EEE3097S: Final Report
2D Ranging and Direction Finding using SONAR

Thomas Gwasira (GWSTHO001)
Jonah Swain (SWNJON003)
Callum Tilbury (TLBCAL002)
Justin Wylie (WYLJUS002)

October 16, 2019

Contents

Plagiarism Declaration

1. We know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. We have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this project report from the work(s) of other people, has been attributed and has been cited and referenced.
3. This project report is the collective work of the team members of this group.
4. We have not allowed, and will not allow, anyone to copy our work with the intention of passing it off as their own work or part thereof.

Signed:

Thomas Gwasira

Thomas Gwasira

Jonah Swain

Jonah Swain

Callum Tilbury

Callum Tilbury

Justin Wylie

Justin Wylie

1 Introduction

1.1 Background

SOund NAvigation Ranging (SONAR) is a mature and useful technology that has been around for many years. Of course, much before any *human* used sound waves to detect objects, there were plenty of animals doing so – a process termed 'echolocation'. Examples of this are certain species of bats and dolphins.

Electronic SONAR developed in the early 1900s, and much research came out of the World Wars. Today, the technology is still common in maritime applications, self-driving cars, and more. [3]

1.2 Theoretical Foundations

A one-directional SONAR ranging system is fairly easy to understand: sound waves are emitted by a transmitter (TX) and interact with objects in a scene. The waves bounce off the objects, and these 'echoes' are detected at a receiver (RX) after a certain time delay. This time delay is directly related to the distance that the object is from the receiver, due to sound travelling at a relatively constant speed in a given medium. The essence of this technology is depicted in figure 1.

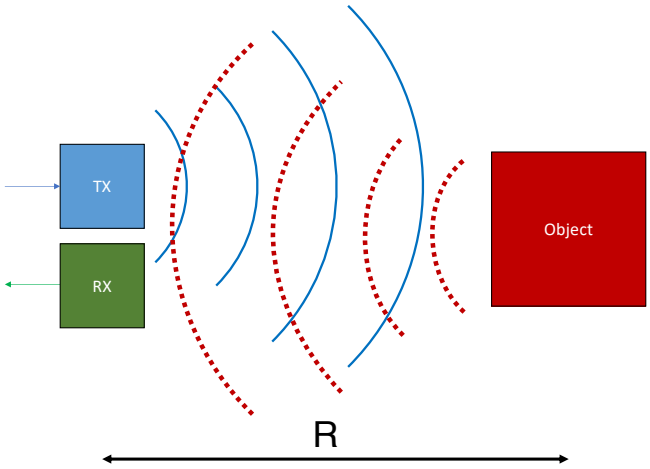


Figure 1: Conceptual diagram of one-dimensional ('ranging') SONAR [Authors' Own Diagram]

Expanding this system to measure so-called 'angle-of-arrival' (and hence angular position of the detected object) requires additional theory. Consider a conceptual diagram of the situation in figure 2.

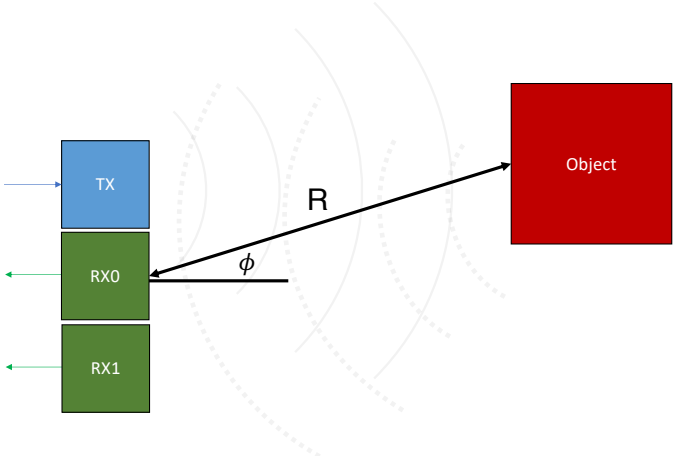


Figure 2: Conceptual diagram of SONAR with ranging and angle-of-arrival capability [Authors' Own Diagram]

Observe more closely what is happening at the two receivers in figure 3.

By definition, on a single 'wavefront' – depicted as a dotted-line – the signals are the same. At an instance in time, t_1, receiver RX0 detects the object's echo, denoted f_A. Receiver RX1, however, detects a *different* signal at this time, f_B. Only some time later, $t_2 = t_1 + \Delta t$, does RX1 detect f_A. Notice that signals f_A and f_B are actually identical, other than a *phase shift* – a delay in time. This shift is caused by the additional distance that an echo must travel to arrive at RX1, when compared to RX0. Importantly, this distance, and corresponding phase shift, is affected by the angle at which the object lies relative to the receivers, denoted in figure 2 as ϕ.

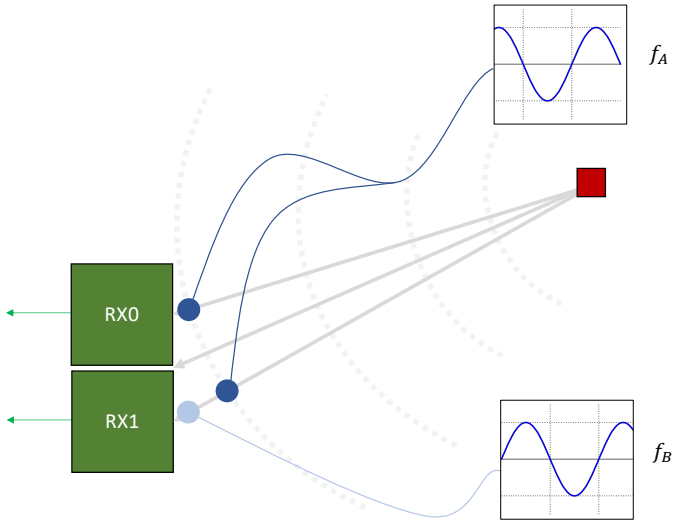


Figure 3: Conceptual diagram of SONAR Receivers [Authors' Own Diagram]

Suppose the phase angles of the signals at the two receivers are θ_0 and θ_1 respectively, and the distance between the receivers is d. It can then be shown that:

$$\theta_1 - \theta_0 \approx 2\pi \frac{d \sin \phi}{\lambda} \quad \Longrightarrow \quad \phi \approx \arcsin \left[\frac{\lambda(\theta_1 - \theta_0)}{2\pi d} \right] \tag{1}$$

where λ is the wavelength of the signal.

Hence, the angle-of-arrival can be determined. In practice, there exists a problem with 'ambiguous' angles, where multiple angles result in the same phase difference used in equation 1. Further investigation into this is outside the scope of this report; instead, a user should simply be told the angles over which the system is correct.

In addition to the angle-of-arrival calculation, the range calculation can be performed as before. The result is a point defined in polar co-ordinates, (R, ϕ), which can easily be displayed graphically.

In general, multiple targets can be detected provided their respective ranges are unique. If two objects are at the same range yet at different angles, their received echoes will overlap, and the resulting phase measurements will be distorted. The system would instead plot a *single* point at the incorrect angle, yet at the correct distance.

1.3 Project Deliverables

The deliverables of this project consisted primarily of two parts: a working 2D SONAR device capable of ranging and angle-of-arrival detection (termed 'direction finding'); and a detailed report outlining the design, implementation and results of this device. The latter requirement is fulfilled by this document. Additional 'milestones' were assigned during the course of the project which involved computer simulations, calculations, and a 1D range-finding prototype device.

1.4 System Requirements

Active SONAR imaging encompasses a wide range of ideas, use cases, and scenarios. This poses the need to clearly define the goals of the system as well as certain rules and parameters. For the project being undertaken, outlined below are details of the user requirements and technical specifications.

1.4.1 User Requirements

The following user requirements were to be satisfied:

UR1. The system must detect targets within the region over which the imaging is to be done.

To test this, an analysis of the data obtained from running the scan would have to be done. Certain data elements (which map to the targets placed in the environment) should significantly deviate from the others. Actually determining whether this data is correct would be better tested in UR2; therefore, simply noticing these deviations would constitute a pass.

UR2. The system must allow for visualisation of information about location of all the reflectors in the scene relative to location of the receiver i.e. an plot of the position of targets must be obtained.

To test this requirement, the scanning process would need to be done and a plot of the position of the targets generated. Their location must be correctly represented for the test to be rendered a pass.

UR3. The SONAR image scanner must require no knowledge of 'what's under the hood' to use. The user must be able to seamlessly run the scan and obtain the result through use of an/some interface/s without having any programming knowledge.

To test this requirement, three or more random testing candidates (preferably without any knowledge of programming) should be asked to perform a scan of a region with very minimal prior briefing. A pass for this test would constitute the user being able to successfully perform the scan and obtaining a result without having to write any code.

UR4. The SONAR image scanner must not be too expensive and the overall cost of the project must be around ZAR2 000.00.

To test this, a breakdown of the cost of the whole project would have to be done and a pass for this test would be if the total expenditure was indeed ZAR2 000.00 +/- 500.00.

1.4.2 Technical Specifications

The following technical specifications were to be satisfied:

TS1. Image Specifications
The SONAR imaging system locate targets in an indoor environment for:

- **TS1.1.** range of up to 10m with an angular resolution $\leq$ 10 degrees
- **TS1.2.** range resolution: $\leq$ 10 cm
- **TS1.3.** minimum field of view: 60 degrees in horizontal plane.

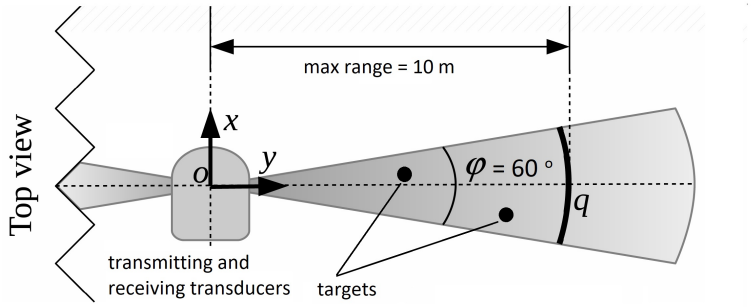


Figure 4: SONAR Imaging System Setup

To test whether these specifications are met, an empirical analysis of the system's behaviour would have to be carried out through experimenting with the system to see how it responds and if the image generated satisfies the above criteria. A pass for this test would be each of the sub-specifications of TS1 being met within experimental uncertainty.

TS2. Signal Parameters
The SONAR imaging system must use a 40 kHz (center frequency) chirp pulse with a bandwidth of 4 kHz to drive a set of ultrasonic transducers.

To test this specification, the signal used to run the SONAR scan must be strictly monitored and ensured to be within the stipulated values. The test is only passed if the SONAR imaging system behaves as expected (i.e. the correct image can be generated) using this signal.

TS3. User Interface
The user interface of the system must comprise of a GUI integrated with plotting tools such as Matplotlib or Julia.

To test this, the plot generated using the SONAR imaging system must be analysed. It must correctly map all the targets in the scene to the image generated.

TS4. Hardware Specifications
The following hardware elements must be used:

- **TS4.1** Micro-controller (Teensy or STM32)
- **TS4.2** Ultrasonic transducers

To test this specification, an audit of the hardware used should be carried out and compared to the above list. The hardware specifications may evolve if an alternative piece of hardware that perhaps does the job better or is cheaper without compromising performance can be attained. A pass for the test is if the above list is matched.

2 Design

2.1 Approaches

Naturally, for every challenge there are a variety of approaches towards a possible solution. Each approach may have benefits over the others, along with specific weaknesses. Following a particular approach is extremely context-specific, and a simple prescription cannot work. Instead, a designer must consider all the factors at hand, and decide appropriately.

Table 1 depicts a summarised list of *some* of the possible problem solving techniques, along with each one's pros and cons.

Problem Solving Theme	**Overview**	**Pros**	**Cons**
Purely Analytical	Thinking about problems mathematically; considering the theoretical implications of requirements	Rigid, well-defined, neat solutions	Unrealistic due to non-linearities
Purely Experimental	'Build, test, repeat', continue this until working solution is found	Learning about system dynamics as one goes, faster time to first idea	Possibly longer time to successful idea, can miss important caveats
Wildly Creative	Out of the box thinking, constantly reframing project to come up with new ideas	Fresh/unique ideas due to alternative perspectives	Untested solutions that may not work or be effective in solving the problem
Tried-and-tested	Follow the steps of other projects, drawing inspiration from existing methods and tools	Reliable, safer	Unimaginative, unable to drastically improve existing solutions
Hybrid Approach	Balance between aforementioned tools: a weighted mix of the theoretical and the empirical, of existing knowledge and crazy new ideas	Gaining multiple, diverse insights into the same problem, allowing for a more robust yet creative solution	Not as specialized as a single approach

Table 1: Table showing some of the possible problem solving themes one can embrace when approaching a challenge

In the context of *this* project, the team decided to follow the **hybrid** approach. Given the time constraints that were in place, following a purely experimental route would have been nonsensical. And yet, considering the large practical ('real-world') component of the project, embracing a purely analytical route would have resulted in frustrating errors. Encouraged to present a unique offering, we chose not follow a rigid, pre-defined, tried-and-tested solution. However, bearing in mind the level of expertise of the team (3rd year students), as well as the relatively complex task at hand, simply being wildly creative would have been silly.

Hence, the hybrid approach was ideal. By balancing a theoretical foundation with iterative prototyping and testing in the laboratories, the team was able to achieve a fairly decent solution. Knowledge was drawn from external sources — from existing literature, from the knowledge of the course lecturer – A/Prof. Wilkinson, and from the internet. However, the team nevertheless offered a unique perspective to the project.

2.2 System Description

There are a variety of ways to understand- and convey information about a particular system. Due to their clarity and ubiquity, UML diagrams describing the system design are given below.

2.2.1 Use-Case Diagram

A use-case diagram is a simple yet useful way of presenting information about a user's possible interactions with a system, and the results of such actions. This diagram is intended to be a high-level overview, easy to access and painless to understand. Figure 5 shows a use-case diagram for this system.

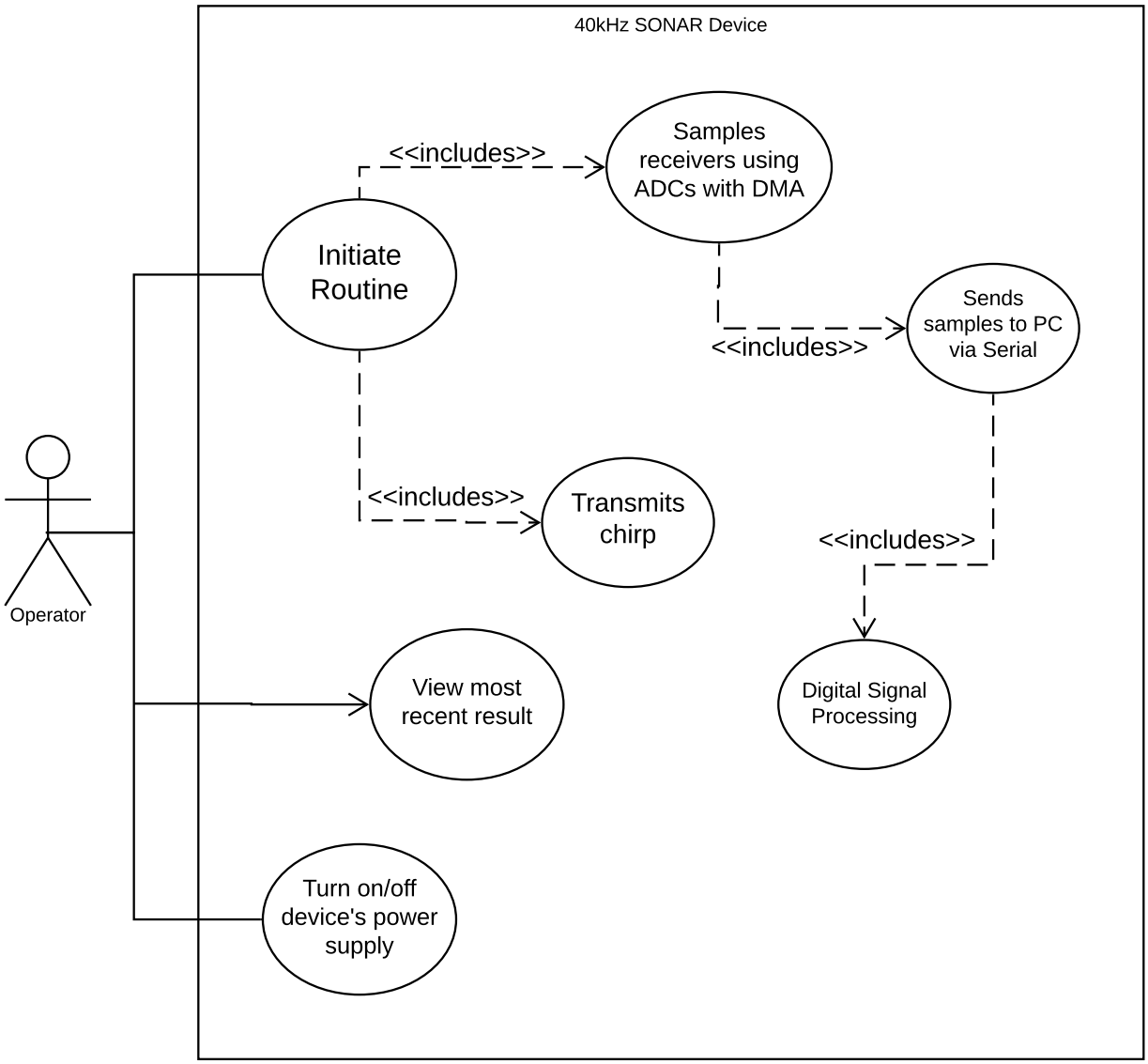


Figure 5: Use-case diagram for this SONAR imaging system

Notice that the user's interaction with the system is fairly simple. After turning the device's power on, they can 'initiate' the chirp-sample-process routine. Once this has completed, they can view the result. For convenience, this can be done in a loop for continuous monitoring.

2.2.2 Sequence Diagram

A sequence diagram is a form of graphical communication which displays the sequential activities that occur between various objects within a system. *Actors* are able to send *messages* to various *lifelines* – elements within the system, and thus the flow of data can be depicted. Various other information is displayed within the diagram, including the lifecycle of each lifeline, as well as the a/synchronous nature of all messages. Figure 6 shows the sequence diagram for the proposed SONAR imaging system.

The diagram shows a high-level activity timeline for a single-shot recording. Firstly, the user opens the Julia IDE, and indicates runs the script. This indicates to the Teensy microcontroller (MCU) via Serial that it should begin the routine.

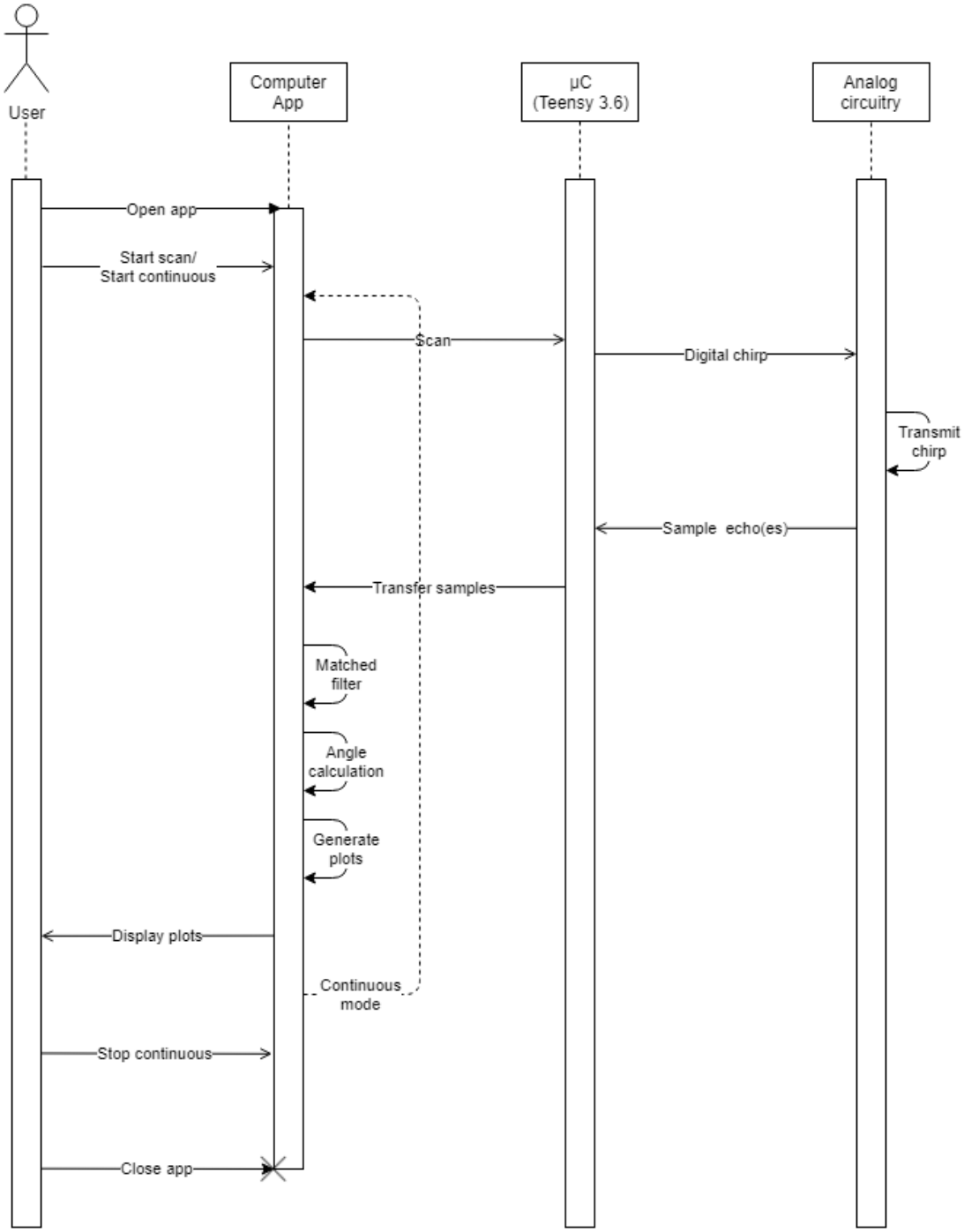


Figure 6: Sequence Diagram for the SONAR imaging system

The MCU generates a chirp pulse, which it sends to the analog circuitry driving the transmitting transducer/s. The ultrasonic chirp wave moves through space and interacts with objects in its path. Various reflections occur, and echo(es) are detected by the receiving transducer/s. The received signal is amplified in the analog circuitry, and is then sampled by the MCU. These samples are transmitted back to the computer via Serial, for digital signal processing (DSP). Finally, the result is displayed to the user.

Note that the process can run in a 'continuous mode', where the chirp-scanning-processing routine is simply repeated at the desired refresh rate.

Finally, if the user wishes the close the app, they may do so on the Julia IDE.

2.2.3 Activity Diagram

An activity diagram is important tool that enables depiction of the dynamic nature of a system, as one moves through various 'activities'. Various processes are identified and control-statements (if/else) are placed between them. This diagram captures the essence of *flow* within a system.

The interaction of subsystems (from the starting a SONAR scan session to obtaining a plot displayed onscreen) is illustrated in the activity diagram in figure 7.

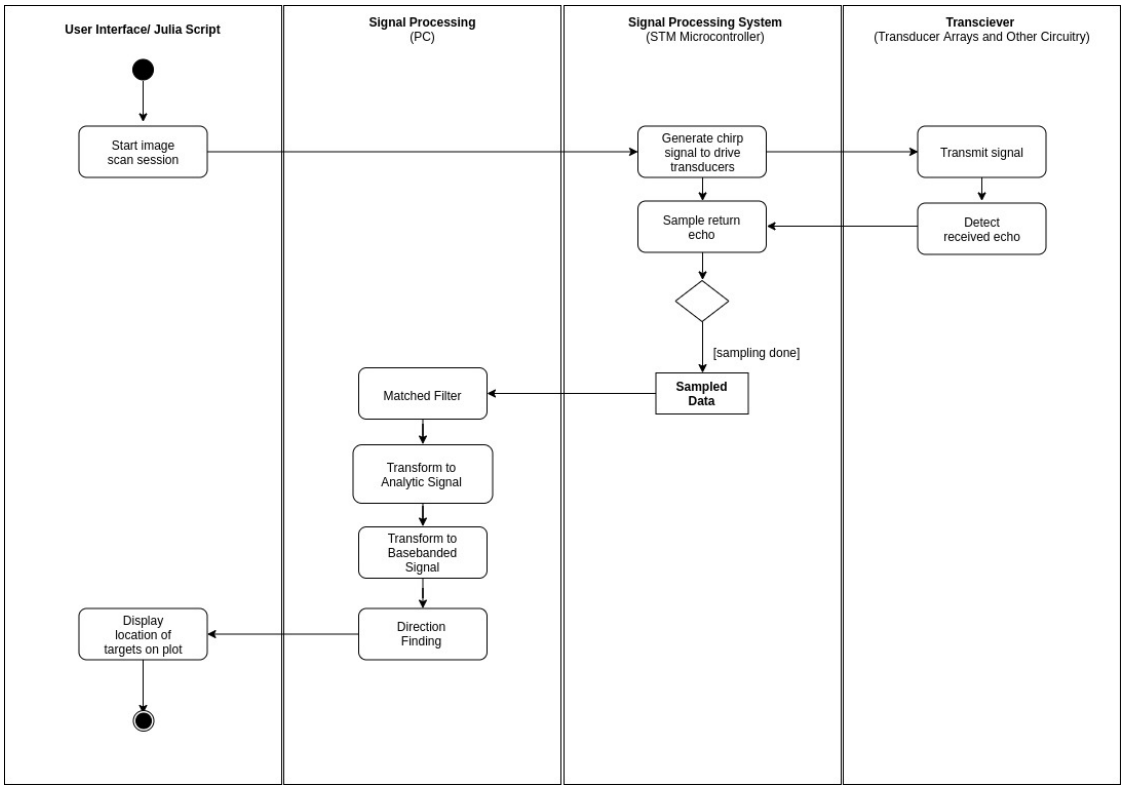


Figure 7: Activity diagram for the SONAR imaging system

The process starts off with the user triggering a scanning session from some kind of user interface or script. Subsequently, the microcontroller generates a chirp pulse transmitted by the transmitting transducers. As the signal reflects off different elements within the region, the receiving transducers pickup the reflected echoes which are sampled by the microcontroller, and this data is sent via Serial to a PC. On the PC, the digital signal processing (DSP) occurs using the Julia programming language, and the results are plotted using PyPlot.

2.3 System Block Diagram

The block diagram (figure 8) is an extremely important and useful tool in engineering design. It is a high level description of a system. It allows the system as a whole to be broken into modular parts which may each be considered a subsystem or a 'block'. Blocks in the block diagram have the ability to communicate with other blocks, this allows the designer in charge of a specific block to be aware of what his subsystem must take as an input and the required output.

Figure 8 shows the block diagram for the ultrasonic sonar system. At the heart of the system is the Teensy 3.6 micro-controller. This block is responsible for generating the digital chirp to drive transmitting transducer, processing incoming signals from the two receiving transducers and to transfer the received waveform data to the PC for digital signal processing (DSP). Communication with the PC is two way, allowing the PC to signal when to take measurements. The Teensy samples through the two built in ADCs, allowing simultaneous sampling of both receivers using DMA.

The transmitter and receiver transducer blocks consist of one and two transducers respectively. These are used to physically generate and receive the chirp.

The receiver amplifier and signal conditioning block is where the signal present at the terminals of the receiving transducers is amplified and adjusted to meet the requirements of the Teensy's ADC interface. The workings of this block are explained in more detail in the hardware section. The output of this block is terminated at the Teensy board and is interfaced by the on board ADC peripheral.

The Teensy board generates a digital chirp signal via its GPIO pins, this is processed by the transducer driver circuit block. This block is responsible for taking the signle supply, low voltage square wave chirp and generating a 8V peak to peak chirp waveform capable of driving the transmitting transducer. This stage is discussed in more detail in the hardware section.

The PC block handles the transmission and retrieval of data from the DSP and GUI applications on the PC and the Teensy board. The Teensy board communications occur over USB, through a serial interface. The DSP application performs the signal processing and then outputs the data to be displayed. A separate display application then provides the user with a graphical plot of the scanned environment. At the time of the system demonstration, time constraints prevented us from fixing issues with the GUI interfacing with the signal processing Julia code and as a result Julia was used to generate the plots.

The digital signal processing block is responsible for receiving the raw ADC samples, processing this data and into objects in a 2D plane with a radial distance and angles. This data is then transferred to the GUI display application.

The GUI visualization/display application provides the user with buttons to interface with the system. These buttons allow the user to perform actions like taking a single shot of the scene or running a continuous scan. The GUI is also responsible for retrieving the detected objects and displaying these on a 2D plot.

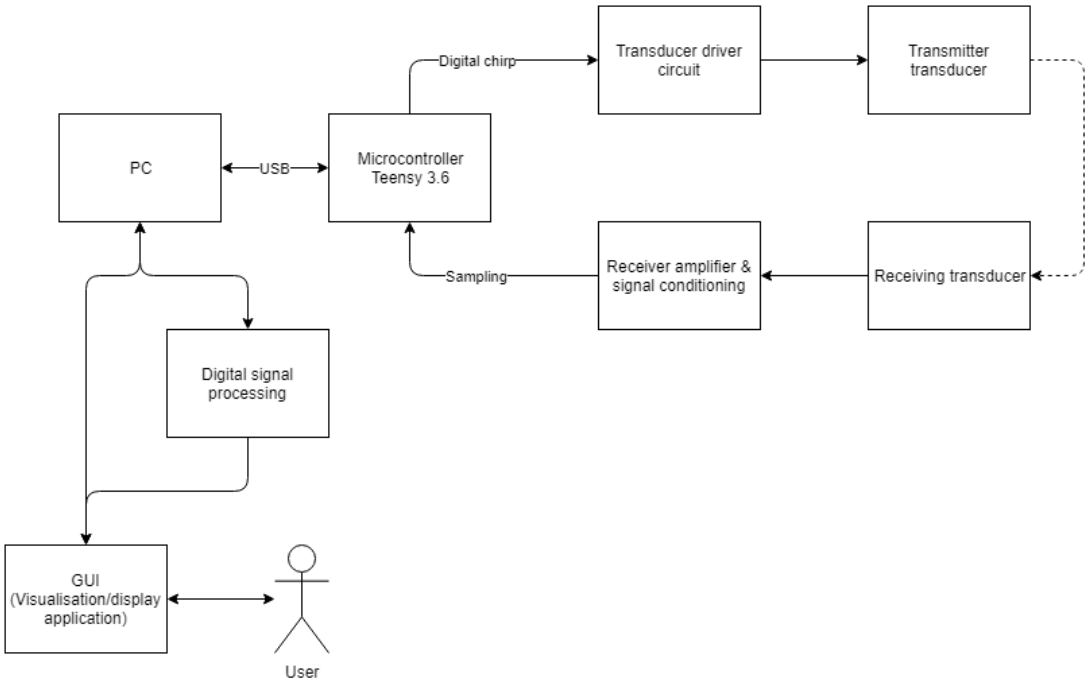


Figure 8: Block diagram of the SONAR imaging system

3 Implementation

3.1 Sub-system: Analog Circuitry

3.1.1 Transmission Block

The transmission circuity is based around the 74HC54 logic chip. This chip offers 8 not gates of which 7 are utilized in the circuit. The 74HC54 is powered with 4V, which is supplied by the positive supply rail. The digital drive signal is asserted by pin 20 of the teensy board.

To ensure the transducer does not exceed the rated current of any individual output pin of the 74HC54, the transducer is driven by three not gates in parallel on each of its inputs.

This driving circuit allows the transmitting transducer to be driven with a peak to peak voltage of 8 volts. This achieved by inverting the logic value being asserted by the teensy board at the input of the first set of not gates and applying this inverted signal as the input to the second set of not gates which are used to drive the other leg of the transducer.

This configuration is such that when the Teensy asserts a high signal, the transducer will receive 4V on one of its terminals and ground on the other. Conversely when the Teensy asserts a low, the leg of the transducer that was receiving 4V is grounded and the other leg is powered with 4V.

The schematic for this circuitry is shown in figure 9.

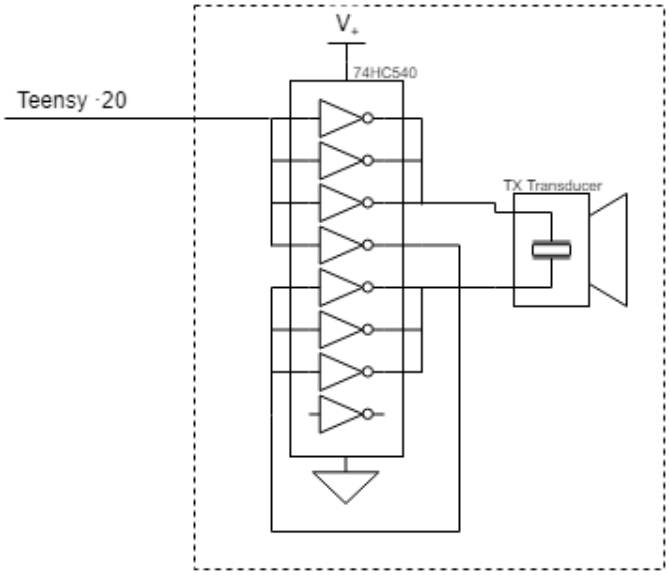


Figure 9: Circuit schematic of transmitter components

3.1.2 Power Supply Block

The entire system (excluding the Teensy board) is powered by a single 9V supply. This conveniently allows a 9V battery to be used as the power supply. The system requires three voltage supplies, namely a 4V, 3.3V and -5V supply.

This is achieved using a 5V regulator and a 3.3V regulator. The 9V supply is placed across the 5V regulator, and the output of the 5V regulator is defined as ground (0V), this provides a supply rail of 4V and -5V. The 4V supply rail is then used by the 3.3V regulator to produce a voltage of 3.3V.

The schematic for this circuitry is shown in figure 10.

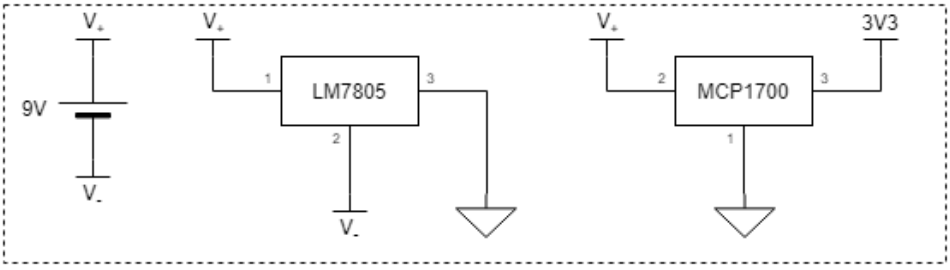


Figure 10: Circuit schematic of power supply components

3.1.3 Decoupling

The entire system draws a quiescent current of 10mA. However during the transmission of a chip there is a large spike in demand for current. This current demand comes mostly from the transmission block. To ensure sufficient current is delivered to the transmission block a 100uF decoupling capacitor is present on the 4V line.

The transmission circuity operates at 40kHz and the generated chirp is in the form of a square wave. Square waves contain high frequencies at their transitions, because of this it was important to consider the inductive characteristic of the electrolytic capacitor which act to reduce the capacitance at higher frequencies. To reduce this side effect of the electrolytic capacitor, three 1uF capacitors are placed on the 4V supply rail. When not using any decoupling a large amount of ringing was present in the chirp signal present at the terminals of the transmitting transducer, when decoupling was implemented the ringing was radically reduced.

The 3.3V rails is generated through a 3.3V linear regulator. This rail is used to clip the amplified, received signal before it is sampled by the ADC. Linear regulators are not typically used to sink current, but rather to supply current. To

ensure that any brief spikes above 3.3V can be sunk into the 3.3V rail, a large 100uF decoupling capacitor was placed across the supply rail and ground.

The schematic for this circuitry is shown in figure 11.

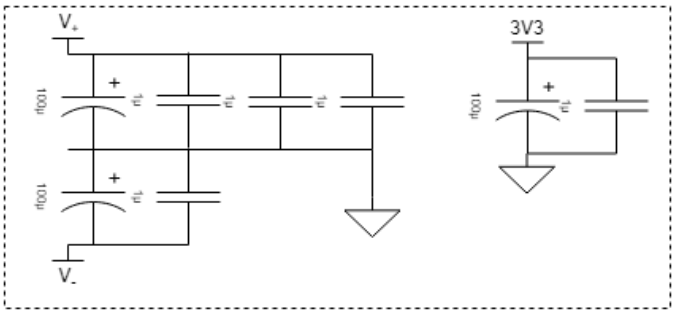


Figure 11: Circuit schematic of power decoupling components

3.1.4 Reference Voltage

The receiving circuity utilizes a -1.65V source to add an offset to the amplified, received signal. This reference is generated by using a potentiometer, allowing for fine adjustment. The high impedance of the op-amp circuitry ensures that loading is negligible.

The schematic for this circuitry is shown in figure 12.

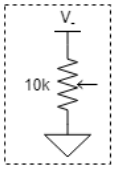


Figure 12: Circuit schematic of -1.65V reference components

3.1.5 Receiving Block

The system contains two receiving blocks, both blocks are identical. The output of the first block is connected to pin A1 and the output of the second block is connected to pin A16.

The receiving block contains two operational amplifier stages. The first stage is an inverting amplifying stage. This stage has a gain of 33 and phase shifts the signal by 180°. The second stage is responsible for offsetting the signal.

The Teensy ADC has an input range of 0V to 3.3V, but the amplified, received signal is centered about 0V. To sample both the negative and positive part of the time domain signal, an offset of 1.65 volts was implemented using a inverting summing amplifier stage. The amplified and inverted signal is summed with the -1.65V reference signal to produce an amplified version of the input waveform. This final waveform is in-phase with the received signal but offset by 1.65V and increased in magnitude by 30dB.

The last part of the receiver block is a pair of clipping diodes, these diodes ensure that the Teensy is not exposed to a voltage outside the input range of the ADC. This is achieved by placing a schottky diode between the signal and the 3.3V rail and another between the signal and the ground rail. The 1kΩ resistor ensures the voltage excess voltage is dropped before it reaches the Teensy board.

The schematic for this circuitry is shown in figure 13.

3.2 Sub-system: Microcontroller

The microcontroller acts as the interface between the digital domain and the real world, sitting between the PC and the analog circuitry. The chosen microcontroller for this project was the Teensy 3.6 development board. The responsiblities of the microcontroller are to generate the outgoing chirp pulse, sample the incoming signal (echoes), transfer the sampled signal to the PC, and provide status feedback to the user. These requirements were achieved using code written in the Arduino environment. Much of the code was adapted from that provided by Jonathan Whitaker. Important snippets of the code and explanations can be found below.

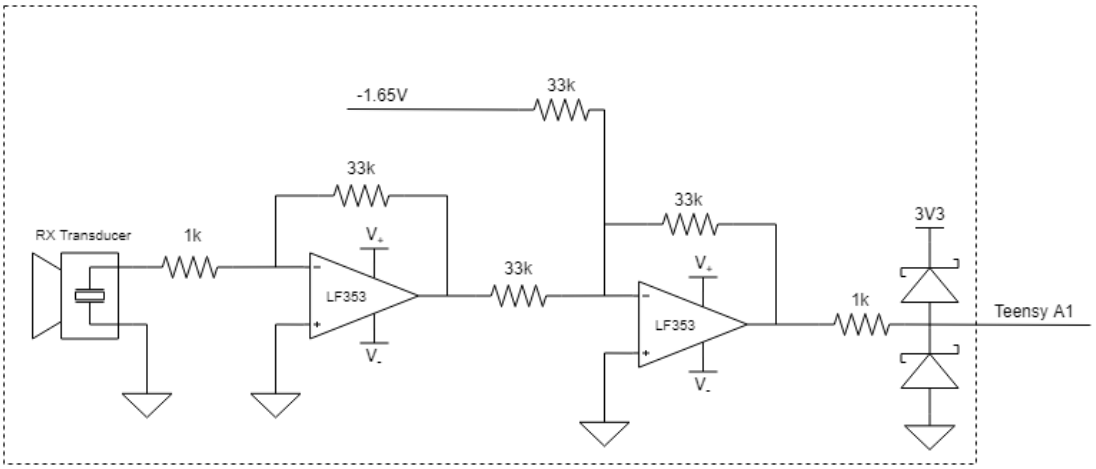


Figure 13: Circuit schematic of receiver components

3.2.1 Chirp pulse

The outgoing signal was chosen to be a digital 'chirp' pulse. This decision was made because there are only two digital to analog converters (DACs) on the Teensy microcontroller, and hence a maximum of two analog chirp pulses could be generated. Since the original project entailed beam-forming using many output channels, an analog chirp pulse using the DACs would have been unsuitable.
The digital chirp pulse was implemented as a frequency stepped digital chirp pulse using the Arduino Tone library.

```
const int tx = 20; // Transmitter pin number
for (int f = 38000; f <= 42000; f += 1000) {
    tone(tx, f);
    delay(1);
}
noTone(tx);
```

As seen in the above code snippet, the frequency is stepped from 38kHz to 42kHz in increments of 1kHz, with each frequency step being transmitted for 1ms. The details of the implementation of the tone() function are abstracted by Arduino, however it does use a timer to generate the output, making it more robust than a simple CPU delay.
Note that whilst a digital chirp pulse contains many harmonics of the fundamental frequencies, the band-pass properties of the ultrasonic transducers ensure that actual transmitted harmonics are significantly attenuated. Additionally, whilst it may seem like having a stepped chirp contains only five distinct frequency components, using the FFT function of an oscilloscope verifies that the transmitted signal frequency spectrum does in fact resemble that of a chirp pulse - this is illustrated by the figure below.

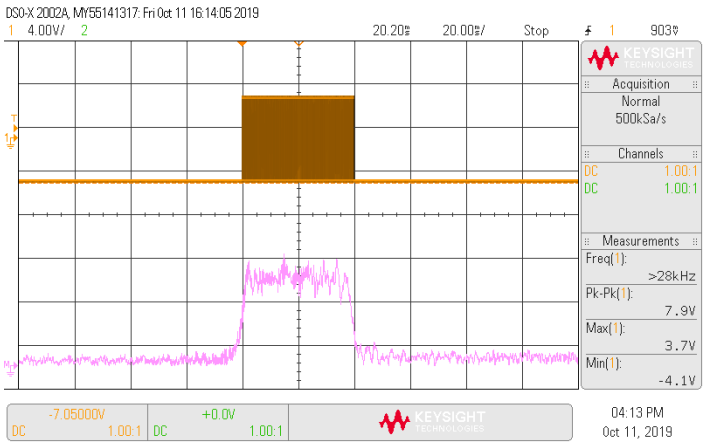


Figure 14: Frequency spectrum (Bounds: 20-60kHz) of stepped digital chirp pulse (pink)

3.2.2 Analog to digital converter (ADC) set-up

The two incoming signals (from the two receiving transducers) were sampled using the two analog to digital converters (ADCs) on the Teensy microcontroller. The ADCs were configured identically, in the manner described below. Note that only the configuration of a single ADC is shown, for simplicity.

Firstly, the memory to be used for the sample buffer is declared and initialized to zero, as shown in the code excerpt below:

```
#define SAMPLES 18000
DMAMEM static uint16_t adc_buf_0[SAMPLES]; // buffer for ADC0
memset((void*)adc_buf_0, 0, sizeof(adc_buf_0)); // clear ADC0 buffer
```

Note that the DMAMEM keyword is used to ensure that buffer is placed in a contiguous block of memory, in an area accessible by the Teensy's Direct Memory Access (DMA) controller. The static keyword is used to declare that the buffer is an instance/method independent variable.
The memset() command writes a specific value (0 in this instance) to every byte in a block of memory.
Next, the operating parameters of the ADC are configured, as shown in the code excerpts below:

ADC PGA Gain configuration

```
#define SAMPLING_GAIN 1
uint8_t sgain = SAMPLING_GAIN;
if (sgain >1) {
adc->enablePGA(sgain, ADC_0);
} else {
adc->disablePGA(ADC_0);
}
```

In the above excerpt, the gain of the Teensy's internal Programmable Gain Amplifier (PGA) is set to 1 (PGA disabled). The PGA is not used because the signal is pre-amplified by the analog circuitry.

ADC Reference voltage configuration

```
ADC_REFERENCE Vref = ADC_REFERENCE::REF_3V3;
adc->setReference(Vref, ADC_0);
```

In the above excerpt, the reference voltage of the ADC is set to the Teensy's internal 3.3V rail (supplied by the onboard voltage regulator).

ADC Oversampling configuration

```
#define SAMPLE_AVERAGING 0
uint8_t aver = SAMPLE_AVERAGING;
adc->setAveraging(aver, ADC_0);
```

The Teensy's ADC has optional oversampling functionality, which, if enabled, performs multiple conversions, and averages them together for a higher effective resolution, at the expense of a lower effective sample rate. In the above code excerpt, the oversampling (averaging) functionality of the ADC is disabled, because it is not required for this sonar application.

ADC Resolution configuration

```
#define SAMPLE_RESOLUTION 12
uint8_t res = SAMPLE_RESOLUTION;
adc->setResolution(res, ADC_0);
```

In the above code excerpt, the resolution of the ADC is configured to 12 bits. A 12 bit resolution was shown to be sufficient for this sonar application through experimentation. Additionally, using a resolution lower than the maximum (16 bits) allows for a higher conversion rate to be used. This is a feature of successive approximation register (SAR) ADCs, in which there is a trade-off between conversionn speed and resolution.

ADC Sampling speed configuration

```
ADC_SAMPLING_SPEED samp_speed = ADC_SAMPLING_SPEED::VERY_HIGH_SPEED;
adc->setSamplingSpeed(samp_speed, ADC_0);
```

In the above code excerpt, the ADC's sampling speed is set to its maximum (VERY_HIGH_SPEED). This equates to the minimum sampling time. Using a shorter sampling time allows for a higher overall sample rate, however a longer sampling time is required for higher impedance inputs (because the sample/hold circuit forms an RC circuit with the

input impedance, which, with high impedance inputs, results in long charge times). Since the input from the analog circuitry is buffered, and has a relatively low (1k) input impedance, the shortest sampling time can safely be used.

ADC Conversion speed configuration

```
ADC_CONVERSION_SPEED conv_speed = ADC_CONVERSION_SPEED::VERY_HIGH_SPEED;
adc->setConversionSpeed(conv_speed, ADC_0);
```

The Teensy's ADCs have configurable conversion speeds, with the lower speeds consuming less power. Hence for a low power application, lower conversion speeds can be used. Since the sonar application does not have a power constraint, the ADC sampling speed is configured to its maximum, as shown in the above code excerpt.

ADC Compare function configuration

```
float Vmax = 3.3;
if (((Vref == ADC_REFERENCE::REF_3V3) && (Vmax > 3.29)) || ((Vref == ADC_REFERENCE::REF_1V2) && (Vmax > 1.19))) {
    adc->disableCompare(ADC_0);
} else if (Vref == ADC_REFERENCE::REF_3V3) {
    adc->enableCompare(Vmax/3.3*adc->getMaxValue(ADC_0), 0, ADC_0);
} else if (Vref == ADC_REFERENCE::REF_1V2) {
    adc->enableCompare(Vmax/1.2*adc->getMaxValue(ADC_0), 0, ADC_0);
}
```

The Teensy's ADCs have the functionality to compare the converted value to a set value, and generate an interrupt depending on the result of the comparison. An example application for this functionality would be battery voltage monitoring, where if the voltage drops below a certain threshold, an interrupt is triggered to indicate that the battery is low. Since this functionality is of no use to the sonar application, it is disabled, as shown in the code excerpt above.

3.2.3 Direct Memory Access (DMA) Controller set-up

Since the ADC's output register can only hold a single value (the result from a single conversion), it is necessary to copy the result from the output register to a buffer after each conversion. This was done using the Direct Memory Access (DMA) controller, which can copy a value from one location in memory (i.e. the ADC output register) to another (i.e. the buffer) independently of the main processor. Additionally, the DMA controller can be triggered by the ADC's conversion completion. Two DMA channels were used - one for each ADC - and were configured in the same manner, which is shown below:

```
dma0.source((volatile uint16_t&)ADC0_RA);
dma0.destinationBuffer(adc_buf_0, sizeof(adc_buf_0));
dma0.triggerAtHardwareEvent(DMAMUX_SOURCE_ADC0);
dma0.interruptAtCompletion();
dma0.attachInterrupt(&dma0_isr_single);
```

First, the source of the transfer is set to the ADC's output register. Then the destination of the transfer is set to the buffer (as declared previously). The trigger of the DMA channel is set to the ADC's end-of-conversion. Finally, the DMA controller is set to generate an interrupt, calling dma0_isr_single(), when the buffer is full.
The definition of the dma0_isr_single() function is shown below:

```
void dma0_isr_single(void) {
aorb_busy = 0; // clear the busy flag
adc0_full = 1; // set the buffer full flag
dma0.clearInterrupt(); // clear the interrupt
dma0.clearComplete(); // clear the DMA complete flag
}
```

The function of the above code excerpt is to set flags signalling to the rest of the code that the sampling is complete.

3.2.4 Sampling using the ADC and DMA

As previously described, the Teensy's ADCs were used to sample the incoming signals. Each of the ADCs and DMA channels were enabled using the process described below.

```
const int readPin0 = A1; // channel 1 input pin
#define SAMPLE_RATE 300000
uint32_t freq = SAMPLE_RATE; // ADC sample rate

aorb_busy = 1; // set busy flag
adc0_full = 0; // clear buffer full flag
adc->adc0->startSingleRead(readPin0); // call single read to configure channel
adc->adc0->startPDB(freq); // set ADC trigger to required sample rate
```

```
adc->enableDMA(ADC_0); // config/enable DMA for ADC
dma0.enable(); // enable the DMA channel
```

First, the busy flag is set (to tell the rest of the program that sampling is currently in progress), then the buffer full flag is cleared. Thereafter, a call is made to startSingleRead(), which configures the ADC's multiplexer to the channel input pin (A1). The ADC is then configured to be triggered by a timer at a specific frequency (300kHz/300kSps) using the startPDB() function. Thereafter, the ADC is configured to trigger the DMA controller, using the enableDMA() function. Finally, the attached DMA channel is enabled.
The result of this code excerpt is that the ADC samples at 300kSps, and the DMA controller copies the result of the conversion into the channel buffer.
The program then waits for the sampling to complete. This is achieved by waiting for both the buffer full flags to be set, as shown in the code excerpt below:

```
while(!adc0_full || !adc1_full);
```

Once the sampling is complete (and the buffers are full), each ADC and DMA channel is disabled in the following way:

```
PDB0_CH0C1 = 0; // disable ADC trigger
dma0.disable(); // disable DMA channel
adc->disableDMA(ADC_0); // disable ADC DMA triggering
adc->adc0->stopPDB(); // stop ADC repeated sampling (timer)
aorb_busy = 0; // clear busy flag
```

Firstly, the ADC trigger is disabled, to prevent any further conversions from taking place while the ADC is being disabled, then the DMA channel is disabled, to prevent any more data from being copied into the buffer. Thereafter, the ADC is configured to not use/trigger the DMA controller, using the disableDMA() function. The ADC's repeated sampling mode/timer and timer triggering is then disabled, using the stopPDB() function. Finally, the busy flag is cleared, to signal to the rest of the program that sampling is complete.

3.2.5 USB Serial communication with the PC

Once the incoming signals are sampled, as described above, it is necessary to transfer them to a PC, on which the Digital Signal Processing (DSP) algorithms are performed in order to generate the final outputs. This communication with the PC is achieved using USB. The Teensy microcontroller (when paired with the Arduino environment) enumerates as a virtual serial port when connected via USB to the PC. The data is then transferred over the virtual serial port, using various Arduino library functions, via the process described below. The first step is to initiate the serial communication, as shown in the code excerpt below:

```
Serial.begin(9600);
```

This opens the virtual serial port at a *virtual* BAUD rate of 9600bps. Note that since the serial port is emulated, the BAUD rate is for standards adherence only, and actual data transfer occurs faster than 9600bps. The sampled signal data is transferred as a string with each sample in its ASCII hexadecimal representation. The process of conversion from a sample to a hexadecimal string is shown in the code excerpt below:

```
1 uint16_t u = adc_buf_0[x]; // u is the x-th sample
2 byte * b = (byte *) &u; // get u as an array of bytes
3 for(int i=1; i>=0; i--) { // iterate through the two bytes (MSB then LSB - little endian)
4     byte b1 = (b[i] >> 4) & 0x0f; // isolate the upper nibble
5     byte b2 = (b[i] & 0x0f); // isolate the lower nibble
6
7     char c1 = (b1 < 10) ? ('0' + b1) : 'A' + b1 - 10; // convert the upper nibble to a hexadecimal character
8     char c2 = (b2 < 10) ? ('0' + b2) : 'A' + b2 - 10; // convert the lower nibble to a hexadecimal character
9 }
```

The expressions on lines 7 and 8 are each equivalent to the following if statement:

```
if (b1 < 10){
    c1 = '0' + b1; // add the value of the nibble to ASCII 0
} else {
    c1 = 'A' + b1; // add the value of the nibble to ASCII A
}
```

This is done because the capital letters do not follow from the numbers in ASCII text.
Once each sample is represented as a pair of ASCII hexadecimal characters, they are transferred over the virtual serial port to the PC, as shown in the code excerpt below:

```
Serial.print(c1);
Serial.print(c2);
```

This once again uses the Arduino serial library.

The Teensy microcontroller also receives commands via the virtual serial port, to either transmit a chirp pulse or transfer the sample buffer to the PC. This is done in a similar fashion to data transfer to the PC, using the Arduino serial library.

3.2.6 User feedback with RGB LED

In order to provide some feedback to the user as to the status of the microcontroller in all the above processes, a common-anode red-green-blue (RGB) LED was used. The colour of the LED is changed depending on which stage of the process is in progress. Green is used to indicate that the Teensy is idle/ready, red is used to indicate that the Teensy is sending a chirp pulse and/or sampling the incoming signal, and blue is used to indicate that data transfer between the Teensy and the PC is in progress. The process used to control the RGB LED is described below.

```
const int red = 3; // LED red pin number
const int grn = 4; // LED green pin number
const int blu = 5; // LED blue pin number
// set LED pins to output mode
pinMode(red, OUTPUT);
pinMode(grn, OUTPUT);
pinMode(blu, OUTPUT);
// set LED pins to high (LED off) by default
digitalWrite(red, HIGH);
digitalWrite(grn, HIGH);
digitalWrite(blu, HIGH);
```

In the code excerpt above, the LED pins are initialised. First, they are each set to output mode, using the pinMode() function. They are then all set high using the digitalWrite() function. Since the RGB LED is a common-anode type, pulling the relevant cathode pin low allows current to flow, and the section of the LED to light up.
The code used to change the colour of the LED is shown in the excerpt below:

```
void led(int col) {
  if(col == red) {
    digitalWrite(grn,HIGH);
    digitalWrite(blu,HIGH);
    digitalWrite(red,LOW);
  }
   if(col == grn) {
    digitalWrite(red,HIGH);
    digitalWrite(blu,HIGH);
    digitalWrite(grn,LOW);
  }
  if(col == blu) {
    digitalWrite(red,HIGH);
    digitalWrite(grn,HIGH);
    digitalWrite(blu,LOW);
  }
}
```

3.3 Sub-system: Digital Signal Processing

Since the purpose of the sonar system is to show the distance and angle to target(s), something needs to be done to the sampled signals in order to achieve this. Hence digital signal processing (DSP) techniques are used to process the sampled signal and produce the meaningful plots. All the DSP was implemented in the Julia programming language, using the JuliaPro/Juno development environment, based off of Atom. The important elements of the DSP are explained, with accompanying code excerpts below.

3.3.1 USB Serial communication with microcontroller

Before any DSP can be done on the sampled signal, the samples have to be transferred from the Teensy microcontroller to the PC. The Teensy microcontroller (when paired with the Arduino environment) enumerates as a virtual serial port when connected via USB to the PC. The Julia LibSerialPort library is used to handle communication between the program and the Teensy microcontroller.
The LibSerialPort library can be included in the Julia environment by using the following line of code:

```
using LibSerialPort;
```

The first step in obtaining the sampled signal data from the microcontroller is to initiate the serial communication, as shown in the code excerpt below:

```
sp = open("/dev/cu.usbmodem5871330", 9600);
```

This opens the virtual serial port at a *virtual* BAUD rate of 9600bps. Note that since the serial port is emulated, the BAUD rate is for standards adherence only, and actual data transfer occurs faster than 9600bps.
The data from the Teensy is transferred as a string with each sample in its ASCII hexadecimal representation, and the samples from each of the two input channels interleaved. This data is read and parsed by the Julia program as shown in the code excerpt below.

```
dataSize = 18000 # number of samples
d0 = zeros(UInt16,dataSize); # initialise channel 0 data array to zeroes
d1 = zeros(UInt16,dataSize); # initialise channel 1 data array to zeroes
# Load samples into the data arrays over serial
count = 1
while (bytesavailable(sp) > 0) # parse all data in the serial buffer
    d0[count] = parse(Int,readline(sp),base=16); # parse channel 0 sample as hexadecimal
    d1[count] = parse(Int,readline(sp),base=16); # parse channel 1 sample as hexadecimal
    count += 1
end
```

As shown in the excerpt above, the program iterates through the serial buffer, parsing the interleaved samples from hexadecimal strings into integers. LibSerialPort's readline() function reads a line from the buffer, and the parse() function parses it from a string into an integer. The count variable is used to keep track of the position in the buffers.
The Julia program also sends commands to the Teensy over the serial port, to instruct it to transmit a chirp pulse and sample the return signals, or to transfer the recorded samples to the PC. These commands are implemented in the following code excerpt:

```
write(sp, "c") # Tell the Teensy to send a chirp and start sampling
write(sp, "s") # Tell the Teensy to load sampled data into Serial Buffer
```

3.3.2 Removing DC offset

Since the Teensy's ADCs can only operate from 0-3.3V, the analog circuitry was constructed to center the incoming signals about 1.65V (halfway between 0 and 3.3V). This, however, results in the signal having a DC offset of approximately 1.65V (or a converted value of 2048 for a 12 bit ADC). This offset is undesirable for performing DSP as it significantly reduces the signal-to-noise ratio (SNR) at the output of the matched filter. It is hence necessary to remove the DC offset from the sampled signal. This was implemented by taking the discrete fourier transform (DFT)/fast fourier transform (FFT) of the signal, nulling the low frequency components (close to DC), and taking the inverse DFT/FFT to get back to the time domain. This is shown in the code excerpt below:

```
# take FFT of sampled signals
D0 = fft(d0)
D1 = fft(d1)
# remove low frequency (DC) components
for i in 1:5
    D0[i] = 0;
    D1[i] = 0;
end
# take inverse FFT
v_rx0 = ifft(D0);
v_rx1 = ifft(D1);
```

Note that the Fastest Fourier Transform in the West (FFTW) library was used to compute the fourier transforms. This library can be included in the Julia environment with the following line of code:

```
using FFTW;
```

3.3.3 Removing directly coupled chirp

Since the receiving transducers are located physically close to the transmitting transducer, they receive a directly coupled chirp pulse when the chirp pulse is being transmitted. If included, this directly coupled chirp appears as a large amplitude at the output of the matched filter, and can overwhelm the peak detection algorithm (used later for angle of arrival calculation). It is hence necessary to remove this directly coupled chirp from the sampled signal. This is done by nulling all the samples which occur whilst the chirp is being transmitted (≈8ms/2500 samples). This is implemented in the following code excerpt:

```
for i in 1:2500 # iterate through first 2500 samples (~8ms)
    v_rx0[i] = 0 # null (zero) the samples
    v_rx1[i] = 0
end
```

3.3.4 Matched filtering and transmitted signal

In order to be able to identify objects through their echoes, a means of processing the sampled signals (containing the echoes) that accurately identifies the positions of the echoes within the signals is required. A matched filter is used to accomplish this. A matched filter has the following formula:

$$Y(\omega) = H(\omega)X(\omega); \quad H(\omega) = R^*(\omega) \tag{2}$$

where R is the reference signal. It outputs a peak at the time at which the known signal occurs.
In order to implement a matched filter, the transmitted signal needs to be known. The transmitted signal was recorded by placing one of the receiving transducers opposite the transmitting transducer, at a distance of 1 meter, and recording the sampled signal when a chirp pulse was transmitted. The resulting reference/transmitted signal is shown below:

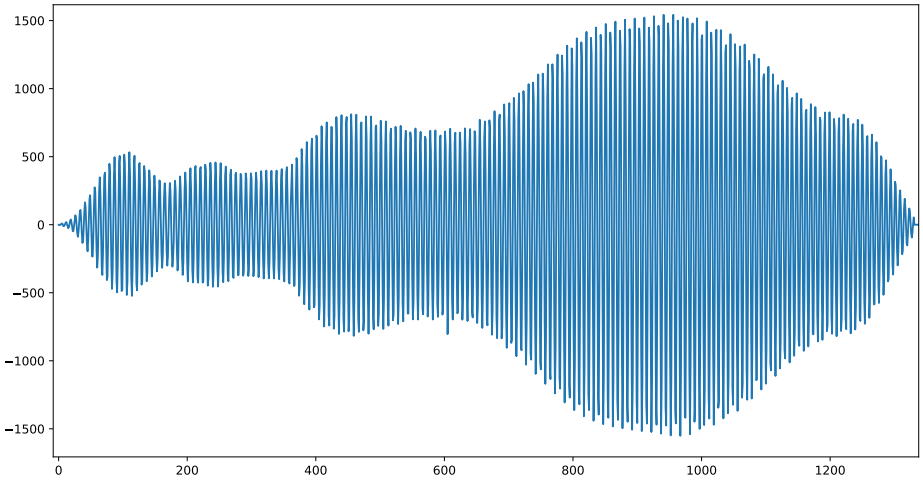


Figure 15: Reference (transmitted) chirp pulse signal

The reference signal (recorded transmitted chirp) is stored as a CSV file, and loaded by the Julia program on instantiation, as shown in the code excerpt below:

```
io = open("vtx_data.csv","r") # open csv file in read mode
global v_tx = readdlm(io, '\t', Complex{Float64}, ','); # parse csv data into array
close(io) # close csv file
```

The matched filter is then built from the reference signal.

```
V_TX = fft(v_tx); # take FFT of reference signal
H = conj(V_TX); # take complex conjugate to get matched filter
```

As shown in the code excerpt above, the FFT of the reference signal is taken, and the complex conjugate of that calculated to obtain the matched filter.
The output of the matched filtering process is then calculated by multiplying the matched filter with the received sampled signals. This is shown in the code excerpt below.

```
V_MF0 = H.*V_RX0;
V_MF1 = H.*V_RX1;
```

3.3.5 Analytic and baseband signals

The output of the matched filter contains both positive and negative frequency components. Additionally, the frequency components are centered at 40kHz. This means that the matched filter output has an envelope, containing the carrier. In order to create the range profile plots, and calculate the angle of arrival, only the envelope is desired. Creating the analytic signal, which contains only positive frequency components, is part of this process – the magnitude of the analytic signal follows the envelope. The other part of the process is obtaining the baseband version of the signal, which contains no carrier frequency components. The calculation of the analytic version of the matched filter output is implemented using the following method:

```
# multiply matched filter output by 2
V_ANAL0 = 2*V_MF0;
V_ANAL1 = 2*V_MF1;
# find negative frequency range
```

```
N = length(V_MF0);
if mod(N,2)==0 # case N even
    neg_freq_range = Int(N/2):N; # define range of neg-freq components
else # case N odd
    neg_freq_range = Int((N+1)/2):N; # define range of neg-freq components
end
# zero out negative frequencies
V_ANAL0[neg_freq_range] .= 0;
V_ANAL1[neg_freq_range] .= 0;
```

As shown in the code excerpt above, first, the matched filter output is multiplied by two. The position of the negative frequency range in the signal array is then calculated, depending on whether the array length is odd or even. Finally, the negative frequency components are set to zero. The time-domain versions of the analytic signals are calculated by taking the inverse FFT of the frequency domain representations, as shown below.

```
v_anal0 = ifft(V_ANAL0);
v_anal1 = ifft(V_ANAL1);
```

The basebanded signals are calculated by multiplying the analytic signals with the carrier (40kHz). This is implemented in the code excerpt shown below.

```
f0 = 40100; # Actual chirp empirically determined to be centered slightly above 40kHz
v_bb0 = v_anal0.*exp.(-j*2*pi*f0*t);
v_bb1 = v_anal1.*exp.(-j*2*pi*f0*t);
```

The baseband signals are then normalised to an amplitude of 1 with respect to their maxima. This is implemented as shown in the code excerpt below:

```
v_bb0_norm = abs.(v_bb0)./(maximum(abs.(v_bb0)));
v_bb1_norm = abs.(v_bb1)./(maximum(abs.(v_bb1)));
```

3.3.6 Peak detection algorithm

Since the output of the matched filter (and hence its baseband representation) is at a maximum at the point at which the reference signal is detected, a peak detection algorithm can be used to determine the range to the targets (time/distance of echoes). The implementation of the peak detection algorithm follows the following process:
First, all local maxima are located, by finding the turning points (points at which the signal magnitude starts decreasing). This is shown in the code excerpt below:

```
for i = 1:length(v_bb)-1 # iterate through the signal samples
    if abs(v_bb[i+1])>=abs(v_bb[i]) # check if next sample is greater than current sample (increasing)
        max_found = false # flag going up to new maximum
    else # decreasing
        if max_found == false # only log as a maximum of previous gradient was positive
            push!(localmaxima,i) # add current index to array of local maxima
            max_found = true # flag going down from maximum
        end
    end
end
```

The amplitudes of the recorded local maxima are then compared to a threshold value (chosen to be 0.5), and decided to be peaks if they are greater than the threshold. This is shown in the code excerpt below:

```
thresh = 0.5;
for i = 2:length(localmaxima)-1 # iterate through maxima to find main peaks
      if abs(v_bb[localmaxima[i]])>thresh # compare to threshold
        push!(peakindexes,localmaxima[i]) # add to peaks if greater than threshold
    end
end
```

3.3.7 Angle of arrival calculation

The given requirements for the sonar system include being able to determine both range and angle to target(s). Range calculation is covered by the peak detection - as the time to peak can be converted to a distance. This leaves the final DSP requirement: to calculate the angle to target (or angle of arrival of the echo). This is done by comparing the phase difference between the two input channels at the respective peaks. The formula for this was given in equation 1.

Code for this is given in the snippet below. Note that v_1 and v_2 are the samples from the two receivers.

```
function arrivalangle(v_2,v_1,wavelength,d,k)
    phase_difference = angle.(v_2.*conj(v_1)) # obtain phase difference from peak samples
    arrival_angle = asin.(((2*pi*k)+wavelength*phase_difference)/(2*pi*d)) # calculate angle of arrival
    return arrival_angle
end
```

Note that there is an additional parameter, k, which has not been explained. Different values of k can be used to compensate for 'ambiguous angles'. However, doing this is beyond the scope of this project, and was set to zero when the function was called. It is included here for extensibility.

3.4 Sub-system: Graphical User Interface

User interaction with the system is a critical design consideration and for the purposes of the SONAR scan system, a Graphical User Interface which used Blink and PyPlot was built. Independent of the GUI, PyPlot could also be used to directly interface with the digital signal processing results.

3.4.1 PyPlot

PyPlot is a module within the Matplotlib plotting library that can be used with Julia. This package takes advantage of Julia's multimedia I/O API to display various plots of data in any Julia graphical backend. [2] To use this package, it must be imported, as shown below.

```
using PyPlot;
```

To faciliate the continous plotting of data in a smooth way, the plotting function must be as 'lightweight' as possible. Rather than calling *plot( )* every time a new set of data arrives, PyPlot allows one to define a plot, and then simply change the data associated with the plot. This ends up being a faster implementation of presenting a new result. The code for setting this up is given below:

```
# Create plots that will be populated with real data later
fig, axs = plt.subplots(4, 1)
figP, axsP = plt.subplots(1,1)
# Mag/Phase plot:
magn0, = axs[1].plot(r,zeros(dataSize))
angl0, = axs[2].plot(r,zeros(dataSize))
magn1, = axs[3].plot(r,zeros(dataSize))
angl1, = axs[4].plot(r,zeros(dataSize))
axs[1].set_ylim(0, 1.1)
axs[2].set_ylim(-3.5, 3.5)
axs[3].set_ylim(0, 1.1)
axs[4].set_ylim(-3.5, 3.5)

axs[4].set_xlabel("Range [m]")
axs[1].set_ylabel("Magnitude")
axs[3].set_ylabel("Magnitude")
axs[2].set_ylabel("Phase")
axs[4].set_ylabel("Phase")

# For adding red-dots as peak idenitifiers
points0, = axs[1].plot(0,0,"ro",markersize=5);
points1, = axs[3].plot(0,0,"ro",markersize=5);

# Polar plot:
polarPoints, = axsP.plot(0,0,"bo",markersize=10);
axsP.plot(0,0,"ko",markersize=20)       # Plot a blob at the origin to depict the device's position
axsP.set_xlabel("horizonal range [m]")
axsP.set_ylabel("vertical range [m]")

axsP.set_ylim(0,10)
axsP.set_xlim(-2,2)
axsP.grid(true, which="major")
```

Then, to update the plot data, one can simply 'set' the data, and the plot will be updated. The code below shows how the range and phase plots are processed:

```
# ---------------------------------------------------------------------
# RANGE & PHASE PLOT:
magn0.set_ydata(v_bb0_norm)
angl0.set_ydata(angle.(v_bb0))
magn1.set_ydata(v_bb1_norm)
angl1.set_ydata(angle.(v_bb1))
add_peak_points(v_bb0_norm, v_bb1_norm, 0.5, points0, points1)
# ---------------------------------------------------------------------
# POLAR PLOT:
```

```
b0_max = locatemaxima(v_bb0_norm,0.5)
b1_max = locatemaxima(v_bb1_norm,0.5)
if (length(b0_max) == length(b1_max))
    x_vals = Float32[]
    y_vals = Float32[]
    for i in 1:length(b0_max)
        R = r[b0_max[i]]
         = arrivalangle(v_bb1[b1_max[i]],v_bb0[b0_max[i]],c/f0,0.017,0)
        println()
        push!(x_vals, R*cos())
        push!(y_vals, (-1)*R*sin())
    end
    polarPoints.set_data(y_vals, x_vals);
else    # Can only plot polar result if same # of peaks detect in both channels
println("error! Mismatching number of peaks detected")
end
# ------------------------------------------------------------------
# Add red-dots as peaks
function add_peak_points(v_bb0, v_bb1, thresh, points0, points1)
    peaks0 = locatemaxima(v_bb0,thresh)
    r_peaks0 = zeros(length(peaks0))
    y_peaks0 = zeros(length(peaks0))
    for i in 1:length(r_peaks0)
        r_peaks0[i] = r[peaks0[i]]
        y_peaks0[i] = v_bb0[peaks0[i]]
    end
    points0.set_data(r_peaks0,y_peaks0)

    peaks1 = locatemaxima(v_bb1,thresh)
    r_peaks1 = zeros(length(peaks1))
    y_peaks1 = zeros(length(peaks1))
    for i in 1:length(r_peaks1)
        r_peaks1[i] = r[peaks1[i]]
        y_peaks1[i] = v_bb1[peaks1[i]]
    end
    points1.set_data(r_peaks1,y_peaks1)
end
# ------------------------------------------------------------------------
```

3.4.2 Blink

Blink.jl is a wrapper for Electron.js - A framework which allows for building of cross-platform desktop applications with Javascript, HTML and CSS. This works by serving HTML pages to a window where the webapp is hosted on a cloud server.

Shown below are some HTML code snippets used to create the dashboard from which Julia functions to interact with the system were called.

Listing 1: Code snippets for Dashboard creation

```
<!-- ======================  DASHBOARD PAGE FOR SONAR IMAGING GUI ======================-->
<!DOCTYPE html>
<html>
<head>
  <title>SONAR Scanner</title>
  <link rel="stylesheet" type="text/css" href="style.css">
    <script src="https://kit.fontawesome.com/817ac82d8e.js"></script>
</head>
<body>
  <div class="dashboard">
    <div class="center">
      <div>
        <img src="./stop.png" style="display: none;">
        <div class="dashboard-item-wrapper">
          <div class="dashboard-item icon"><button id="start_stop_scan" class="dashboard-btn">
          <img id="start_stop_scan_icon" class="dashboard-btn-icon" src="./play.png"></button></div>

                                                ....

    <script type="text/javascript">
      document.getElementById("start_stop_scan").onclick = function() {
        if (document.getElementById("scan_loader").style.display == "block") {
          (document.getElementById("scan_loader").style.display = "none");
          (document.getElementById("start_stop_scan_icon").src = "./play.png");
          (document.getElementById("start_stop_scan_label_text").innerHTML = "START SCAN");
        }
        else {
          (document.getElementById("scan_loader").style.display = "block");
          (document.getElementById("start_stop_scan_icon").src = "./stop.png");
```

```
30              (document.getElementById("start_stop_scan_label_text").innerHTML = "STOP SCAN");
31              Blink.msg("press", "Scan started");
32            }
33          }
34        </script>
35      </div>
36    </body>
37    </html>
```

Figure 16 shows a screenshot of the GUI window running using Blink.

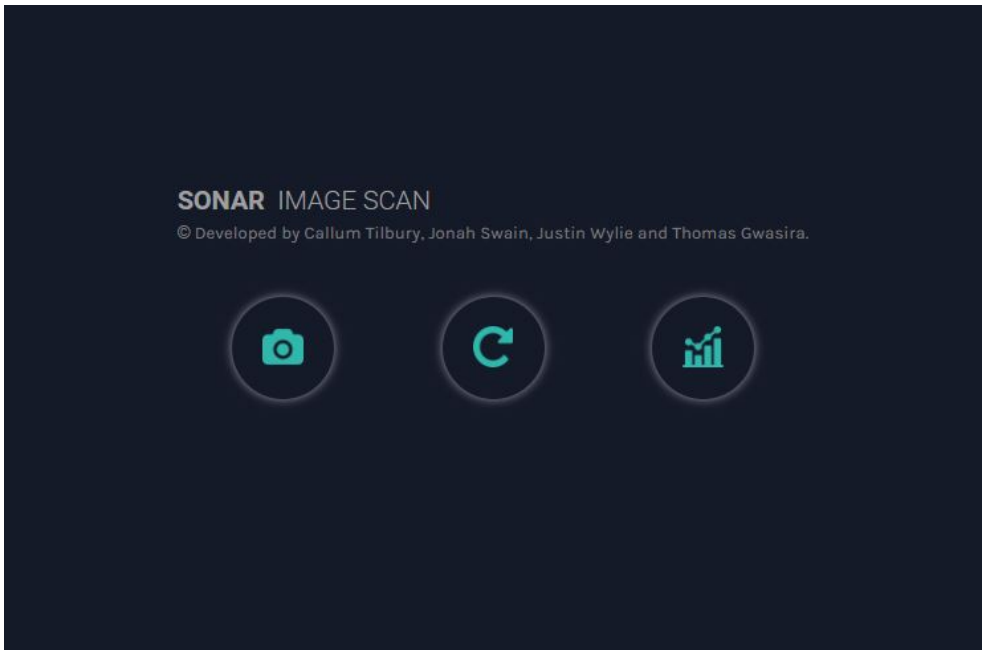


Figure 16: Graphical User Interface for SONAR Scan System

3.5 Enclosure

3.5.1 Purpose

The enclosure created for the sonar system is designed to give the system a 'finished product' look, however there are various additional design considerations and advantages that contribute to the reasoning for creating an enclosure.
The enclosure provides protection for the internal circuitry of the sonar system. A circuit with exposed wires and connectors is prone to damage, short circuits and ESD, the enclosure protects against these risks. The enclosure allows for precise positioning of the transmitting and receiving transducers. By using the enclosure to position the transducers, it allows the transducers to be detached from the vero-board freeing up space which allows for a smaller form factor.
The enclosure allows the sonar system to be easily positioned to point in the desired direction.

3.5.2 Design

The enclosure was chosen to be created from laser cut, hardboard. Hardboard was used because it is the material provided by UCT for projects. Laser cutting was chosen because it is very precise, quick to cut the design and material costs are low. The free SVG (Scalar Vector Graphic) software was used to design the enclosure. Figure 17 shows a screen shot of the front panel as it is seen from within the InkScape design environment.
The enclosure consists of 6 pieces of board that are assembled and glued to form the final enclosure. As a result of the symmetry of the enclosure, the outline of the each pair of opposite side pieces is the same. The front panel had three holes placed on it for the three transducers, the spacing between the centres of the two receiving transducers was set to be 17mm. One of the side pieces had a small hole for the RGB status led and a larger hole for the USB cable cut into it. Labeling and aesthetics where added to the box to ensure usability and to give the system a polished appearance.

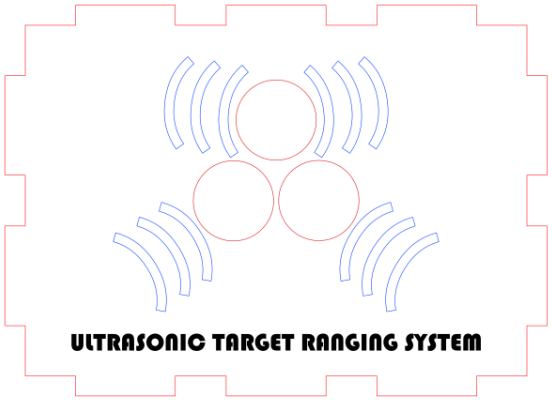


Figure 17: Front Panel in InkScape Design Software

3.6 Integration of Sub-systems and Testing

In order to arrive at a functional system, the aforementioned subsystems had to be properly integrated, in an appropriate order. Failing to do this would have resulted in a frustrating debugging process.

The first step in this process was ensuring that the analog circuitry was performing as required. An *Agilent 33210A Waveform and Function Generator* was initially used to create a constant amplitude, constant frequency sinusoidal waveform. This was connected to the transmitting transducer, and a 2-channel digital oscilloscope was used to view the echoes on the receiving transducer. This was primarily done to ensure correct operation of the transducers. Each received signal was then scoped at the output of the respective amplifier blocks, as well as after the level shifting (summer) blocks. The clamping circuits were tested by increasing the transmitted signal amplitude, and ensuring the output (to be fed into the microcontroller) did not exceed 3.3V.

Using this setup, an approximate idea of the frequency response of the transducers was also investigated. Frequencies surrounding 40kHz were swept through, and the amplitude of the received signal was measured. An approximate band-pass response was discovered.

Next, the microcontroller was connected to the transmitter's driving circuit. The digital chirp pulse was initiated in regular intervals for testing purposes. As before, the receiving circuit was scoped at various points to ensure that the amplification was working and was clamped when necessary.

To test the analog-to-digital conversion on the microcontroller, the signal generator was again used. A simple 40kHz, level-shifted sinusoid was fed into each analog input pin, and the ADC routine was started, sampling values straight to memory using DMA. The digital signals were considered afterwards and confirmed to be a sufficiently accurate version of the continuous input.

Only then were the receiving pins on the microcontroller connected to the outputs of the receiving amplification blocks in the analog circuitry. This completed the integration between these two subsystems.

Independent to the above testing, the microcontroller-to-PC connection was tested. A simple Serial echo server was created: the microcontroller wrote an array of increasing numbers to its Serial port, and the PC read these values. Then, the PC wrote these values back to the microcontroller, where they were read and compared to the original values. Once this was working, sample datasets were sent from the microcontroller to the PC – what would eventually be the received signal data.

With a working sampling routine and serial communication process, the system was fully integrated, and system-wide testing could begin.

3.7 Bill of Materials

Table 2 shows the bill of materials for the project. Notice that the total cost is less than half of the proposed budget.

Item	Unit Cost	Units	Cost
Teensy USB Development Board (V3.6)	R 491.40	1	R 491.40
5V Linear Regulator (7805CT)	R 9.37	1	R 9.37
3V3 Linear Regulator (LM317LZ)	R 7.86	1	R 7.86
470uF Electrolytic Capacitor	R 3.63	2	R 7.26
100uF Electrolytic Capacitor	R 3.33	1	R 3.33
22uF Electrolytic Capacitor	R 3.33	2	R 6.66
Inverting Octal Buffers (CD74HC540E)	R 13.00	1	R 13.00
Dual Operational Amplifier (TL082)	R 12.25	2	R 24.50
10K Trim Potentiometer	R 11.94	1	R 11.94
2-pin Molex Male Connector	R 2.57	3	R 7.71
2-pin Molex Female Connector	R 3.93	3	R 11.79
RGB LED (Common Anode)	R 13.04	1	R 13.04
SPST Switch	R 45.54	1	R 45.54
9V Battery	R 70.00	1	R 70.00
9V Battery Snap Connector	R 7.11	1	R 7.11
Schottkey Diode (1N5819)	R 6.96	4	R 27.84
Row of 24 Female Headers	R 20.71	2	R 41.42
20-pin DIP IC Socket	R 24.95	1	R 24.95
8-pin DIP IC Socket	R 7.56	2	R 15.12
33k Resistor (5%)	R 0.10	6	R 0.60
1k Resistor (5%)	R 0.10	4	R 0.40
29k Resistor (5%)	R 0.10	2	R 0.20
470R Resistor (5%)	R 0.10	1	R 0.10
Veroboard (5cm x 7cm)	R 74.00	1	R 74.00
Total Cost			**R 915.14**

Table 2: Bill of Materials required for the project (All prices from Digikey)

4 Results

Validation and Testing were carried out for the SONAR image scanner and a clear picture of how well the realisation of the design matched the desired performance was obtained. Specifically, the following types of tests were done:

- **Unit Test:** This entails testing each individual subsystem and ensuring that it behaves as expected. The purpose of unit testing is to ensure units are working before being assembled, which makes debugging much easier.In the context of the project, this was done by independently testing the driver and receive circuits, sending a receiving signals using the Teensy board, Digital Signal Processing using Julia as well as testing the user interface.
- **Integration Test:** Individual elements of the system are combined and how well they interact with each other is observed. This was achieved by combining immediately related elements such as the Teensy board and the circuitry and examining how they functioned together.
- **System Test:** In System Testing, the complete and integrated system is tested to evaluate the system's compliance with the specified requirements.[1]
- **Acceptance Test:** After the aforementioned, prior tests, the prototype and its functionality can then be analysed to see if it matches the user and/or business requirements. This was done in form of a demonstration to the client, through which useful feedback was received.

Of particular importance was the System Level Testing as this is what the tests preceding it build up to. This was done by assembling the entire system and performing a White-box assessment of its operation when running. Section 4.1 shows the results for a single shot scan with the roof as a reflector while Section 4.2 shows the results of performance tests for a continuous scan.

4.1 Single Shot Scan Of Roof

The following results were obtained for testing each of the sections given below:

4.1.1 Driver and Receive Circuitry

During the test runs, the voltage provided across the transducers was constantly monitored. In addition, an oscilloscope was connected across the receiving transducers and it was noted that they both detected chirp pulse echoes.

4.1.2 Teensy Board

The operation of the Teensy Board in conjunction with the circuitry was assessed by placing tracing statements in the code i.e. printing flags or values of specific variables when a particular function is done running. Futhermore, the values of the received signal sampled by the Analog to Digital converter were printed to have a sense of how well the system was functioning.

4.1.3 Digital Signal Processing

Visual representations of the received signals and the outputs of digital signal processing stages were obtained in this section test. This was carried out using Julia and PyPlot to perform Mathematical operations and plot the results.

To begin with, the time domain received signal as sampled by the Teensy Board subsystem was passed into a Julia script as arrays and the results of this, along with the frequency domain representation are shown in figures 18 and 19.

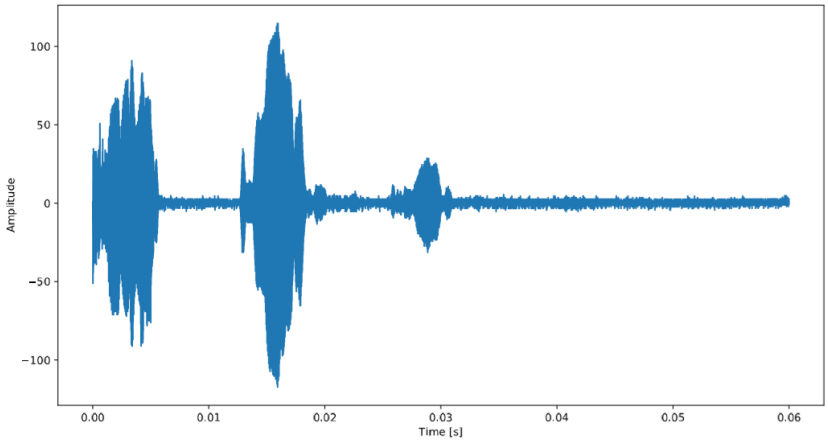


Figure 18: Time Domain Representation of Received Echo Detected by Single Transducer

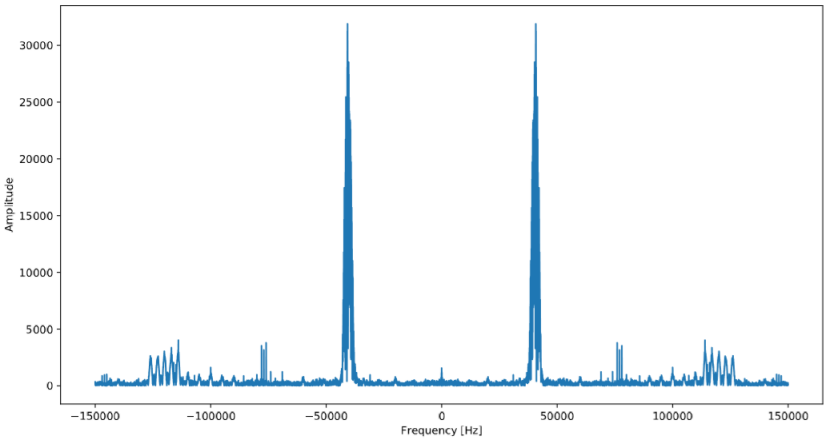


Figure 19: Frequency Domain Representation of Received Echo Detected by Single Transducer

Figures 18 and 19 show the reflected chirp pulses detected by the receiving transducers. This is exactly as was expected: replicas of the original transmitted signal.

The signal, in its frequency domain form was passed through a matched filter and the result was as shown in figure 20.

A matched filter optimises the signal to noise ratio in the presence of additive stochastic noise at the point where the input signal matches the signal for which the matched filter was designed. This is clearly illustrated by the peaks in Fig 20.

The output of the matched filter was converted into an analytic signal - as is illustrated in figure 21 - which was converted into a basebanded signal as illustrated in figure 22

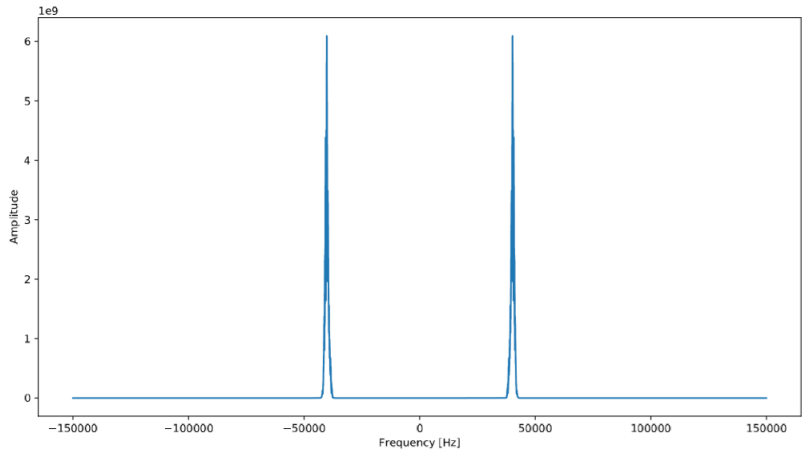


Figure 20: Matched Filter Output for Received Echo

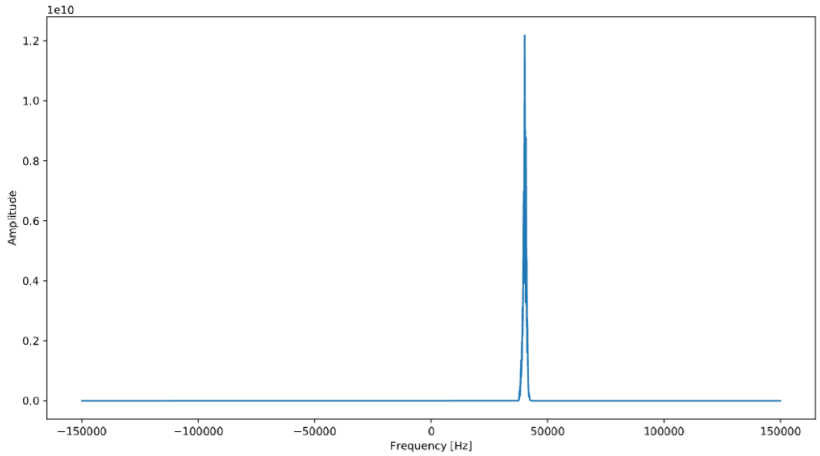
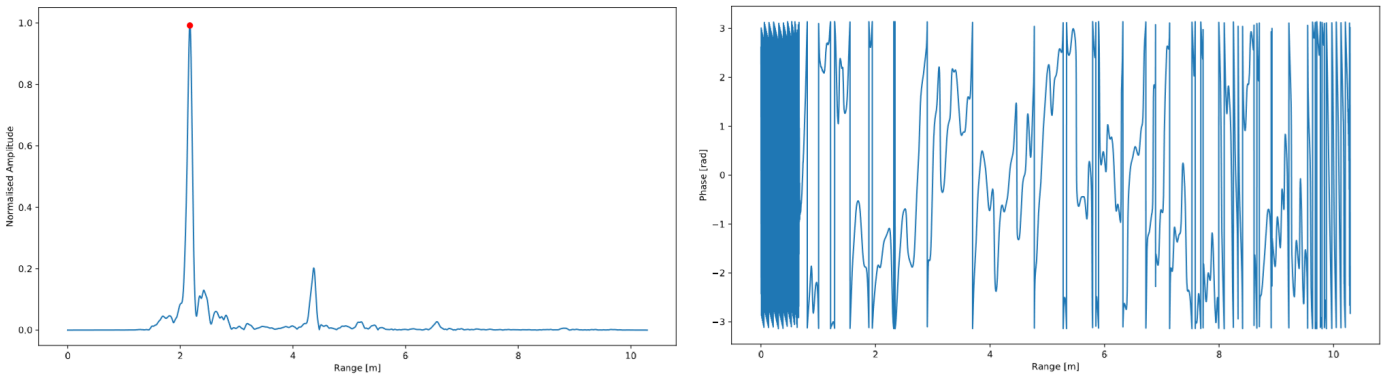


Figure 21: Analytic Signal



(a) Amplitude

(b) Phase

Figure 22: Normalised Basebanded Signal

The analytic signal is seen to be twice the positive frequency component as expected and the normalised basebanded signal is seen to be analytic signal shifted down in frequency with a maximum amplitude of 1.

Finally, consider figure 23. This shows the basebanded signal (magnitude and phase) for each channel, zoomed into the peak. Notice how the phase is essentially constant over the width of the peak – this is the phase value that is extracted for angle-of-arrival calculations. Using these values, a polar plot can be determined for this scene. The result is shown in figure 24.

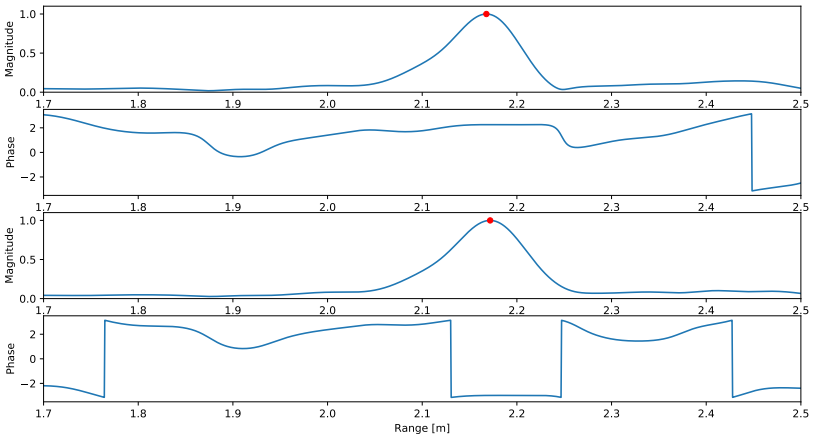


Figure 23: Result on two channels, zoomed into peaks

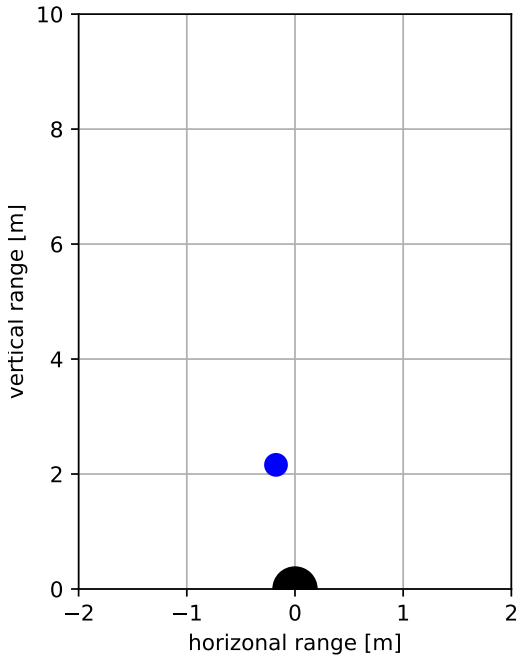


Figure 24: Polar plot of single shot scan of roof

4.2 Performance Specific Tests

Accuracy tests, range resolution and angular resolution of the system were tested and the results obtained were as shown in this section. In addition, these results show the plot of angle against range for detection of multiple reflecting targets.

4.2.1 Accuracy Tests

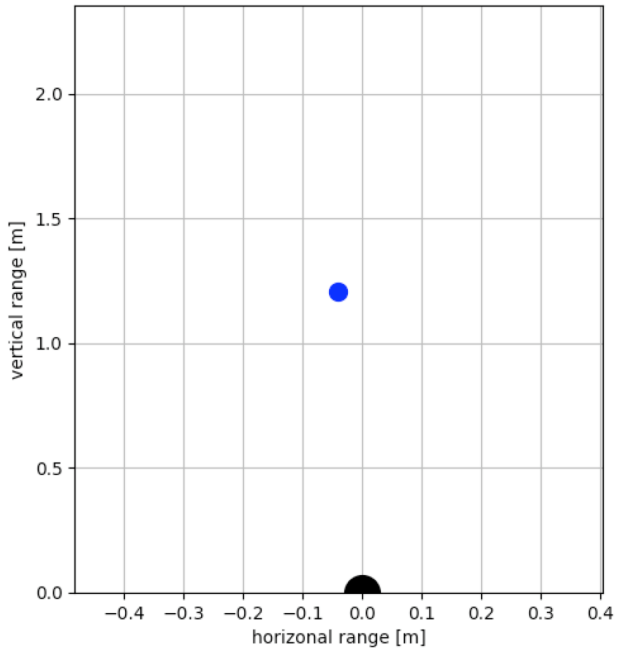
The SONAR ranging system proved to be quite accurate. To test the accuracy of the system, markings where made at several distances from the system. A reflector – a simple blackboard duster – was then placed on a particular marking and the distance measured by the sonar system was recorded. This was repeated for each distance marking. The results are summarized in table 3.

Table 3: Distance Accuracy Test Results

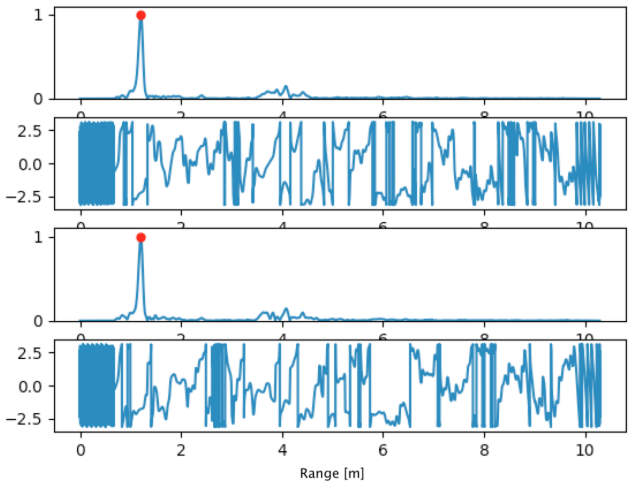
Distance	Measured Distance	Error
90cm	90cm	± 5cm
120cm	120cm	± 5cm
150cm	150cm	± 5cm
180cm	175cm	± 5cm

Figure 25 shows an aerial view of the test setup, and figure 26 shows a sample result of these tests at 120cm. Due to the resolution of the plots it is not possible to quote the result to more than 5cm of certainty.

Figure 25: Test Setup



(a) Polar Plot



(b) Range and Phase Plot

Figure 26: Range Test at 120cm

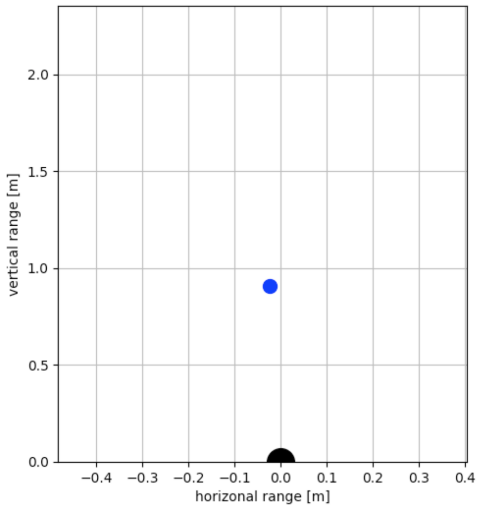
Figures 27, 28, 29 show the results of other ranges tested.

4.2.2 Range Resolution Test

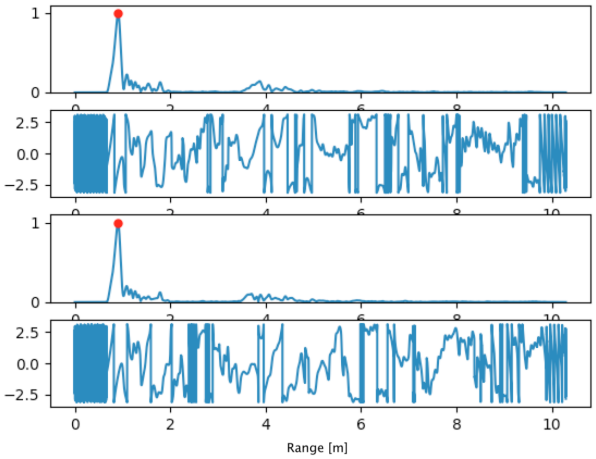
In order to check the achieved range resolution, one must consider the width of the peaks in the range plot. Notice in figure 30 that the 3dB width of the peak is approximately 6cm. This is the range resolution as it limits the distance between two peaks that can be accurately detected.

4.2.3 Angular Resolution Test

To get a feeling of the system's angular resolution, an object was placed at the point (90cm, 0°). Then, a second object was placed to the left of the first object. The distance between them was increased until both objects were visible in the output. The minimum distancefor both objects to be detected was approximately 20cm. Hence, the angular resolution between two objetcs is approximately 12°. This setup is shown in figure 31a, and the two objects in figures 31b and 31c.

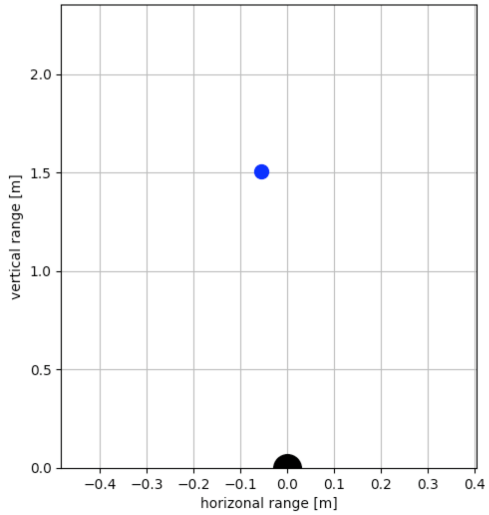


(a) Polar Plot

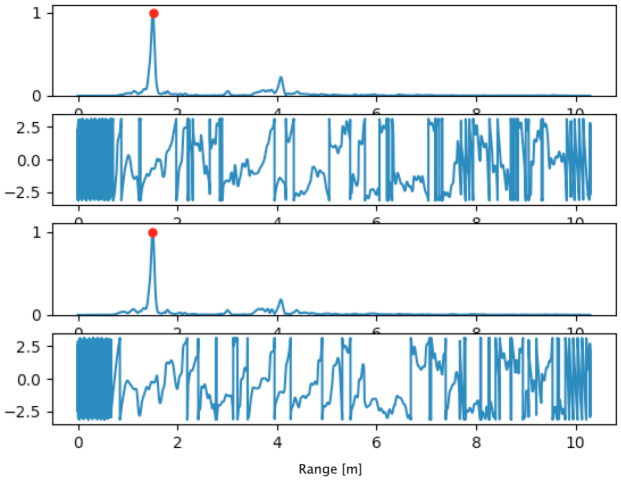


(b) Range and Phase Plot

Figure 27: Range Test at 90cm

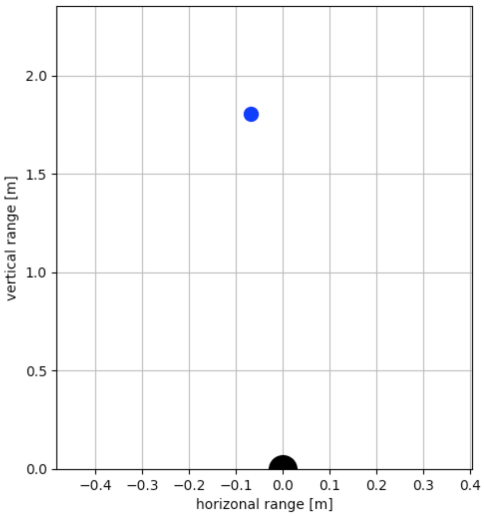


(a) Polar Plot

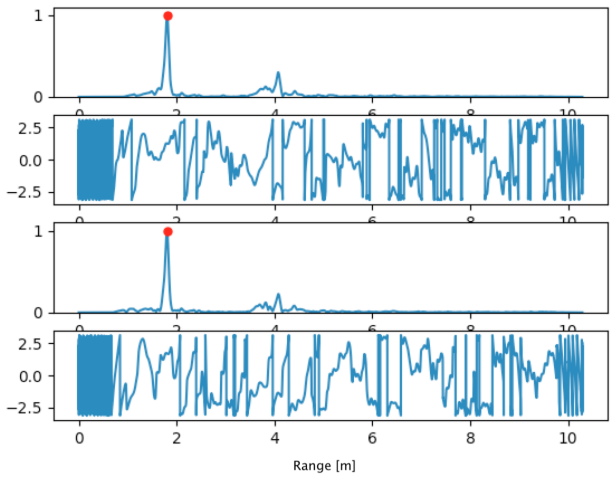


(b) Phase

Figure 28: Range Test at 150cm



(a) Polar Plot



(b) Range and Phase Plot

Figure 29: Range Test at 180cm

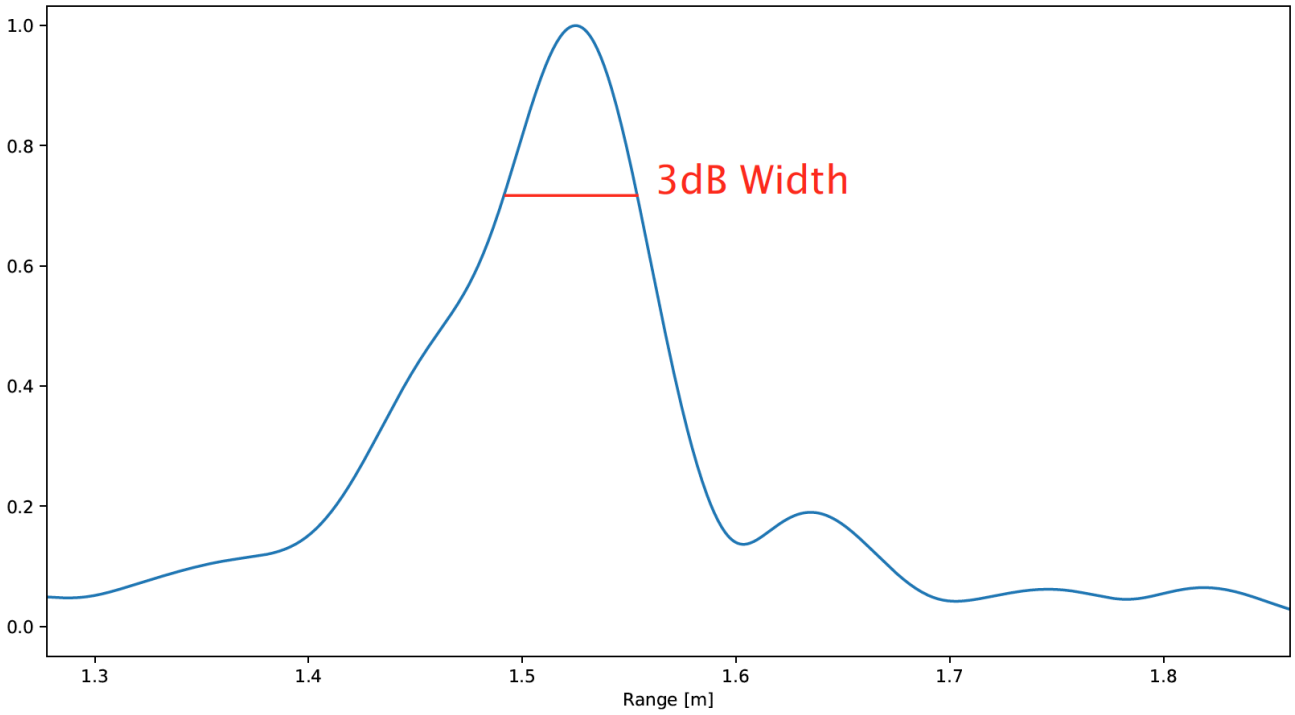
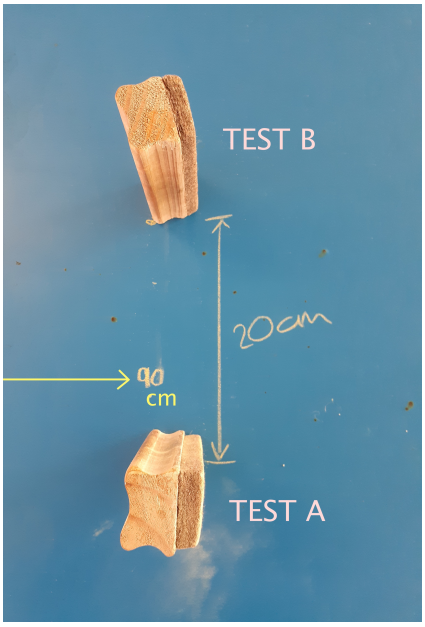
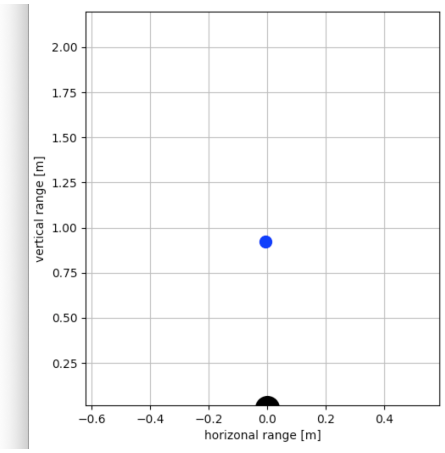


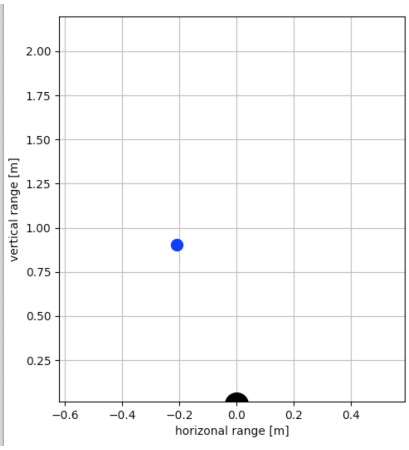
Figure 30: Zoomed into peak showing range resolution



(a)



(b)



(c)

Figure 31: Test for Angular Resolution

4.3 Sample Scene

Figure 32 shows the system in action, with the picture of the scene captured with a webcam.

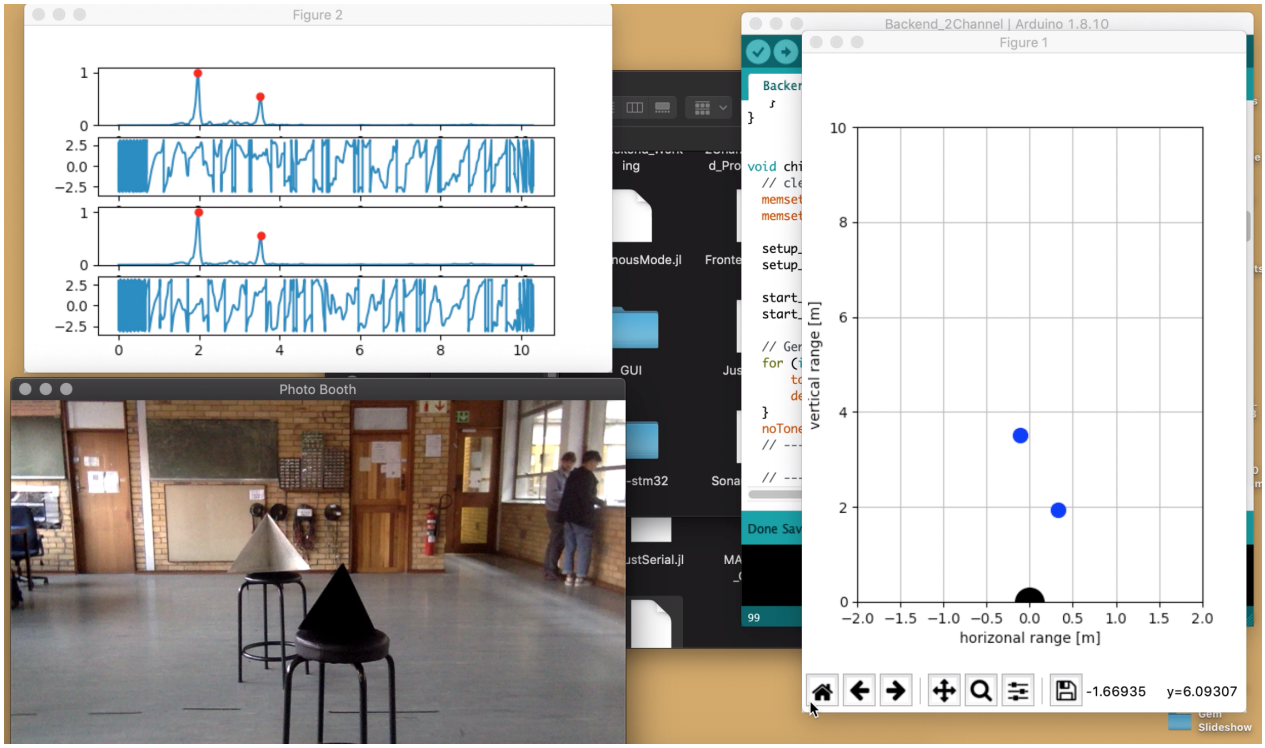
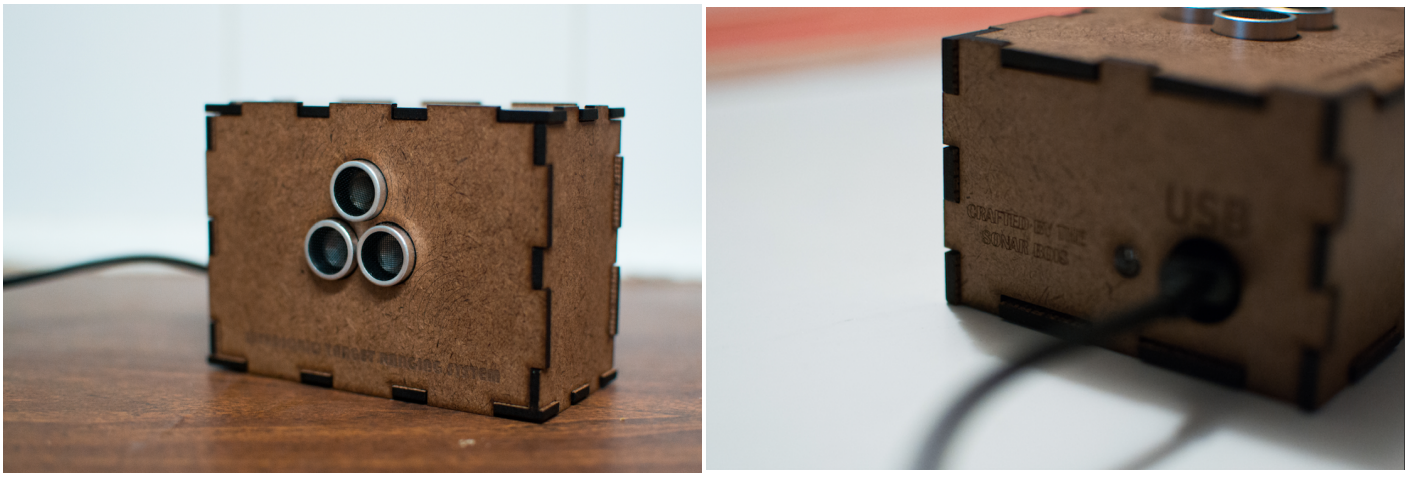


Figure 32: Sample scene test

4.4 Photos

The photos in figure 33 show the final box for the project.



(a) Top View

(b) Side View

Figure 33: Photos of Final Project Box

Figure 34 shows the final veroboard used.

5 Conclusions

5.1 Project Reflections

With regard to the final product - the working sonar system - the project can be considered a success, however it was not without its challenges and limitations.

The initial project scope called for a 3D sonar system, using arrays of transmitting and receiving transducers to determine the exact position of objects in a space. This turned out to be an unrealistic goal given the time constraints imposed both by the time frame and work required by other courses. As the semester progressed, and it became clear that the original

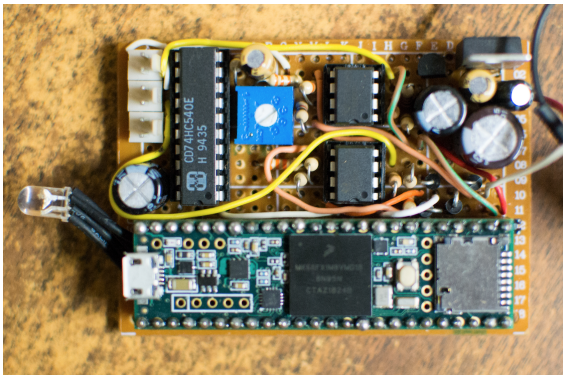


Figure 34: Photo of Project's Veroboard

goal was unrealistic, the goal was shifted to a 2D sonar system using two receiving transducers and DSP to calculate the distance and angle to targets. This is the final result that was achieved.

At the outset of the project, the decision was made to use the STM32F767ZI microcontroller. This decision was made based on its hardware and peripheral merits - having a higher clock speed, more RAM, and more ADCs with higher sample rates than the suggested Teensy board. Additionally, all group members have been exposed to programming the STM32 lineup through the Embedded Systems (EEE2046) course. The main disadvantage, and ultimately Achilles heel, to this board is that it has to be programmed at a register level, with minimal abstraction available through libraries. This means that programming takes a long time. With the pressure of the milestone 3 deadline, the group made the decision to switch to the Teensy 3.6 microcontroller development board, because it is supported by the Arduino IDE, and has a multitude of libraries, abstractions, and example code available for it. This made it relatively easy to get a working product/prototype in the short available time frame. The hardware advantages of the STM32F7 might have resulted in a better, more polished finished product eventually, but ultimately the ease of use of the Teensy made it a better choice for this project.

With the above being said, the Teensy microcontroller was not without problems. The group faced issues getting the Teensy to communicate with the PC over serial. Initially, samples were transferred directly (as their binary values), however a buffer overflow problem was encountered, resulting in only the first 640 samples being transferred. This was resolved by transferring samples encoded as their hexadecimal values in a string. This method of transfer, however, is less efficient, both in terms of USB bandwidth and processor time.

Additionally, the group had difficulty getting two channels to be sampled by the ADCs using DMA. Example code was provided by Johnathan Whittaker for using a single channel (and single ADC) with DMA, but this did not immediately scale to two channels. The solution to this was to use *separate* ADCs to sample each channel, making it relatively straightforward to adapt the sample code. This solution was not, however, immediately apparent, because the information containing the mappings between GPIO pins and ADCs is hidden in a long-winded reference manual.

Because one of the team members was well-versed in HTML and Javascript, it was decided that the front-end Graphical User Interface (GUI) would be written using these languages. While the result was both aesthetic and functional, integrating this GUI with the Julia digital signal processing (DSP) side of things was difficult. The use of the Blink library helped, but did not eliminate the problems. Unfortunately, by the time the end of the project arrived, this integration had not yet been achieved. In future, unifying the DSP with the GUI should be tested earlier in the project.

In order to make a more robust and professional-looking final product, the circuitry for the sonar system was constructed on veroboard. As anyone who has ever attempted to use veroboard will know, it brings its own set of challenges. These include bridged tracks, unconnected tracks, and routing and layout difficulties. These were faced in the construction process, and took time to debug and resolve, ultimately extending the time taken to implement the circuitry.

Since Julia is a relatively nascent programming language, it lacks some of the documentation and community support available with a more established programming language, like Python or C. This lack of support and documentation meant that when issues were encountered, it was not easy to debug them. Eventually, all issues were resolved, as is evident by the working sonar system. It must be noted, though, that the Julia language has many appealing and compelling features for use with DSP.

Despite all of the issues and complications described above, the results achieved by the sonar system are positive. As examined in the results section, testing revealed the target ranging accuracy of the system to be consistently within 5cm of the actual value. This is well within the design requirement of a 10cm accurate resolution. Additionally, the output of the matched filter achieves a high signal-to-noise ratio (SNR), even at the maximum rated range of 8m. This indicates that the actual maximum range of the device is greater than the rated 8m, and hence its capabilities could be easily extended. Furthermore, the sonar system is able to distinguish individual targets with close separation.

With consideration to all the challenges faces, results obtained, and the final product - a working 2D ultrasonic sonar system, the project was a success.

5.2 Recommendations and Future Work

- Adding an adapter for connecting a tripod would allow the user to much more precisely control the positioning of the sonar ranging system. In addition to giving the users more precise control, a tripod allows the user to position the sonar in orientations not possible with only the box enclosure.
- The addition of a laser above the transmitting transducer would allow the user to precisely center the scene being observed by positioning the laser beam through the center of the systems field of view. With the addition of the laser, a calibration feature could also be implemented to allow the user to finely tune the target angle.
- The creation of a mobile app along with a Bluetooth connection would make the system completely wireless and extremely convenient to setup. Either a Bluetooth transceiver could be directly interfaced with the Teensy (and all calculations performed on the mobile device) or a single board computer (such as the Raspberry Pi) with on board wireless connectivity to could be used to perform the DSP and send the results to the mobile app.
- To achieve 360° of view in the horizontal, a servo motor could be mounted underneath the system. This could allow a stationary scene to be scanned over some period of time, producing a radar style 2D plot.
- Currently the system is powered of a 9V battery. A big improvement would be to implement a built-in rechargeable battery (and charging circuitry). This would allow the user to save money of batteries and allow the user to top up the energy of their battery on demand, before taking the system into the field.
- Currently the system is not able to reliably distinguish targets that are at the same radial distance from the transducer, this because the device only has two receiving transducers. If the system was extended to use several more transducers, it would be able to distinguish targets at the same radial distance. The resolution at which targets can be distinguished is directly proportional to the number of receiving transducers. This process of distinguishing targets is as a result of beam steering.
- The use of additional transmitting transducers would allow for vertical beam steering. This would allow the system to distinguish objects at some vertical angle. As with the addition receiving transducers, the vertical resolution for distinguishing targets at the same radial distance is directly proportional to the number of transmitting transducers.
- The Teensy 3.6 microcontroller development board includes onboard programmable gain amplifiers (PGAs) on the input to the ADCs. These could be used to extend the effective range of the sonar system by dynamically adjusting the gain depending on the time since transmission (and hence current echo distance) to compensate for the $\frac{1}{R^2}$ dependence.
- Using a better microcontroller/development board would enable better performance and/or additional features. Two potential options for improved microcontrollers are the Teensy 4.0 (ARM Cortex-M7F, 600MHz, 1M SRAM, 2M flash, 2x 12 bit ADCs), not released at the beginning of the project, or the STM32H743ZI (ARM Cortex-M7F, 480MHz, 1M SRAM, 2M flash, 3x 16 bit ADCs).

There are endless possible means to improve or add features to the system. The device, is a 'vanilla' system that is able to be adapted according to an individuals unique user requirements. For some use cases, certain features would be indispensable while for others those same features may be a compromise.

References

[1] *System testing*, Mar. 2018. [Online]. Available: http://softwaretestingfundamentals.com/system-testing.
[2] JuliaPy, *Juliapy/pyplot.jl*, Sep. 2019. [Online]. Available: https://github.com/JuliaPy/PyPlot.jl.
[3] T. E. o. E. Britannica, *Sonar*. [Online]. Available: https://www.britannica.com/technology/sonar.

A Appendix

A.1 Project Timeline

Figure 35 shows a Gantt Chart depicting a broad overview of the 12-week project timeline.

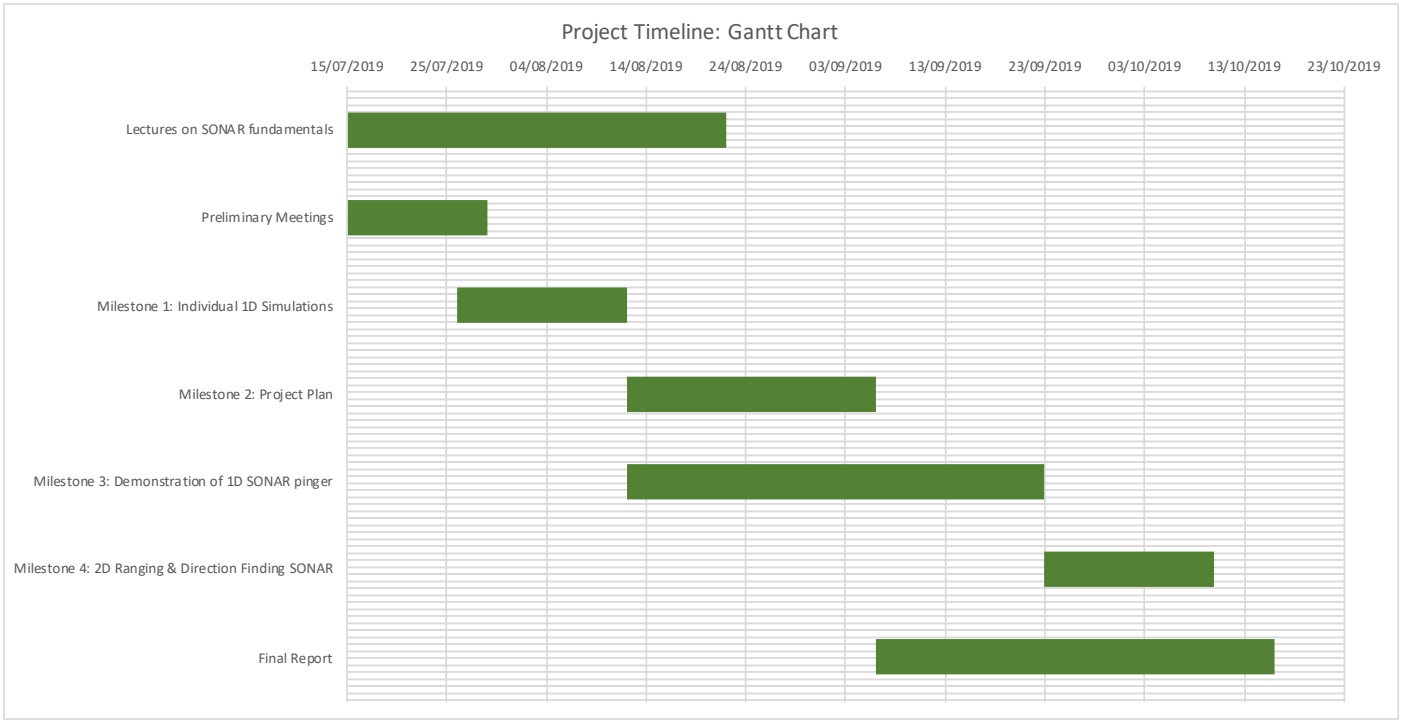


Figure 35: Gantt Chart depicting Project Timeline

A.2 Team Member Contributions

Table 4 shows a summary of the respective contributions that each team member made to the project.

Team Member	**Contribution**
Thomas Gwasira	- Direction finding algorithms - Graphical User Interface - Reports (System Requirements and Specifications, Activity Diagram, Implemenation: GUI, Results)
Jonah Swain	- Contributions to both reports - Consultation on the analog circuitry - Consultation on the microcontroller code - Some work done on STM32F7, which ended up not being used due to time constraints favouring the Teensy
Callum Tilbury	- Teensy Microcontroller Code - Digital Signal Processing in Julia - Veroboard Design and Assembly - Report compilation and proofreading
Justin Wylie	- Circuit Design and Testing - Enclosure Design and Creation -Reports (Block Diagram, Circuit Diagram, Explanations of both) - Presentation (Responsible for delegation of work) - Debugging Hardware and Software issues - Recording Chirp waveform for the matched filter

Table 4: Team Member Contributions