McGill Physical Journal

Ultrasonic Depth Perception and Applications to Object Mapping

Chloé Robeyns (260712228), Noah LeFrancois (260706235), William Frost (260767513)

McGill University Department of Physics

April 22, 2019

Abstract

This project explored the capabilities of the Arduino Uno micro-controller when used to measure distances through ultrasonic sensors. Calibration was performed by measuring the speed of sound in air over known distances, extrapolating the slope from a linear least squares fit, and comparing accuracy and precision to this known value. This measurement gave a speed of $(350\pm4)\frac{m}{s}$, compared to the accepted value of $(344.6\pm0.3)\frac{m}{s}$. Then a collection of these low-cost, adaptable sensors was used to map the shape of various 2D and 3D objects in order to create a digital representation of the targets via Python's matplotlib environment. An effective prototype was designed and constructed to mount and maneuver the three HC-SR04 sensors required for 3D coordinate mapping. An Arduino-Python serial interface was used to automatically record mapping coordinates in order to accelerate measurement speed and thereby increase feasible image resolution with recording times of around 5-10 minutes for objects on the scale of 10-40cm in size. This study found that the object's distance to the sensor had the most impact on image accuracy. For close objects (10cm from sensor), their shapes and sizes were well represented by our method. However, holes in these objects were not recognized in some cases; in order to quantify this effect holes of varying side lengths were tested for visibility and the minimum required size for detection was found to be 4cm and 5.5cm at distances of 10cm and 22cm, respectively.

1 Motivation

Ultrasonic technology has developed enormously since it was first discovered in 1794 by Lazzaro Spallanzani. [1] When conducting experiments to determine how bats were able to navigate in darkness, he concluded that bats used sound instead of sight to navigate, leading to the discovery of echolocation. Since then, many other animals such as dolphins and whales have been observed using this method of navigation. It involves them emitting a high frequency sound imperceptible to humans which reflects off objects in their environment and enables them to determine the distance to said objects.

Spallanzani's research was the beginning of the study of ultrasound in physics, with a key development being the discovery of piezoelectricity [1] in 1877 by Pierre and Jacques Currie. [2] This in turn led to sonar detection systems being developed for underwater navigation, which became especially important for submarines and ships and the detection of icebergs after the sinking of the Titanic in 1912.

Over the next few decades ultrasonic technology has developed and its uses have grown exponentially [3], not only for medical purposes but also in scientific research, in the military, in various industrial applications (such as ultrasonic soldering) and in space through NASA's virtual guidance program [2]. This report will focus on the use of ultrasound for the mapping of two- and three-dimensional objects.

The sensor used in this lab was the ultrasonic sensor HC-SR04 for Arduino [4], shown in figure 1. It contains two pins, a trigger pin which emits a high frequency sound wave towards an object and an echo pin which receives the reflected sound wave. The distance, d, between the ultrasonic sensor and the object can then be determined using the time, t, taken for the transmitted signal to reach

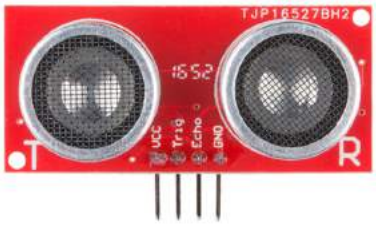


Figure 1: The HC-SR04 ultrasonic sensor. [4]

[1] The piezoelectric effect is the process by which mechanical stress on certain materials, such as crystals, leads to the material being deformed and the resulting kinetic or mechanical energy is converted into electric energy. This energy conversion is how an ultrasound transducer (the part of the ultrasound machine which emits and receives ultrasound waves) is able to detect sound waves [5].

the object and be reflected onto the echo pin, and the speed of sound in air, v_s, using the following well-know relation [6],

$$d = v_s t. \tag{1}$$

2 Speed of Sound as a Function of Air Conditions

Although the laboratory where this experiment was being carried out was a fairly controlled environment, there are several factors which need to be taken into account which could affect the speed of sound in air and consequently affect the calculated distance between the sensor and the object.

One of these factors is the temperature, T, of the air and it affects the speed of sound according to the following equation [7],

$$V_s \approx 20.055\sqrt{T}, \tag{2}$$

where T is measured in Kelvin. The variation of speed with temperature is due to the particles in the air gaining kinetic energy as the temperature increases which gives more energy to the sound wave and enables it to travel faster.

Although not as significant as the temperature, the humidity also affects the speed of sound in air; at a temperature of 20°C the effect of humidity is approximately 0.15% [7], and at a distance of 0.3m between the sensor and the object, this results in a standard uncertainty of 0.3mm. An increase in humidity means an increase in the number of water molecules in the air, and since the mass of a water molecule is about half that of the oxygen and nitrogen molecules which make up most of the air, the density of air decreases as humidity increases which leads to a very slight increase in the speed of sound. [8]

The final factor which can affect the speed of sound in air is a disturbance of the air due to wind. Clearly there was no wind in the laboratory but some disturbances were nevertheless caused by the vent situated directly above our station and people occasionally walking past which also caused the object being mapped to move. This results in a change in the length

of the path travelled by the sound wave, and therefore a change in the distance measured between the sensor and the object. This change in distance, Δd, is shown by the following equation [7],

$$\frac{\Delta d}{d} \approx \frac{1}{2}\left(\frac{v_w}{v_s}\right)^2, \tag{3}$$

where v_w is the velocity of the object. In this experiment however, the velocity of the object due to wind and the distance at which it was being measured were not significant enough for the change in distance to be noticeable.

The speed of sound in air has no pressure dependence since an increase in pressure causes the density of the air to decrease by the same factor and vice versa.

It is also important to consider the frequency of the ultrasonic signal, as this determines the resolution of the image obtained from mapping the object. A higher signal frequency means that the sound wave has a smaller wavelength which implies a better resolution. However, the higher the frequency the more the signal is attenuated so to measure greater distances a lower signal frequency may be preferable. For this experiment however, measurements were taken well within a meter of the ultrasonic sensors used. As such, the default frequency of the HC-SR04 sensors should be adequate for our purposes.

2.1 Calculation of the Theoretical Speed of Sound

For the purpose of this calculation we assume that the air is an ideal gas. If it were a pure gas, the speed of sound would be [8]

$$v_s = \sqrt{\frac{\gamma P}{\rho}}, \tag{4}$$

where P is the pressure of the gas, ρ is the density and γ is the adiabatic gas constant. [2] The value of this constant depends on the shape of the molecules making up the gas. For air, which is composed mostly of the diatomic molecules N_2 and O_2, the typical value of γ is 1.4. [9]

[2] The adiabatic gas constant is given by the ratio of the heat capacities at constant pressure and volume, C_p/C_v, and describes the amount of expansion or compression energy going into a temperature T against the pressure, P. [9]

This equation can be rewritten using the ideal gas law, $PV = Nk_bT$ (where the number of particles, N, is constant) to give [8]

$$v_s = \sqrt{\frac{\gamma k_B T}{M}}, \tag{5}$$

where k_B is Boltzmann's constant, T is the absolute temperature, and M is the mass of a molecule of gas.

Assuming an ideal gas with an adiabatic constant of 1.4, a temperature of 293.17K and a relative humidity of 100%, the speed of sound in air is $(344.6 \pm 0.3)\frac{m}{s}$. [10]

This section has focused on accurately measuring the speed of sound in air and the various factors which could affect it, but these are also important as they can be a source of error when mapping objects. For this experiment, the effect of small variations in the variables mentioned above were deemed small enough that we were able to use Eq. 1 to calculate the speed of sound in air.

3 Materials and Methods

3.1 Sensor Characterization

Before mapping any 2D or 3D objects, we characterized the ultrasonic sensors we were using in order to understand the range at which they were able to accurately measure distances.

The first measurements performed were to determine the speed of sound in the lab environment. At first, two ultrasonic sensors were placed facing each other with one's trigger pin active and the other's echo pin active, so we could use the measured distance between them and the time for the sound wave to travel to calculate the speed of sound using Eq. 1. This method turned out to be ineffective as it was hard to position the sensors so that the signal emitted by one could be picked up by the other. The failure of this method constrained the experiment to only measure the speed of sound in air as opposed to our original goals of measuring it in different media, since transmission through an interface would lead to a reflection giving an interfering signal.

We therefore tried a second method, where one HC-SR04 sensor was placed at nine differ-

ent distances (3 to 80 centimeters) from a cardboard wall and time intervals in microseconds were recorded using a simple Arduino-Python interface. It relayed this time interval information to a Python script which allowed 10 sets of 100 measurements to be taken for each distance. The weighted mean of the means of these measurements was then taken to obtain a final time interval value for a given distance. A plot of distance travelled over time intervals measured was used to perform linear least squares fitting of the data and output a slope value equivalent to the speed of sound. This value was then compared to the expected speed of sound in air introduced in Section 2.1.

In order to determine the distance resolution possible with the HC-SR04 sensor, measurements like those used in the speed of sound calculation were performed from 37 and 40 centimeters with millimeter increments. Each data point is the result of the weighted mean of 3 sets of 10 time interval measurements. A linear least squares fit to the data was performed and corresponding slope parameters were again extrapolated.

3.2 Object mapping

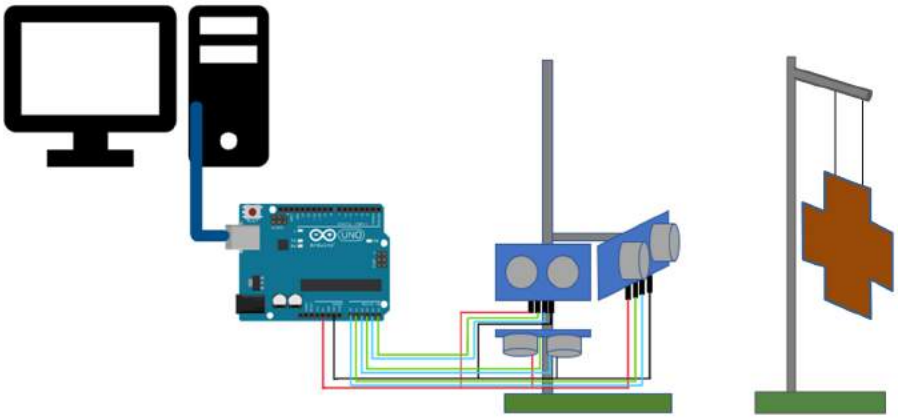


Figure 2: Diagram showing the setup of the equipment used in this experiment. When mapping objects (such as the cross shown in this diagram) in three dimensions, the ultrasonic sensor facing the cross measures the z coordinate, the sensor facing out of the page measures the x coordinate, and the sensor pointing downwards measures the y coordinate.

Our objective in this part of the experiment was to use the ultrasonic sensors to map two- and three-dimensional objects, using the setup depicted in figure 2 above. The three sensors were moved up and down the retort stand simultaneously, and the stand itself was moved

along the table in a restrictive x-y plane so that the entire object area could be covered. The first object mapped was a simple 20cm by 25cm cardboard rectangle which served as a proof of concept for subsequent object mappings and is discussed in Section 3.3. For future reference, all objects mentioned in this report can be seen as images in Appendix A.

Our first goal was to characterize the hole detection capabilities of the sensor, i.e. what is the main limitation to detecting holes in various objects? Evidently, the chosen mapping resolution has an impact on hole detectability, so our smallest resolution was used (0.25cm). We first placed the sensors in front of a long strip of cardboard out of which different sized square holes had been cut, starting from a side length of 9.5cm and getting progressively smaller, the goal being to detect the holes at two different measurement distances. Additionally, mapping of a square window (hole length 8cm and outer length 35cm) was also performed at two different distances.

Along with this, the sensor's ability to resolve sharp edges was also subject to rigorous testing. By mapping a 2D cardboard cross of height 18cm and width 19.5cm at different resolutions and two different distances, comparisons were made between those mappings to determine whether resolution or object distance to sensor was the dominant limiting factor to image accuracy.

Furthermore, the objects ability to map smooth 3D curvatures was also tested. An 80cm-diameter sphere placed onto a supportive cylinder was mapped at the same 30cm distance from its center using all four of our implemented resolutions to see whether or not the sensor is actually capable of this sort of detection, and if so, what resolution was necessary.

Finally, the last objects mapped were a water bottle, a mason jar and a cut-out cardboard face. These were meant to test the sensors capacity to resolve mundane things and to combine certain characteristics (i.e. holes, curves) analysed separately thus far. Actual images (which can be seen in Appendix A) were compared to the object mappings to determine if the digital reproductions were accurate.

3.3 Software Implementation

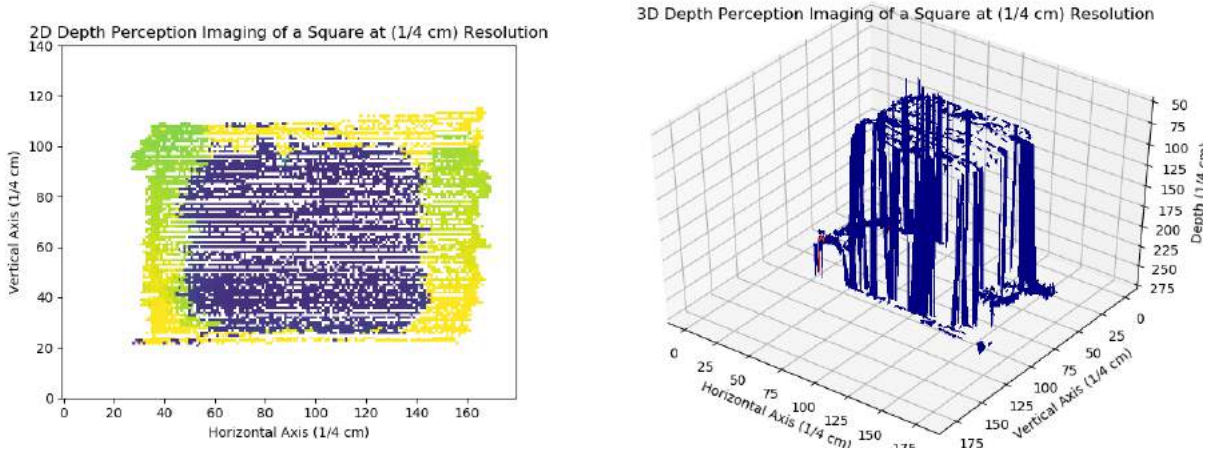
The code discussed here can be found in Appendix B.

As mentioned previously, automatic data acquisition was made possible through an

Arduino-Python serial interface. The underlying principle is that on the Python end, a 2D array awaits to receive x and y values as indexes and fills those indexes with the z values coming from the Arduino. On its end, the Arduino fires a sound pulse for the x, y and z ultrasonic sensors, records the time it takes for the echo pins to detect them and converts them to a distance in either 2cm, 1cm, 0.5cm or 0.25cm units using the expected speed of sound value introduced in Section 2.1. It then sends a string containing the information via the Serial.println() command which Python then receives and parses. And thus the cycle continues. On most of our mappings, the data transfer rate allowed for up to 80 (x,y,z) measurements to be taken per second.

Precisely, it all starts with the Python program initialising a 2D array corresponding to the resolution we expect to receive from the Arduino. The chosen resolutions of 2cm, 1cm, 0.5cm and 0.25cm were used to cover an interesting experimental range. In addition, we specify the total number of serial exchanges we want the Arduino-Python to have (i.e. the max amount of data we want to receive) and also a percentage value used to specify what percentage of the array indexes we want filled. Both these conditions can be used to stop the Arduino-Python interaction. While this interfacing takes place, data is added to an image array index with the condition that newer data replaces older data at that given index.

Once the Arduino-Python interface closes, data manipulation can now begin. Right before graphing, it is recommended to set to "None" (equivalent of NULL) all values of z which are 0 in order to make the plot look better. 0 values in the array are places where no measurements were taken. In most cases, especially at small resolutions, it will be practically impossible to fill all of the 2D array elements with z depth values. As an example, here is what one raw mapping of a carboard square looked like at 0.25 cm resolution with the square being 20cm away from the sensor. The darker, purple pixel coloration in Fig. 3a represents closer z values (the object):



(a) 2D image rendering of the square

(b) 3D rendering of the square

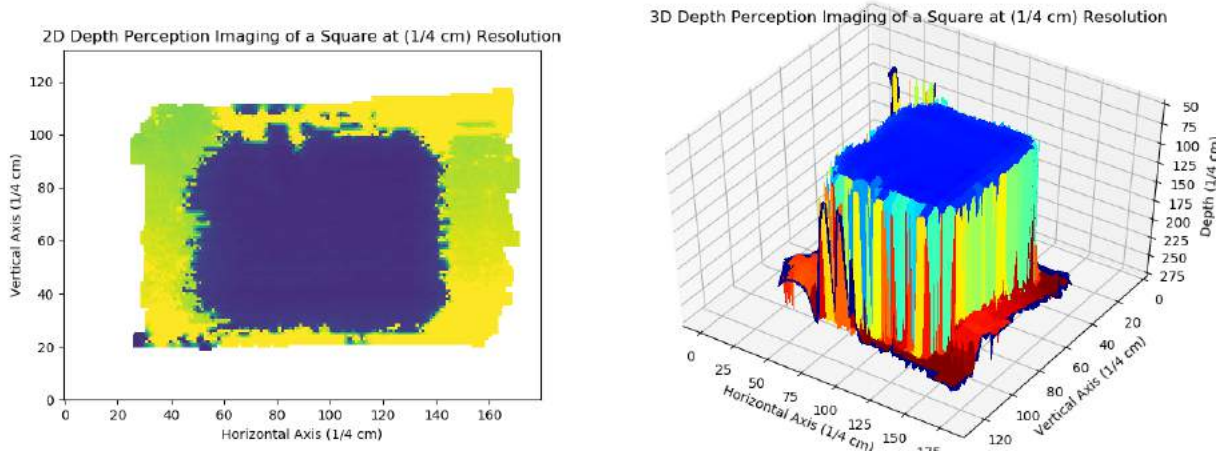
Figure 3: Images of the square target scanned at a distance of 20cm with 0.25cm resolution. The 2D image of Fig. 3a shows the shape well, but the 3D rendering using Python's matplotlib package in Fig. 3b is clearly underwhelming.

The gaps visible in the 2D image are directly responsible for the inadequate 3D graphing. This particular one-shot mapping run took approximately 5 minutes to do and required very delicate increases in y when sweeping the target object horizontally. This trial alone shows that a fast data acquisition rate and delicate area sweeps are not sufficient for our purposes. Therefore, additional data manipulation will be required to produce adequate 3D surfaces.

To solve this problem, two solutions were designed. The first one was to simply combine multiple mapping runs of the same object into one array, hoping that gaps in one run would be filled in by hits in the other. Although this technique was used in conjunction with the second one about to be described, by itself it was not sufficient.

The second approach was to take a mean of points around all zero values contained within the object portion of the image and use that to replace the gaps. We first start by scanning the finalized 2D array and storing the location of elements that are not on the array border, that have a 0 value and that also possess one neighboring element that is not zero. This last condition is important since it prevents 0 values that are far from the object's location in the 2D array to be unnecessarily considered. This first sweep only stores the locations of the values to be changed. Otherwise, if those points were to be updated immediately, the algorithm would end up filling in all the points after a certain vertical y height.

A second sweep of the array is then performed only at the locations previously determined. Each location's 0 value is updated to become the mean of it's non-zero neighbor values. This whole process can be run numerous times to progressively fill up the gaps in an image. As a proof of concept, here is the same square mapping seen previously in Fig. 3, but with only one run of the correcting algorithm applied to it:

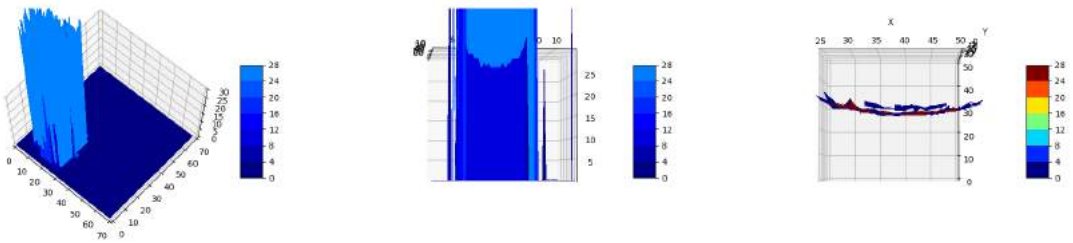


(a) 2D enhanced image of the square

(b) 3D enhanced image of the square

Figure 4: Images of the square target scanned at a distance of 20cm with 0.25cm resolution. The enhanced 2D image of Fig. 4a shows no gaps, and the 3D rendering using matplotlib in Fig. 4b is remarkably improved.

Clearly, this method works in smoothing out the sensor data. In addition to this, short for-loop algorithms were used to determine optimal y-axis limits for the graphs as well as setting all zero or undesirable values to None (NULL) so that they are not considered in the graphing process. Finally, for certain images, certain ranges of points are omitted to make some 2D and 3D images clearer. An example of this is demonstrated below when mapping a mason jar:



(a) Unprocessed, top view (b) Unprocessed, side view (c) Processed, side view

Figure 5: Images of the circular mason jar target scanned at a distance of 30cm, before and after removal of misses. This processing converts the unintelligible raw plots into discernible object images. Fig. 5c is the front of the mapped mason jar viewed from the top.

4 Results

4.1 Sensor Characterization

The speed of sound as measured by linear least squares fitting of Fig 6 is $(351 \pm 1)\frac{m}{s}$. This is a slight overestimate compared to the theoretical value of $(344.6 \pm 0.3)\frac{m}{s}$ under our lab conditions as discussed in the introduction.

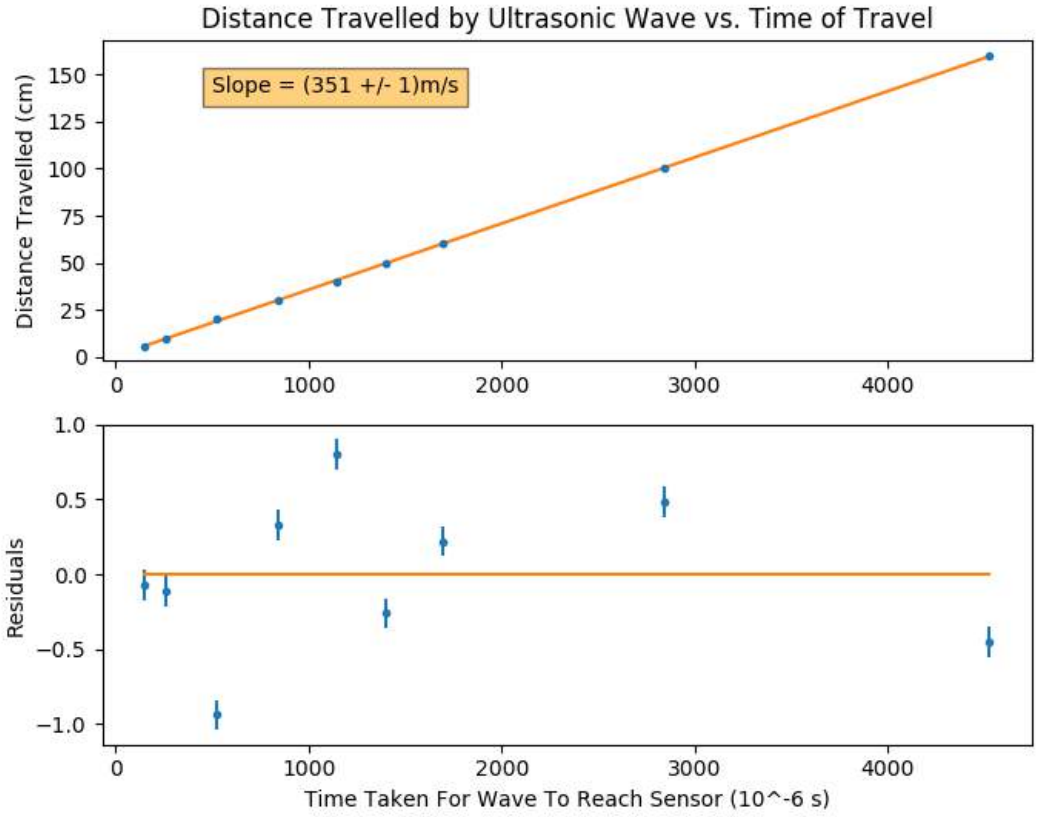


Figure 6: Plot of distance travelled as a function of travel time for the ultrasonic wave over a range from 3cm to 80cm. Residuals show no pattern, and the data is spread randomly around the line, indicating that the linear model is a good fit. The slope was found to be $(351 \pm 1)\frac{m}{s}$ which gives a measurement of the speed of sound.

In order to examine the resolution of our ultrasonic distance measurements, a number of data points were taken at small distance increments and their corresponding travel times were plotted. This data is fit well by a linear model, and shows that the HC-SR04 ultrasonic sensor is efficient when resolving millimeter increments (or bigger) of distance given an accurate measurement of the speed of sound.

In addition, the smallest distance the ultrasonic sensor could accurately measure was determined to be 2cm. Placing it in front of a wall at that distance,we proceeded to move it back in increments of a few centimetres, recording the measured distance at regular intervals until sensor's measurements were no longer accurate which was at a distance of around 200cm.

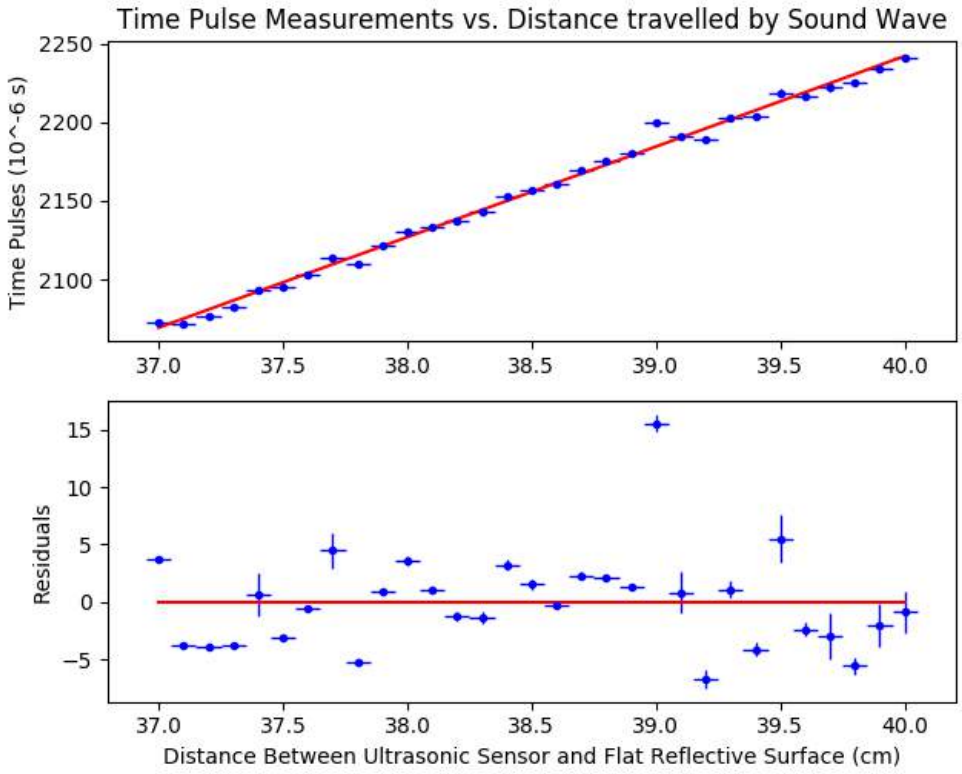
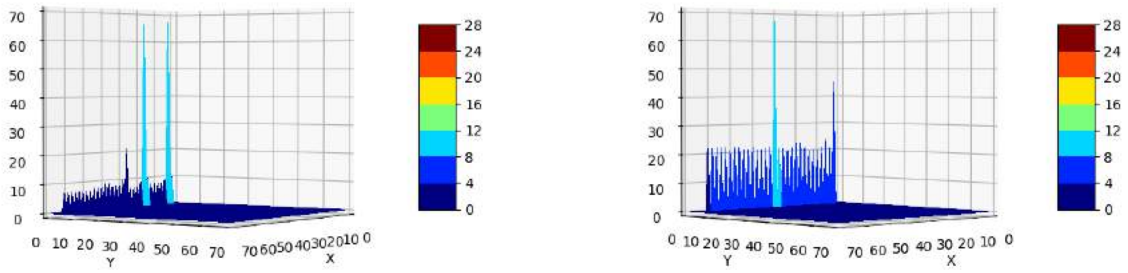


Figure 7: Plot of travel time as a function of distance travelled by the sound wave over a range of 37cm to 40cm. The horizontal error is from the uncertainty in the distance measurements, determined to be half the measurement resolution of a meter stick ((±0.05)cm). The vertical error is the error in the mean of means of 3 sets of 10 measurements. The residuals show no pattern, indicating that the linear model is a good fit.

4.2 Object Mapping

For all graphs presented in this report, the uncertainty in the distance data is equivalent to half the resolution used to map the objects. This is true for distances shorter than 114.9cm and our objects were all scanned at distances of 10, 20, or 30cm. A deeper understanding of this is presented in Section 5.1 of the Discussion.

When testing the scanner's ability to detect holes, this sensitivity was found to improve for shorter distances from the sensor. At 10cm the sensor detected a square hole of side length 4cm; at 22cm this hole was not detected while a hole of 5.5cm length was. Both distances allowed detection of the largest hole, with side length 9.5cm.

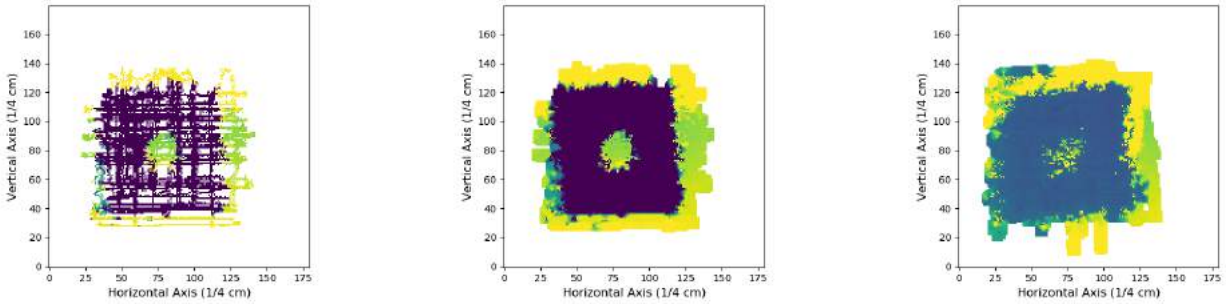


(a) Observing hole test-target at 10cm

(b) Observing hole test-target at 22cm

Figure 8: Results of scanning the hole test-target with holes of varying sizes. A hole is indicated in these plots by the taller spikes rising above the flat line of lower spikes, showing that the wave is passing through instead of hitting the target. Holes of side lengths 9.5cm, 5.5cm, and 4cm were detected at 10cm from the sensor, while only the two largest of these holes were detected at a distance of 22 cm.

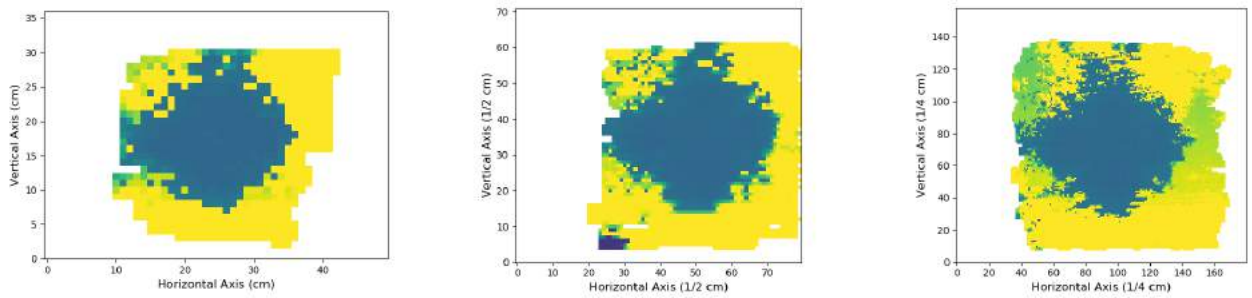
Imaging of the square window target further demonstrated the sensitivity to measurement distance of hole resolution; this feature was much more recognizable at a distance of 10cm compared to a distance of 30cm. The side length of the actual hole was 8cm, while the side length of the image's hole can be seen to be approximately 6cm on the 0.25cm scale used. This smaller measured hole is not likely due to the processing algorithm used, since comparing Fig. 9a with Fig. 9b, which was processed with 3 enhancement runs, shows the same apparent hole size. Therefore the sensor mounted on the experimental setup was likely the cause of this accuracy loss.



(a) Unprocessed window at 10cm (b) Processed window at 10cm (c) Processed window at 30cm

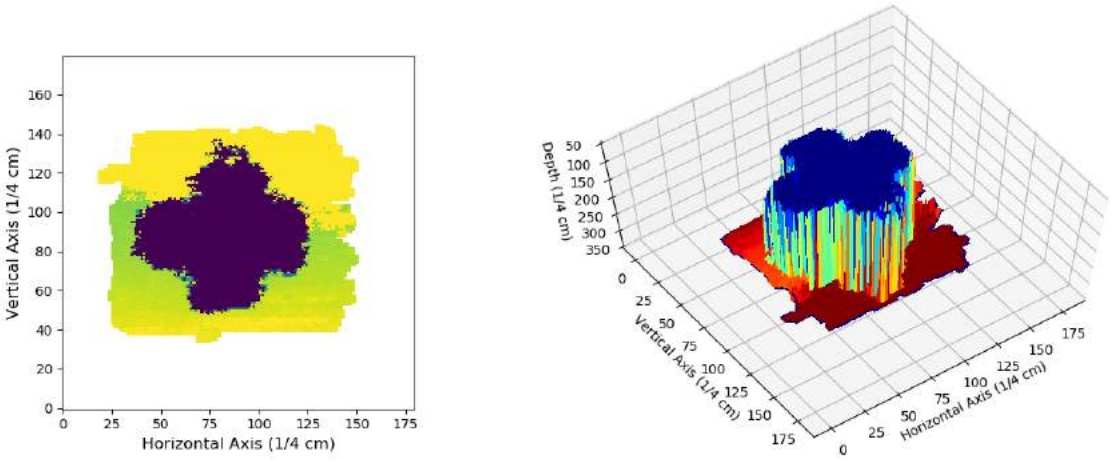
Figure 9: Images of the cardboard window scanned at distances of 10cm and 30cm. The hole is clearly visible in Fig. 9b, but is poorly resolved at the further distance in Fig. 9c.

In Fig. 10, images of the cross at 30 cm with 3 different resolutions showed increasingly better defined corners. Whereas the cross looks more like a diamond when resolved at 30cm, when compared to the cross imaging of Fig. 11 taken at 10cm, we see drastically better definition. This indicates that image accuracy is affected more by object distance than image resolution. The upper portion of the cross lacks sharpness compared to the other areas, likely due to the sensor picking of the strings that supported the cross in mid-air.



(a) Cross at 30cm with cm resol. (b) Cross at 30cm with 0.5cm resol. (c) Cross at 30cm with 0.25cm resol.

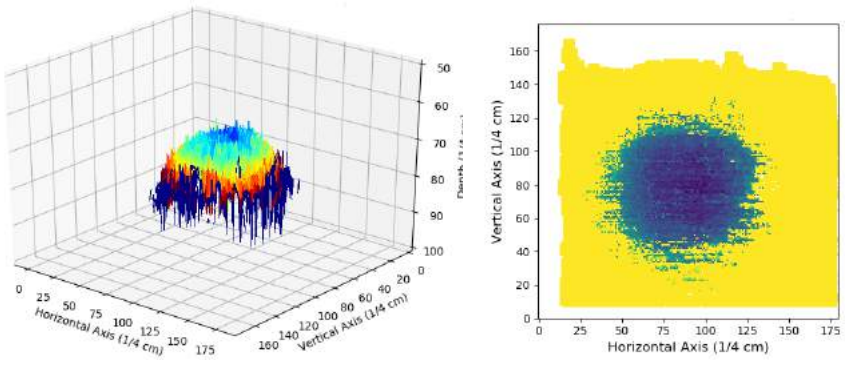
Figure 10: Images of the cross target scanned at distances 30cm with various resolutions. The corners do seem to be more defined as resolution lowers, but are still unable to reproduce the cross' sharp edges.



(a) Cross at 10cm with 0.25cm resol. (b) 3D Mapping of cross at 10cm with 0.25cm resol.

Figure 11: Images of the cross target scanned at a distance of 10cm with 0.25cm resolution. Contrary to Fig. 10, the corners are well defined in Fig. 11a. A 3D rendering is also improved.

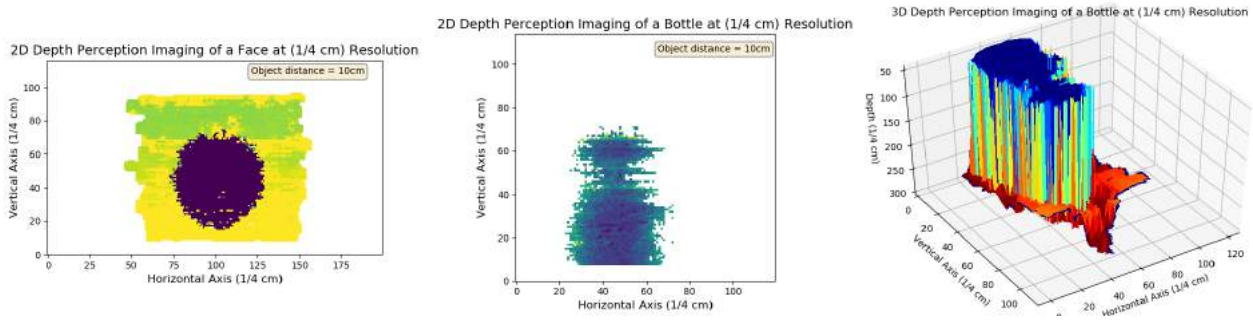
Scanning data of the sphere was used to create a 3D mapping illustrating the object's curvature, as well as to produce a 2D heat-map plot from the front view of the sensor. The uniform curvature of the sphere is clearly visible in both plots, with the 3D colour scale going from blue for closer points to yellow/red for the furthest points. Darker points below the red strip are not considered as hits on the sphere. For the 2D plot the outline is blue-green and the wall is yellow. Other plots showing different resolutions of sphere mapping can be found in Appendix B



(a) (b)

Figure 12: Images of the sphere at 10 cm. The 3D curvature of the object is displayed in Fig. 12a, while Fig. 12b displays the 2D resolution observed.

Imaging of the face-shaped target failed to detect the eyes even at the relatively short distance of 10cm. A few pixels of the mouth hole were detected but not enough to form any recognizable shape. The circular outline of the figure was well resolved however. Only one enhancement run was performed on the image, which did not have the effect of mistakenly filling in the eyes and mouth. Similarly, the water bottle outline was recognizable at the same distance. Fig. 13b shows that the curvature of the bottle was well resolved, with darker colors corresponding to closer distances. The smaller-scale curvature at the neck was slightly blurred but still visible and the bottle's top tip was not detected.



(a) (b) (c)

Figure 13: Images of the face target (a) and the water bottle (b and c) at a distance of 10cm.

5 Discussion

5.1 Sensor Characterization

The result of the speed of sound measurements performed in Fig. 6 using linear least squares fitting was $(351 \pm 1)\frac{m}{s}$, a discrepancy of 4.9σ from the expected value for sound travelling in a 100% humidity, 20^0 C environment, which is $(344.6 \pm 0.3)\frac{m}{s}$. Additionally, when taking each adjacent pair of data points in Fig. 6 and computing the speed of sound with each individual pair, the result still holds above $350\frac{m}{s}$ for all points. However, this unusual discrepancy may have its origins in a small systematic error. In fact, it was not trivial to estimate by visual inspection where in the ultrasonic sensor did the sound wave start and where it was received by the echo sensor (See Fig. 1 and Fig. 2 for illustration of the sensor). This small error on the order of 2-3mm from roughly looking at the sensor to determine the

origins and destinations of the waves itself has interesting implications. As an example, the speed of sound calculated at a distance of $40cm$ is $(350 \pm 2)\frac{m}{s}$ when doing error propagation with $(\pm 0.05)cm$ of distance error. When taking into account the uncertainty of not knowing exactly where the actual piezos are in the sensor (estimated to be $(\pm 0.2))cm$, the new error propagation yields $(350 \pm 4)\frac{m}{s}$. This value is within 2 standard deviations of our expected value at 20^oC environments. Therefore with a better understanding of the sensor build and it's precise geometry, it seems more accurate speed of sound results could have been obtained.

The data of Fig. 7 shows that for millimeter increments in distance, the HC-SR04 sensor seems to respond linearly. Interestingly enough, when computing the inverse of the slope obtained by linear least squares fitting, we obtain a more accurate speed of sound value with less precision of $(345 \pm 5)\frac{m}{s}$, which is within one standard deviation of the expected value of $(344.6 \pm 0.3)\frac{m}{s}$. This better accuracy (not precision) is most likely due to the increased amount of separate data points taken compared to the initial speed of sound calculation. Because the sensor shows the capability for millimeter resolution (of flat, perpendicular surfaces at least), it implies that the only limiting factor should be the precision and accuracy of the speed of sound value used to convert to distance. As an example, our speed of sound obtained in Fig. 6 of $(351 \pm 1)\frac{m}{s}$ would enable us to have mm precision or better below $35.1cm$ since

$$\frac{(351 \pm 1)(100cm)}{(s)} \cdot \frac{2000}{10^6} s \cdot \frac{1}{2} = (35.1 \pm 0.1)cm \ . \tag{6}$$

In this case, the relative error in the time pulse measurement is smaller than in the speed of sound measurement, so the latter's uncertainty dominates during error propagation. Therefore, it can be approximated that resolution capability is proportional to speed of sound uncertainty. Using the speed of sound value introduced in Section 2.1 in Eq. 6, the maximum distance where millimeter precision would be possible becomes $(114.9 \pm 0.1)cm$. Since this speed of sound value was used during experimentation to convert time to distance, we can safely assume that all data acquired for our figures was at least at millimeter precision. Therefore, the truncated resolutions of no smaller than 2.5mm should be well resolved by the sensor. In the context of this experiment, it can thus be assumed that the only limit

to resolution for any single object data point will be the forced truncation of the data sent to the Python array (i.e. are we choosing to send data in the form of 2cm, 1cm, 0.5cm or 0.25cm pixels).

5.2 Object Mapping

One significant difficulty encountered in this project was the quality of visualization provided by the matplotlib.pyplot library's Axes3D environment. A commonly cited issue with this 3D plotting software is that from some viewing angles, an object may appear in front of another object even if it is physically behind it [11]. As explained in the matplotlib documentation:

"*This problem occurs due to the reduction of 3D data down to 2D + z-order scalar. A single value represents the 3rd dimension for all parts of 3D objects in a collection. Therefore, when the bounding boxes of two collections intersect, it becomes possible for this artifact to occur.*"

This issue heavily restricted the choice of viewing angles for which our images were coherent and well-represented. Future work could explore the use of plotting environments more specialized for use in 3D imaging, such as MayaVi as recommended by the matplotlib documentation itself, in order to achieve optimal imaging in a more efficient and less limited fashion.

The major takeaway from our testing was that the clarity of small-scale features such as holes and sharp corners was highly sensitive to object distance, limiting the effective operational range of our device to under 20cm for objects in the 10-40cm size scale range. This poor performance occurs at much shorter distances than is explained by the distance uncertainties examined in our calibration shown by Fig. 6, which displays only small precision losses when going from object distances of 10cm to 50-70cm. One factor that could account for this discrepancy is dispersion occurring while the sound pulse travels: if the wave spreads out in the x-y plane it may hit surfaces not directly in front of it and register false positives (when in front of a hole, for example) due to hitting a nearby surface which returns the pulse to the receiving sensor even though it is not directly in front of the intended line of sight.

This effect would not significantly affect the calibration measurements which only need to hit a solid wall at the fixed known distance, but could distort or cover up small scale features as was observed here. Since this dispersion would grow as distance travelled increases, it would accordingly become more noticeable for more distant objects as we found, matching the observed behaviour. An additional source of accuracy loss was hypothesized to have been due to the filling algorithm explained in Section 3.3, which could have slightly exaggerated certain object features by filling gaps with inaccurate data. This would have been apparent for images at higher resolutions with a lot of gaps to fill. However it was not a prominent source of error when using low resolutions, as seen in Fig. 9. Rather the physical limitations of the setup seemed to be the cause. In addition, the contour of curved objects was found difficult to resolve, sometimes leading to drastically overblown distance measurements. This is likely due to the sound wave not deflecting off of the surface directly back at the echo sensor due to this curvature. Therefore, a polar coordinate method of viewing 3D curved objects could be advised instead of a strictly in-plane viewing as was done in this experiment.

Another potential source for image blurring could be the sensor array mounting and the manual scanning technique it requires. Because the retort stand clamp must be repeatedly adjusted in order to change the sensor height, the angle at which it points relative to the x-y plane may not be perfectly held stationary throughout the measurement which could cause y-z value pairs to be attributed to incorrect x coordinate locations.

Within this limited range for optimal performance, features such as the curvature of a sphere, corners of a cross, and hole in the middle of a square window were all recognizably imaged. As a proof of concept prototype, these results demonstrate the feasibility of proximity imaging with an ultrasonic sensor array; however a number of improvements to the setup could be implemented. The sensor array could be mounted on a mechanically controlled track instead of manual manipulation, allowing more precise movement as well as fully automating the measurement process. This would be a significantly more complicated and time-intensive setup than was feasible given the time scale and budget of this project due to the added complexity of automating motors through the serial interface and building the required mechanical apparatus along which the sensors would be precisely moved. However, it would make the scanning process faster, easier, and likely allow high-quality resolution of

small-scale features at further distances. It would also provide a more realistic characterization of the limitations inherent to ultrasonic mapping using the HC-SR04 sensors by greatly suppressing human-induced error.

A set-up similar to this extended concept was explored by Yao, C.W. [12], who built a fully motorized device to automate measurements with 0.1mm resolution; with such a setup it was possible to achieve scans of such high quality that a coin could be scanned and used to 3D print a recognizable representation of it. Clearly this ultrasonic approach to object imaging shows feasibility and effectiveness when fully developed.

6 Conclusions

In this lab we demonstrated how the HC-SR04 ultrasonic sensor for Arduino could be used to measure the speed of sound in air and map 2D and 3D objects. The speed of sound was found using a linear least squares fitting to be $(351 \pm 1)\frac{m}{s}$, which is a discrepancy of 4.9σ from the theoretical value of $(433.6 \pm 0.3)\frac{m}{s}$. This difference stems from the small systematic errors which were endemic in each measurement we took. When taking into account an estimate of this systematic error, a more consistent result of $(351 \pm 1)\frac{m}{s}$ was obtained.

The precision of the ultrasonic sensors was an important factor to consider before trying to map objects, and our measurements indicated that the precision was proportional to the uncertainty in the speed of sound. This result makes sense since ultrasonic sensors make use of the speed of sound to measure distances, and therefore the lower the error in sound speed, the more confidence we have that our results are accurate. In this instance, we wished to be able to measure distances on the order of millimeters, and the maximum distance at which such precision was possible was (114.9 ± 0.1)cm.

The final part of this experiment was the mapping of 2D an 3D objects and this was the most challenging component in terms of data processing, as each set of measurements had to be processed multiple times in order for the 2D and 3D plots to accurately represent the objects being mapped. It was found that the distance between the sensor and the objects had a more significant effect than the resolution chosen to map them. While it was possible to map the general outline of certain objects such as the cross, the square, the globe and the

water bottle, smaller details were not detected by the ultrasonic sensors.

Further work on extending this project could yield a greatly improved image resolution, enabling sufficient performance to be useful for a variety of applications. The most obvious of these is 3D modelling of objects for replication via 3D printing, but it could also be used as a visual input for machine vision algorithms for projects in which speed of image-scanning is not an important factor. Another sector that could benefit from rough ultrasonic imaging would be first responders. Even though sound waves are more difficult to process into images than light itself, some scenarios could arise where thermal imaging cameras, for example, are rendered inefficient due to raging fires. In this scenario, having an alternative ultrasonic mid-range imaging method implemented in a HUD could help guide firefighters safely through intense burning buildings, for example.

References

[1] When Was Ultrasound Invented?, www.ultrasoundtechnicianschools.com/articles/when-was-ultrasound-invented/. 1

[2] "History of Ultrasound." – Overview of Sonography History and Discovery, 25 Mar. 2019, www.ultrasoundschoolsinfo.com/history/. 1

[3] "Ultrasonics - Real-Life Applications." Science Clarified, www.scienceclarified.com/everyday/Real-Life-Physics-Vol-3-Biology-Vol-1/Ultrasonics-Real-life-applications.html. 1

[4] SparkFun Electronics, *Ultrasonic Distance Sensor - HC-SR04 Data Sheet* (April 28 2016) 1

[5] Nadrljanski, Mirjan M. "Piezoelectric Effect — Radiology Reference Article." Radiopaedia Blog RSS, radiopaedia.org/articles/piezoelectric-effect. 1

[6] Crang. "Speed = Distance/Time : Chronotopographies of Action." Durham Research Online, Stanford University Press, 1 Jan. 2007, dro.dur.ac.uk/5027/. 2

[7] Carullo, A., and M. Parvis. "An Ultrasonic Sensor for Distance Measurement in Automotive Applications." IEEE Sensors Journal, vol. 1, no. 2, Aug. 2001, p. 143., doi:10.1109/jsen.2001.936931. 2, 3

[8] Suits, B.H. Speed of Sound in Air, Michigan Technological University, pages.mtu.edu/ suits/SpeedofSound.html. 2, 3, 4

[9] Smith, Julius O. Adiabatic Gas Constant, $ccrma.stanford.edu/\ jos/pasp/Adiabatic_Gas_Constant.html$. 3

[10] Zuckerwar, Allan J. Handbook of the Speed of Sound in Real Gases. Academic Press, 2002. 4

[11] Hunter, J. et. al., *mplot3d Frequently Asked Questions*, matplotlib Version 2.2.2 Documentation. 18

[12] Yao, C.W., *An Ultrasonic Method for 3D Reconstruction of Surface Topography* J. Phys. Commun. (2018) 2 055034 20

[13] Farrington D. et. al., *Ultrasonic Mapping Device; Major Qualifying Project*, Worcester Polytechnic Institute (April 28 2016)

A Images of Scanning Targets

Fig. 14 below shows the 2D objects we mapped.

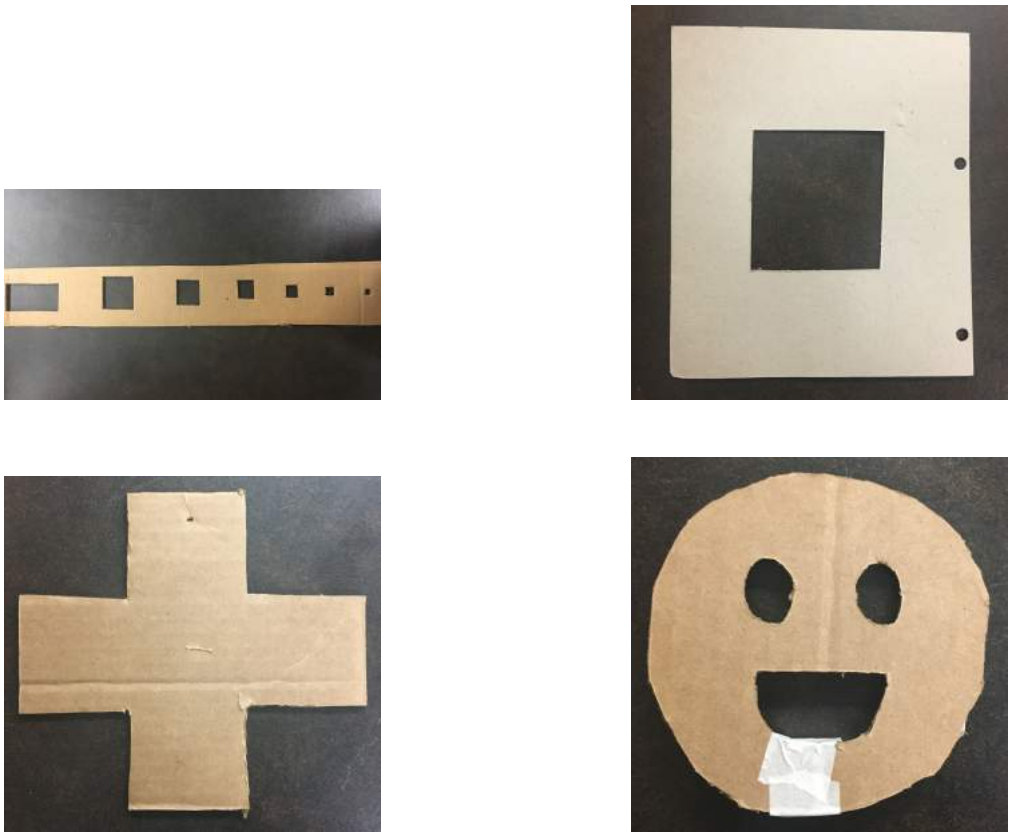


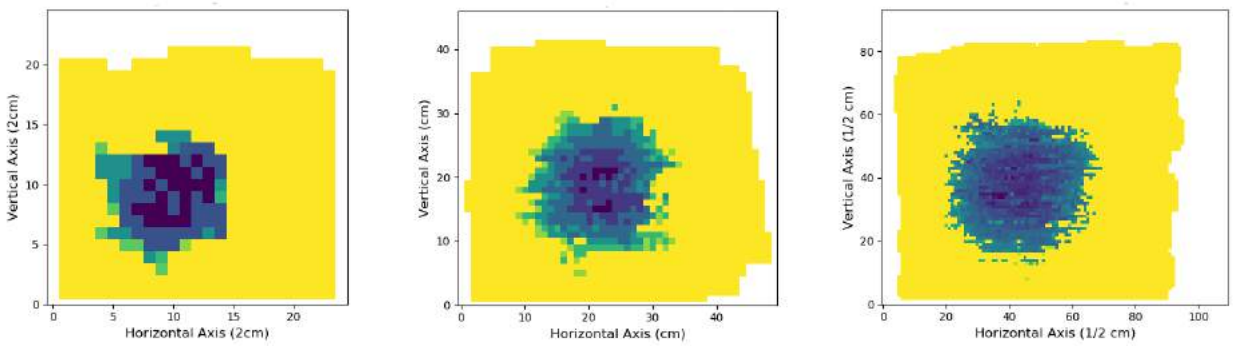
Figure 14: Pictures of the different sized holes used to test the precision of the ultrasonic sensors, as well as the window (hollow square), the cross and the face.

Fig. 15 shows the water bottle and the sphere that were mapped as 3D objects.

Figure 15: Pictures of the water bottle and sphere used for the 3D mappings.

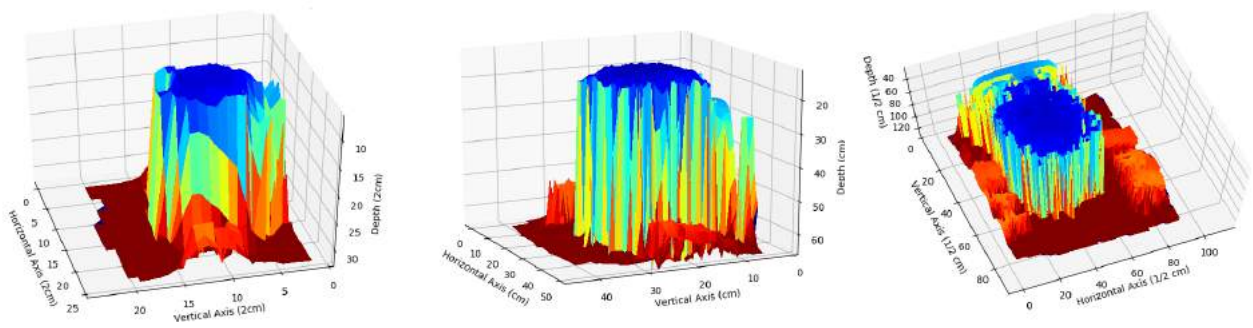
B Additional Mappings

Here can be found 2D and 3D mappings of our sphere object with 2cm, 1cm and 0.5cm resolutions:



(a) Sphere at 2cm resolution (b) Sphere at 1cm resolution (c) Sphere at 0.5cm resolution

Figure 16: 2D mappings of the sphere with its center at 30 cm from the sensor.



(a) Sphere at 2cm resolution (b) Sphere at cm resolution (c) Sphere at 0.5cm resolution

Figure 17: 3D mappings of the sphere with its center at 30 cm from the sensor.

C Python Code Used

```
import serial
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

#xyLim = [25,25,15000]      # for 2 cm resolution
#xyLim = [50,50,20000]      # for cm resolution
#xyLim = [100,100,20000]    # for 1/2 cm resolution
xyLim = [200,200,30000]     # for 1/4 cm resolution
image = np.zeros(shape = (xyLim[0],xyLim[1]))
completeness = 0.8  # stops the serial interface if sufficient array values obtained
count = 0
countTot = 0
data = [0,0,0]
dcount = 0
ok = 0
```

```
# START OF DATA TAKING WITH ARDUINO-PYTHON INTER(IN YOUR)FACE

serialPort=serial.Serial()
serialPort.baudrate=9600
serialPort.port="COM?"     # Define COM port number
serialPort.open()
serialPort.flushInput()
# if certain % of array is filled or max number of data points taken, end loop
while (count < completeness*xyLim[0]*xyLim[1] and countTot < xyLim[2]):

    string = serialPort.readline()
    for x in string.split(" "):
        if (x != "\n" and x != "" and x != ''):
            data[dcount] = int(x)
            print(int(x))
            dcount+=1
            ok = 1
    dcount = 0

    if (ok == 1 and data[0] < xyLim[0] and data[1] < xyLim[1]):
        if (image[data[0]][data[1]] == 0):
            count+=1
        image[data[0]][data[1]] = data[2]
        print("Data taken")
        countTot+=1
        print(countTot)
    ok = 0

serialPort.close()
```

```
# END OF DATA TAKING WITH ARDUINO-PYTHON INTER(IN YOUR)FACE

# START OF DATA MANIPULATION FOR GRAPHING
objekt = ["Square","Window","Cross","Sphere","Face","Bottle"]
obj = objekt[5]
dist = 20
rezol = ["2cm","cm","1/2 cm","1/4 cm"]  # used in graphing
adjustit = [0.5 , 1 , 2 , 4]  #used in graphing
rezolut = rezol[3]
adjust = adjustit[3]
col = [12.5*adjust,65*adjust]  # used in color mapping
over = 350   #TO BE CHANGED BASED ON IMAGE AND RESOLUTION
maxObjDist = over # can be lowered if certain parts of graph are to be ignored
again = 2  # governs how many correction runs are performed on the data

#This part fills in the holes in our figure
for runit in range(again):

    toFill = []
    # This first nested loop scans the image for holes to fill
    for i in range(xyLim[0]):
        for q in range(xyLim[1]):

        # checks we are not at array border and if datum is undesirable
            if (i > 0 and q > 0 and i < xyLim[0]-1 and q < xyLim[1]-1):
                if (image[i][q] == 0 or image[i][q] > over):

                    # if any of the 8 other array values around this datum are
                    # not zero, then add this datum's location to the toFill list
                    if (image[i-1][q] != 0 and image[i-1][q] < over):
```

```
                    toFill.append([i,q])
                elif (image[i-1][q-1] != 0 and image[i-1][q-1] < over):
                    toFill.append([i,q])
                elif (image[i][q-1] != 0 and image[i][q-1] < over):
                    toFill.append([i,q])
                elif (image[i+1][q-1] != 0 and image[i+1][q-1] < over):
                    toFill.append([i,q])
                elif (image[i+1][q] != 0 and image[i+1][q] < over):
                    toFill.append([i,q])
                elif (image[i+1][q+1] != 0 and image[i+1][q+1] < over):
                    toFill.append([i,q])
                elif (image[i][q+1] != 0 and image[i][q+1] < over):
                    toFill.append([i,q])
                elif (image[i-1][q+1] != 0 and image[i-1][q+1] < over):
                    toFill.append([i,q])

#This part of the code actually fills in the holes located in the first loop
sumAreMean = 0.0
count = 0
for h in range(len(toFill)):

    i = toFill[h][0]
    q = toFill[h][1]
    # if the data around the point is non-zero, then, excluding zeros, take
    # the mean of the surrounding data and make it the new value for the point
    if (image[i-1][q] != 0 and image[i-1][q] < over and image[i-1][q]):
        sumAreMean += image[i-1][q]
        count+=1
    if (image[i-1][q-1] != 0 and image[i-1][q-1] < over):
        sumAreMean += image[i-1][q-1]
```

```
            count+=1
        if (image[i][q-1] != 0 and image[i][q-1] < over):
            sumAreMean += image[i][q-1]
            count+=1
        if (image[i+1][q-1] != 0 and image[i+1][q-1] < over):
            sumAreMean += image[i+1][q-1]
            count+=1
        if (image[i+1][q] != 0 and image[i+1][q] < over):
            sumAreMean += image[i+1][q]
            count+=1
        if (image[i+1][q+1] != 0 and image[i+1][q+1] < over):
            sumAreMean += image[i+1][q+1]
            count+=1
        if (image[i][q+1] != 0 and image[i][q+1] < over):
            sumAreMean += image[i][q+1]
            count+=1
        if (image[i-1][q+1] != 0 and image[i-1][q+1] < over):
            sumAreMean += image[i-1][q+1]
            count+=1

        if (count>0):
            image[i][q] = sumAreMean/count
            sumAreMean = 0.0
            count = 0

# (end of big loop)
# Setting an optimal y-limit for graphing
yG_lim = 0
boo = True
booY1 = False
```

```
booY2 = False
for q in range(xyLim[1]):
    if (boo == True):
        if (image[q][xyLim[0]/2] != 0 and booY1 == False):
            booY1 = True
        if (image[q][xyLim[0]/2] == 0 and booY1 == True):
            yG_lim = q+5*adjust
            print(q)
            boo = False

# Setting 0 and other selected depth values to NULL such that they aren't graphed
for i in range(xyLim[0]):
    for q in range(xyLim[1]):
        if (image[i][q] <= 1 or image[i][q] > over or image[i][q] > maxObjDist):
            #print(image[i][q])
            #print("yes")
            image[i][q] = None

# END OF DATA MANIPULATION FOR GRAPHING

# BEGIN PLOTTING
# The following can produce a 3D surface, a 2D image and a 3D scatter plot

figgy = plt.figure()
ax = Axes3D(figgy)
xdat,ydat = np.meshgrid(range(xyLim[0]),range(xyLim[1]))
ax.plot_surface(xdat, ydat, image ,cstride=2,rstride=2, antialiased = False , cmap = cm.
ax.set_xlabel('Horizontal Axis (%s)' % rezolut)
ax.set_ylabel('Vertical Axis (%s)' % rezolut)
ax.set_zlabel('Depth (%s)' % rezolut)
```

```
ax.set_title('3D Depth Perception Imaging of a %s at (%s) Resolution' % (obj,rezolut))
ax.set_zlim(col[0],maxObjDist)
ax.set_ylim(0,yG_lim)
plt.show()

gofigor = plt.figure()
plt.imshow(image , vmin=col[0], vmax=col[1] , origin = "lower")
plt.colorbar
plt.ylim(0,yG_lim)
plt.title('2D Depth Perception Imaging of a %s at (%s) Resolution' % (obj,rezolut) , fon
plt.xlabel('Horizontal Axis (%s)' % rezolut , fontsize = 12)
plt.ylabel('Vertical Axis (%s)' % rezolut , fontsize = 12)
plt.text(xyLim[1]-(19.7-6)*adjust,yG_lim-2.1*adjust , "Object distance = %scm" % dist ,

igor = plt.figure()
axel = Axes3D(igor)
axel.scatter(xdat,ydat,image,marker = '.')
axel.set_xlabel('Horizontal Axis (%s)' % rezolut)
axel.set_ylabel('Vertical Axis (%s)' % rezolut)
axel.set_zlabel('Depth (%s)' % rezolut)
axel.set_title('3D Depth Perception Imaging of a %s at (%s) Resolution' % (obj,rezolut))
axel.set_zlim(col[0],over)
axel.set_ylim(0,120)

# END PLOTTING
```

Additionally, if one mapping run was deemed insufficient, subsequent ones could be taken and then combined to form a more complete image:

```
import numpy as np
```

```
data1 = np.loadtxt('data1.csv' , delimiter = ',')
data2 = np.loadtxt('data2.csv' , delimiter = ',')
xyLim = [len(data1) , len(data1[0])]
data3 = np.loadtxt('data3.csv' , delimiter = ',')
xyLim = [len(data1) , len(data1[0])]
image = data1

for i in range(xyLim[0]):
    for q in range(xyLim[1]):

        if (image[i][q] == 0 and (data2[i][q] > 0 and data2[i][q] < 400)):
            image[i][q] = data2[i][q]
        elif (image[i][q] == 0 and (data3[i][q] > 0 and data3[i][q] < 400)):
            image[i][q] = data3[i][q]

np.savetxt('combinedData.csv' , image , delimiter = ',')
```

D Arduino Code Used

```
const int trigPinX = A0;
const int echoPinX = A1;
const int trigPinY = A2;
const int echoPinY = A3;
const int trigPinZ = A4;
const int echoPinZ = A5;

void setup(){
  Serial.begin(9600);
}
void loop(){
```

```
long tymeX, tymeY, tymeZ;

pinMode(trigPinX, OUTPUT);
digitalWrite(trigPinX, LOW);
delayMicroseconds(2);
digitalWrite(trigPinX, HIGH);
delayMicroseconds(10);
digitalWrite(trigPinX, LOW);
pinMode(echoPinX, INPUT);
tymeX = pulseIn(echoPinX, HIGH);

pinMode(trigPinY, OUTPUT);
digitalWrite(trigPinY, LOW);
delayMicroseconds(2);
digitalWrite(trigPinY, HIGH);
delayMicroseconds(10);
digitalWrite(trigPinY, LOW);
pinMode(echoPinY, INPUT);
tymeY = pulseIn(echoPinY, HIGH);

pinMode(trigPinZ, OUTPUT);
digitalWrite(trigPinZ, LOW);
delayMicroseconds(2);
digitalWrite(trigPinZ, HIGH);
delayMicroseconds(10);
digitalWrite(trigPinZ, LOW);
pinMode(echoPinZ, INPUT);
tymeZ = pulseIn(echoPinZ, HIGH);

//Serial.println(String(microsecondsTo2Cm(tymeX))+' '+String(microsecondsTo2Cm(tymeY
```

```
    //Serial.println(String(microsecondsToCm(tymeX))+' '+String(microsecondsToCm(tymeY))
    //Serial.println(String(microsecondsToHalfCm(tymeX))+' '+String(microsecondsToHalfCm
    Serial.println(String(microsecondsToQuarterCm(tymeX))+' '+String(microsecondsToQuart
    delay(5); \\ sometimes used 10 or 15 millisecond delays
}

int microsecondsTo2Cm(long microseconds){
  return microseconds / 29.017 / 4.0;
}

int microsecondsToCm(long microseconds){
  return microseconds / 29.017 / 2.0;
}

int microsecondsToHalfCm(long microseconds){
  return microseconds / 29.017;
}

int microsecondsToQuarterCm(long microseconds){
  return microseconds / 29.017 * 2;
}
```