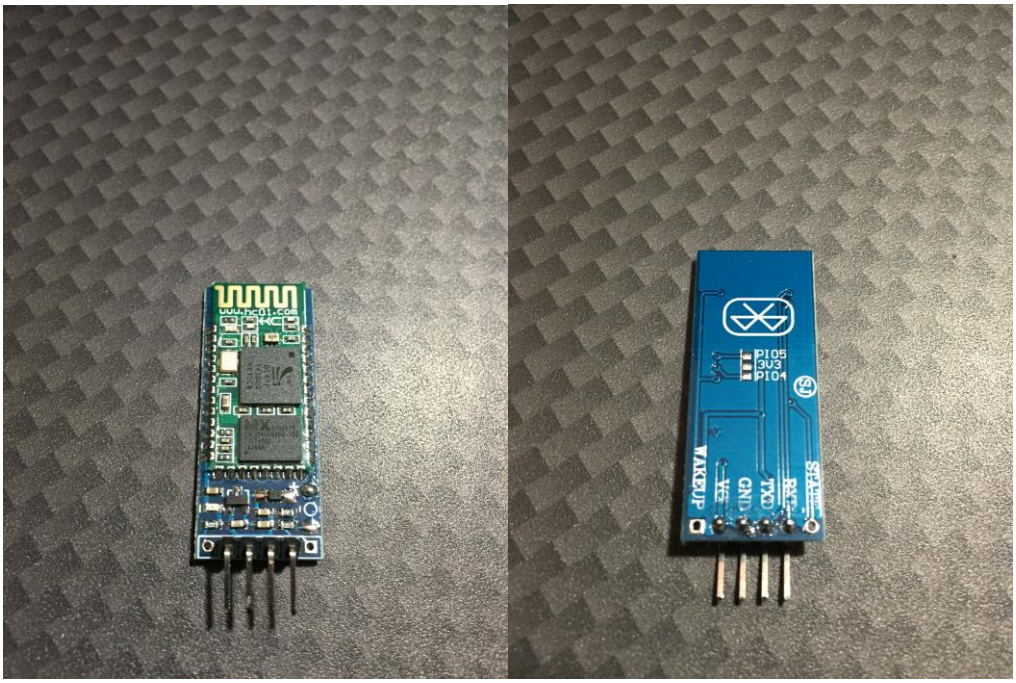


Figure 3 (a, b): HC-06 Bluetooth Module

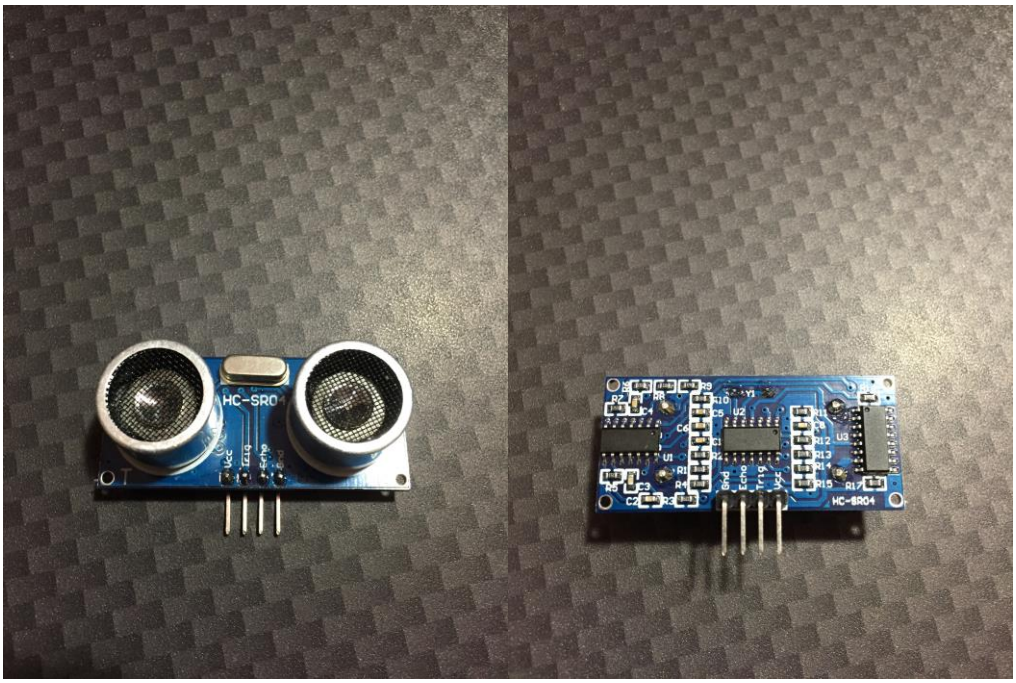


Figure 4 (a, b): HC-SR04 Range Finder

3) Circuits and Physical Component Guide:

For this project, I have used two programming languages: python for processing the data transferred from the Arduino and the user interface to display commands, and C for the Arduino microcontroller to send and relay commands to the components of the device. The python code can be edited using most IDEs, I have used Pycharm as it has similarities to Eclipse. Pycharm provides code correction, tracks and installs libraries easily, and version control is available to connect to Github. The Arduino has its own IDE to code and upload to the microcontroller easily as well as open source code and documentation available to beginners, all of which is available on their website.

The Arduino board will be acting as the brain of the device to relay the information coming in and executing the instructions that has been uploaded prior to beginning tests. Since the system will be operated over Bluetooth the board operates off a 9V battery, however any other battery is acceptable so long as sufficient current is provided. For each module, I will provide where each circuit connections will connect to between the modules and the Arduino board. Before utilizing any of the components it's necessary to upload code to the Arduino through USB and using its provided IDE to edit and compile the program that will be used. I will cover the code and algorithms used for this project in a later section. Once the code is compiled and uploaded onto the board it is necessary to unplug the device from the computer and begin making connections between the components, it is not recommended to start plugging in components while the board still has power.

The system is constructed atop the vehicle chassis. This car uses two DC motors connected to a gearbox which turn the wheels. For this kit, some of the wheels have teeth which are to be used with the provided treads, giving the mobility of a tank with each motor having control over either side. Both motors are connected to and controlled using the L298N motor driver (**Figure 2**). This motor driver uses electrical inputs from other sources, in this case the Arduino board, to change the speed and direction of motors that are connected. The polarity of the motors will not be important to remember as this will be adjusted in the code later when testing the pulses from the driver. See figures below to follow how the connections should be setup with the Arduino board. I have used a guide as reference to better understand how to use the driver with the Arduino board as well as understanding the power requirements and purpose of each pin connection (see reference 1). Note that out pins 1 & 2 are for the right motor and 3 & 4 are for the left, in addition this is where the polarity of the DC motors will not be too important to take note as switching will simply reverse the direction. The direction can also be adjusted within the Arduino code when uploading to the board. The figure below (**Figure 5**) illustrates how the motors and driver will connect to the Arduino.

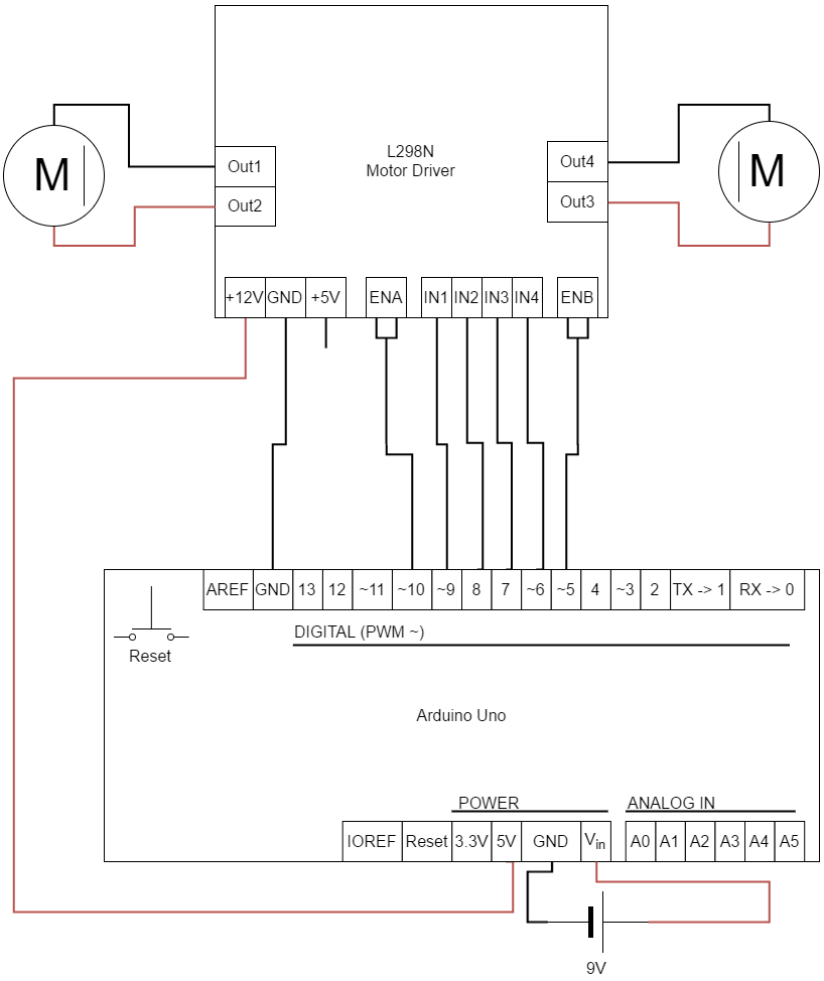


Figure 5: Circuit connection between Arduino, L298N, and two DC motors

Note that if using a power supply greater than 12V then connect the positive lead to the +12V connector on the motor driver and remove the 12V jumper that sits above the connection. Inorder to control the speed of both motors, enables A & B must be connected to a digital pin that has tilde "~" on its label. The tilde denotes that the pin supports pulse with modulation (PWM). The connections are as follows:

- DC motor 1 (-) -> Out1
- DC motor 1 (+) -> Out2
- DC motor 2 (+) -> Out3

- DC motor 2 (-) -> Out4
- ENA (right motor) -> Digital ~10
    - IN1 -> Digital ~9
    - IN2 -> Digital 8
- ENB (left motor) -> Digital ~5
    - IN3 -> Digital 7
    - IN4 -> Digital ~6
- If power supply is 12 or greater:
    - Motor Driver +12V (remove jumper if greater)
- If power supply is +5V or less:
    - Motor Driver +5V
- GND (Motor Driver) -> GND (Arduino)

Both the Arduino and driver can share a power supply and the same ground. For this project, I had a 9V battery connected to the Arduino board at the V_{in} power input and distribute power from the 5V connection to a common line for each component to connect on a bread board.

The next component is the HC-SR04 ultrasonic range finder, this will act as the main source of data collection as it will get the distances of objects. The range finder has four pins: VCC, Trigger, Echo, and GND. An electrical signal would be sent to the trigger pin from one of the digital outputs on the Arduino and the range finder will send a pulse at 40kHz with a range between 2 and 400 centimeters (approx. 13 ft.). Once the pulse is sent it will bounce off a surface and return to the echo sensors, which the echo pin will receive a signal from a digital pin to make the sensor listen for the incoming pulse that was sent. In the code the delay between the trigger and echo will be used to find the distance with the formula for the speed of sound. **Figure 6**

illustrates how I have added the range finder to the current circuit with the trigger connected to digital pin ~11 and echo connected to digital pin 12.

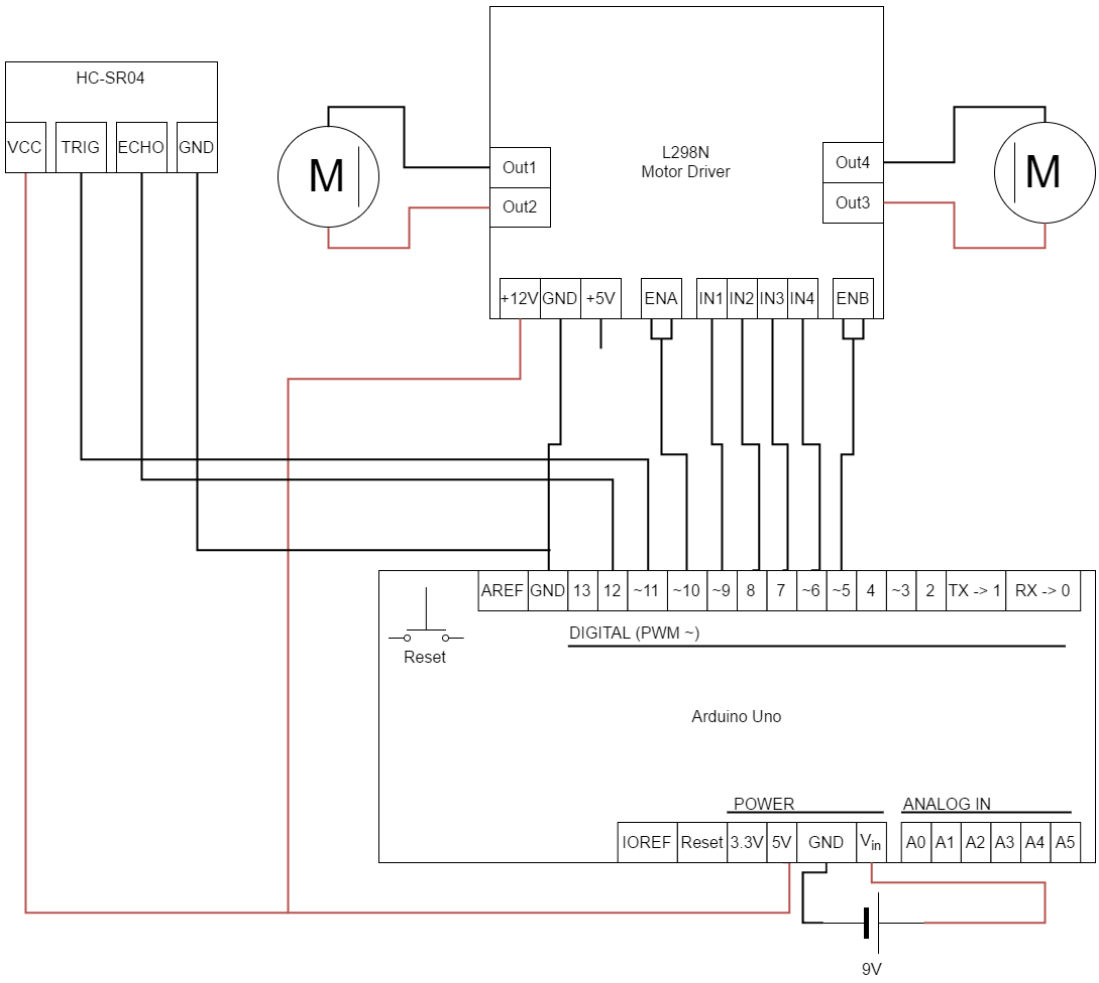


Figure 6: Circuit with Motor Driver and Range Finder

The final component to incorporate is the HC-06 Bluetooth module, the Bluetooth is necessary to allow the system to be mobile and relay the instruction from the python software and send back data to process. Refer to **figure 3b** for the pin labels and connections. On the Bluetooth module, the RX pin will connect to the digital pin 1 or TX, the TX pin on the module is connected to digital pin 0 or RX. The module will act as a serial port to communicate through, much like if the Arduino was connected via USB. The schematic below (figure 7) shows the

final circuit connections required for this system. It's important to remember that the Bluetooth module cannot be connected to the circuit while trying to upload code onto the Arduino board through a USB connection. Having the Bluetooth connected will cause the Arduino IDE to upload the program to the wrong serial port and result in a compiling error. Only attach the module when not connecting the board to a computer for code editing.

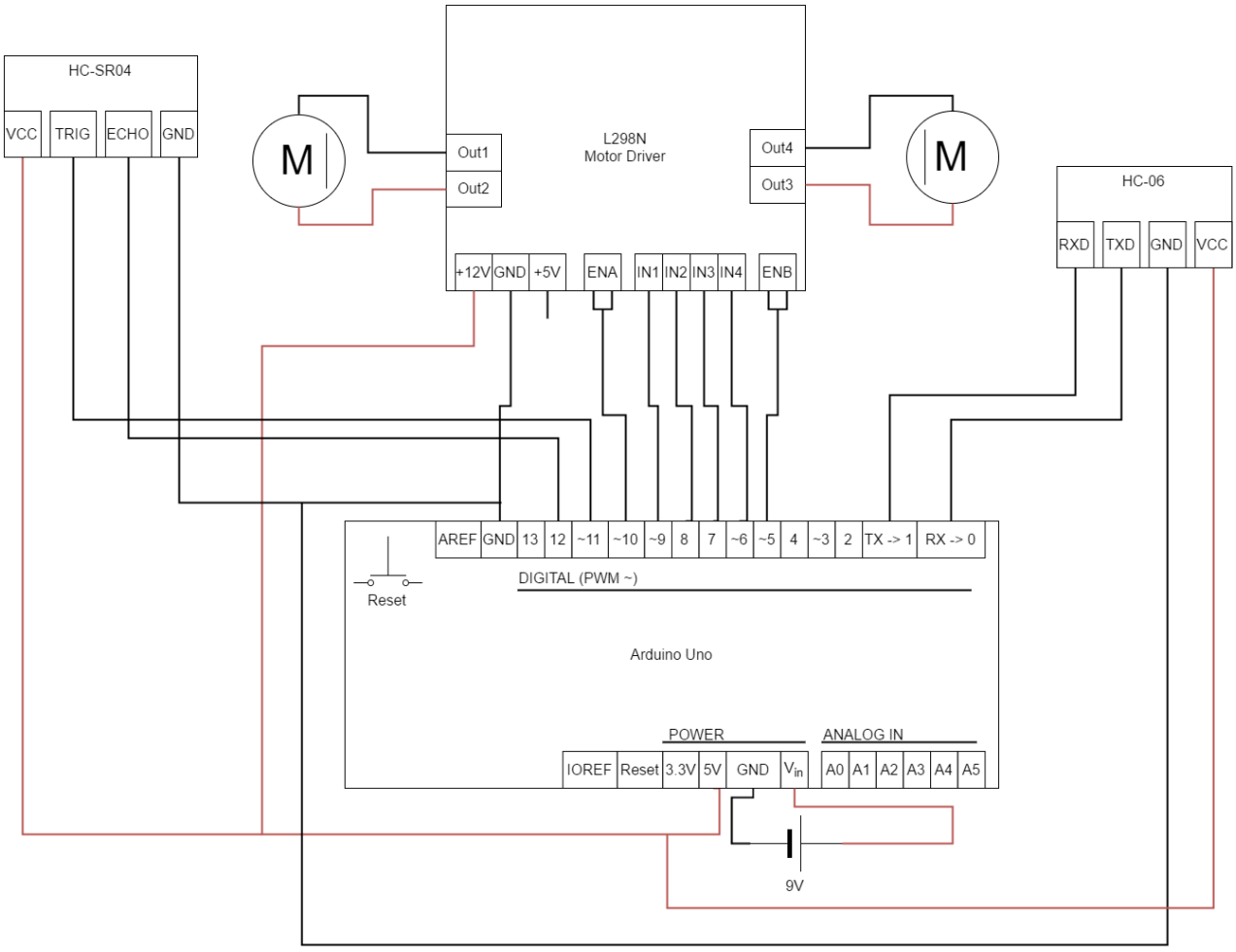


Figure 7: Final circuit connections, includes the motor driver, DC motors, Bluetooth module and range finder

4. Python & C Implementation:

Source code available at following Github link:

https://github.com/misterSchmaltz/UltrasonicRoomMapping

Now that the physical components are setup I will go over snippets of code used in my implementation to operate the various modules as well as data processing. In addition, some libraries and additional software will be mentioned and required as code is explained further along. I will denote the required library as well as the link to download the software, all of which are free and readily available.

As mentioned earlier the components will be operated with electrical signals delivered to their respective digital pins that they are connected to. I will begin with code that will go into the Arduino board. Arduino board are programmed via their own open-source IDE that's provided by the manufacturer on their website, follow the link and download the IDE to begin coding to the board (https://www.arduino.cc/en/Main/Software). The way the IDE functions is programs are saved as 'sketches,' some of which there are pre-made sketches from communities and the developers of the board. These sketches are uploaded and compiled to the selected board that's currently connected to the computer. Before beginning coding or uploading any code the corresponding Arduino board and serial port must be selected to upload to. The board can be selected in the menu Tools->Board->Board Name (**figure 8**). The selected board must correspond to the model that is being used. It's also worth noting that a serial port is selected and keep in mind of the port name as it will be used later when interfacing with the board through serial port. Serial ports can be selected and viewed under Tools->Port (**figure 8**). The current serial port in use can also be seen in the bottom right corner of the editing window.

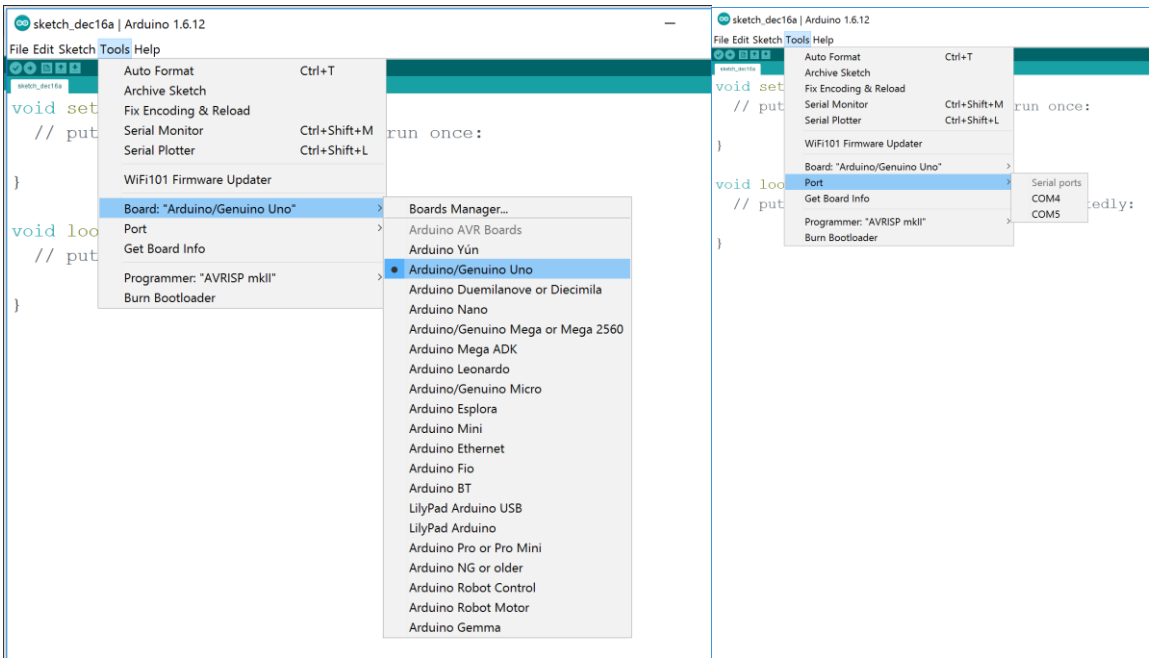


Figure 8: board selection (left), serial port selection (right)

The language that Arduino runs on is generally much like C or C++ however the initial design of how the code is implemented is a little different than what's normally done. The sketch window has two methods to use, the setup method initializes any values that will be used and will run once, in the loop method any code that is inserted here will run repeatedly which where the main code will reside in, refer to figures 9-11 to follow as well the supplied source code on Github. In my code, I have defined the trigger and echo pins of the ranger finder to be 11 and 12 respectively, much like the digital pins that they are connected to in the circuit. The character "val" variable is initialized to hold the commands from python, this variable will be used in if statements in the main loop to execute the given command that corresponds to the character that was sent over the serial port. The other connections such as for the motor driver for the left and right motor are also define in lines 5-12. The enable pins will be used to set the motor speeds and "IN" pins 1-4 control the direction the motors will be turning in. This is where the polarity of the initial hardware setup didn't matter as this can be adjusted in the code itself. Within the setup

method the baud rate for serial data transmission must be started and initialized. The baud rate is the data rate of bits per seconds that's transferred, for this I've been recommended to use a rate of 9600. After the trigger pin is set to act as an output and the echo as an input corresponding to their function. All pins from the motor driver will be set as an output since there they will only be used to move the cart itself.

In the loop two floating point variables for duration of the trigger flight and the calculated distance is created for later use as well as reset the value after each passing of the loop. There's also an if statement that checks if the serial port that's being used is available to be used and if so then to read the next value coming in from the port. This is where the input commands from the user view will be received and be used to execute specific commands by checking the value of the character received. An example of a specific command to execute would be in figure 10 where a value of the character "O" would set all the pins for to the drive to low, this forces the motors to stop spinning if a previous command had them set to high.

```
#define trigPin 11
#define echoPin 12
char val;
//Right motor = enable A
int enA = 10;
int in1 = 9;
int in2 = 8;

//Left motor = enable B
int enB = 5;
int in3 = 7;
int in4 = 6;


void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  pinMode(enA, OUTPUT);
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
}
```

Figure 9: Initializing values

```
31  float duration, distance;
32  if(Serial.available())
33  {
34    val = Serial.read();
35  }
```

```
63  else if(val == 'O')
64  {
65    digitalWrite(in1, LOW);
66    digitalWrite(in2, LOW);
67    digitalWrite(in3, LOW);
68    digitalWrite(in4, LOW);
69    start = 0;
70  }
```

Figure 10: Checking serial connection and first command checked

```
72  else if(val == 'S')
73  {
74    digitalWrite(in1, HIGH);
75    digitalWrite(in2, LOW);
76    // set speed to 200 out of possible range 0~255
77    analogWrite(enA, 225);
78    // turn on motor B
79    digitalWrite(in3, HIGH);
80    digitalWrite(in4, LOW);
81    // set speed to 200 out of possible range 0~255
82    analogWrite(enB, 225);
83
84    //start sensor ping
85    digitalWrite(trigPin, LOW);
86    delayMicroseconds(2);
87
88    digitalWrite(trigPin, HIGH);
89    delayMicroseconds(10);
90    digitalWrite(trigPin, LOW);
91
92    duration = pulseIn(echoPin, HIGH);
93    distance = (duration/2) * 0.0344;
94
95    //delay(150);
96    if (distance >= 400 || distance <= 2)
97    {
98      Serial.println(-1, DEC);
99    }
100   else {
101     Serial.println(distance, DEC);
102     //delay(150);
```

Figure 11: If command to check scan execution

Figure 11 depicts an essential code that is used to gather initial data to determine nearby adjacent surfaces or walls. This statement will change the input pins for the driver to different combinations of low and high which affects the direction the treads will spin (see lines 74 – 80). Writing integer values to the enable pins also sets the speed which can range from 0 – 255 (see lines 77 and 82). Here it can be seen that the trigger pins for the range finder is first set to low and then high and back to low with some delays in-between. The delays are to allow the pulses to be sent out without having another pulse follow behind immediately and interrupted the flight of the first. Once the trigger pin is set back to low the echo pin is set to high to listen for any incoming pulses that were sent which results in the duration of how long it could bounce off surfaces before returning. This duration is then used to calculate the distance the pulse traveled is then store and printed out later in the serial line. If the distance falls outside the range that the sensor is capable of then -1 will be printed to denote that the pulse was either too close or too far from the sensor.

As of now a sweep and scan are the only commands available to collect data. Now python code can be written with some prior setup. Since serial ports will be used to relay and communicate between the boards it's necessary to download the necessary library for the version of python that is being used. This requires pySerial to enable python to communicate with serial ports, the version downloaded is dependent on the current version of python that's installed on the machine. The link download and documentation link can be found here: (documentation) https://pythonhosted.org/pyserial/ , (download) https://pypi.python.org/pypi/pyserial. Pycharm can also organize install any libraries that are available to search through their IDE. For ease I've used the settings within Pycharm to import the libraries for this project. Go to File -> Settings -> Your Project -> Project Interpreter, once here the plus sign can be clicked to search and install

the library needed. For this project, I have used maplotlib, numpy, pyserial, ttk, and Tkinter. Although ttk and Tkinter came packaged by default with Pycharm.

For the user interface and processing the data I've used python for its compatibility with making applications that involve working with Arduino boards. I have used ttk and Tkinter to display the interactive window for the user send commands to the Arduino system and generating a map once the data has been processed. The user view can be seen in figure 12, a three-dimensional map is not generated until a scan is performed.

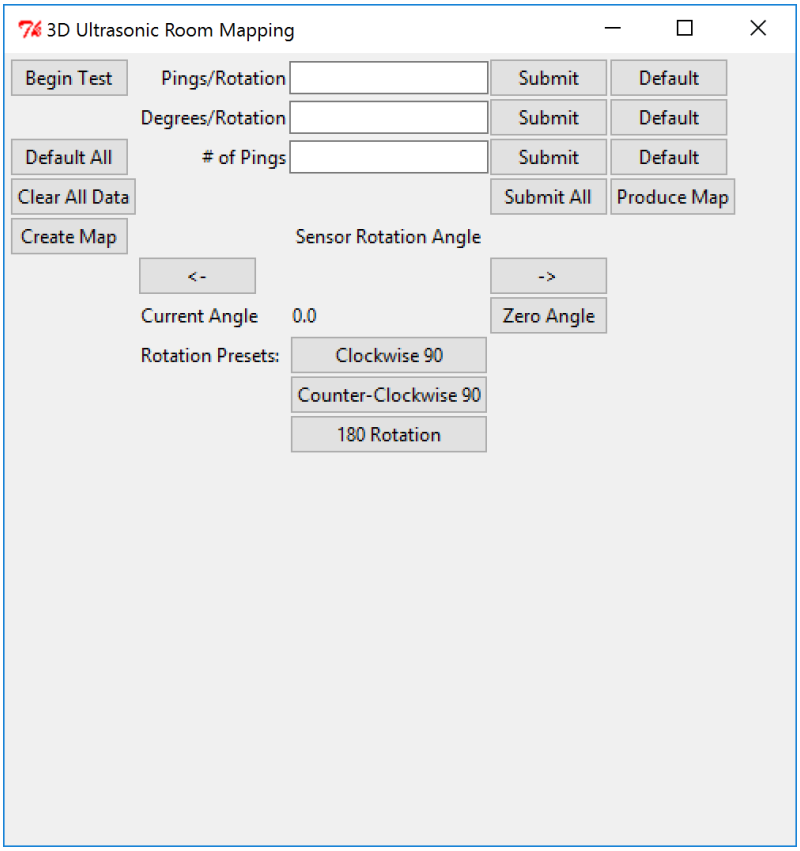


Figure 12: Basic user interface with several options to adjust testing parameters

Since communication between the software and the system will occur Bluetooth, python will need to be provided a serial port for it communicate through. Take note that to find the

Bluetooth that's connected to the Arduino it first needs to be powered on and find the device under the Bluetooth devices settings on a computer. The module will show up as HC-06, select to pair with the device, if asked for a password the default is usually "0000." Once paired the serial port that the module is operating on can be viewed under the Bluetooth settings of the computer under the COM ports tab. The port that will be needed is the port marked with the direction "Outgoing." Once that's completed one line can be added to designate data transfer to the Arduino.

```
arduinoSerialData = serial.Serial('com5', 9600, timeout=1)
```

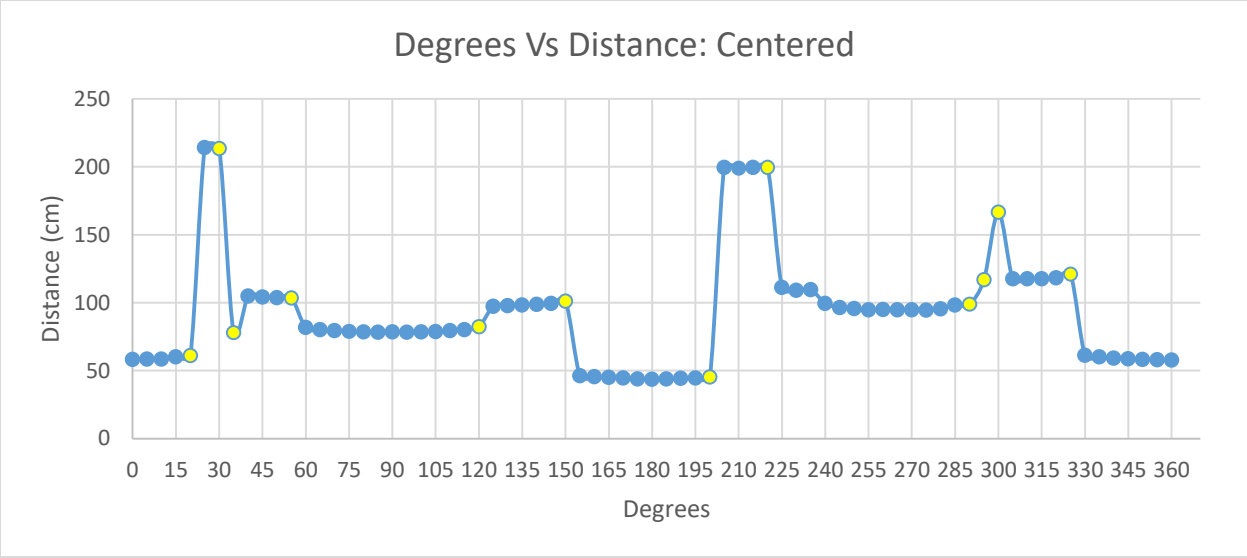
In the above python code arduinoSerialData will be used to read and write to the Arduino. The values that Serial() will be using are the COM port that the Bluetooth module will be operating on, the baud rate which would need to match with the baud rate that's written to the Arduino, and the timeout of the serial connection. The following lines are a method that's used to begin a simple test to send the instruction 'S' to the Arduino to rotate and collect data, and as that's happening while the cart has not made a full rotation the distances reported from the board will be appended to an array that will hold the unprocessed data of the first test:

```
def startTest():
    print("Test has started")

    arduinoSerialData.write('S')
    start = time.time()
    #11.65 with no delay
    count = 0
    while ((time.time() - start) < 11.65):
        sweepData.append(arduinoSerialData.readline())
        count = count + 1
    arduinoSerialData.write('O')
    #print(count)
    print(len(sweepData))
    print(sweepData)
    normSweepData = ProcessSweep(sweepData)
    print(normSweepData)
```

Once the cart is done rotating python then sends the instruction ‘O’ to the board to set the motor pins to turn off and stop reading the data, the sweep data is then passed through another method to convert the distances from string data type to decimal in a new array.

```
def ProcessSweep(x):
    list = []
    #length = len(x)
    #x = [y[:-4] for y in x]
    temp = []
    for i in range(len(x)):
        temp.append(x[i].strip('\r\n'))

    for index in range(len(temp)):
        list.append(float(temp[index]))
    return list
```

Next was analyzing the array of distances to identify the adjacent surfaces that surround the cart. This posed some challenges because if the test area was a controlled shape like a rectangular room or contain easily identifiable walls, the data collected should look like a trigonometric function. The data collected should look like the graph below where I had initially recorded by rotating the sensory by hand to examine how the dataset would look like and how an algorithm could be written to identify adjacent surfaces. This became more an issue of algorithm design and an extensive background in mathematics, particularly subjects involving advanced calculus and understanding trigonometric functions.



As an alternative I had attempted in writing a flagging system to determine when the range finder would have pinged an adjacent surface rather than an incorrect distance because of the ping bouncing off too many surfaces. The graph shows that walls would most likely exist in a region of the data were the distances recorded have remained in a consistent plateau, usually within fifteen degrees above and below positions of where the range finder is adjacent to the surface. Any pings retrieved as a result of hitting the surface at an obtuse angle about 30 degrees away from its starting position is indicative of a dramatic spike in the distance recorded. As such I've chosen to center my algorithm in determining whether the change in distances from one data point to another is within a 10% tolerance of one another, meaning that if the difference is past a certain threshold then the data point is marked to not be used for generating the distance to a wall. Here is the code that I have implemented for the flagging:

```
def flagPositions(caseNum, flagList, flagPos):
    for index in range(0, len(caseNum)-1):
        x = caseNum[index]
        y = caseNum[index + 1]
        # This loop will iterate and test if there was a spike in distance between
index and index + 1, this should determine the event of a peak from a plateau
        # append 1 if there was a spike, if not append 0 to show no change has
occurred
        if((x * 1.10) < y) or ((x - (x * 0.10)) > y): # decided to test for a ten
percent tolerance between the two points
```

```
            #print(x,y)
            flagList.append(1)
            flagPos.append(index)
        else:
            flagList.append(0)
    # This will test the case for degree position 360 and 0, but since both are the
same starting point this should yield a 0
    # this is simply to have a complete cyclical list of peak changes for each
recorded position
    if((caseNum[len(caseNum) - 1] * 1.10) < caseNum[0]):
        flagList.append(1)
        flagPos.append(len(caseNum))
    else:
        flagList.append(0)
```

As stated before the data points in the array will be compared to see if they fall outside this range of tolerance. If there is evidence of a major spike in distance then a 1 is appended to a list to denote that a spike is about to occur and to set a perimeter of when to not count the following distance towards calculating walls. If the difference between the two points is small than it will be flagged with a zero and be used in a set to get that adjacent wall. With these algorithms, I am assuming that these tests may not start with the range finder adjacent to a surface as to allow some different test cases to pass through with minimal errors.

After determining the possible corners and walls I ran the flag list and the array of recorded distances through a separate method to check for plateaus. This again is a challenge as some false positives could pass through in the form of a test case where the cart is starting at a corner and confuses two walls to be one if the distances recorded don't very greatly in that period. Below is the code to determine the average distance to these adjacent surfaces:

```
# This method will be to determine how far away is the device is from the four walls
using the flag list as a reference
def determinePosition(distancesFromWalls, caseNum, flagList, flagPos):
    count = 0
    avgdist = 0
    # This loop will iterate through flagList, each plateau (several 0's) should be
between the peak/spikes (1's)
    # e.g. a plateau would be represented as 10000001, the plateau in this case will
be wall in the 30 degree window of consistent distances
    for index in range(0, len(flagList)):
        # Check to see if it's a peak or plateau, if it is a plateau then increase
count and add distance to avgdist
```

```
        if(flagList[index] == 0):
            count = count + 1
            avgdist = (avgdist + caseNum[index])
        # Once a peak is detected, check to see if enough 0's have been detected to
constitute a plateau, in this case
        # if count is greater than or equal to 6 since adjacency will lie at least 15
degrees above and below
        # points where the device is adjacent to the wall then calculate and append
average distance
        else:
            if(count >= 6):
                count = count + 1
                avgdist = (avgdist + caseNum[index])/count
                distancesFromWalls.append(avgdist)
            # reset counters for the next set
            count = 0
            avgdist = 0
```

In order to calculate the average distance to an adjacent surface from the flag list I decided to test if the list contained occurrences of zeroes appearing six or more times in a row. The reason six was chosen was assuming if the system could rotate five degrees at a time, and that the point that the recorded distance is pinged at is at least less than thirty or fifteen degrees away from an adjacent position then if this were to occur at least six times then it has already passed the threshold for accuracy and must stop and calculate the average so far. A wall in the flag list would like similarly to this “10001,” with the 1s denoting a spike in distance. Once done then the average distance to that surface is calculated and appended to an array. Afterwards it is a matter of matching up the average distances calculated and if test area fits within the approximate dimensions of the area.

3. Testing:

Initial testing began with seeing how well the algorithm developed faired against a static data set, specifically distances recorded in five degree intervals placed at different points in the testing area. The algorithm is capable of generating the approximate dimensions of the testing area to some degree of accuracy for two of the three test cases. The test cases included placing the cart

centered to the test area, centered but close to one side of a wall, and close to a corner. The corner test yielded some errors with regards to the data collected which indicated that the algorithm was confusing the two walls that the system was placed in as just one wall. Next phase of testing was relaying data and commands between the python code and the Arduino board using the Bluetooth module. I succeeded in having the two systems communicate and relay data with minimal problems. Lastly I've tested the cart to perform a live rotation and scan of the surrounding area and take the data relayed from that test and produce a map after calculating the points. This function did not work as well as there was no exact control over how well the cart rotated with such factors like fresh batteries or the appropriate number of delays. The algorithm also didn't test so well against actual demonstrations of the cart meaning that the algorithm still needs work or that the current method is not as efficient as it could be. The current system still shows room for improvement, particularly with algorithm design and providing a more efficient battery source. The code could also use work with how well it executes commands to operate the motor driver.

4. **Challenges:**

Due to the complexity of how analysis of the data should have been I felt as though my attempts to construct an appropriate algorithm to test this fell beyond the scope of this project. A better understanding of how to analyze trigonometric functions in addition to a recommended algorithm that serves the same function could have aided me to understand the development problems further. This was strongly evident as data collecting produced negative results from experimental testing seeing as the cart does not rotate in accurately measurable increments of angles. In addition, differences in technical prowess showed as the system performs a rotating scan the cart would start to slightly move to one side more so despite setting both motors to the

same speed. Circuit design as well an inadequate power source could have used improvement as the whole system was powered by 9V battery source connected to the board.

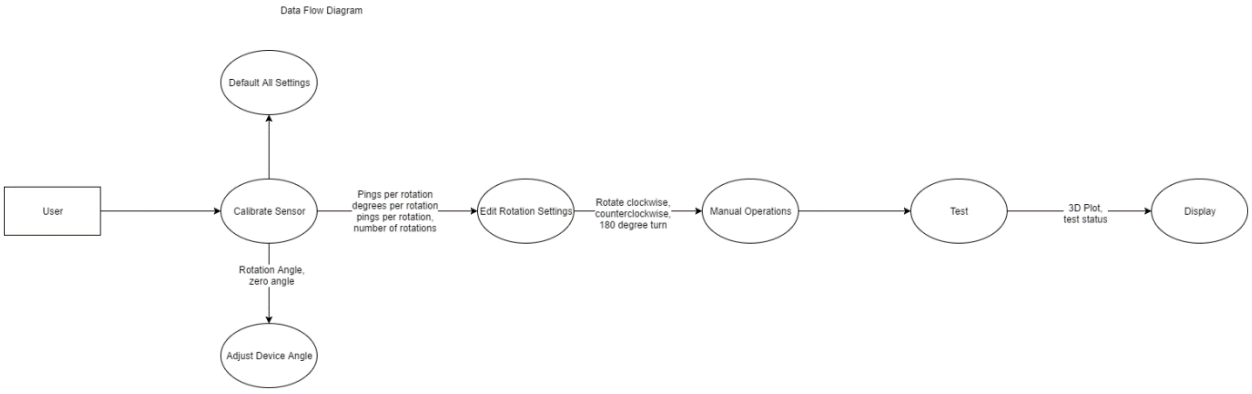
All in all, this was a great learning experience to push myself to new areas of software and hardware development. I do plan to return to this project once I've gained a little more knowledge of data analysis, particularly involving analyzing trig functions. I hope to also work to develop an AI that could operate the cart to move itself and generate a map of an area.

5. **Use Cases:**

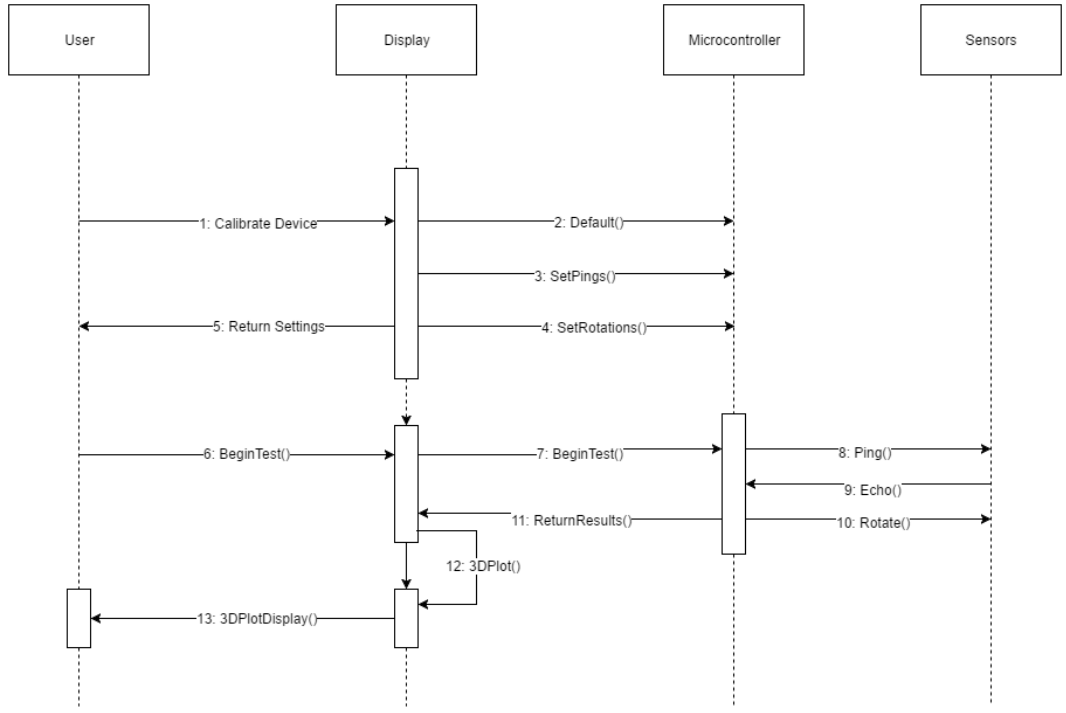
6. **Data:**

There was no database needed to store the information collected as processing could soon follow data collection. Future work may include saving test sessions to an external file or saving generated maps.

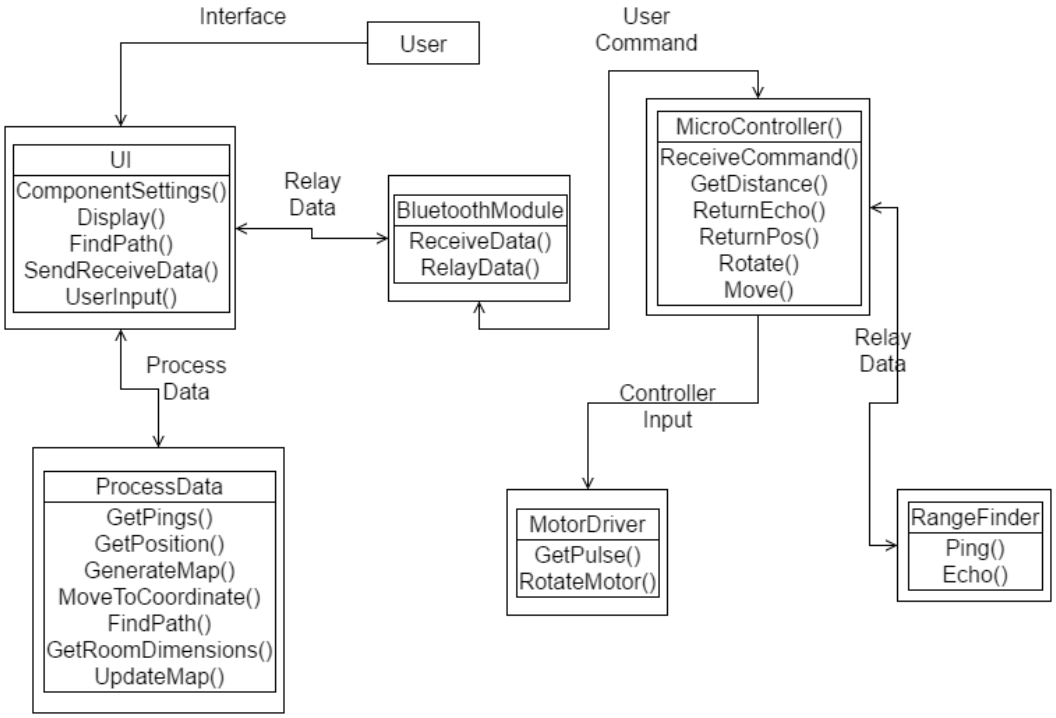
7. Detailed Design:



Data Flow Diagram

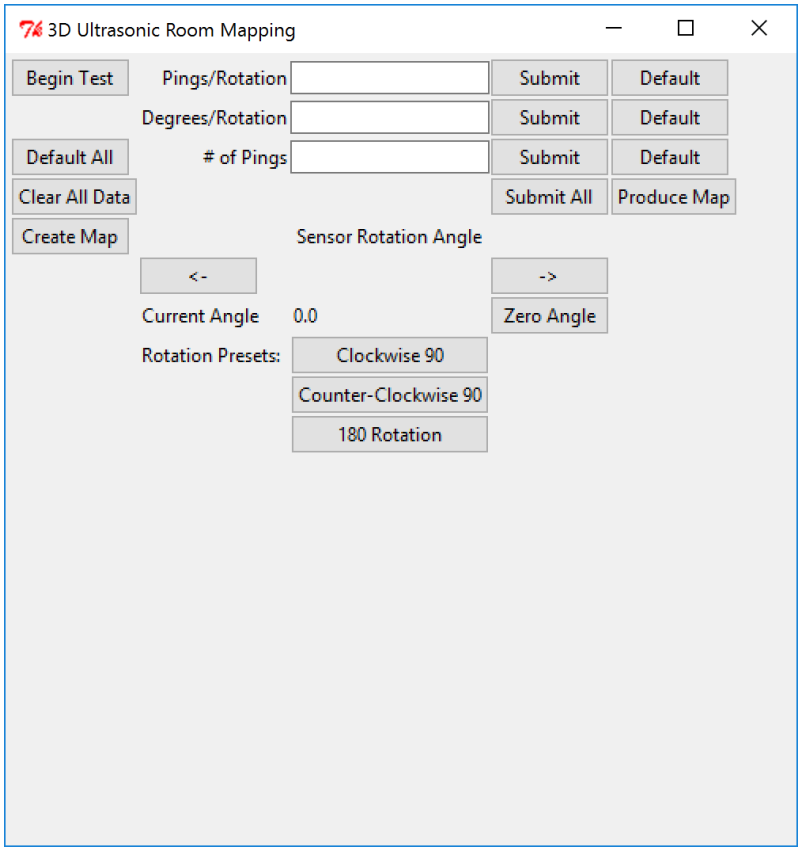


Sequence Diagram

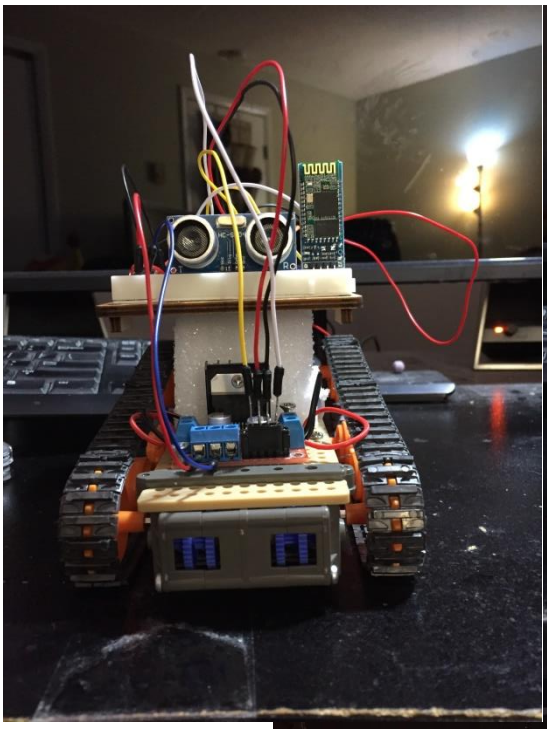


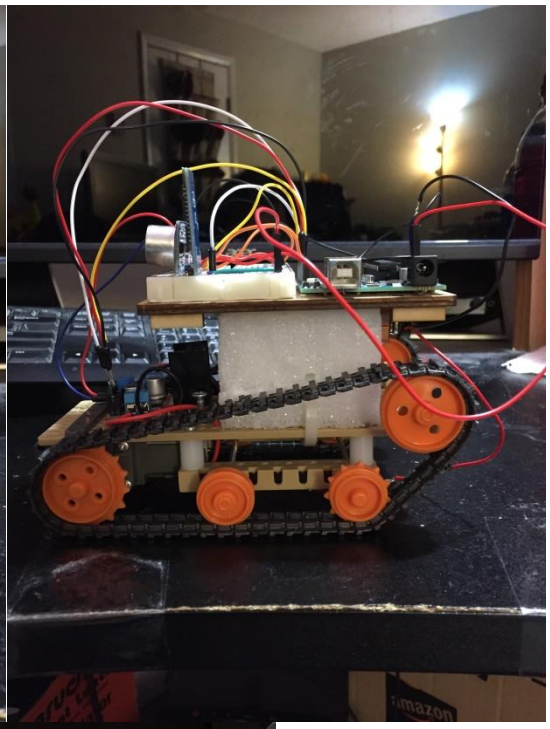
Design Diagram

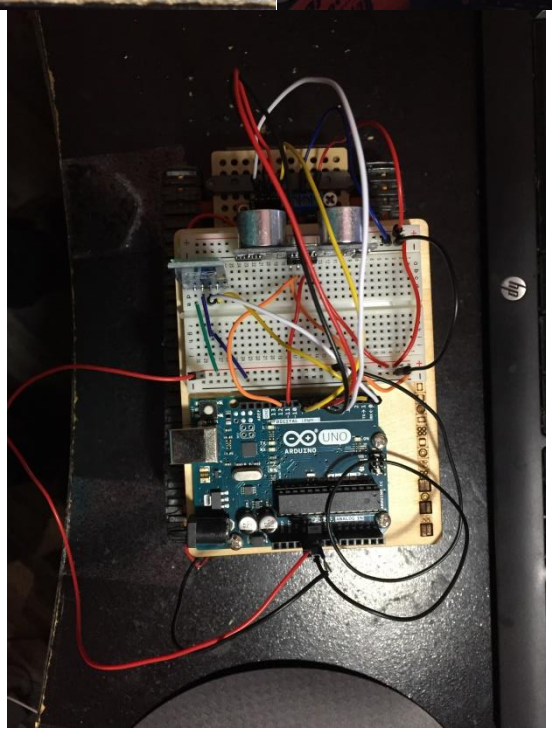
8. User Interface Design:



9. Final Build:







10. References:

1) https://tronixlabs.com.au/news/tutorial-l298n-dual-motor-controller-module-2a-and-arduino/

2) http://www.micropik.com/PDF/HCSR04.pdf

3) http://silabs.org.ua/bc4/hc06.pdf

4) https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf

5) https://www.arduino.cc/en/Main/Software

6) https://pythonhosted.org/pyserial/

7) https://pypi.python.org/pypi/pyserial

8) https://www.jetbrains.com/pycharm/specials/pycharm/pycharm.html?&gclid=CjwKEAiAvs7CBRC24rao6bGCoiASJABaCt5DpWtUIl6FSBTTXBmUfw8AhNdYKGrG-Tpr-BhY0o1GJxoCfPPw_wcB&gclsrc=aw.ds.ds&dclid=CKDhsPyl-NACFVEJDAodU2wFBw