Kienan O’Brien
Assignment 2
Technical Write Up

Ultrasonic Distance Sensing

Introduction

The task for this project was to interface an Arduino with an ultrasonic distance sensor to a personal computer. An ultrasonic distance sensor pulses high frequencies of sound and measures the distance it takes that sound to return to the sensor. In doing so, the sensor is able to measure the distance between itself and objects with a relatively high degree of accuracy. The data from this sensor is sent to an Arduino that then outputs that information to a computer. Once on the computer, a programming language by the name of processing animates the distances from the sensor over a radial graph.

Another important aspect of this project was to design and 3D print both an L-Bracket for the sensor and a housing for the project to rest on. This was done through CAD software and the user of in-class 3D printers and laser cutters.

3D Design – Fuse Holder:

In order to become familiar with 3D CAD software I began by designing an enclosure in order to hold the fuse board for my Arduino. The first step was to measure the fuse board and create relevant dimensions for the enclosure.

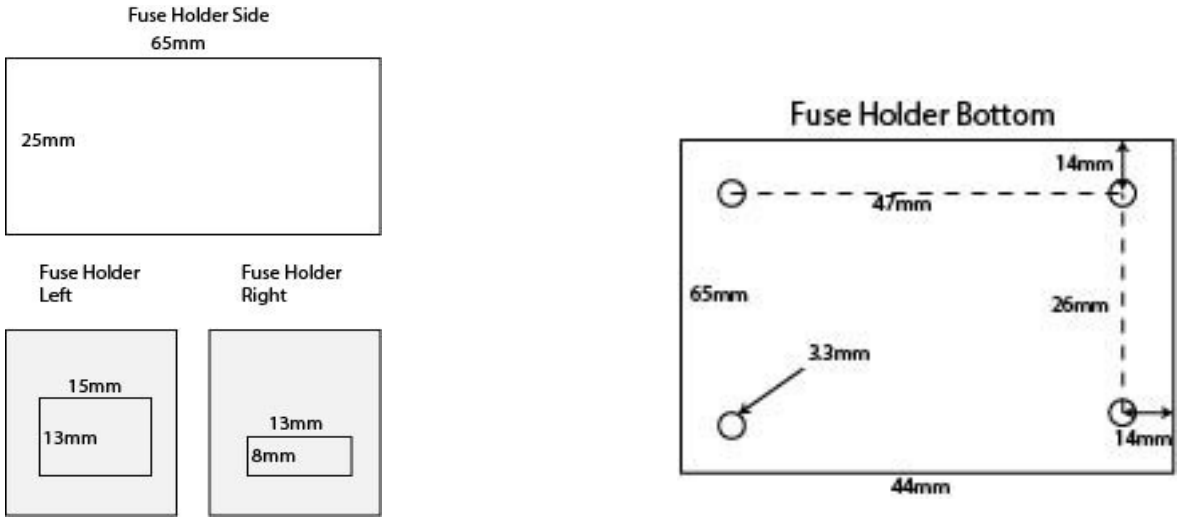


Figure 1: Relevant Fuse Holder Dimensions.

The resulting 3D model from these images was made in both OpenSCAD and Google Sketchup. It should be noted that a wall thickness of 5mm was used in both programs. The render created in Sketchup is in Figure 2. The render created in OpenSCAD is in Figure 3.

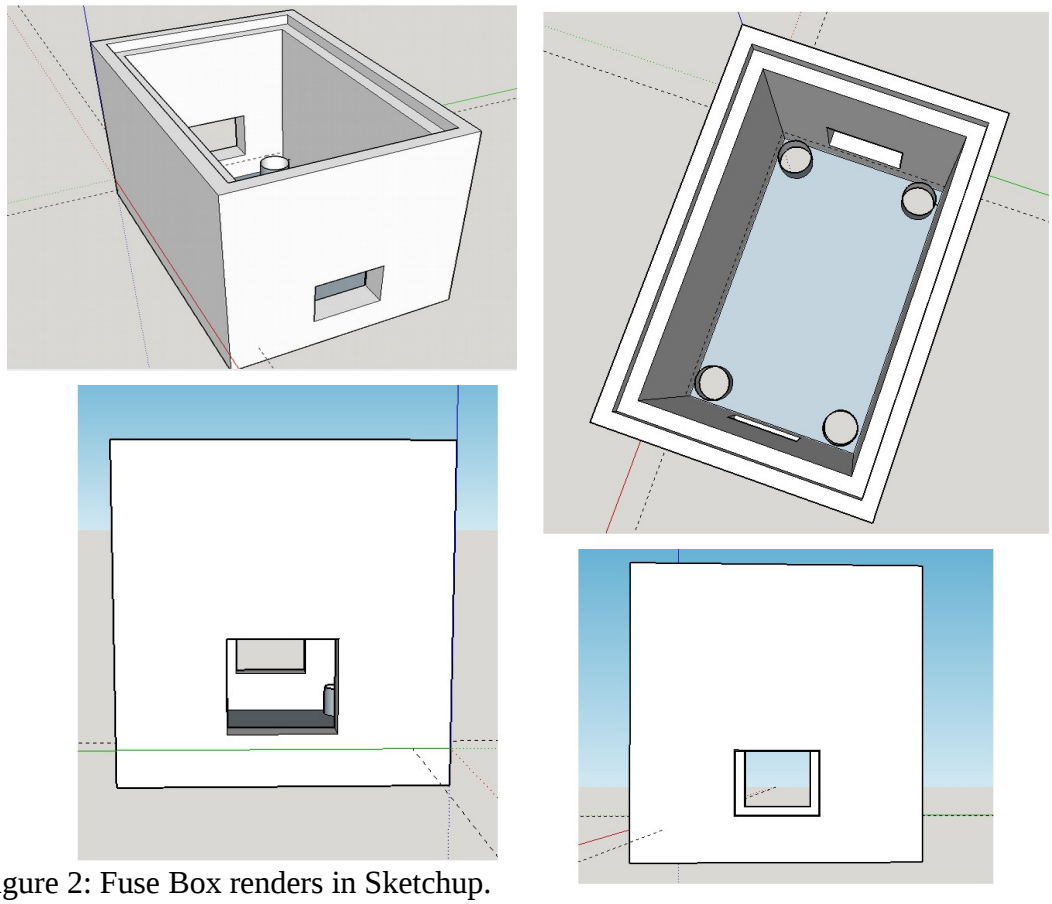


Figure 2: Fuse Box renders in Sketchup.

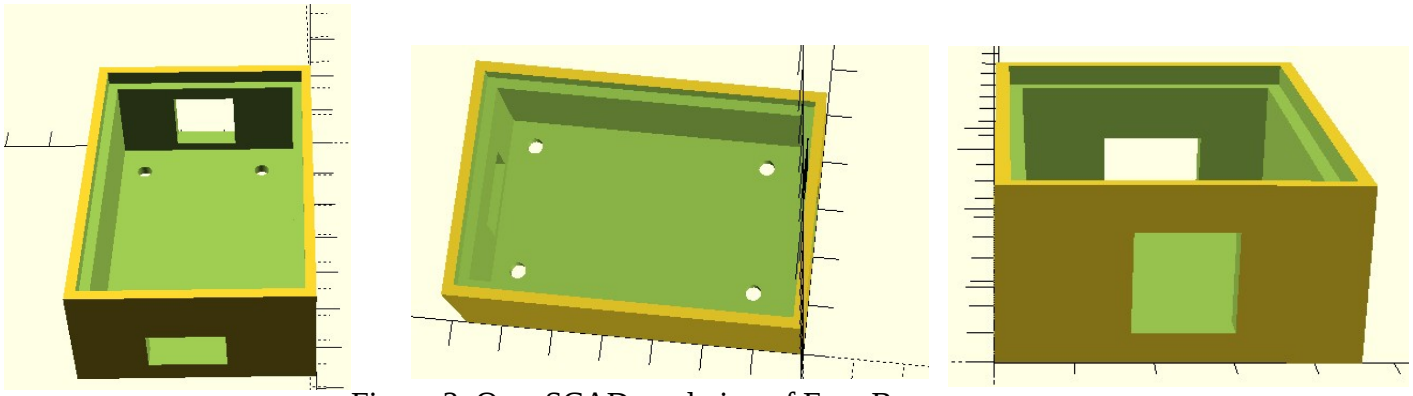


Figure 3: OpenSCAD rendering of Fuse Box.

Both renderings were created with the same dimensions as above. The thickness of the walls and bottom pieces were all 5mm, with the shelf as seen from the top view being 2mm wide. Both CAD programs created workable models that would be ready to 3D print.

3D Bracket Design – Laser Cut/3D Printed

A large portion of this project was dedicated to designing a 3D bracket to hold the ultrasonic distance sensor. Two separate models were designed for this, a 3D printable model and a 2D laser cut

model. As before, the first step was to measure the sensor as well as a servo that the entire unit would be mounted on. Figure 4 documents these measurements.

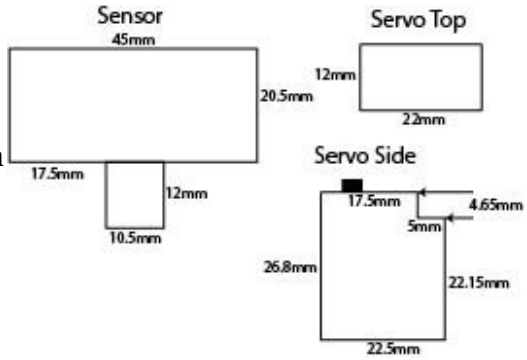


Figure 4: Servo and Sensor dimensions.

With the sensors measured the next step was to design the 3D printable version. This version of the L-Bracket was designed in Solidworks. My initial idea was to be able to press fit the fin that the servo came with, but was later scrapped due to minor inconsistencies in the 3D printed product. The final product turned out as seen in Figure 5.

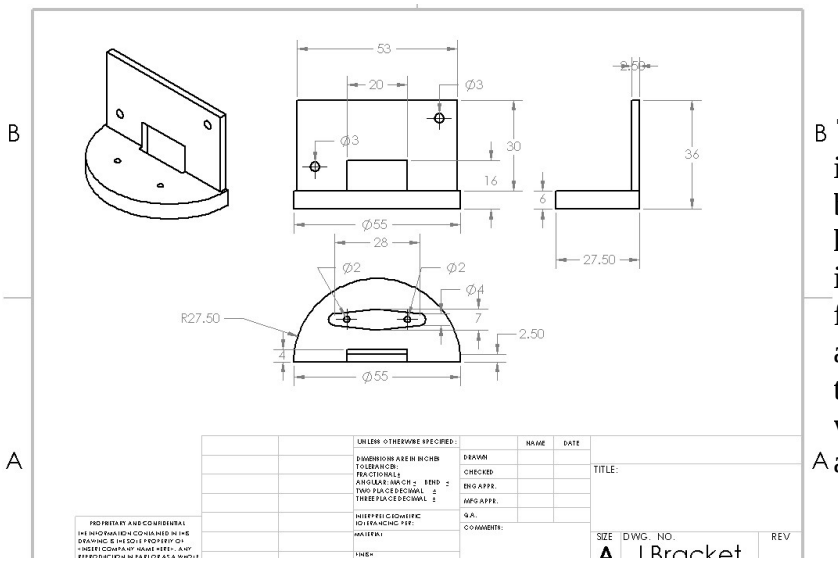


Figure 5: L-Bracket design in Solidworks

The L-Bracket allows the servo fin to be inserted below and then screwed down. The back plate on the L-Bracket has mounting holes meant for the sensor the be screwed into. The gap in the center allows the pins for the sensor to extrude out of the back for a uniform fit. The dimensions were chosen to accommodate the sensor and servo fin while still having some additional space to account for manufacturing tolerances.

The next step was to create a 2D version of an L-Bracket in Adobe Illustrator. This version of the bracket featured a press fit mount for the servo fin as well as a back plate that allowed the sensor to be mounted to it. The back plate was fit to the bottom press fit bracket through the use of ‘t’ slots. These slots allowed a 3.3mm bolt to hold both pieces together. The dimensions for the bracket feature a 6.93mm center hole that was almost perfect in size to account for the servo fin. The result was a tightly fitting servo mount coupled with a mounting plate. Figure 6 displays the results as well as dimensions.

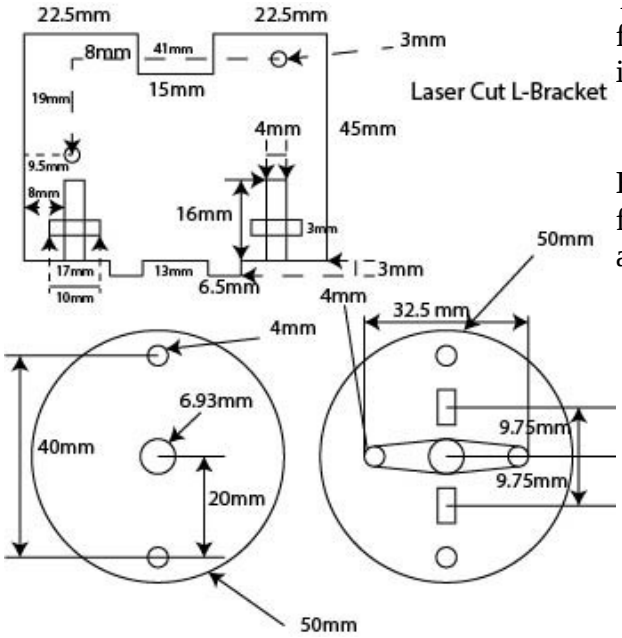


Figure 6: 2D designed L-Bracket. Cut using laser cutter.

The two disks mount on top of each other to form a press fit. The bolts are then threaded through the disks and held in place by their respective ‘t’ slots.

Both methods resulted in usable products. However, the 3D printed version of the bracket was used for the final product. This was due to the fit and overall aesthetic preference of the 3D-Printed version.

Arduino/Processing

The class provided Arduino Uno brought the project together. The Arduino read data from the sensor over serial and then transmitted that data through serial into the USB bus. Once transmitted to the computer, processing reads the data and reacts accordingly. The Arduino also controlled the servo that the sensor was mounted to via the L-Bracket. The servo sweeps180 degrees in 3 degree increments with a 250ms delay in between, measuring the distance from the sensor at each increment. This is done by outputting a new position on the ‘Servo’ pin. When the servo reaches the 180 degrees it resets to 0 degrees. This sweeping pattern continues until the Arduino is powered off. The wiring diagram for the Arduino, sensor, and servo are seen in Figure 7.

For each increment of the servo, the sensor receives a 10 microsecond pulse and then responds. The sensor has 4 pins: Vcc, Gnd, Trigger, and Echo. The trigger pin receives the pulse and then holds the echo pin high for a duration. This duration is measured and then converted into centimeters through the following formula:

cm = (Echo time on)/58.

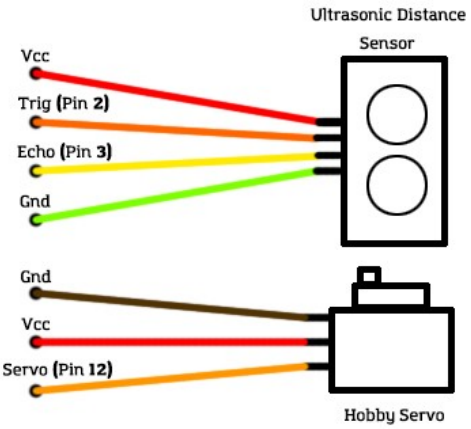


Figure 7: Wiring diagram for the Arduino, sensor, and servo

(Echo time on) is the time in microseconds that the sensor holds the echo pin at a logical high level. The data is then transmitted to the computer in the form of ordered pairs ie. (x,y) where x is the current angle and y is the distance. Both x and y are floating point values.

Once the floating point ordered pair is passed to the computer, processing takes over. Processing is a visual oriented programming language built with Arduino in mind. The ultimate goal of processing is to visualize the distances sent from my Arduino in a radial bar graph. My processing code is broken down into four major parts, the initial setup, the loop, reading from serial, and animation.

The initial setup sets the size and color of the canvas as well as instantiates two 2d arrays with their first value being an angle and second value being a height. The height is initially set to 0. The size of this array depends on the number of bars that you want to use in the graph. For my graph I divided 180/3 to get 60 bars, one for each position that the servo moves. This is readily changeable due to the scalability of the code. Once set, the angles never change however the height does.

The next major part of code is the draw() loop. This loop runs 120 times a second which was set with framerate(120) in the setup. In addition there is a 10ms delay on each run. The first step reads data in from the usb port, then moves on to viualizing the data read. For each iteration of the loop, all 60 bars are draw with their respective heights around a semi-circle. The semi circle is created using the following trigonometric functions which sets the position of a rectangle:

```
translate(width*0.5, height*0.5);//center of sketch
translate( 100*cos( radians( 180+((i+1)*(180/numBars)) ) ),
100*sin( radians( 180((i+1)*(180/numBars)) ) ) );
rotate(radians( 90+(i*(180/numBars)) ));
```

To explain, the first translate statement sets the origin of everything to be drawn to the center of the screen. Thee next statement multiplies a radius(100) by the cosine of a particular bar to find the corresponding X position and does the same again, but rather with sine to find the Y position. The rotate follows to create a radial graph that visualizes the position of the sensor accurately.

Reading from the serial port is done in a readPort() function. This function is ran every draw() loop. The function first starts by waiting for data on the serial port. Then after sanitizing this data it splits the given ordered pair by comma ie “180.0, 6.0” turns into “180.0” and “6.0”. This is then cast to an integer and stored in an array called bars where the angle that is received corresponds to a particular index of the array(180 from above) and that index is set to the ‘height’(6 from above).

Animation will be covered in the additional section below.

Additional – Animation/Stand

In order to further the functionality of my project I added two additional features: animation and assembling a stand for everything to sit on.

In order to create a more fluid animation in processing I animated the radial bar graph to move each bar up and down smoothly. To do this I created a second 2d array that is called ‘oldbars’ and initialized it in the same manner as the original ‘bars’ array. Essentially the ‘oldbars’ array acts as a “before” version of the ‘bars’ array. When a coordinate is passed via serial to the processing sketch there are a few things that happen. First, the coordinates height value is stored at it’s respective index in the ‘bars’ array. I then compare that index of the ‘bars’ array to the same index of the ‘oldbars’ array. If there is a mismatch ie. there is new information in the ‘bars’ array, then I send the height value(second index of the 2d array) of ‘oldbars’ to a function called ‘animateBar(int, int, int)’. ‘animateBar()’ takes in three integer parameters: the old value of height which will be from ‘oldbars’, the target height from ‘bars’, and a modifier to increment by. When ‘oldbars’ height is passed, ‘animateBar()’ returns a positively or negatively incremented integer depending on whether the bar needs to increase or decrease in height. This return value is then set as the new height for for the given index in ‘oldbars’ This runs each loop of the draw() function until the height value in ‘oldbars’ is equal to the height value in ‘bars’. Along with running this every loop, the draw() function redraws every bar in the graph every iteration. This produces a smooth effect that is happening so fast that it does not interfere with the rate at which the Arduino is sending new coordinates. The visual of this effect is seen in Figure 8.

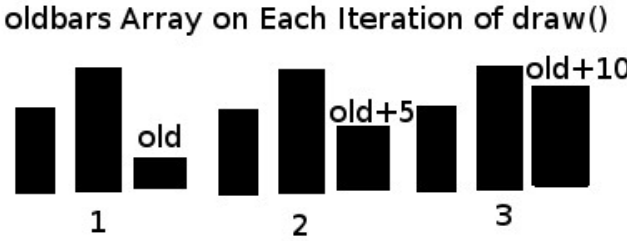


Figure 8: changing and drawing the oldbars array through each iteration of draw() in order to match the bars array.

bars array height values

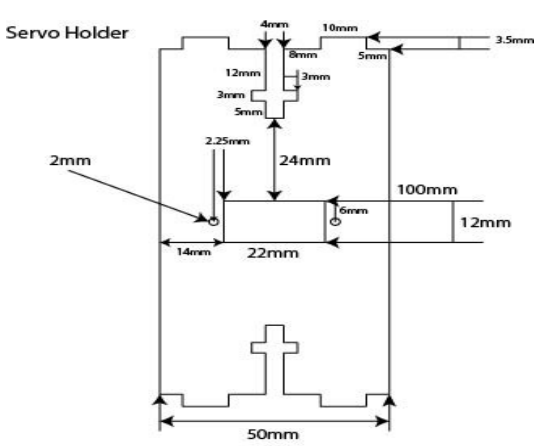
The value by which to increment a bar, denoted ‘mod’, changes throughout the lifespan of the program. The difference between a given index of ‘bars’ and ‘oldbars’ is then divided by a constant being 12. This results in 12 steps to “animate” an index of ‘oldbars’ to its new height.. The equation written out is:

modifier = absoluteValue(oldbars(index) – bars(index))/12

12 was chosen because it provided a smooth visual while still being fast given any change in height.

Stand

The next addition was to make a stand for the entirety for the project to mount on. The idea was to create a 2d stand and laser cut each individual part. There are four parts total: the top – mounts the servo, the sides, and the bottom – mounts the Arduino. The stand sits 75mm tall and 107mm wide. Each part is labeled with dimensions in Figure 9.



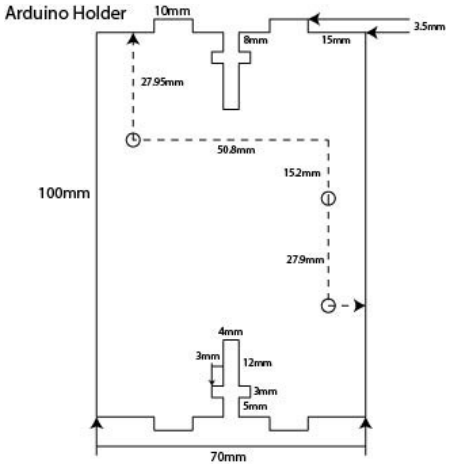


Figure 9: All parts for holding project

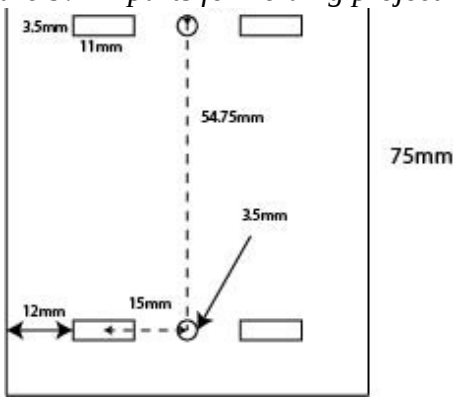


Figure 9 Continued

The individual pieces are then held together using ‘t’ slots and m3 screws and their respective nuts.

Once assembled the stand alone unit animates accordingly in processing. This creates for a fully functioning final product.