Faculty of Electrical Engineering

ECED 4901 Senior Year Project

Final Report

Curling-Rock Tracking System

Ryan Haley B00320636

Matt Thomson B00612496

Victor Wong B00560156

William Perrett B00507880

3 December 2014

To Joseph Murphy of the Train Smart for Curling Team,

Enclosed is the final report detailing the development of a curling-rock tracking system using acoustic, difference, propagation-delay calculations. This report is being submitted to complete the requirements ECED 4901 as required by the Department of Electrical and Computer Engineering.

This report presents the design process taken to develop an alpha prototype of a curling-rock tracking system. The objectives, as determined by the client, include having a system that has a spatial accuracy of better than 15cm, working in typical curling conditions and interfacing directly with a laptop. It should be noted that this system was developed with the assumption that it is necessary to track in only a section of a sheet and that the environment is quiet.

The report presents the design of the system, breaking it three parts: algorithms, hardware and software. After the design is explained, the major tests that were completed are documented to show the evolution of the design and the final results. A budget and schedule are also provided, and finally an analysis is performed on the design, with recommendations for future modifications.

Overall, the acoustic, difference, propagation-delay solution was successfully adopted to achieve a median spatial accuracy of 4.17cm with no spatial errors higher than 15cm. All other required objectives were met. Future recommendations for this project include purchasing better equipment, developing a dynamic threshold for pulse detection and using ultrasonic sensors and transducers.

This report was written by all members of the group and submitted only to Dalhousie University. If there are any comments or concerns please do not hesitate to contact us at ry321235@dal.ca or mt848213@dal.ca.

Sincerely,

Ryan Haley, Matt Thomson, Victor Wong, William Perrett

Table of Contents

Acknowledgments

Our design team would like to thank Joe Murphy, Dr. Peter Gregson, Dr. Kamal El-Sankary and Dr. John Newhook for their continuous encouragement and support throughout this project.

1.0 Abstract

This report details the development of a curling-rock tracking system using acoustic, difference, propagation-delay calculations. The prototype is capable of measuring the position of a rock once a second to a median spatial accuracy of 4.17cm or better over an 8.24m by 4.37m area, with no spatial errors larger than the specified accuracy of 15cm. It does this with a material cost of less than $400 and the luxuries of wireless access and graphical plotting. Using consumer electronics and development boards, the position was located using acoustic pulses. A method of synchronization was developed to bring asynchronous audio streams into a single time reference. A pulse detection algorithm was created using a Goertzel filter that specialises in removing noise from the signal band. Many tests were conducted to refine the algorithms in order to overcome a variety of issues, such as multipath or sampling frequencies errors. All required objectives specified by the client were met showing the viability of the acoustic, difference, propagation-delay solution.

2.0 Introduction

This report outlines the final design and results for the curling-rock-tracking, senior-year project. The project was completed in partnership with Train Smart for Curling, a group based in Halifax, Nova Scotia. The goal of the project was to develop a tool for training curlers and characterizing ice conditions. The tool needs to track a curling rock for a given section of a curling rink. Further details regarding curling and the objectives for this project are covered in Sections *3.1 Background* and *3.2 Objectives*.

The reasoning for the project's final design and design details are covered, demonstrating the adequacy of the acoustic, difference, propagation-delay solution. This includes the algorithms, hardware, and software needed to track a curling rock. When integrated into the final tool, the described systems become the curling-rock tracking system.

The design was tested thoroughly in the Halifax Curling Club in order to develop and validate the algorithms. Progression was visible as the testing was conducted. The final test showcases the capability of the design as well as its ability to meet all objectives and constraints. The end result is a solution that meets all performance objectives while being contained within the given constraints of a $400 budget and an eight-month schedule. The primary measure of performance was the spatial accuracy. The goal was a spatial accuracy of better than 15cm. This goal was exceeded as the design was able to achieve a median error of 4.17cm. See Section *3.5.7 Testing-Final Test* for more details.

The final budget, which covers all material costs, and purchases during development, is included. A schedule is included which outlines the progression of the project over the term along with completed milestones and deadlines.

A design analysis was conducted after the project to reflect on successes and possible failures. Overall, the project was a success, however there are components of the design that are not optimal. Ideas for future improvements are given, and should be used as a starting point for continuing development by future design groups.

Finally, the Appendix includes all source code, scripts, and data generated for the project. Therefore, with the design and the Appendix, one could recreate the project's results.

3.0 Project Contents

This section details the work completed over the course of the project. It begins by providing background on the problem and the objectives set out for the project. Then, the section continues by describing the selection process for the solution to the rock-tracking problem. The chosen solution is then described in detail, focusing on the algorithms, hardware and software used in the final iteration of this design. The major tests are documented, along with their impact to the design process, showing the evolution of the acoustic, rock-tracking system throughout the term. The budget and schedule are shown, and finally an analysis of the pros, cons, and recommended future improvements of the design is performed.

3.1 Background

Curling is a winter sport invented in medieval Scotland. The goal of the game is to get your rocks as close to the center of a target (called the button) as possible. Teams alternate throwing rocks by sliding them along the ice, and releasing them before the curler crosses the “hog line”. This is called a “throw”. The ice is swept immediately in front of the traveling rock. Each team throws eight rocks in an end, after which the points are tallied by counting the closest rock to the button, along with all other rocks from the same team that are closer to the button than the rocks from the opposing team. After ten ends are played the team with the most points is declared the winner.

While curling has a long and storied history, in more recent years the sport has been focused on developing new technologies. These range from synthetic broom heads to more advanced sliders. However, despite all these advances in technology, currently there is no system in place to track the location of the rock throughout a throw and store it in a digital format. These data could prove valuable for many different aspects of the game. In terms of athletic development, rock tracking could be a great tool for teaching the effect that different throwing techniques have on the rock. It would also be beneficial for sweepers, as the rock tracker could show over two runs the effect that sweeping has on the rock. In terms of ice making, the rock-tracking system could help with the study of how different ice conditions affect the path of rock. For all these reasons, there has been much international interest in the development of a rock-tracking system, with a potential showcase for this project in the World Men’s Curling Championship in the Halifax Metro Centre in 2015.

Since curling is an indoor sport based upon consistency, the conditions on the ice are kept nearly constant. This means that the air temperature inside the rink is between 4.5 and 5.5 degree Celsius, while the temperature of the ice itself is only slightly below zero. The humidity is kept at about 60% percent. The dimensions of the curling rink can be seen in Figure 1. It should be noted that while the dimensions between the back lines are kept consistent throughout the world, the total length of the curling sheet can vary between 138’ to 146’. The Halifax Curling Club, where all testing was performed, has a length of 140’. The curling rock itself weighs between 40 and 45 pounds, and is a smooth piece of granite with a detachable handle fastened to the rock with a bolt.

In order to develop a rock-tracking system, the design team worked with Train Smart for Curling. This is a group based in the Halifax Curling Club that emphasizes the development of new technologies and ideas for the game of curling. Train Smart for Curling is headed by Joe Murphy and Dr. John Newhook. Throughout the past year, Train Smart for Curling has had success developing an instrumented broom that is able to measure the pressure that a sweeper applies as they sweep. Train Smart for Curling is bringing its curling and technology development experience to the rock-tracking system, along with a budget of $400 to be used for prototype construction.

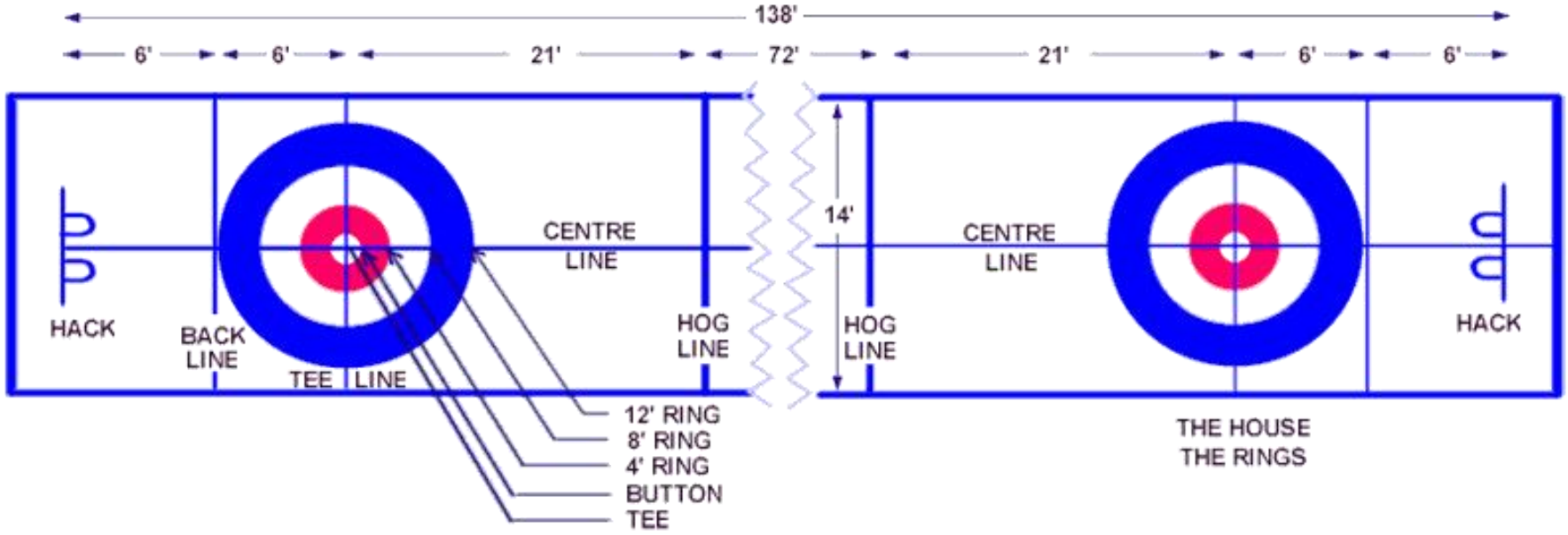


Figure 1: Dimensions of a Curling Sheet

Currently, there are two systems in curling that are related to rock tracking. The first is Eye on the Hog, a system developed by Startco Engineering Ltd [1]. This is a famous system that is used in all major tournaments. Using a magnetic strip underneath the ice, as well as a touch and magnetic sensor on the rock, the system can determine if the rock was released by the thrower before crossing the hog line. If the system detects that the thrower has not released the rock, a red LED is flashed and the rock is removed from play (this is known as hogging). The electrical components of the system are built into the handle, resulting in minimal interference with the user. The Eye on the Hog system has been well received and respected by the curling community.

The other system is a video, rock-tracking system. This system is new, and so far has only been shown in the 2014 Sochi Olympics. It traces the path of the rock in replays on television. It should be noted that so far, this system is only used for television replays and it is unknown if any usable data (such as time and velocity) can be recovered from it. It should also be noted that this system would be very expensive, as it uses a high-definition camera strung along the ice using a cable that physically follows the rock overhead. A similar system can be considered to be outside the budget of this project.

3.2 Objectives

The goal of this project is to design a rock-tracking system for athletic and facility development. During the course of this project, an alpha prototype was constructed to meet the following objectives:

1. Must have a spatial accuracy better than 6 inches (or 15cm) in order to provide usable data.
2. Must be able to function in typical curling conditions (4.5-5.5 degrees C, 60% humidity).
3. Must be able to communicate data to a laptop for further analysis.
4. Must be able to store location data, as well as user inputs, including temperature, humidity and ice pebble.
5. Must be able to output data to an Excel file.
6. Must be capable of improvement for increased spatial accuracy and expandable for tracking multiple rocks in future versions of the design.
7. Tracking only a single rock is permissible in this version.
8. Can be sheet specific, and only track for a specified section of the sheet.
9. Can be designed for testing-only conditions to avoid interference. This includes limiting the amount of acoustic noise and people within the testing area.

If the above required objectives are met, then the following list of desired objectives can be added to the design:

1. Provide a fully portable solution that could work in any curling club.
2. Be able to track the rock for the entire length of the sheet.
3. Be able to track 2 to 4 rocks simultaneously.
4. Be able to capture rotation and vibration data from the rock.
5. Display the rock's path in real time.
6. Indicate when the rock crosses specified locations, such as the hog or back line.

Testing for design verification, as set by the client, is to include:

1. Static tests to determine the accuracy and precision of the system. By setting up a pre-measured grid, the actual position of the rock is compared with the results provided by the system. This test must demonstrate the spatial accuracy of the system, as well as the precision of the system over a thirty to forty second run.
2. Dynamic tests to determine the functionality of the design during an actual curling shot. While it is difficult to test the accuracy of the system under these conditions without interfering, the

results of the test are compared with a video recording of the throw to determine if they follow the same trend. Also, specific points, such as stopping point, are measured and compared to the results.

3. All tests are completed in an acoustically quiet environment with minimal sources of interference. This includes talking, clapping and people in the testing area.

3.3 Selecting the Design

Over the course of the first semester of the project, five initial solutions that were considered. These included acoustic, difference, propagation-delay tracking; video tracking; accelerometers; rotating lasers and grid tracking. The pros and cons of each system were analyzed, and the acoustic, difference, propagation-delay solution was chosen in the end for its low cost, simple theory, and the experience of the team in acoustic signal processing and system setup.

While video tracking was an appealing option at first, it was quickly discarded. This decision was made due to the high cost of a precision camera. A good camera can cost hundreds of dollars, well above the budget of $400. Also, since only a quarter of the team is skilled at programming, having a purely software solution would be an inefficient use of resources. Finally, a video system would likely be a stationary system that could not easily be moved from rink to rink. This is because the camera would need to be calibrated for each move, which could prove to be a time-consuming process. Due to these three reasons, other solutions were considered.

The grid system was also quickly eliminated. In order to implement a proper magnetic grid, the ice would need to be torn up so the lines could be laid. This process does not lend itself well to the iterative design process, and a mistake laying the grid could take hours of work to fix. Even implementing a laser grid above the ice would require a lot of precise measuring for each sensor. A more flexible and portable system was desired.

Another laser solution involved two or three rotating laser beams located on tripods around the play surface and a light detector on the rock. Using the known position of the rotating laser beams, the dimensions of the rink, and the two or three measured angles the position could be determined. However, this solution would involve the use of constant speed motors. Given the inexperience of the group with motors, it would be difficult to ensure that all motors are rotating at the exact same speed, resulting in large errors. Also, this solution would involve the use of a visible laser in order to help meet FDA requirements. These rotating lasers could be annoying to the thrower. A solution that involves less moving parts and annoyance to the user would work better.

The accelerometer was an appealing option, as it involves the least amount of hardware. Since only an accelerometer and microcontroller are required on the rock, there would be minimal interference with the thrower. Also, no external sensors would be necessary, making the system portable. An accelerometer could even capture the rotational information of the rock. However, the accelerometer solution was not chosen for a couple of reasons. The first is the fact that determining the position of the rock from the accelerometer would require double integration. Any errors in the accelerometer readings would be amplified in the position data. Also, since the accelerometer would use no external sensors, the rock's position would always be relative to its last position. This means that in order for the stored throws to have any usefulness, they would all need to begin in the same location. This can be considered a limitation if the rock tracking system was being used to determine ice characteristics.

Finally, it was decided to go with the acoustic, difference, propagation-delay solution. Set-up would be simple, as the system would only require a few sensors being placed at measured locations along the ice. Since the rock would only require an emitter on it, it would provide minimal interference with the thrower. The acoustic system could also use different frequencies to track multiple rocks, and would allow the rock to be thrown from any position on the ice. Finally, the cost fit the budget of $400.

The decision was made to use audible frequencies instead of ultrasonic ones. While ultrasonic frequencies would not irritate the user, they also require more specialty parts. Since most of the design uses consumer electronics, use of ultrasonics would significantly increase the cost. Also, for early testing, it is beneficial to have audible pulses as it helps simplify the troubleshooting process. For these reasons it was decided to work in the audible sound range for the first prototype. Future implementations of the acoustic, difference, rock-tracking systems could use ultrasonic frequencies and would require only minimal changes to the system software.

3.4 Final Design:

The final design of the project has an emitter on the curling-rock that emits an audible pulse once per second. The pulse is picked up by four receivers located around the sheet of ice. Each of these receivers has a Raspberry Pi which runs a Goertzel filter to detect the time of arrival for each pulse. The data from each of the receivers is then sent to a central computing unit where, based on propagation-delay, difference calculations and a synchronizing pulse emitted every second the location of the rock is triangulated. Once the location of the rock is found, it is then sent to an Excel spreadsheet where velocity and acceleration are calculated. The design prototype was constructed for $342, as detailed in Section *3.6 Budget*. A diagram of the hardware flow for the design can be seen in Figure 2. It demonstrates the emitter transmitting a pulse acoustically to the sensors. The sensors then communicate their processed data over Wi-Fi to the user.

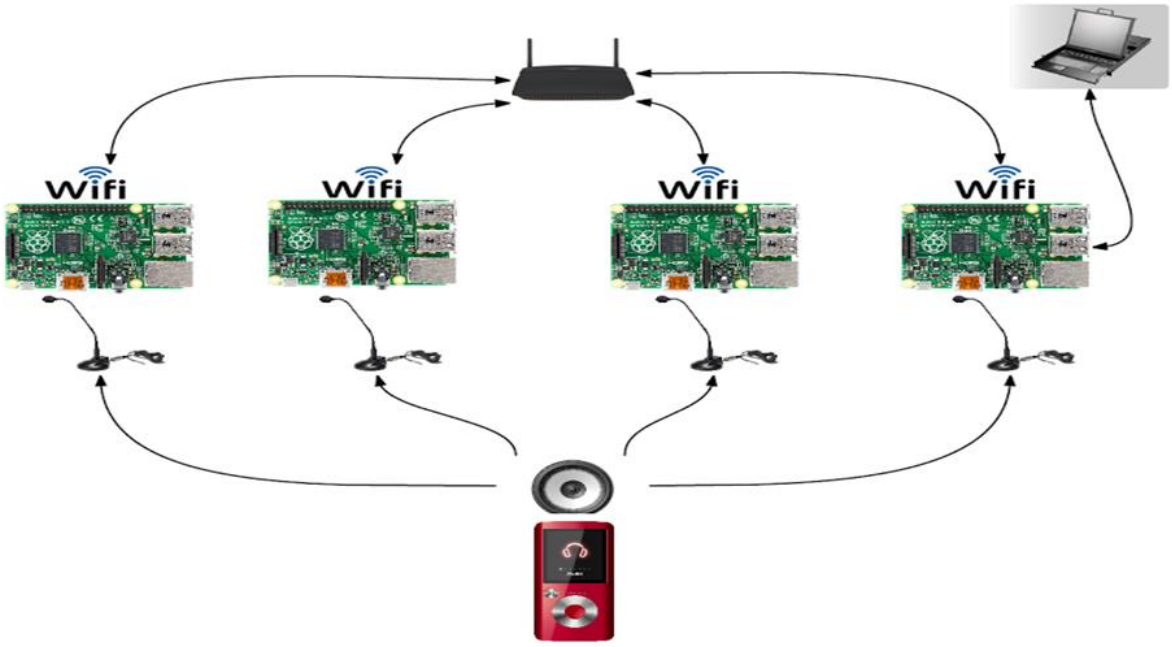


Figure 2: Hardware Flow for Rock Tracking System.

3.4.1 Algorithms

The following sections outline the algorithms that were needed for the design. The three primary algorithms are the positioning algorithm, the synchronisation algorithm, and the pulse-detection algorithm.

3.4.1.1 Positioning

The positioning module will convert time delays measured at the sensors into a location on the curling rink. This location is the output of the whole rock-tracking system. The time delays must be synchronised to each other in a single time reference, as they are all due to a single pulse. This is covered in the Section *3.4.1.2 Synchronisation*. Note that both the positioning and synchronization algorithms use data from each sensor and execute on the central processor.

This algorithm uses the difference between arrival times of a single pulse on the sensors to localise the rock. The difference between arrival times is compared with the simulated difference between propagation delays for a point on the ice to obtain a best position. The synchronisation must be applied to the arrival times before they can be used.

The positioning algorithm is executed for every rock pulse independently. Three valid sensors are needed to ascertain a valid location. For four sensors there is a basic fault tolerance system that will attempt to locate the rock with every combination of three sensors possible. The system omits a sensor for each combination. This allows the system to be fault tolerant for a failing sensor. This system was developed for four sensors only. If three sensors are used there will be no fault tolerance. This system will need to be adapted for N sensors in the future.

The following section covers the derivation of the algorithm to localise an emitter given a set of pulse arrival times. Figure 3 shows the curling rock at a location on the rink with unknown distances between it, and the sensors in the corners.

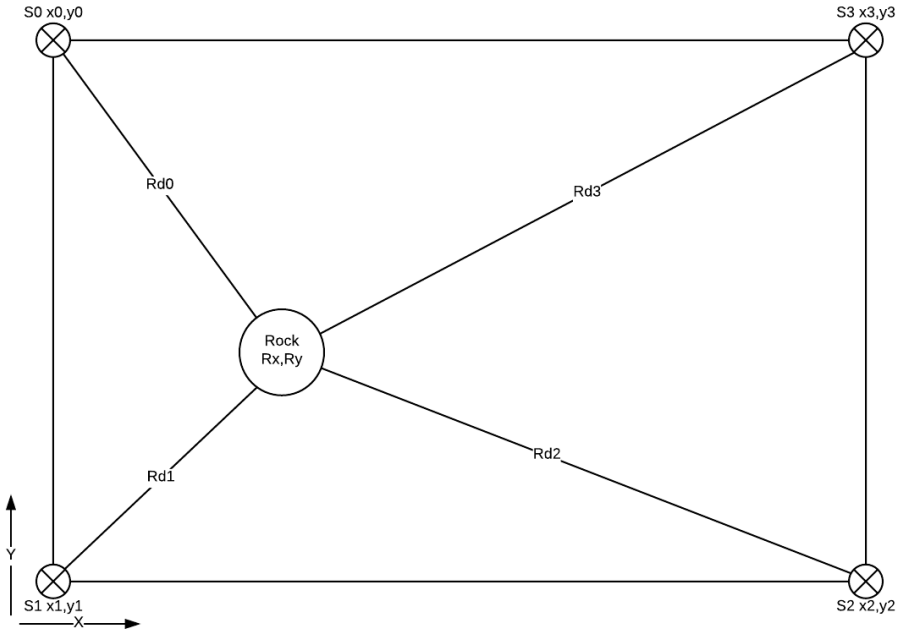


Figure 3: Example sensor and rock configuration

Table 1: Postioning Variable Table

Sensor number	i	Status
Rock pulse occurrence time:	R	unknown
Rock pulse arrival time for sensor i:	R_i	known
Rock distance from sensor i:	Rd_i	unknown
Pulse propagation time to sensor i:	$T_i = Rd_i$ / (speed of sound)	unknown
Sensor i internal delay:	D_i	unknown
Position of Rock:	Rx,Ry	unknown
Position of sensor i:	x_i, y_i	known

The synchronization can be assumed to produce the difference between internal delays of sensor i and sensor 0: $D_i - D_0$ = **Synchronization.** The rock-pulse arrival time is the sum of the rock-pulse fire time, the pulse propagation time, and the sensor's constant, internal delay. The rock-pulse fire time is unknown, which removes the need to have a communication system on the rock.

$$R_i = R + T_i + D_i$$

The difference between the received time for a sensor and the received time for sensor 0 is as follows:

$$R_i - R_0 = (R + T_i + D_i) - (R + T_0 + D_0)$$

$$R_i - R_0 = (T_i - T_0) + (D_i - D_0)$$

$$R_i - R_0 = (Rd_i - Rd_0)/(\text{speed of sound}) + \textbf{Synchronization}$$

The difference in distance from the rock to sensor i minus the rock to sensor 0 will be known after synchronization.

$$Rd_i - Rd_0 = (\text{speed of sound})*((R_i - R_0) - \textbf{Synchronization})$$

The algorithm requires simulation over the Euclidian grid of x and y to find (Rd_i-Rd_0)'s. The point in the grid with the closest Rd_i-Rd_0 values is taken as the location of the rock.

$$Rd_i-Rd_0 = ((x_i - R_x)^2 + (y_i - R_y))^2 - ((x_0 - R_x)^2 + (y_0 - R_y))^2$$

Simulation is performed to retrieve a location from a set of time delays. This is a least squares algorithm very similar to what was done by Yazici, Yayan, and Yücel [2]. The word simulation is used because the algorithm simulates the propagation delay for points on the rink, and then compares them to the received values to determine the rocks position.

Simulation of delays could be conducted for the desired resolution over the entire rink. For a rink of 4mx4m and a resolution of 1cm this would require 160000 simulations and comparisons of delays, clearly a computationally intensive operation.

$$\text{Resolution} = 1\text{cm}$$

$$\text{Simulations} = (4\text{m}/0.01\text{m})*(4\text{m}/0.01\text{m}) = 400*400 = 160000$$

Instead the system uses a zooming in functionality. The rink is simulated for a low resolution of 10x10 grid points. The best of these points is zoomed in on for a smaller x and y range. This smaller subset of the rink is then simulated with the same resolution of 10x10 grid points. This process is repeated until the desired resolution is reached. For a 4mx4m rink with a resolution of 10x10 and zooming in to ⅓ of the rink for every iteration, after 8 iterations the resolution is 1.8mm:

$$\text{Resolution} = 4\text{m}*(1/3)^{(8-1)} = 1.8 \text{ mm}.$$

$$\text{Simulations: } 8*10\text{x}10 = 800$$

Just 800 simulations results in a five times better resolution in 1/200th the simulation time. Through this method the time delays can be converted into locations on the ice in a rapid and accurate manner. A graphical representation is shown in Figure 4:

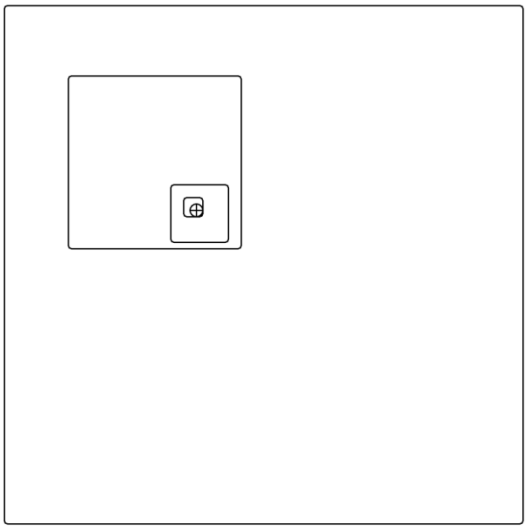


Figure 4: Example of 1/3 ratio zooming. Squares represent the simulated area. Circle is the actual location

3.4.1.2 Synchronisation

The purpose of synchronisation is to obtain a time reference across all sensors, which is applied to the data. The synchronisation of sensor's time references meant that the design could avoid any real-time hardware or telemetry systems. Using asynchronous hardware, it was possible to shift complexity to the post processing and use simple, cost-efficient hardware. The only physical component of synchronisation is the sync emitter. This emitter is placed at a known location. The input to this algorithm is asynchronous time stamps of the sync and rock pulses.

The method used for synchronization is a primitive form of Reference Based Broadcasting. Unlike the method covered by Elson, Girod, and Estrin [3] this project's network is a simple grid of N

sensors that all receive a pulse. It does, however, use acoustics across a grid to obtain a common time reference. This project's method uses N sensors located around the rink that are at known locations in an x,y,z zone. A single synchronisation emitter is located at a known location in x,y,z which is received by all sensors.

Assuming that there is a constant clock difference between sensor systems, the recording start difference is the result of the script starting at slightly different times on each sensor. This is because the recording script is initiated by the notoriously asynchronous internet protocol. Using the synchronisation-pulse, propagation delay and known locations, this difference can be measured to the microsecond and factored out. It is assumed that the recording is conducted at a constant rate. Only non-uniform sample rates can cause a drift after the initial synchronisation.

The propagation delay between the synchronisation emitter and each sensor is calculated using the distance multiplied by the speed of sound. This propagation delay between the emitter and each sensor is then subtracted from the recorded sync receive time for each sensor. This effectively removes the propagation delay between each sensor. The remaining delay is the difference between clock sources for each sensor. Each sensor's data is then corrected to the first sensor's time reference.

All sensors have unknown delays D_i due to buffers, ADCs, and recording start times. All of these are assumed constant throughout the duration of the recording. This can be assumed because the sound card records at a constant rate. The operating system pulls the data from the sound card in batches.

Table 2: Synchronization Variable Table

Sensor:	i	Status
Sync pulse fire time:	S	unknown
Sync pulse rec time:	S_i	known
Sensor internal delay:	D_i	unknown
Sensor distance from sync:	X_i	known
Sensor delay due to distance:	$X_{di} = X_i/(\text{speed of sound})$	known

The sensor recorded time is the sum of the sync-pulse emission time, the internal delay, and the propagation delay.

$$S_i = D_i + X_{di} + S$$

$$D_i = S_i - X_{di} - S$$

The internal delay of a sensor when compared to sensor 0 can be computed as follows:

$$D_i - D_0 = (S_i - X_{di} - S) - (S_0 - X_{d0} - S) = S_i - X_{di} - S_0 + X_{d0}$$

$$D_i - D_0 = S_i - X_{di} - S_0 + X_{d0}$$

Notice how the actual time that the sync pulse was emitted (S) becomes irrelevant. The system uses differences instead of actual times.

Once the synchronization, correction differences are calculated they are applied to the rock timestamps. Using the time that the synchronisation pulse was received, the resulting correction is applied to all data samples following the sync pulse until a new sync pulse is received.

In this project it was possible to correct for the different clocks at the sensors, assuming that each sensor would sample at the same frequency. In reality, this is not entirely valid. The system used three Raspberry Pi B+ systems and one Raspberry Pi Alpha. One of the sensors was a different version of the same hardware. This led to a very interesting drift in its synchronisation times.

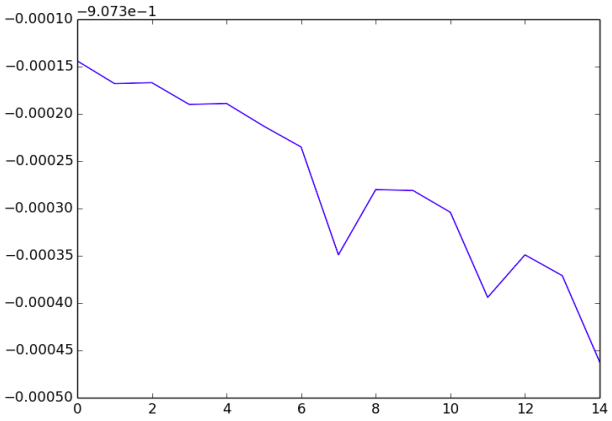


Figure 5: Raspberry Pi Alpha Drift. X axis time (s). Y axis delay (s)

As can be seen in Figure 5 the synchronisation allows for the measurement of sampling differences. In this case the delay appears to be drifting by 300µs over 14 seconds when compared to the other Raspberry Pis. Without synchronisation, this could lead to 10cm of error.

3.4.1.3 Pulse Detection

For pulse detection, a Goertzel filter is used along with a time-constrained acoustic pulse to time stamp the midpoint of each pulse. The time stamps are obtained every second, and are used as differences for the spatial algorithm. In ideal circumstances, this system is able to obtain time stamps with less than 114µs error. This results in a spatial error of less than 3.78cm, which is well within the specifications. It is also able to simultaneously detect 3 kHz and 6 kHz pulses with no interference. Note that pulse detection runs on every sensor after the recording is complete.

Before the pulse-detection algorithm can be explained, the acoustic pulses that it is detecting must be defined. Each pulse is a sin wave emitted at a specified frequency, centered at zero. The pulse lasts either 10 or 20 sin-wave cycles in length. At first glance, the harmonics of the pulse would appear to make it difficult to detect multiple pulses at different frequencies at the same time. However, through testing it was found that the magnitude of the harmonics is small enough that they have no effect on the detection of pulses with different frequencies, even if those frequencies are integer multiples. Therefore, 3 kHz and 6 kHz, or 4 kHz and 8 kHz, do no interfere with each other, validating their use.

In order to meet the requirements of the spatial positioning system, an algorithm is needed to reliably detect the arrival times of the acoustic pulses. It is essential that this system is consistent, as a timing error of a few samples can lead to errors of many centimeters in spatial location, and perhaps not meeting the required spatial accuracy of 15cm. It is important that the pulse-detection system is able to clearly detect acoustic pulses in the presence of ambient noise, such as talking and the noise of sliding rocks. Since the positioning system uses an acoustic sync pulse in addition to the rock pulse, the pulse-detection system should be able to detect both types of pulses without interference. Note that the positioning system uses difference-based time stamps from the sync pulses, as opposed to time of flight calculations from the rock emitter. This means that the pulse detector does not need to detect the beginning of the pulse, it only needs to detect a consistent point within it, such as the middle.

After some research into various pulse-detection methods, it was determined that a Goertzel filter would be an effective method of detecting a pulse of a specific frequency, while eliminating outside noise and remaining computationally efficient. The Goertzel theorem, as described by Zaplata and Kasal [4], is an infinite impulse response filter that maintains linear phase characteristics. It is a common algorithm used in tone detection systems, because it has a very narrow band in the frequency domain. This can be seen in Figure 6. Another more intuitive way to describe the Goertzel filter is as a single FFT-bin energy detector. Running a set of samples through a Goertzel filter will determine how much energy from that set is located in the frequency band of the Goertzel filter.

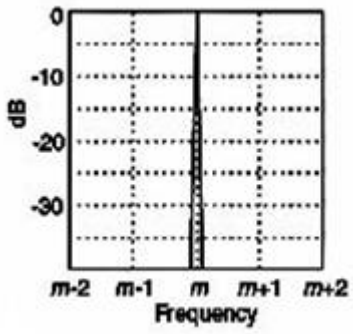


Figure 6: Frequency Bandwidth of Goertzel

Since the rock transmitters will be emitting a known frequency for their acoustic pulses, a Goertzel filter is ideal for detecting that frequency while eliminating outside noise. The Goertzel filter has been tested in various high-noise conditions to confirm its effectiveness. These tests were conducted by having an MP3 player emit pulses at a specified frequency (3 kHz, 4 kHz, 6 kHz and 8 kHz depending on the sampling frequency) every second with noise in the background. Some results can be seen in Figure 7.

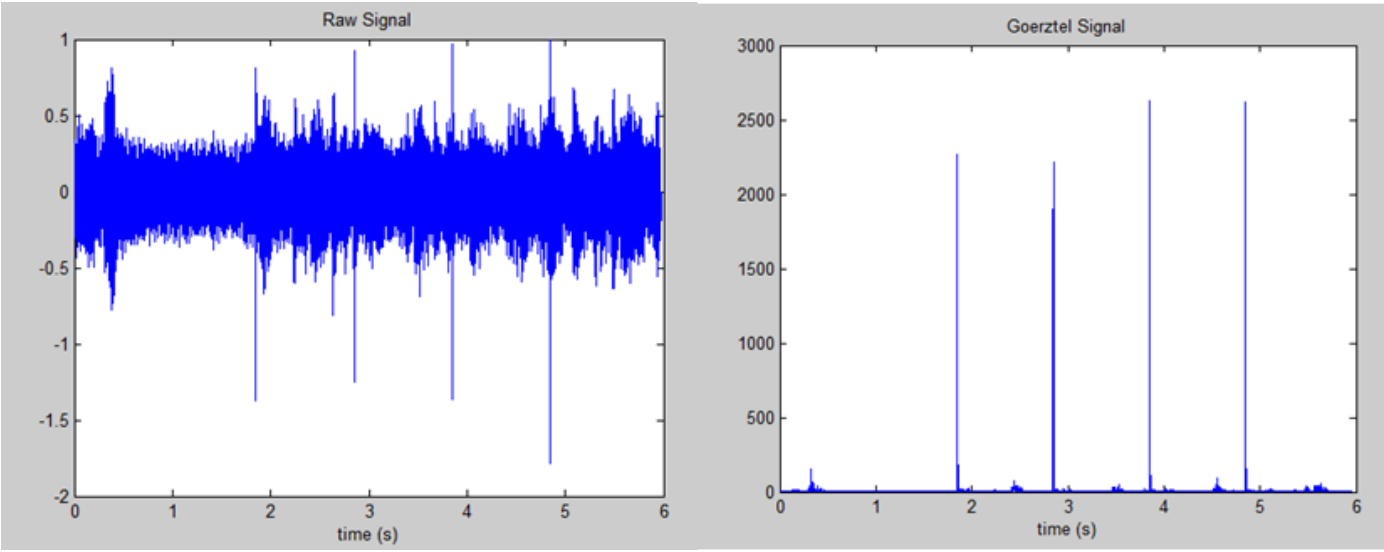


Figure 7: Raw Signal (Right) vs Goertzel Signal (Left)

It is easily seen that the Goertzel filter has no problem picking out pulses from the sound file. In some cases the pulse is not detectable to the human eye, but is still found by the Goertzel filter.

In order to provide more detail on the Goertzel filter, a description of the calculations required to set up the Goertzel filter at the required frequency and length are given. These instructions follow an article posted at Embedded.com [5]. In order to set up Goertzel filter, three numbers are required: the sampling frequency (f_s), the target frequency (f_{targ}) and the block size (N). The sampling frequency is normally set by the rest of the system, and in this case a sampling frequency of 44100 Hz was used. 44100 Hz was chosen because it is a common audio sampling rate that most recording hardware and software can use. The next selections are the target frequency and block size; however caution must be taken when deciding these two numbers. Since the Goertzel filter acts like a bin of an FFT, it is important that the targeted frequency is in the center of the bin in order to obtain the best results. For this to be true, the ratio as defined by the following equation must be an integer value:

$$\frac{f_s * f_{targ}}{N}$$

Through an iterative process, it was found that using N=147 and f_{targ}=3000 Hz or 6000 Hz performed well. It should also be noted that setting N and f_s determines the bandwidth of the Goertzel filter. For the rock-tracking system, the bandwidth is $\frac{fs}{N} = 300\ Hz$. Note that the roll-off for the Goertzel filter is very step. For a 300 Hz bandwidth Goertzel filter centered around 3 kHz, the magnitude of the response at 2800 Hz is down 14.85 dB from the center frequency, and 7.43 dB from 2850 Hz. At 2750 Hz, the magnitude of the response is well below -20 dB. This shows how even if the bandwidth appears to be large, the Goertzel filter can still effectively cut out all outside noise. The coefficients of the Goertzel filter are then set to the following:

$$k = floor(0.5 + \frac{N(f_{targ})}{f_s})$$

$$w = \frac{2\pi k}{N}$$

$$coeff = 2cos(w)$$

The iterative variables Q_1 and Q_2 are zero at the start. The algorithm then runs through the data for N samples using the following process:

$$Q_0 = coeff * Q_1 - Q_2 + sample$$

$$Q_2 = Q_1$$

$$Q_1 = Q_0$$

At the end of the block, the relative magnitude of the energy can be calculated using:

$$mag = Q_1^2 + Q_2^2 - Q_1 * Q_2 * coeff$$

This magnitude is then used as the relative energy of the targeted frequency within the block.

While the Goertzel filter is effective at detecting pulses, it cannot be used solely by itself to determine an arrival time for a pulse. This is because the Goertzel filter acts like an energy detector and

outputs a magnitude, therefore losing the phase information and eliminating the possibility of using an envelope detector. Also, because of the ambient noise and multipath effects, it can be difficult to determine the exact beginning, middle and end to the pulse. This can be overcome by shaping the transmitted pulse. An analogy would be to think of the Goertzel filter block length as a sliding window that moves along a received acoustic pulse longer than the block. As the Goertzel filter moves along the sampled pulse, the magnitude of the energy will increase until all samples within the block contain the pulse. The magnitude should then remain constant until the Goertzel filter begins to slide beyond the sampled pulses. At this point, the magnitude will go down. This is shown in Figure 8.

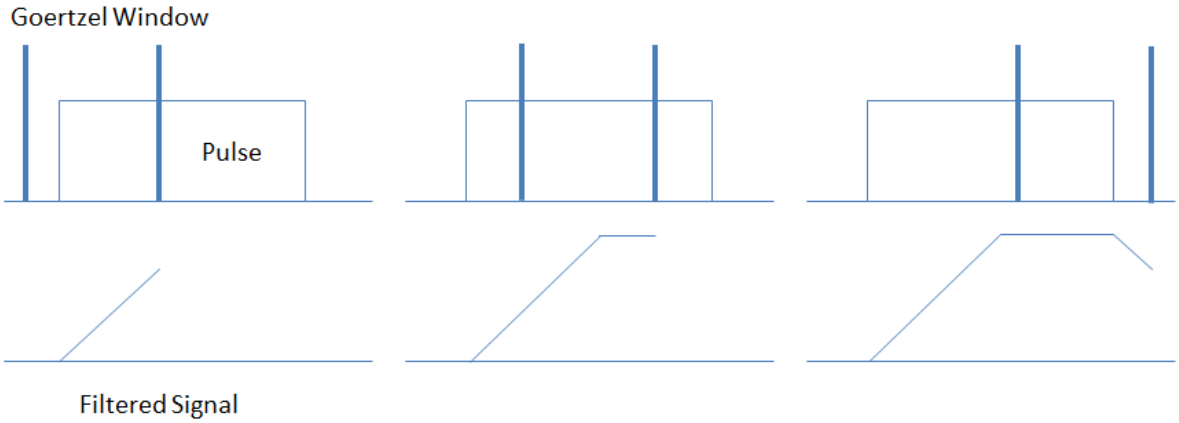


Figure 8: Graphical Representation of Goertzel Filter Pulse Shaping, Long Pulse

By ensuring that the transmitted pulse is exactly the same length as the Goertzel filter (N samples), the conditioned data should form a triangle, as seen in Figure 9. This peak will consistently occur in the middle of the pulse. The peaks can be easily found using a simple maximum function that occurs when a certain threshold has been passed. While determining the maximum is not an ideal method as it is susceptible to noise, it did function adequately well for this version of the system. Ideas to improve the peak finder can be seen in Section *3.8.2 Ideas for Future Improvement*. When this theory was tested in early September in minimal noise conditions, it was found that the midpoint of the pulse could be found with only 1 to 5 samples error. For a sampling frequency of 44100 Hz, this results in less than 114µs error, or a position error of 3.78cm. In actual trials multipath and clock drift caused additional errors, up to 1.11m, but these issues are mitigated in other parts of the rock-tracking system. For more information on how multipath and clock drift affected test results, see Section *3.5 Testing*.

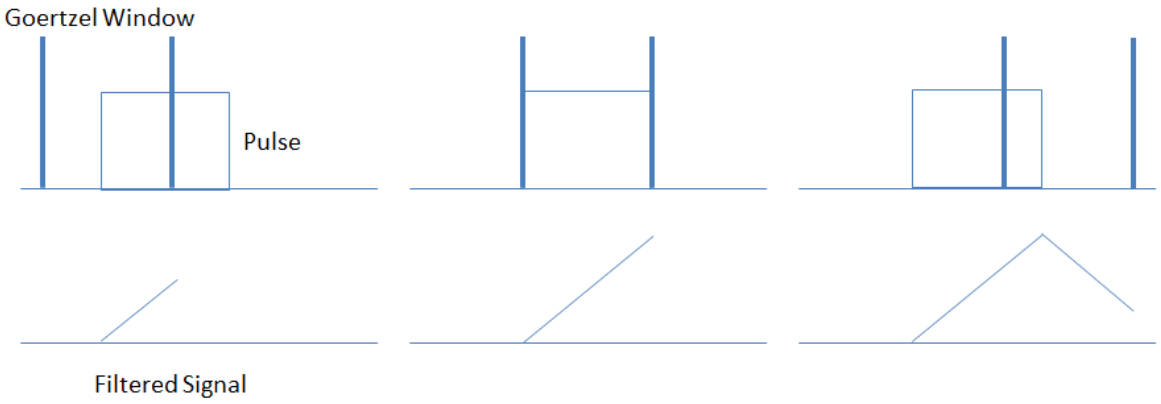


Figure 9: Graphical Representation of Goertzel Filter Pulse Shaping, Equal Pulse

For the prototype, it was decided to use a sampling frequency of 44100 Hz as it is one of the most common sampling frequencies supported by recording systems, and because it provided a decent temporal resolution of 22µs. This led to a block length of 147 samples, a rock emitter frequency of 3 kHz, and a sync emitter frequency of 6 kHz. It was found through testing that 3 kHz and 6 kHz pulses can be run simultaneously with no interference. While beat frequencies could prove a problem, as the difference between the sync and rock emitter frequencies is 3 kHz, no issues of this nature were found during testing.

Doppler shift could be a problem because as the rock moves down the sheet, it causes a Doppler shift on the emitted pulse. Since the motion is always changing, this could, in principle, cause some errors with an envelope detector as the frequency is also changing slightly. In fact, since the rock moves at a maximum speed of about 3m/s and the speed of sound in cold air is about 333m/s, a 3 kHz pulse could shift by as much as 26Hz. However, because the Goertzel filter bandwidth is set to 300 Hz, the shifted frequency is still easily detected, with no errors induced.

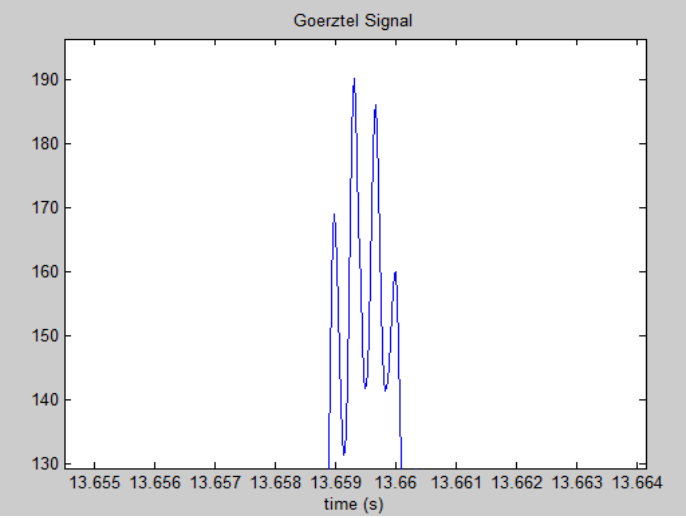


Figure 10: The Effect of Multipath on Multiple Peaks

While this system is effective at determining consistent timestamps for the acoustic pulses, it does make a few assumptions that can cause errors. The first is the precision of the rock emitter. The acoustic pulse must be 147 samples long, with a maximum error threshold of 1 sample (or 22.7µs at 44100 Hz sampling frequency). This means that the rock emitter must be extremely consistent, and resistant to drift. Any errors would cause a flat top in the conditioned data, allowing noise to dictate the maximum and induce more errors.

Another assumption is that the multipath effect is relatively small or is delayed by longer than the length of the Goertzel filter. Since multipath occurs at the same frequency as the transmitted signal, it can very easily corrupt the Goertzel filter and cause double or triple peaks, in which the multipath peak can be higher than the initial pulse. This can cause a large error, as seen in Figure 10. To mitigate this, the peak-finding function only looks over a short period of time to determine the time-stamped maximum. The theory is that the initial impulse will always occur first, and trigger the threshold. Then the initial maximum will be sampled and the sensor locked out for 0.75 seconds to avoid false multipath detections. This strategy is good for solving most multipath issues, however multipath that occurs a short delay after the initial pulse can still corrupt the signal. To mitigate the situation, the sensors and the emitters should be placed far from the walls. This would cause longer multipath delays, which are easily ignored.

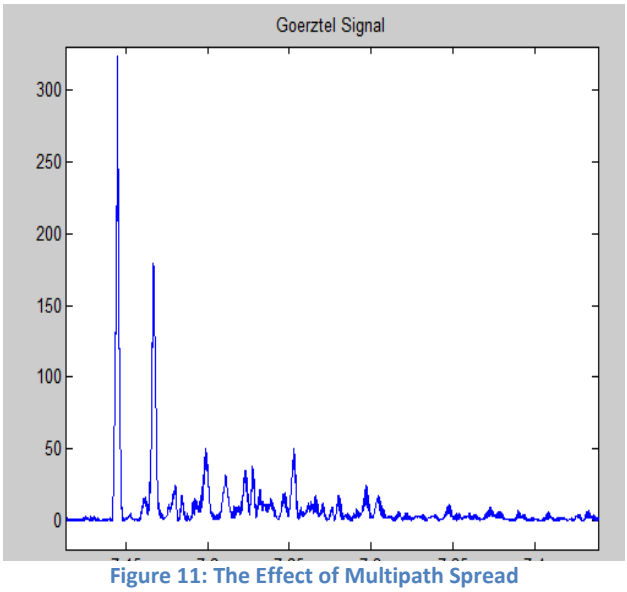


Figure 11: The Effect of Multipath Spread

Multipath sets the minimum period between pulses. To ensure good results, a pulse should only be emitted once all multipath from the previous pulse has

dissipated. As can be seen from Figure 11, this can take up to 0.3 seconds. The pulse emitting frequency was set at 1 Hz to overcome this, with potential to increase this frequency to 2 Hz in the future.

The final assumption is that noise will not trigger the peak-finding function. Currently, the threshold is set to a static level. While the Goertzel filter is effective, it is still an energy detector. A large amount of energy, even outside of the band of the filter, can cause the magnitude to rise and the threshold to be triggered. An example of a noise that could trigger the threshold would be a clap. This is because a clap acts like an impulse, spreading energy throughout the entire frequency spectrum. The energy put in the Goertzel filter's bandwidth would likely cause a trigger. Loud noises close to the sensors can cause an issue. This is because it would cause the sensor to clip, which produces a large amount of energy throughout the entire frequency spectrum. However, since one of the conditions of this prototype is a low-noise, testing environment, the amount of loud sounds during test runs can be controlled. Methods to improve the performance of the system in higher-noise conditions are detailed in Section *3.8.2 Ideas for Future Improvement.* It should be noted that the sync and rock pulses do not interfere with each other due to their frequency spacing.

Overall, the use of the Goertzel filter with a time-limited emitted pulse results in consistent time stamps for the acoustic pulses. While there are some errors due to multipath effects, overall both accurate and precise time stamps can be obtained for at least two frequencies at the same time.

3.4.2 Hardware

The following section outlines the hardware necessary to localise a curling rock using acoustic, propagation-delay calculations. A full system can be seen in Figure 13.

3.4.2.1 Receivers

For the alpha prototype of this project, four receivers are used. The current positioning algorithm requires at least three sensors for a 2D plane. All algorithms were developed with the possibility of using many more sensors in later revisions of the system.

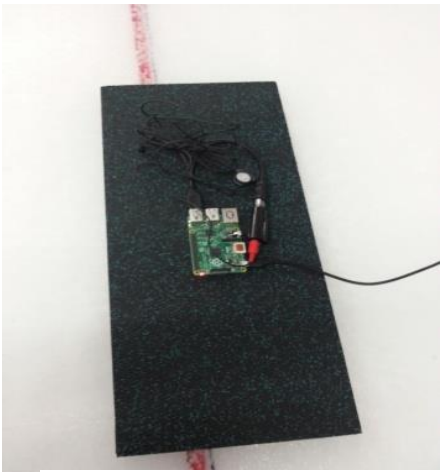


Figure 12: Picture of Sensor Setup

The receivers consist of a microphone, a USB sound card, a Raspberry Pi, and a USB Wi-Fi card. The microphone picks up the audio which is then sampled with the USB sound card. The Raspberry Pi temporarily stores data and performs pulse detection to generate pulse times. The Raspberry Pi is used as a server for the pulse-time data. It is controlled and transmits its pulse times over the Wi-Fi card. A picture of the system can be seen in Figure 12. It should be noted that at this time, there is no enclosure for the sensors, and that curling rocks are used to elevate the microphones. Suitable enclosures are necessary before regular use.

3.4.2.2 Emitters

There are two emitters needed for the system to be operational. One emitter is located on the curling rock to be tracked and the sync emitter is placed at a known location. Additional rock emitters could feasibly be added to the system and operate at different frequencies. This would allow for the tracking of multiple objects. The maximum amount of objects that can be tracked has not yet been determined, but as long as the rock-emitter frequencies are spaced over 300 Hz apart, the system should function.

The emitters consist of an amplified speaker and an MP3 player. The MP3 player plays a pre-generated string of pulses spaced by ½, 1 or 2 seconds. The MP3 player must be able to play an uncompressed .wav file. Compression can distort the transmission signal which can harm the performance of pulse detection. The signal is amplified by the speaker so as to be audible across the whole rink. The rock emitter is attached to the top and centre of a curling rock with Velcro.

Figure 13: Picture of Full Setup

3.4.2.3 Auxiliary Equipment

In addition to the emitters and receivers, there is a router that is located within Wi-Fi range of the receivers. The receivers are configured to automatically connect to, and authenticate with the router on start-up. The receivers are configured with static IP addresses to allow for simple networking (No DNS).

3.4.3 Software

The software is separated into two groups. The client application that runs on the user's machine, and the server application that is always running on each of the receivers. The client application handles user interaction. The server application records data, runs the Goertzel filter and detects pulses. The two applications communicate with each other through basic socket messages. Figure 14 presents the flowchart for the software.

3.4.3.1 Client Application

The client application uses python. It runs on the client machine or one of the receivers. The IP address of the client application does not need to be known. The system that runs this application must be connected to the same router as the receivers. The order of operations is listed below:

1. The client application first prompts the user for the desired running time in seconds. This will be the duration of recording.
2. The client application then sends the duration as a message to each of the four servers. The servers begin recording.
3. The response from each of the servers is received. It contains the detected pulses.
4. The pulses are combined and the sync and position algorithms are performed on the data using python.
5. The output of the position is made available to the user through the console and a plot of the rocks path.

After completion the data can be inspected and modified directly from the python console.

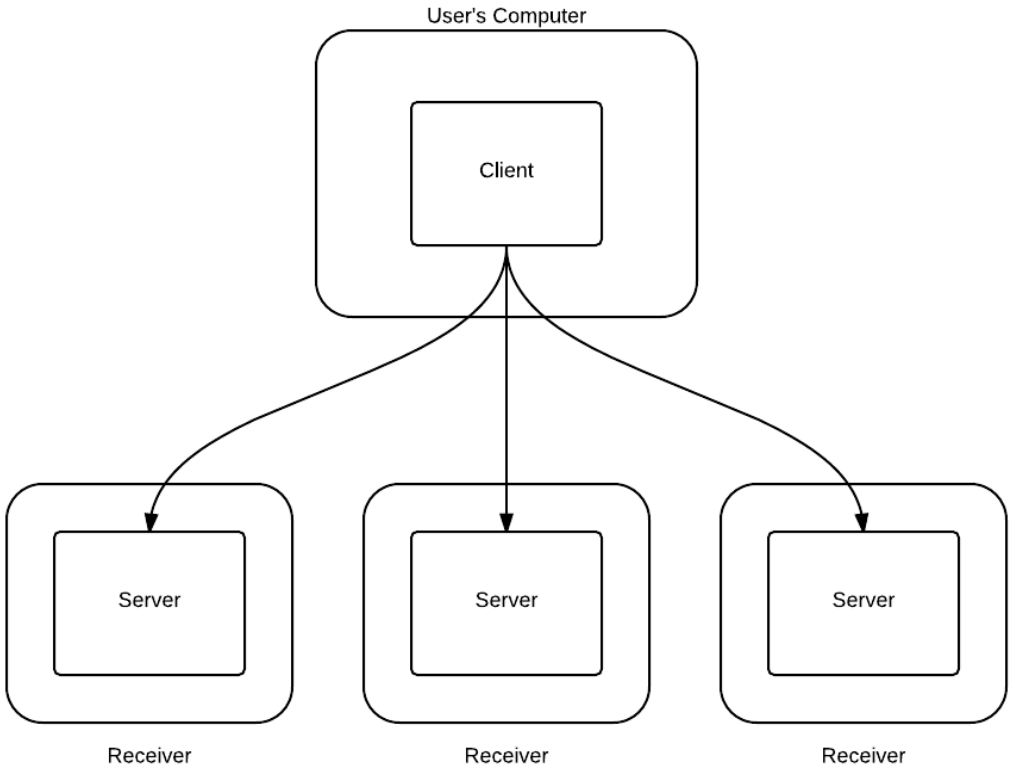


Figure 14: Software Connections

3.4.3.2 Server Application

The server application is run on each of the Raspberry Pis. It is started up before any recording is done. The server does not need to know the IP address' of any of its clients. The order of operations is listed below:

1. The server binds and listens to its IP address.
2. When a message is received from a client it is checked that it is a valid duration.
3. The recording script is then called from python with the duration argument set as the received, desired recording duration.
4. The results of the recording script are run through a formatting script.
5. The formatted results are then passed through the pulse-detection software which has been optimized by being written in C.
6. The pulse-detection results are read into python and are sent as a response back to the client.
7. The server application then waits for another request for data.

The server application should be run from an SSH session with the receiver. This way any errors or issues can be observed by the user.

3.5 Testing

This section details the major tests that were completed during the second half of the project. There are seven tests described that showcase the evolution of the rock-tracking system. Each test description includes a purpose, procedure and results section, along with comments on what was learned or what needed to be improved.

3.5.1 Algorithm Proof-of-Concept Test

The purpose of this test was to ensure the validity of the pulse detection and spatial positioning algorithms at the curling club. This test was completed early in the term, before any of the hardware had been ordered. It was important to confirm the algorithms at a general level before any of the hardware was ordered. Cell phones with the "Easy Voice Recorder" application were used as sensors, sampling at 44100 Hz. At this time, the algorithms did not include any sync pulses or pulse shaping. These were developed later in the design process.

Figure 15: Linear Test Setup

The test procedure consisted of running two different experiments. The first experiment was a linear test using two cell phones as receivers spaced 10m apart and a 3 kHz buzzer as the emitter. The buzzer was moved 1 meter at a time down a line, buzzing once at each meter marker (see Figure 15) except the first and last markers where it buzzed twice. The second buzz at these two points was used for synchronisation. The sound files from the cell phones were transferred to a computer where they were fed through the program to get the results.

The second experiment used a 2D layout with four phones as receivers in the corners of a 4x4 meter grid as shown in Figure 16. By moving the buzzer from intersection point to intersection point, the combination of the four recordings should have allowed mapping of the path taken. The data files were sent to the laptop for analysis.

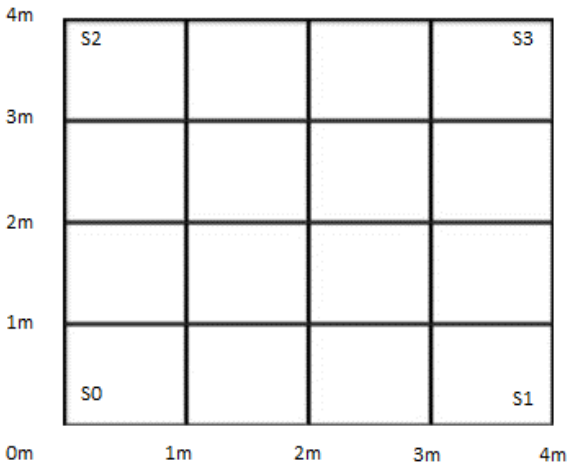


Figure 16: 2D Test Setup

The results from the tests can be seen in Appendix A. For the linear test, the differences between the two sensors were used to calculate the relative distance between them, after syncing to the first point at a known location. The linear results showed that there was potential with the Goertzel-filter pulse detection, as all the data points were within a reasonably close area to the actual value. However, the maximum error (77cm) was much higher than specification, and the error was increasing for every data point, showing some drift. The 2D tests also showed some promise. Excluding a couple of misfired data pulses, the results were within 71cm of the actual position. The extra sensor helped provide some redundancy to get better data than the linear test. However, once again the error drift was detected.

Much was learned from this initial test. In terms of the Goertzel filter, it was determined that the main issue was that the buzzer was sounding for an unknown and inconsistent amount of time. This led to a flattening of the Goertzel-filter magnitude, allowing noise to affect the values and making it difficult to detect a proper maximum (see Figure 17). Also causing issues was the inconsistency of the buzzer frequency. Not only was the Goertzel filter not optimized for 3 kHz, but the buzzer frequency would shift throughout the emission. This caused less precise data. It was decided that a more precise method of controlling the pulse frequency and length would be needed. Another lesson was the need for a synchronising mechanism. The error drift was likely due to imprecise sampling frequencies between sensors, and synchronising with the first pulse is an undesirable solution, as a bad first sample will cause errors throughout the rest of the run. A method for resynchronizing the pulses throughout the run was needed.

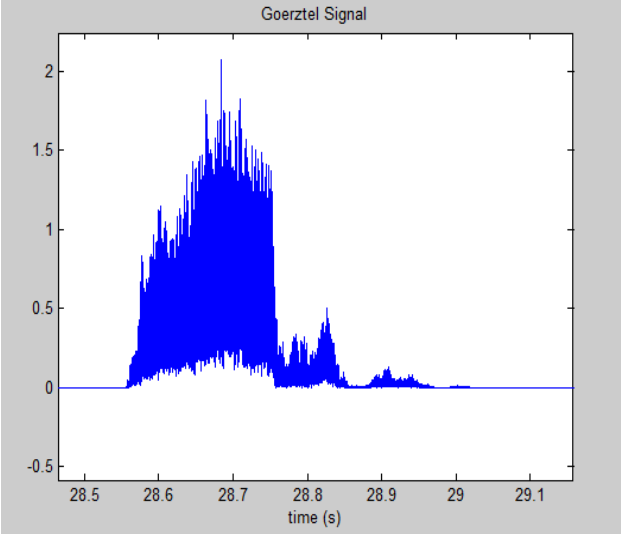


Figure 17: Unusable Pulse

Overall the algorithm proof-of-concept test was valuable in demonstrating the ability of the rock-tracking algorithms, while simultaneously showing areas where performance needed improvement.

3.5.2 Controlled Emitter Test

This purpose of this test was to repeat the linear test using a more controlled emitter. Using cell phones sampling at 32000 Hz as receivers, the setup shown in Figure 18 with 7m spacing was constructed. However, instead of a buzzer, a cell phone playing a pre-constructed .wav file was used as the emitter. The .wav file emitted 4 kHz pulses for exactly 4ms (corresponding to 128 samples in the Goertzel filter) spaced exactly 2 seconds apart. The emitter was then moved between the receivers at 1m intervals, beeping twice at each location. The double emission per location allowed for a comparison between two data points at the same location.

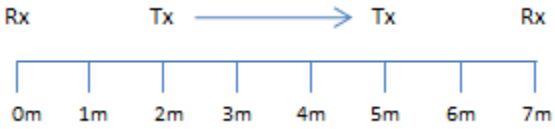


Figure 18: Second Linear Test Setup

The results for this test can be seen in Appendix B. Excluding the one outlier, all results were within 6cm of their actual locations, well within the specification for spatial accuracy. The outlier point has an error of less than 23cm which is a much better result than most of the earlier tests. This shows the effectiveness of the Goertzel filter, and how it can be reliably used to achieve results within specification. However, it was that one outlier point that showed where the algorithm could be further improved.

Upon further examination of the outlier data point, it was found that the reason that the error was so large was due to a double peak, in which the second peak was much larger than the initial one as shown in Figure 19. It is posited that this is due to the multipath effect, which makes sense as this test took place next to the wall of the curling club. In fact, if the initial peak is used, the error drops down to 6.5cm. A method was needed to circumvent the multipath problem.

3.5.3 Raspberry Pi Receiver Test

The purpose of this test was to test the functionality of the Raspberry Pi as a receiver. This test was completed in a living room and made sure that the Raspberry Pi did not cause any errors in the positioning algorithms.

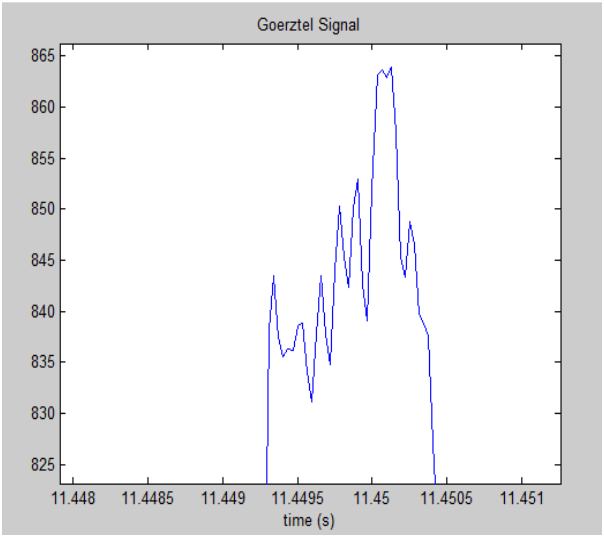


Figure 19: The Effect of Multipath

For the procedure, the four Raspberry Pis, which will henceforth be referred to as the receivers, were set up around the living room with sound cards and microphones. Each receiver sampled at 32 kHz as a stationary emitter played 4 kHz pulses for 4ms at 2 second intervals. After recording, the receivers transmitted their .wav files via a wireless network to a laptop for further processing. Using this method, it was possible to test the precision of the receivers from their recordings by determining how far the peaks were from the initiating and next pulse. The wireless capabilities of the receivers were also tested. The experiment was run several times to ensure reliable results.

In Appendix C, the results for one of the receivers can be seen. Since all the receivers had similar results, only one sensor set is needed. It is clearly evident that none of the pulses have proper 2 second spacing, which may at first seem like a distressing result. However, the offset time delay is consistent, always around 1.78125ms per 2 seconds. What this means is that the Raspberry Pi and sound card combination are not sampling at the correct frequency. Instead of sampling at 32000 Hz, they sampled at 31971.5 Hz. This result is consistent with most of the receivers at both room temperature and rink temperature. This error is likely due to a lack of accuracy in the sound card. When the actual sampling frequency is used in the Goertzel filter, the results are much better. The receivers now have almost exactly 2 second pulse spacing, with less than five samples in error across two trials. This means that the receivers can accurately record data to within 5cm error once the sampling error is accounted for. Also, the wireless capabilities of the receivers showed promise, as all of the receivers were able to access the wireless network and transmit data over it.

While learning that the sound card samples at the wrong sampling rate was an important fact, the method of fixing it can still lead to some error. Since it cannot be assumed that the sampling frequency for all future versions of the Raspberry Pi and sound card combination will be offset by the same amount, it is important that the system has a method to adjust sensors that could be sampling at an unexpected frequency. This reinforces the need for a synchronization function that is able to correct the incorrect sampling times before they get out of hand.

3.5.4 Raspberry Pi Linear Test

The goal of this test was to determine the accuracy of the Raspberry Pi receivers in the linear test. This test would permit estimating the temporal accuracy of the Raspberry Pis by using distance as a surrogate for time.

Two Raspberry Pis were arranged as receivers with 10m spacing between them. Each Raspberry Pi sampled at 32 kHz and had the microphone raised above the ice using a curling rock. A cell phone, which was used as an emitter, played a .wav file with 4 kHz pulses lasting 4ms with 2 second spacing.

The emitter was moved at 1m intervals between 3m and 7m, emitting twice at each location. At the end, the .wav files were transmitted wirelessly to a laptop for analysis.

The results of the test can be seen in Appendix D. It is easily seen that all points are within 11cm of their actual position. This means that the spatial accuracy using the Raspberry Pis are within the specified accuracy of 15cm. Also, note that the mean error is 1.28cm, showing that the results are consistent and do not heavily drift towards one side or the other. This demonstrates the effectiveness of the Raspberry Pis as receivers.

One important development from this test was multipath protection. Since this test was run next to a wall, double peaks were a common occurrence. This gave many of the data points huge errors that considerably exceeded the specifications. To avoid this, the reduced maximum-lookout function was introduced. Instead of finding the peak in a one second interval, the maximum finder only looks for a peak in a short interval (4ms) after the threshold is crossed. The receiver was then locked out for 1.75 seconds to avoid multipath triggers. This reduced the larger errors to within specification values, and even reduced some errors to nearly zero.

While the results were promising, the cell-phone-based linear test outperformed Raspberry-Pi linear test. While this is true, the difference may be due to the longer range of the Raspberry-Pi test (10m) versus the cell-phone test (7m). At longer ranges, the relative magnitudes are smaller, and noise and multipath can more easily corrupt the signal. Therefore, the degradation in performance may be an indicator of distance only, not the hardware's capability.

3.5.5 Precision and Motion Test

The purpose of this test was to experiment with the 2D spatial accuracy using the Raspberry Pis. Since the results for the Raspberry-Pi linear test showed reasonable spatial accuracy, it was important to see if this carried over to the second dimension. This was also the first series of tests that implemented a synchronization pulse at a known location to help avoid clock drift. Finally, it was during this series of tests that the emitter was attached to a rock with Velcro and thrown down the ice. The motion tests were run to determine if the system could handle a moving emitter and still provide reasonable results.

Three different trials were run. In all trials, the four sensors were set up in the corners of a 6.5x4 meter grid (see Figure 20) and all sampled at 32 kHz. The rock emitter, consisting of a speaker and iPod, played 4 kHz pulses for 4ms intervals every 2 seconds, and the sync emitter played 8 kHz pulses for 4ms intervals every 1 second. The sync emitter was located at (3.5m, 0m).

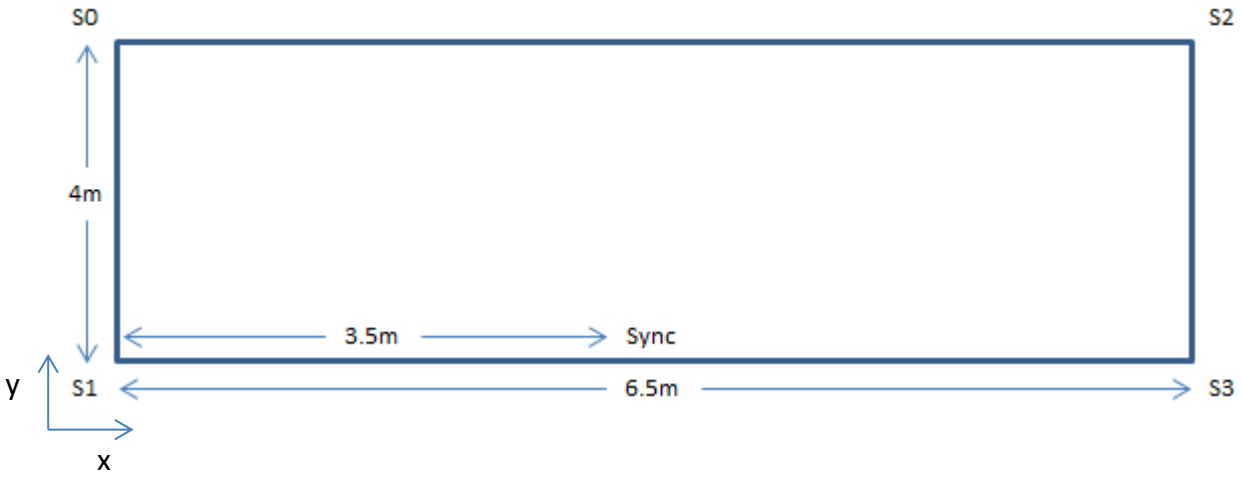


Figure 20: Testing Setup for Precision and Motion Test

The first test was a precision test. The emitter was left at a known location (2.75m,2m) for an extended period of time to determine how much clock drift would affect the system. It should be noted that during this test, no people were inside the testing area.

The final two tests were motion runs. The speaker and iPod were attached to the edge of the curling-rock handle with Velcro. The rock was then pushed through the testing grid. At the same time, the pushed rock was videotaped. The data points from the system were then compared with the video to determine if the results coincided with the trend of the rock path in the video.

The results of the tests can be seen in Appendix E. The precision test showed good results, as all the data points are clumped together and close to the actual location. While the data trends around (2.75m,1.65m) and the actual location is (2.75m,1.8m), the difference may be due to some measurement errors when constructing the grid. It appears that there is no drift error present. It should also be noted that the worst data point at 12.64 seconds occurs at the same instance as a loud impulse, like a clap. The error may be due to the clap causing a large amount of noise across the entire frequency spectrum, triggering the Goertzel filter for all sensors and giving a false data point. Therefore, this point may not be indicative of the rock-tracking system in an acoustically quiet environment. However, the point is indicative of how sensitive the system is to impulse noise. Assuming the point at 12.64 seconds is due to the clap and can be removed for an acoustically quiet environment, the standard deviation of the x and y positions of the rock are small, 5.41cm in the x direction and 5.96cm in the y direction. This shows the precision of the system.

The motion tests' results were good. While the actual position of the rock is unknown, the trend of the rock path from the video can be compared to the data points. For both motion trials, the rock path from the tracking system is similar to the videotaped rock path, in both direction of the throw and ending location. This demonstrates that the system works for a moving emitter.

Throughout this series of tests, a couple other observations were made. The first was that the rock-tracking system consistently had unacceptable errors when there were people in the testing area, as opposed to being well within system specifications otherwise. This may be due to the people blocking the direct acoustic path from either the rock or sync emitter to any receiver. All tests run after this time are conducted with no people in the testing area. While this is disappointing, as the system may not work with sweepers present, it still meets the objectives as set by the client. Ideas to mitigate this are given in Section *3.8.2 Ideas for Future Improvement.*

The other observation was the placement of the speaker on the rock. Using Velcro to attach the speaker to the edge of the rock handle caused some wobble in the results, as the speaker was spinning around the centre of the curling rock. Putting the speaker in the middle of the rock should reduce the amount of wobble in the future.

Overall, this series of tests showed how effective the rock-tracking system could be for spatial positioning during motion, as long as there are no interfering bodies in the way.

3.5.6 Hog to Hog Test

The purpose of this test was to 1) determine the range limit of the rock-tracking system, and 2) determine the performance with the sampling frequency set to 44100 Hz in an effort to remove sampling-frequency error. In this situation, the four sensors were set up at the corners of the hog line and the side line, a 21.82x4.15 meter section as shown in Figure 21.

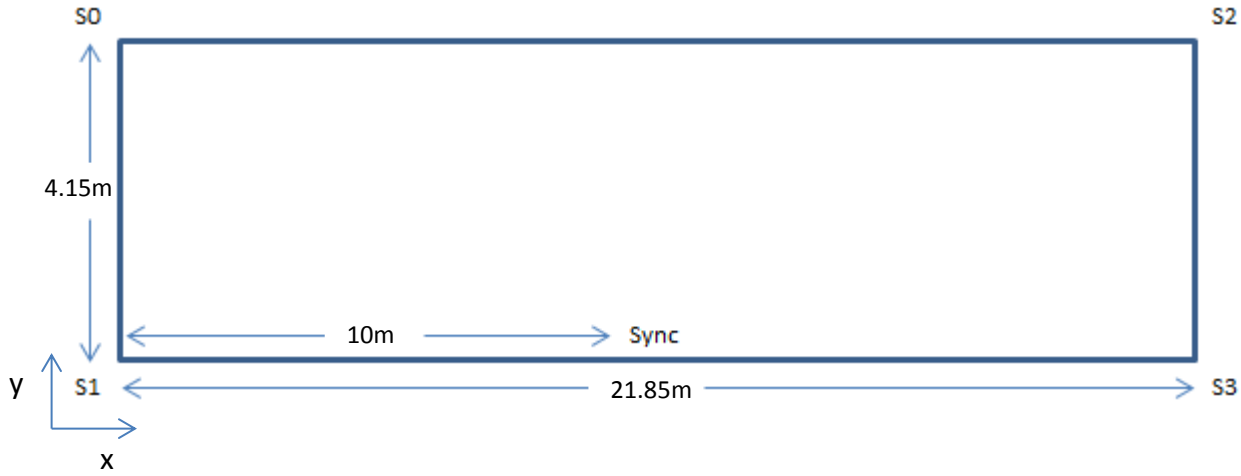


Figure 21: Testing Setup for Hog to Hog Test

The sensors sampled at 44100 Hz. The rock emitter played 3 kHz pulses for 3.33ms at 1 second intervals. The sync emitter played 6 kHz pulses for 3.33ms at 1 second intervals. The sync was located at (10m,0m). Both a stationary and a motion test were run. The stationary test had the rock emitter located at (10m,2.18m). The motion test was a full sheet throw of the rock. For both tests, pulse detection was run on the Raspberry Pis themselves, as opposed to on laptops. Note that the emitter was attached to the centre of the rock handle for this test.

The results for this test can be seen in Appendix F. It is quickly evident that the results are not good. For both tests, the amount of error was so high that the system discarded the data for being unreasonable! It is posited that the reason for this poor performance was the distance at which the sensors were set up away from each other. The distances in this trial were much longer than any previously tested distance. Because of this, the energy of the pulse was reduced by the time it hit the far sensors. This was especially apparent during the motion test, where the rock could be over 21m away from two of the sensors. The reduced energy led to a smaller peak from the Goertzel filter, which may have made it more susceptible to noise, potentially causing the error. Therefore, the hog line to hog line setup is outside of the maximum range of the rock-tracking system.

In terms of the new sampling frequency, it was found that the sound card sampled incorrectly, using 44048 Hz instead of 44100 Hz. While it was disappointing to see that no improvement was made, the decision was to keep the 44100 Hz sampling frequency because a higher sampling frequency gives a smaller temporal resolution, which can lead to a better spatial resolution. For 32000 Hz, the smallest time difference was 31µs, which leads to a spatial resolution of around 1cm. For 44100 Hz, the smallest time difference is 22µs, leading to a spatial resolution of around 0.75cm.

The other aspects of the test worked well. The automation simplified the testing process and worked well. For future tests, the testing area will be kept much smaller to get better results.

Possibilities for improving the range of the system can be found in Section *3.8.2 Ideas for Future Improvement*.

3.5.7 Final Test

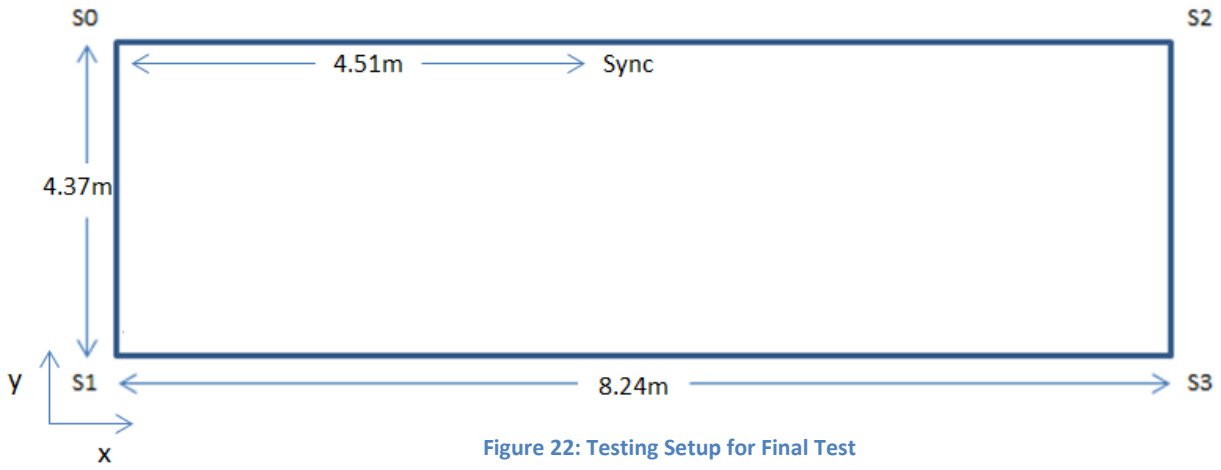


Figure 22: Testing Setup for Final Test

The purpose of this test was to perform a final demonstration for the client to ensure that the system met his specifications. The sensors were arranged in the corners of an 8.24x4.37 meter grid as seen in Figure 22, essentially the space between the back line of the house and the hog line. A stationary test was used to show the spatial accuracy and precision of the system. A motion test was then used to demonstrate how the system works in practice, and the results of the motion test were compared with a video recording of the trial as well as a measurement of the last point to show accuracy. Note that the sensors are sampling at 44100 Hz, the sync is emitting 6 kHz pulses for 3.33ms every second and the rock emitter is emitting 4 kHz pulses for 3.33ms every second. The one exception is the last stationary trial, as the rock emitter is then playing pulses every half second, in an effort to retrieve more data. The sync was located at (4.51m,4.37m).

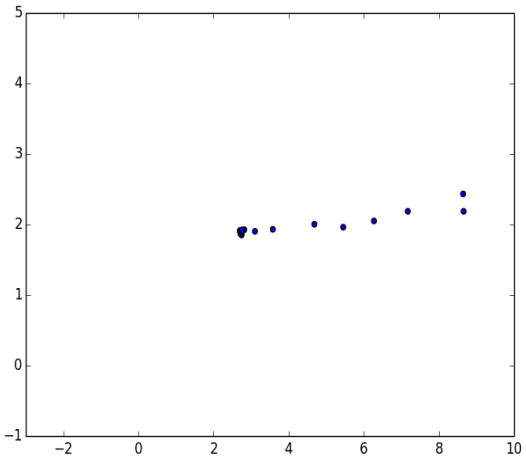


Figure 23: Motion Test Results

The results of the test can be seen in Appendix G. All completed tests showed that the system is ready and meets specifications. For the first stationary test, when comparing the results with the actual position of (3.68m, 2.19m), it is easily seen that calculated points are all clustered around it. The system typically drifts up to 5cm in either direction or axis, with the largest error at around 10.34cm. However, all of these points are within the specified spatial accuracy of 15cm, and the median spatial error is 4.17cm. Therefore, the acoustic rock tracking system meets the required spatial accuracy.

The motion test also showed good results. As can be seen from Figure 23 all the plotted points follow a legitimate path. After comparing the results with the video, the plotted points do follow the videotaped path to the client's satisfaction. Also, the last point was measured to be at (2.78m, 1.92m). When compared to the last point in the trial, the error is 2.92cm,

well within specification and demonstrating the accuracy of the system during a real run. Therefore, the acoustic, rock-tracking system is able to work in the conditions set by the objectives.

The half second trial was extremely successful, as it demonstrated that the system could be run at higher data frequencies than 1 per second. Having measured the rock's location to be (3.6m,2.27m), the system showed less than 4.21cm error in any direction with a median spatial error of 2.34cm. Therefore, the system shows considerable potential for working at a data collection rate of 2 position estimates per second, much higher than the planned 1 per second. Upgrading the system to work at this rate all the time may be possible before the final presentation. The graph of the data can be seen in Figure 24.

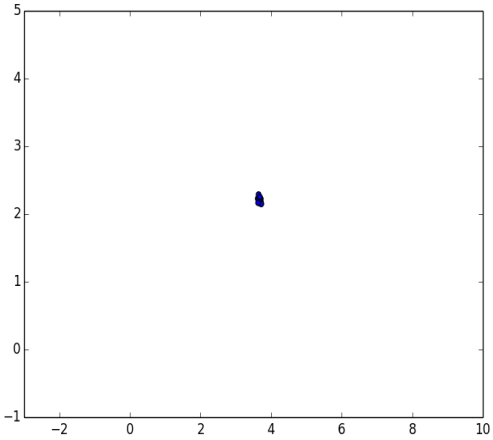


Figure 24: Half Second Test Results

Finally it should be noted that the system was almost entirely automated. All that was needed was for a user to input the recording duration, and the system ran and output a text file to a laptop that could be imported into Excel. This simplifies the system for the user, gets the data in a workable format, and helps with the portability of the system as a whole.

Overall, this final test set showed that the acoustic, rock-tracking system meets all the required objectives. The system is able to work for a single rock on a section of the rink and has a median spatial resolution of 4.17cm with no error larger than 10.34cm. It was able to work in typical curling conditions, sending the data to a laptop as a .csv file for reading by Excel. Finally, the system has lots of future potential in terms of its data collection rate.

3.6 Budget

Table 3: Budget

Name	Price per unit	Quantity	Total Price
USB microB Cable - 6 Foot	$5	3	$15
Wall Charger - 5V USB (1A)	$3	4	$12
Raspberry Pi - Model B+	$40	3	$120
EDIMAX EW-7811Un N150 USB 2.0 Wireless nano	$15	3	$45
StarTech ICUSBAUDIOB 2 Channels USB Interface	$15	3	$45
Kingston 8 GB Secure Digital High Capacity (SDHC) - 1	$10	3	$30
ZALMAN ZM – Mic1 Black 3.5mm Connector High	$19	3	$57
Speaker	$9	2	$18
Wav Player (Estimated)	$20	2	$40
Estimated Total			$382
Actual Total			$342

The requirements of the project were a budget of $400 and an eight month timeframe to design and implement a rock-tracking prototype. Overall, the project was within budget, as the prototype cost $342 to build. This works out to be $108 per sensor and $29 per rock/sync emitter. It should be noted though, that the current prototype makes use of a few personal parts, including one of the four sensors and both .wav players. These parts will be unavailable after the completion of the term. Therefore, to construct a three-sensor system similar to the design defined in Section *3.4.2 Hardware*, additional .wav players will have to be purchased. This brings the total cost up to the estimated value of $382, still within the budget. Within these budget constraints, the prototype equipment can handle a reasonable area from the hog line to the backline, which is about 8.24x4.37 meters.

In the future, additional resources should be spent in two areas. Adding more sensors will help gather more information to help increase the accuracy, as well as provide additional range. The other area of improvement is the parts used in the sensors. This is especially true for the microphones (which would increase range) and the sound card (to fix sampling errors).

3.7 Schedule

At the beginning of the term a Gantt Chart was created. This can be found in Appendix H. Overall, the actual progress followed the schedule very well. This left lots of time for testing the prototype and debugging problems. A few highlights are listed below.

In September, the objective was to focus on building algorithms for pulse detection and spatial positioning. This concluded with proof-of-concept test runs on September 13 and October 4 that demonstrated the potential of the algorithms, and allowed ordering parts to commence.

In October, the objective was to build a prototype device. Parts were ordered during the first week of October. The parts arrived by October 22, after which prototype construction and testing began. This was well ahead of the original goal of Oct 31 prototype construction.

In November, the objective was to troubleshoot the prototype. Trials were run every week to continue developing the system. The prototype was able to meet specifications by November 8, allowing for two weeks of additional development to improve the user interface and run times.

3.8 Design Analysis

This section reviews the design and testing results together, and compares them with the objectives that were set at the beginning of the project. This is done through a pro and con method. Finally, ideas are given to help the project improve in the future.

3.8.1 Pros and Cons of Design

The first pro of the acoustic, rock-tracking design is that it meets all the required specifications. In the last batch of testing, the median spatial resolution was 4.17cm, much better than the specified value of 15cm. Even the worst error (10.34cm), is better than the specified accuracy. The system is able to get a new data point every second, and higher rates seem to be easily achievable. The large number of test runs in the curling club shows that the system is able to function in those conditions. While the data does not export itself directly to an Excel format, it does export to the laptop in a text document. The text document is then easily read into excel using the “Insert Data” function, where all the velocities and accelerations can be calculated from the recorded times and positions. The Excel document can even have cells to store temperature, humidity and pebble, all submitted by the user. Meeting all the objectives shows that the acoustic, rock-tracking design is a viable solution.

Another pro is the future potential for the spatial accuracy of the acoustic system. Developing a dynamic threshold for the pulse-detection algorithm, using higher sampling frequencies, and more sensors will all increase the spatial accuracy of the system. It is reasonable to assume that with a few minor improvements, the acoustic rock tracking system could easily have spatial accuracy of 2.5cm or better.

The ability of the Raspberry Pis to automatically connect with each other is another good aspect of the design. The synchronization algorithm means that the user does not need to worry about different clock speeds within the sensors, or about starting the recordings at the same time. The auto-synchronization makes the system much easier to use for the end user, and more resilient against individual sensor differences. The ability of the system to detect multiple frequency pulses separately with no interference is also good. It makes the addition of more rocks into the system much easier.

The final pro is the ease of setting up this system. Currently the system is composed of only four sensors, a sync emitter and a rock emitter. As long as the sensors and the sync are placed at known locations, setting up the system is relatively easy. One does not need to rip out the ice to set up the system (like a magnetic grid), or have to worry about installing a permanent fixture (like overhead video tracking). The system transmits data wirelessly, so there is no issue recovering the data.

Despite all the positive aspects of the acoustic, rock-tracking design, there are some negatives as well. One that is inherent to an acoustic system is the lack of rotation and vibration data. Since the acoustic system only records differences in propagation delays, there is no way to recover vibration or rotation from the rock as long as the speaker is in the center of the rock. In order to capture this information, another system (like an accelerometer) would need to be integrated.

Another con is the interference potential of the design. This refers to two problems: noise and acoustic blockers. Loud noises close to the sensors can cause them to trigger, even after the Goertzel filter. This is due to the energy detector nature of the pulse detector. Since the system is designed for low-noise, testing conditions, this can be considered an acceptable demerit for the design. In the future, a dynamic threshold should mitigate this risk. Blockers though, are a larger problem. Having a person in the direct path between an emitter and the sensor results in large errors. This is disadvantageous, as having sweepers on the ice could cause the data to become unusable. However, the blocker problem is not isolated to only the acoustic system. Most alternative options, such as a laser grid or rotating laser system would have had similar problems. The only way to mitigate this problem is to have more sensors along the ice, therefore increasing the redundancy of the system. This would allow valid results despite a few bad data points from blocked sensors.

The final con of the system is the limited range of the design. While this is more of a con for the alpha prototype, as opposed to the acoustic design principle, it is disappointing that the entire sheet cannot be mapped with four sensors. Also, since the software is currently ill-equipped to deal with missing pulses, a few missed points due to range can corrupt the entire data set. This problem should be easily fixed in future prototypes, as better microphones or more sensors are purchased.

Overall, the design is considered a success due to it meeting all of the required objectives, and as well it is a portable solution that can easily be upgraded to track multiple rocks. While there are some negative attributes, most of them can be easily fixed in future versions of the design.

3.8.2 Ideas for Future Improvement:

There are several areas where the alpha prototype of the rock-tracking system could be improved in the future. The first area is the power source for the receivers. Presently the receivers are being powered by 120V AC, but this means that extension cords must be laid across the ice surface. This is not an ideal situation. If the receivers could be powered by batteries that were attached to their tripods, it would eliminate the need for long cables on the ice. The batteries would have to be long lasting and have a voltage regulator attached to provide the best performance for the receivers. A charging system, battery level indicators, and brown out system would also need to be implemented.

The setup of the hardware could also be improved. Currently, the microphones sit atop of curling rocks to provide their height, while the Raspberry Pis sit on rubber mats. While acceptable for the initial prototype, in the future it is necessary to build a more long-term solution for the sensor-hardware packaging. Getting tripods to mount the system and provide the microphone height in a stable and repeatable way would be a good start. A box to enclose the Raspberry Pis would also provide some protection from the environment. For the rock emitter, the current speaker is quite bulky. This causes some interference with the curler as the rock is thrown. Finding a way to shrink the speaker or to embed it within the rock handle would decrease the impact that the system has on the thrower.

The next area to improve would be the range of the system. One way to do this would be to improve the microphones on the receivers. Due to a limited budget, low quality microphones had to be used. These microphones have a limited sensitivity that makes it difficult to use them for large portions

of the ice surface. A higher quality, long-range microphone would extend the range of the rock-tracking system, hopefully to the entire sheet. The other method would be to purchase additional sensors. While this is an expensive option, it also comes with the added benefits of improved accuracy and redundancy for the system. Completing one or both of these options should allow the entire sheet to be tracked.

If more than four sensors are used a sensor-selection algorithm must be developed. Currently it is assumed that all sensors receive every rock pulse. With more than four sensors it will be desired that a subset of the sensors will be listened to at any one time. These are the closest sensors to the rock at this point in time. All other sensors can be ignored until the rock travels close to them. This algorithm was not developed during this project as there was neither the need nor the resources to handle more than four sensors.

A way to improve the reliability of the system would be to develop a dynamic threshold for the pulse detection. Currently, the threshold is set at a static level. However, at long ranges it is possible that the pulses are too small to trigger the threshold, and in high-noise conditions the threshold might be triggered even with no pulse. Note that in both situations, the timing of the pulse that is run through the Goertzel filter can still be detected by the human eye. A dynamic threshold would give the system extra protection against missing pulses, and false triggering due to noise.

Another way to improve the reliability of the system would be to track the centroid of the received pulses, as opposed to the peak. This should increase the systems protection against noise. However, when implementing the centroid tracking, it is important to consider the multipath effect. If the multipath is not accounted for, it could cause the perceived centroid to be heavily dependent on the amplitude and delay length of the multipath. Implementing a short cut off for centroid calculations could help mitigate the multipath effect.

Another simple feature to implement would be tracking multiple rocks simultaneously. Since the Goertzel filter is so efficient at eliminating noise, setting up multiple rocks with different frequency emitters would provide all the information necessary to track multiple rocks. As long as the frequencies are spaced out by farther than the Goertzel filter's bandwidth (300 Hz), the different emitters will not interfere with each other.

One change that could possibly be implemented is to move the pulses from the audible hearing ranges to the ultrasonic range. Converting from audible to ultrasonic would mean that the pulse from the emitters would not be heard by the people on the rink. This means it would be possible to use the system for longer periods of time without aggravating people. To do this, the sampling frequency would need to be much higher (likely over 100 kHz) and ultrasonic emitters, microphones and sound cards would need to be purchased. This feature would be simple to implement in the software, as long as the proper Goertzel-filter calculations are done. In theory, the switch to ultrasonic could improve the spatial resolution of the system, as the temporal resolution will be smaller. However, the range of the system would likely decrease, as higher frequency sounds attenuate faster than low frequencies. To avoid this, more sensors would be mandatory, and the ultrasonic equipment is expensive. So while this is a relatively simply step, it is an expensive one that should only be used on later prototypes.

Finally, if a video camera was set up either on one of the four sensor tripods or on a tripod of its own, then a video could be taken and saved for each shot that was thrown. While this video would not be used directly in the rock-tracking system, it could provide information for coaches, who could compare the release of the throw on video to the projection of the rock from the rock-tracking system.

4.0 Conclusion:

The collaboration with Train Smart for Curling was a great learning experience for the group. The result of the project is a usable measurement system that will enable new training and ice-preparation techniques. All required objectives were met. One additional deliverables was also met. The compliance with the required objectives that were laid out by the intended user is as follows:

Required Deliverables:

1. Must have a spatial accuracy of better than 15cm in order to provide usable data.
   The median spatial error measured in the Final Test was 4.17cm with no point having a larger error than 10.34cm.
2. Must be able to function in typical curling conditions (4.5-5.5 degrees C, 60% humidity).
   All tests were performed in these conditions.
3. Must be able to communicate data to a laptop for further analysis.
   System is interfaced with a laptop. The data is natively stored on the user's laptop.
4. Must be able to store location data, as well as user inputs, including temperature, humidity and ice pebble.
   Excel sheet can be modified on site with the current parameters.
5. Must be able to output data to an Excel file.
   Final data is reported in comma separated variables (.csv) format, which is readable by Excel.
6. Must be capable of improvements for increased spatial accuracy and expandable for tracking multiple rocks in future versions of the design.
   All algorithms were designed for N sensors. Minor modifications to scripts would be needed. Multiple rocks could be implemented by ensuring 300 Hz spacing between rock frequencies.

Additional Deliverable:

1. Provide a fully portable solution that could work in any curling club.
   The system uses wireless communication and can be set up by a user in under 15 minutes.

The current state of science for sports localisation is an array of cameras and reflective nodes that require large image-processing machines and algorithms. It has been shown that indoor position measurement can be done through a network of sensors and an emitter on the target. This is a less costly and more direct method of localisation. The acoustic solution has been explored by other universities. The novel aspect of this project is the utilization of asynchronous, consumer electronics as well as the high reliability of the pulse detection. Using synchronisation the group was able to completely avoid costly, scientific recording hardware and telemetry. The reliable pulse-detection algorithm allowed for the use of low grade DACs, speakers, microphones, and ADCs.

The seven phases of testing allowed for the generation of refined algorithms and software. All tests were conducted at the location of intended use to ensure that no unforeseen variables cause issues. All tests were also conducted with the primary intended user and efforts are being made to provide a smooth transition at project end.

Overall, this project is a successful prototype of a rock-tracking system using acoustic, difference, propagation delay.

5.0 References

[1] Startco Engineering Ltd. (n.d.). *Eye on the hog*. <http://www.eyeonthehog.com/> .

[2] Yazici, A.; Yayan, U.; Yucel, H., "An ultrasonic based indoor positioning system," *Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium on* , June 15-18, 2011 pp.585-589.

[3] Elson, J., Girod, l., Estrin, D., "Fine-Grained Network Time Synchronization Using Reference Broadcasts." *Usenix Association*, 2002.

[4] Zaplata, F., Kasal, M., "Using the Goertzel algorithm as a filter," *Radioelektronika (RADIOELEKTRONIKA), 2014 24th International Conference* , 15-16 April, 2014, pp.1-3.

[5] Banks, K., "The Goertzel Algorithm." *Embedded*. 28 Aug. 2002. <http://www.embedded.com/design/configurable-systems/4024443/The-Goertzel-Algorithm>.

6.0 Appendix

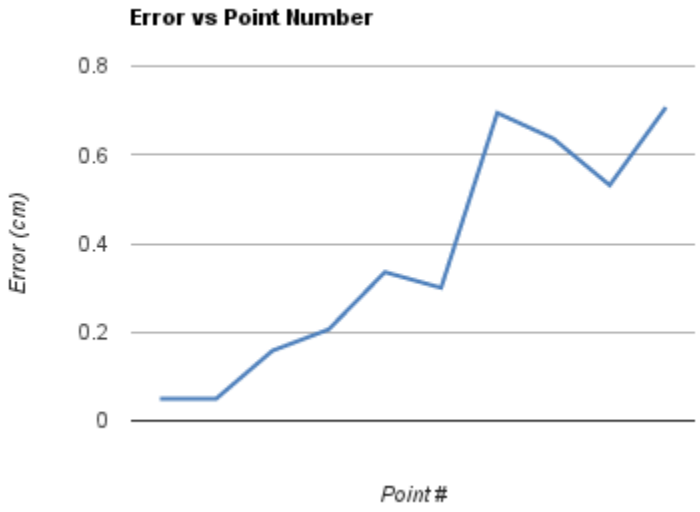
Appendix A: Algorithm Proof-of-Concept Test Results

Linear Test:

Pulse #	1	2	3	4	5	6	7
Measured Distance (m)	1	1.14	2.83	4.72	7.35	9.68	9.77
Actual Distance (m)	1	1	3	5	7	9	9
Error (cm)	0	14	17	28	35	66	77

Spatial Test:

Measured Distance X, (m)	1	2.05	3.05	3.05	1.7	1.3	1.35	2.45	2.6	2.9
Measured Distance Y, (m)	0.95	1	1.15	2.2	1.85	2	3.6	3.45	2.65	3.7
Actual Distance X, (m)	1	2	3	3	2	1	1	2	3	3
Actual Distance Y, (m)	1	1	1	2	2	2	3	3	3	3
Error (cm)	5	5	15.811	20.616	33.541	30.	69.0462	63.64	53.151	70.711


Error vs Point Number
0.8
0.6
0.4
0.2
0
Error (cm)
Point #

Appendix B: Controlled Emitter Test Results

Time (s)	0	2.000125	6.003469	8.002688	12.00578	14.00578	18.00888	20.00888
Measured Distance (m)	5	4.979187	3.773062	4.054031	3.035219	3.056031	2.047625	2.058031
Actual Distance (m)	5	5	4	4	3	3	2	2
Error (cm)	0	2.08125	22.69375	5.403125	3.521875	5.603125	4.7625	5.803125

Appendix C: Raspberry Pi Receiver Test Results

For Sensor 0, Trial 1:

Time (s)	1.46222	3.46044	5.45863	7.45684	9.45506	11.45328	13.45150	15.44972	17.44791
Relative Difference (s)		1.998219	1.998188	1.998219	1.99822	1.998219	1.998219	1.998219	1.998188
Time Delay (ms)		1.78125	1.8125	1.78125	1.78125	1.78125	1.78125	1.78125	1.8125
Time Delay After Adjusted Sampling Frequency (ms)		0	0.03125	0	0	0	0	0	0.03125
Samples in Error		0	1	0	0	0	0	0	1

Trial 2:

Time (s)	1.55403	3.552219	5.550469	7.548531	9.54678	11.54496	13.54318	15.54140	17.53962
Relative Difference (s)		1.99819	1.99825	1.99806	1.99825	1.99819	1.99822	1.99822	1.99822
Time Delay (ms)		1.8125	1.75	1.9375	1.75	1.8125	1.78125	1.78125	1.78125
Time Delay After Adjusted Sampling Frequency (ms)		0.03125	-0.03125	0.15625	-0.03125	0.03125	0	0	0
Samples in Error		1	-1	5	-1	1	0	0	0

Appendix D: Raspberry Pi Linear Test Results

Time (s)	0	2	4.00284	6.00297	8.00594	10.0059	12.0090	14.00907	16.01176	18.01198
Measured Distance (m)	3	2.91667	4.05229	3.88564	4.96919	4.96919	5.91733	5.91733	7.03212	6.99046
Actual Distance (m)	3	3	4	4	5	5	6	6	7	7
Error (cm)	0	8.33242	5.22966	11.4351	3.08086	3.08086	8.26673	8.26673	3.21224	0.95397

Appendix E: Precision and Motion Test Results

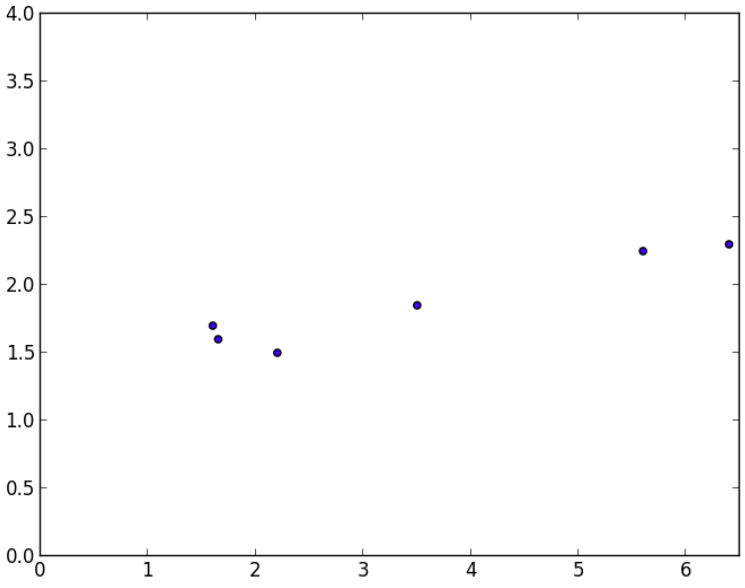
Precision Test (Compared to 2.75m, 1.8m):

Time(seconds)	X(m)	Y(m)
12.64	2.85	2.15
15.10	2.75	1.75
17.10	2.75	1.75
19.10	2.75	1.70
21.10	2.80	1.65
23.10	2.80	1.65
25.10	2.80	1.70
27.10	2.75	1.75
29.10	2.80	1.70
31.10	2.75	1.70
33.10	2.80	1.70
35.10	2.75	1.75
37.10	2.55	1.50
39.10	2.75	1.75
41.10	2.70	1.70
43.10	2.75	1.65
45.10	2.80	1.65
47.10	2.75	1.75
49.10	2.75	1.75
51.10	2.70	1.65
53.10	2.80	1.60
55.10	2.75	1.70

57.10	2.80	1.70

First Motion Trial:

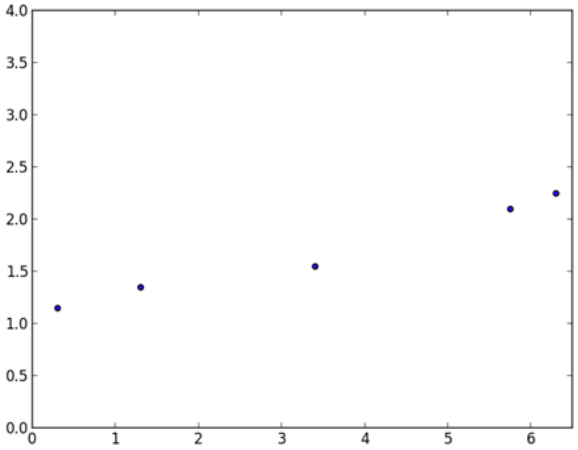
Time (seconds)	X(m)	Y(m)
11.79	6.40	2.30
13.79	5.60	2.25
15.78	3.50	1.85
17.78	2.20	1.50
19.78	1.60	1.70
21.78	1.65	1.60



Second Motion Trial:

Time (seconds)	X(m)	Y(m)
12.46	6.3	2.25
14.46	5.75	2.10

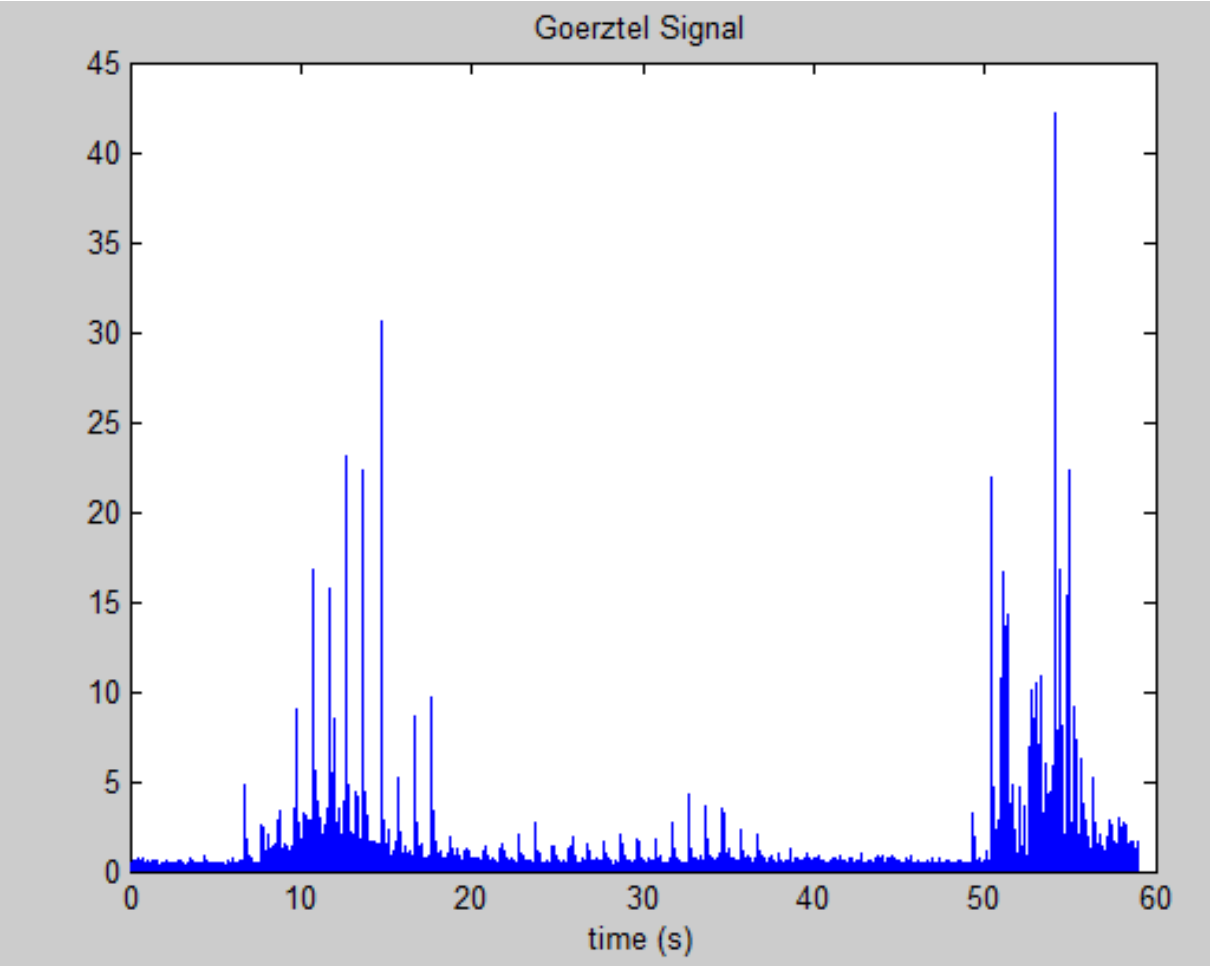
16.45	3.40	1.55
18.45	1.30	1.35
20.45	0.30	1.15



Appendix F: Hog to Hog Test Results

Results were unable to be shown, as the system could not place them in the testing area.

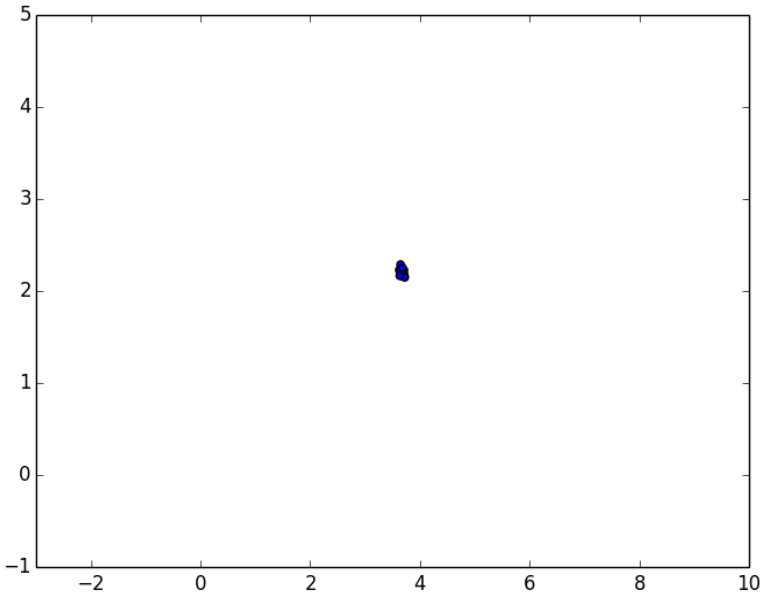
Figure depicts how the pulse magnitude decreased into the noise:



Appendix G: Final Test Results

Precision Test (compared to 3.68m, 2.19m):

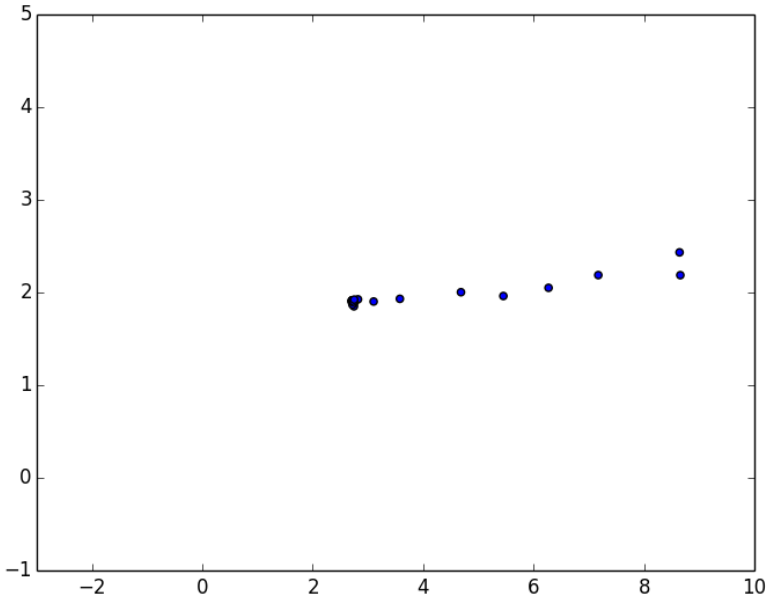
Time (s)	X Position (m)	Y Position (m)	Spatial Error (cm)
5.26	3.647	2.223	4.666905
6.26	3.679	2.23	4.00125
7.26	3.697	2.171	2.54951
8.26	3.638	2.215	4.88774
9.26	3.642	2.178	3.984972
10.26	3.651	2.23	4.940648
11.26	3.679	2.179	1.104536
12.26	3.642	2.169	4.341659
13.26	3.647	2.237	5.742822
14.26	3.679	2.186	0.412311
15.26	3.642	2.23	5.517246
16.26	3.66	2.207	2.624881
17.26	3.674	2.238	4.837355
18.26	3.647	2.288	10.3407
19.26	3.674	2.171	1.992486
20.26	3.711	2.172	3.58469
21.26	3.665	2.171	2.420744
22.26	3.711	2.223	4.527693
23.26	3.697	2.179	2.024846
24.26	3.651	2.223	4.393177
25.26	3.706	2.179	2.823119
26.26	3.624	2.23	6.88186
27.26	3.711	2.172	3.58469
28.26	3.688	2.201	1.360147
29.26	3.638	2.207	4.531004
30.26	3.679	2.171	1.90263
31.26	3.692	2.156	3.605551
32.26	3.715	2.163	4.420407
33.26	3.692	2.201	1.627882
34.26	3.683	2.252	6.207254
35.26	3.72	2.156	5.249762
36.26	3.692	2.179	1.627882
37.26	3.724	2.148	6.082763
38.26	3.633	2.166	5.27731



For Motion Test:

Time (s)	X Position (m)	Y Position (m)
6.2	8.658	2.187
7.2	8.644	2.433
8.19	7.172	2.188
9.18	6.271	2.05
10.18	5.453	1.962
11.18	4.685	2.003
13.18	3.578	1.932
14.18	3.103	1.903
15.18	2.819	1.926
16.18	2.746	1.851
17.18	2.755	1.893
18.18	2.732	1.894
19.18	2.741	1.9
20.18	2.719	1.914
21.18	2.719	1.87
22.18	2.705	1.901
23.18	2.714	1.914
24.18	2.705	1.905
25.18	2.709	1.914
26.18	2.714	1.914
27.18	2.737	1.9

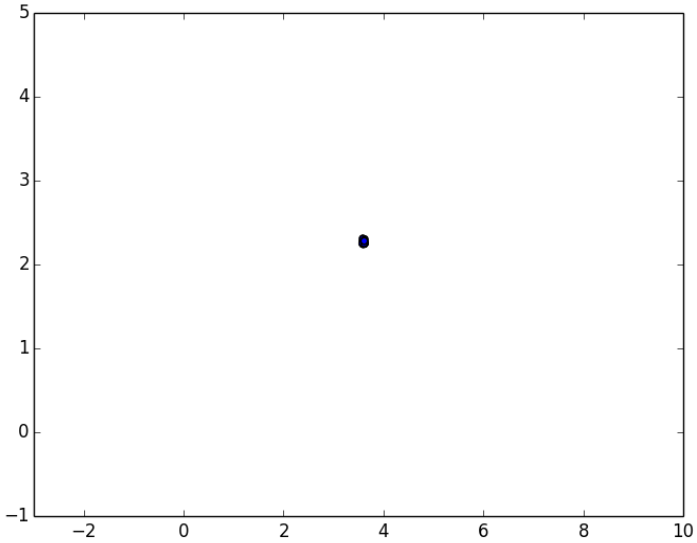
28.18	2.751	1.923



For half second test (compared to 3.6m, 2.27m):

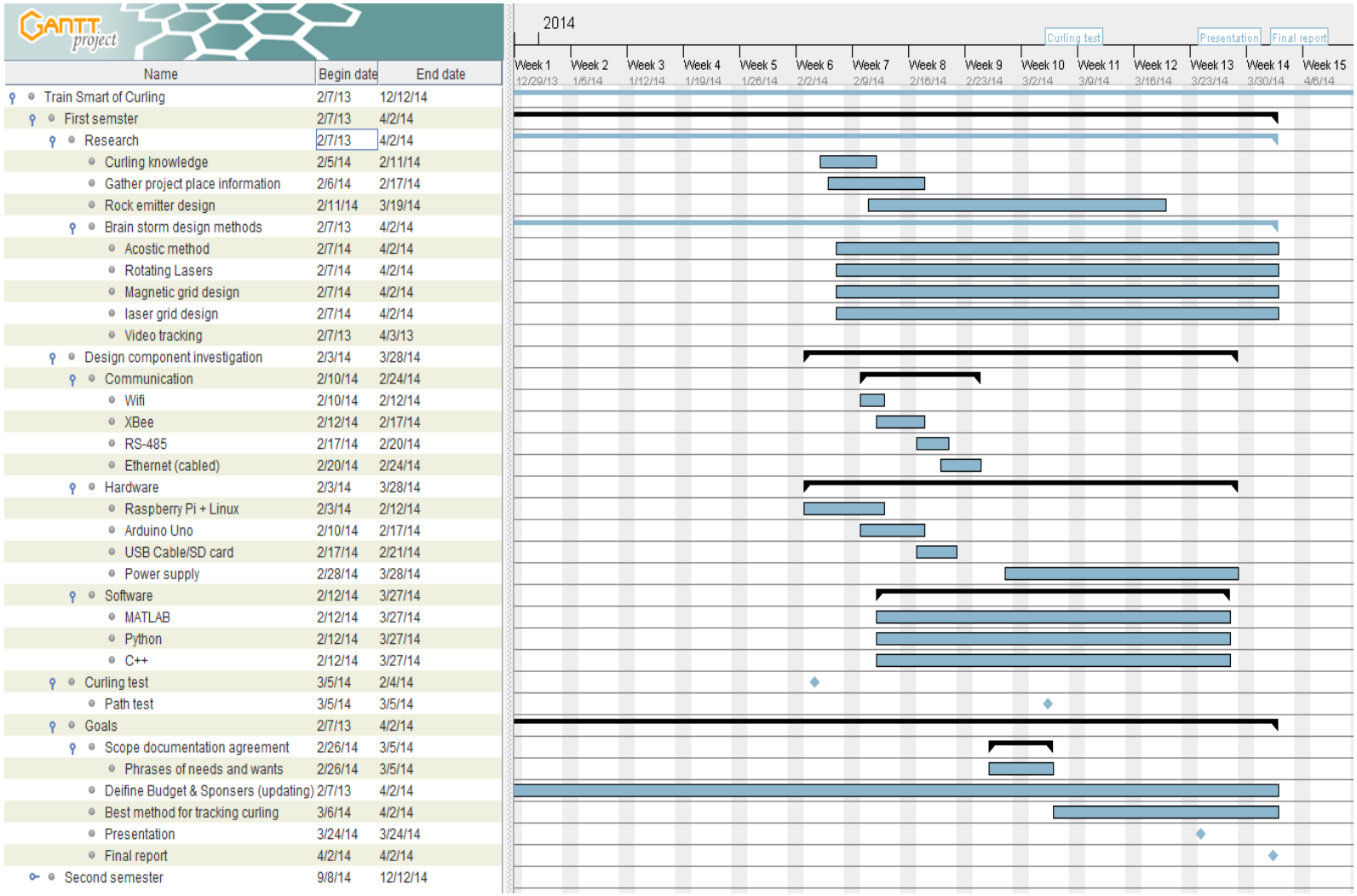
Time (s)	X Position (m)	Y Position (m)	Spatial Error (cm)
5.15	3.615	2.274	1.552417
5.65	3.624	2.252	3
6.15	3.61	2.252	2.059126
6.65	3.628	2.252	3.328663
7.15	3.61	2.252	2.059126
7.65	3.61	2.252	2.059126
8.15	3.592	2.243	2.816026
8.65	3.592	2.243	2.816026
9.15	3.619	2.252	2.61725
9.65	3.624	2.303	4.080441
10.15	3.606	2.251	1.992486
10.65	3.587	2.265	1.392839
11.15	3.587	2.281	1.702939
11.65	3.587	2.281	1.702939
12.15	3.619	2.252	2.61725
12.65	3.583	2.251	2.54951
13.15	3.592	2.244	2.720294
13.65	3.619	2.244	3.220248
14.15	3.615	2.252	2.343075

14.65	3.619	2.259	2.19545
15.15	3.619	2.274	1.941649
15.65	3.587	2.265	1.392839
16.15	3.587	2.31	4.205948
16.65	3.578	2.251	2.906888
17.15	3.615	2.252	2.343075
17.65	3.587	2.309	4.110961
18.15	3.619	2.295	3.140064
18.65	3.615	2.281	1.860108
19.15	3.615	2.281	1.860108



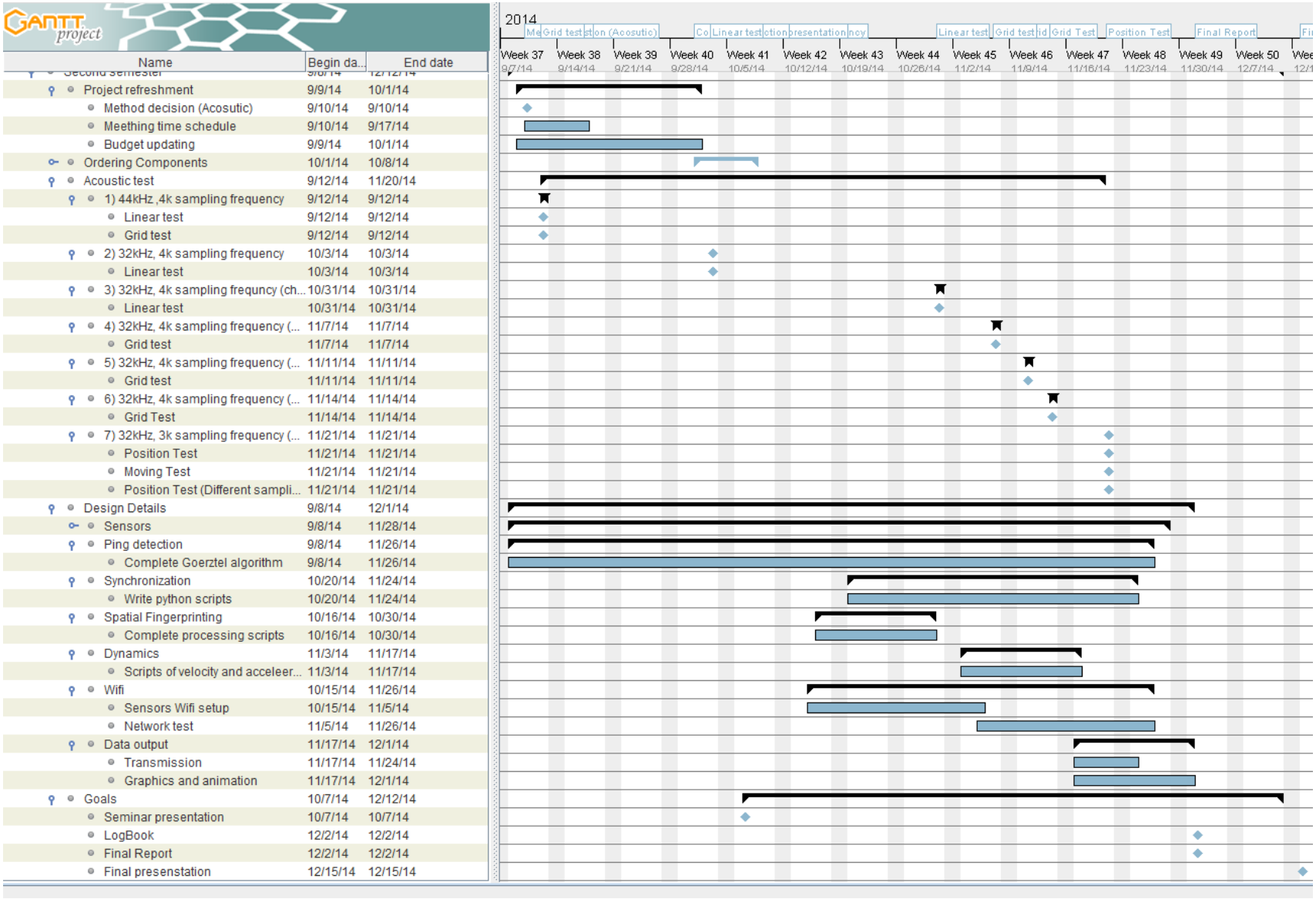
Appendix H: Gantt Chart

First Semester Schedule:



Name	Begin date	End date
Train Smart of Curling	2/7/13	12/12/14
First semster	2/7/13	4/2/14
Research	2/7/13	4/2/14
Curling knowledge	2/5/14	2/11/14
Gather project place information	2/6/14	2/17/14
Rock emitter design	2/11/14	3/19/14
Brain storm design methods	2/7/13	4/2/14
Acostic method	2/7/14	4/2/14
Rotating Lasers	2/7/14	4/2/14
Magnetic grid design	2/7/14	4/2/14
laser grid design	2/7/14	4/2/14
Video tracking	2/7/13	4/3/13
Design component investigation	2/3/14	3/28/14
Communication	2/10/14	2/24/14
Wifi	2/10/14	2/12/14
XBee	2/12/14	2/17/14
RS-485	2/17/14	2/20/14
Ethernet (cabled)	2/20/14	2/24/14
Hardware	2/3/14	3/28/14
Raspberry Pi + Linux	2/3/14	2/12/14
Arduino Uno	2/10/14	2/17/14
USB Cable/SD card	2/17/14	2/21/14
Power supply	2/28/14	3/28/14
Software	2/12/14	3/27/14
MATLAB	2/12/14	3/27/14
Python	2/12/14	3/27/14
C++	2/12/14	3/27/14
Curling test	3/5/14	2/4/14
Path test	3/5/14	3/5/14
Goals	2/7/13	4/2/14
Scope documentation agreement	2/26/14	3/5/14
Phrases of needs and wants	2/26/14	3/5/14
Deifine Budget & Sponsers (updating)	2/7/13	4/2/14
Best method for tracking curling	3/6/14	4/2/14
Presentation	3/24/14	3/24/14
Final report	4/2/14	4/2/14
Second semester	9/8/14	12/12/14

Second Semester Schedule:



Appendix I: Software

Server

```
import socket
import sys
import subprocess


IP = '192.168.2.10'
if (len(sys.argv) == 2):
   IP = IP + sys.argv[1]
else:
   print 'Error please enter the Sensor Number!'


def run(runlen):
   if (runlen < 1 or runlen > 99999):
       return 'ERROR INVALID RUNLEN'
   durr = str(runlen)
   print durr
   try:
#    subprocess.call(['cd','Desktop'])
       print 'Recording'
       subprocess.call(['arecord','-d',durr,'-f','cd','-
c','1','run.wav'])
       print 'Converting'
       subprocess.call(['python','wtt.py','run.wav'])
       print 'PULSE'
       subprocess.call(['./pulse'])
   finally:
       print 'ERROR'

   file = open('s06000_times.txt','r')


   return file.read()


# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


# Bind the socket to the port
server_address = (IP, 1234)
print >>sys.stderr, 'starting up on %s port %s' % server_address
sock.bind(server_address)
```

```
# Listen for incoming connections
sock.listen(1)


while True:
   # Wait for a connection
   print >>sys.stderr, 'waiting for a connection'
   connection, client_address = sock.accept()


   try:
       print >>sys.stderr, 'connection from', client_address



       # Receive the data in small chunks and retransmit it
       while True:
           data = connection.recv(16)
           print >>sys.stderr, 'received "%s"' % data
           if data:
               returndata = run(int(data))

               connection.sendall(returndata)
           else:
               print >>sys.stderr, 'no more data from', client_address
               break

   finally:
       # Clean up the connection
       connection.close()


sock.close()
```

Client

```
import socket
import sys
import AcousticPipe


d = raw_input("Please enter durration: ")


num_sensors = 4
```

```
sensors = []


for i in range(num_sensors):

   # Create a TCP/IP socket
   sensors.append(socket.socket(socket.AF_INET, socket.SOCK_STREAM))


   # Connect the socket to the port where the server is listening
   IP = '192.168.2.10'+str(i)
   server_address = (IP,1234)
   print >>sys.stderr, 'connecting to %s port %s' % server_address
   sensors[i].connect(server_address)


    # Send data
   message = d
   print >>sys.stderr, 'sending "%s"' % message
   sensors[i].sendall(message)


for i in range(num_sensors):
   data = ''
   while (len(data) < 1):
       data = sensors[i].recv(4096)

   print >>sys.stderr, 'received "%s"' % data
   sensors[i].close()
   fname = 's'+str(i)+'.txt'

   file=open(fname,"w")

   file.write(data)
   file.close()


AcousticPipe.run()
```

Pulse Detection

```
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
```

```
#define Pi 3.14159365


//#define FS 31971.5
#define FS 44047.9999
//#define NSAMP 128
#define NSAMP 147
//#define SIGFREQ 4000
#define SIGFREQ 3000
//#define SYNCFREQ 8000
#define SYNCFREQ 6000


#define THRES 15
#define LOOKOUT NSAMP
#define LOCKOUT 0.75*FS


void Goerz(double* input, double* output, double fs, int N, double
ftarg);
void WavThrGoerz(double* input, int size, double* output, double*
outputp, double fs, int N, double ftarg, double ftargp);
int FindTimes(double* input, int size, double* output, double thres,
int lockout, int lookout, double fs);
int FindMax(double* input, int length);



int main(int argc, char *argv[])
{
   char * fileName = argv[1];


    //reading in data
    FILE *fr;
    FILE *fout;
    int size;
    int j=0;

    fr=fopen("s0_6000.txt","rt");
    fscanf(fr, "%d", &size);
    double *wavdata=(double*) malloc(size*8);
    while(fscanf(fr,"%lf",wavdata+j) ==1 ){
        j++;
```

```
    }
    fclose(fr);

    //now can process data
    double *signalmag=(double*) malloc((size-NSAMP)*8);
    double *syncmag=(double*) malloc((size-NSAMP)*8);
    WavThrGoerz(wavdata, size, signalmag, syncmag, FS, NSAMP, SIGFREQ,
SYNCFREQ);

    free(wavdata);

    //now for the max finder
    double signaltime[1024];
    double synctime[1024];
    int numsigs=FindTimes(signalmag, (size-NSAMP), signaltime, THRES,
LOCKOUT, LOOKOUT, FS);
    int numsyncs=FindTimes(syncmag, (size-NSAMP), synctime, THRES,
LOCKOUT, LOOKOUT, FS);

    free(signalmag);
    free(syncmag);

    int i;
/*   for(i=0; i<numsigs; i++)
  {
    printf("%lf\t%lf \n", synctime[i], signaltime[i]);
  }
  for(i=numsigs;i<numsyncs;i++){
        printf("%lf\n",synctime[i]);
    }
  printf("\n");*/
  fout=fopen("s06000_times.txt","w");
  if (fout==NULL)
    {
        printf("Error Opening File!\n");
        exit(1);
    }
    for(i=0;i<numsigs;i++){
        fprintf(fout,"%lf, ",signaltime[i]);
    }
    fprintf(fout,"\n");
    for(i=0;i<numsyncs;i++){
        fprintf(fout,"%lf, ",synctime[i]);
    }
    fclose(fout);
```

```
    return 0;
}


int FindMax(double* input, int length)
{
    double max=0;
    int indmax=0;

    int i;
    for(i=0;i<length;i++){
        if (input[i]>max){
            max=input[i];
            indmax=i;
        }
    }
    return indmax;
}


int FindTimes(double* input, int size, double* output, double thres,
int lockout, int lookout, double fs)
{
    int x=0;
    int flag=0;
    int i;
    for(i=0;i<(size-lookout);i++){
        flag--;
        if ((input[i]>thres)&&(flag<0)){
            int hold=FindMax(input+i,lookout);
            output[x]=(double)(hold+i)/fs;
            x++;
            flag=lockout;}
        }
    x--;
    return x;
}

void WavThrGoerz(double* input, int size, double* output, double*
outputp, double fs, int N, double ftarg, double
ftargp)        //Idea,  adding another output so sync an signal done
at same time
{
    int i;
    for(i=0;i<(size-N);i++){
```

```
        Goerz(input+i, output+i, fs, N, ftarg);
        Goerz(input+i, outputp+i, fs, N, ftargp);
    }
    return;
}


void Goerz(double* input, double* output, double fs, int N, double
ftarg)
{
    int k=floor(0.5+N*ftarg/fs);
    double w=(2*PI/N)*k;
    double cosine=cos(w);
    double sine=sin(w);
    double coeff=2*cosine;

    double Q1=0;
    double Q2=0;
    double Q0;

    int y;
    for(y=0;y<N;y++){
        Q0=coeff*Q1-Q2+input[y];
        Q2=Q1;
        Q1=Q0;
    }

    *output=Q1*Q1+Q2*Q2-Q1*Q2*coeff;

    return;
}
```

Synchronisation

```
#make sure that the first pulse is the first pulse is the same.


# sync rec time - sensor delay due to dist - sync0 rec time + sensor0
delay due to dist.


import fastSpatial as spatial
import matplotlib.pyplot as plt


def process(Nodes,SyncNode,rocks,syncs,speed):
```

```
    #get the delay for each node from the sync
    spatial.findDelay(Nodes,SyncNode.x,SyncNode.y,SyncNode.h,speed)


    newData = []


    node0Syncs = syncs[0]


    nn = len(Nodes)

    for n in range(0,nn):
        newNodeData = []
        DiD0 = []


        nodeSyncs = syncs[n]
        ns = len(nodeSyncs)

        nodeData = rocks[n]
        nr = len(nodeData)


        # find the delays
        averageNodeDelay = 0
        for i in range(0,ns):
            internalDelay = nodeSyncs[i] - Nodes[n].delay -
node0Syncs[i] + Nodes[0].delay
            DiD0.append(internalDelay)
            averageNodeDelay += internalDelay/ns


        plt.plot(DiD0)
        plt.show()


        # give everything at least the average node delay
        for j in range(0,nr):
            firstRunData  = nodeData[j] - averageNodeDelay
            newNodeData.append(firstRunData)

        #fine detail syncing for each sync pulse.
        for i in range(0,ns):
            if i != ns-1:
                for j in range(0,nr):
```

```
                    # if the data is after this sync and before the
next.
                    if nodeData[j] > nodeSyncs[i] and nodeData[j] <
nodeSyncs[i+1]:
                        newNodeData[j] = nodeData[j]-DiD0[i]

            else:
                for j in range(0,nr):
                    # if the data is after the last sync
                    if nodeData[j] > nodeSyncs[i]:
                        newNodeData[j] = nodeData[j]-DiD0[i]

        newData.append(newNodeData)


    return newData
```

Fast Spatial

```
from __future__ import division
import numpy as np
import matplotlib.pylab as plt
from pylab import *
#import mpl_toolkits.mplot3d.axes3d


resolution = 10
scale = 7
narrowing = 3


class Node(object):
    def __init__(self,x,y,h):
        self.x = x
        self.y = y
        self.h = h


    def __str__(self):
        return "x%d,y%d,h%d,delay%d" %
(self.x,self.y,self.h,self.delay)


    def posSet(self,x,y,h):
        self.x = x
        self.y = y
        self.h = h
```

```
    def setDelay(self,delay):
        self.delay = delay

    def dist(self,otherNode):
        dx = otherNode.x - self.x
        dy = otherNode.y - self.y
        dh = otherNode.h - self.h
        sx = np.square(dx)
        sy = np.square(dy)
        sh = np.square(dh)
        return np.sqrt((sx+sy+sh))


    def dist(self,x,y,h):
        dx = x - self.x
        dy = y - self.y
        dh = h - self.h
        sx = np.square(dx)
        sy = np.square(dy)
        sh = np.square(dh)
        return np.sqrt((sx+sy+sh))


    def TimeDist(self,x,y,h,speed):
        dx = x - self.x
        dy = y - self.y
        dh = h - self.h
        sx = np.square(dx)
        sy = np.square(dy)
        sh = np.square(dh)
        return (np.sqrt((sx+sy+sh)))/speed


class Position(object):
    def __init__(self,time,x,y,error):
        self.x = x
        self.y = y
        self.time = time
        self.err = error
    def __str__(self):
        return "Rock at %.2f sec: x(%.3f),y(%.3f)" %
(self.time,self.x,self.y)
    def x(self):
        return self.x
    def y(self):
        return self.y
```

```
def findDelay(Nodes,x,y,h,speed):
   zero = Nodes[0].TimeDist(x,y,h,speed)
   for node in Nodes:
       node.delay = node.TimeDist(x,y,h,speed) - zero


def findCenter(Nodes):
   lowestX = Nodes[0].x
   highestX = Nodes[0].x
   lowestY = Nodes[0].y
   highestY = Nodes[0].y
   for node in Nodes:
       if node.x < lowestX:
           lowestX = node.x


       if node.x > highestX:
           highestX = node.x


       if node.y < lowestY:
           lowestY = node.y


       if node.y > highestY:
           highestY = node.y


   centerX = (highestX+lowestX)/float(2)
   centerY = (highestY+lowestY)/float(2)
   return centerX, centerY



def getError(Nodes,x,y,h,speed):
   error = 0
   zero = Nodes[0].TimeDist(x,y,h,speed)
   for node in Nodes:
       PredictedDelay = node.TimeDist(x,y,h,speed) - zero
       error = error + np.square(node.delay - PredictedDelay)

   return np.sqrt(error)


def process(Nodes, height, speed,xr,yr):
```

```
    numNodes = len(Nodes)
    #zero out node0's delay
    zero = Nodes[0].delay
    for node in Nodes:
         node.setDelay(node.delay - zero)

    Nodes[0].getDelay = 0

    #get the compute area
    [centerX, centerY] = findCenter(Nodes)


#     print '\n CX ',centerX, 'CY ',centerY
    errors = np.zeros((resolution,resolution))

    for s in range(scale):
        leasterror = 9999
        for x in range(resolution):
            xpos = centerX + (xr*(x/(resolution-1)) - xr/2)
            for y in range(resolution):
                ypos = centerY + (yr*y/(resolution-1) - yr/2)
                #print '\nx',xpos,' y',ypos
                error = speed*getError(Nodes,xpos,ypos,height,speed)
                if error < leasterror:
                    leasterror = error
                    bestPosx = xpos
                    bestPosy = ypos

                errors[x,y] = error

        centerX = bestPosx
        centerY = bestPosy
        xr = xr/narrowing
        yr = yr/narrowing


#         figure(1)
#         imshow(errors)
#         colorbar()
#         grid(True)
#         show()
    #print '\nx',bestPosx,' y',bestPosy


    pos = Position(zero,bestPosx,bestPosy,leasterror)
    return pos
```

Acoustic Pipeline

```
import fastSpatial as spatial
import numpy
import Sync
import matplotlib.pyplot as plt


def run():

   Nodes = []


   Nodes.append(spatial.Node(0,4.37,0)) #s0
   Nodes.append(spatial.Node(8.24,4.37,0)) #s1
   Nodes.append(spatial.Node(8.24,0,0)) #s2
   Nodes.append(spatial.Node(0,0,0)) #s3


   SyncNode = spatial.Node(4.51,4.37,0)


   nn = len(Nodes)


   syncs = []
   rocks = []


   # pull from file
   for n in range(nn):
       filename = 's'+str(n)+'.txt'
       f = open(filename, 'r')
       srocks = f.readline()
       ssyncs = f.readline()
       sensorRocks = []
       sensorSyncs = []


       strin = ''


       for c in srocks:
           if (c == ','):
               sensorRocks.append(float(strin))
               strin = ''
           else:
               strin+=c
```

```
    strin = ''


    for c in ssyncs:
        if (c == ','):
            sensorSyncs.append(float(strin))
            strin = ''
        else:
            strin+=c


    syncs.append(sensorSyncs)
    rocks.append(sensorRocks)


#checker
for n in range(nn):
    rockavg = 0
    for s in rocks[n]:
        rockavg+= float(s)/float(len(rocks[n]))
    lastsamp = 0
    i = 0
    for s in rocks[n]:
        i = i+1
        if (lastsamp == 0):
            lastsamp = s
        else:
            if (((s - lastsamp) - rockavg)>0.25):
                print '\nRock Error on node ',n,' sample ',i
            lastsamp = s


#checker2
for n in range(nn):
    rockavg = 0
    for s in syncs[n]:
        rockavg+= float(s)/float(len(syncs[n]))
    lastsamp = 0
    i = 0
    for s in syncs[n]:
        i = i+1
        if (lastsamp == 0):
            lastsamp = s
        else:
```

```
            if (((s - lastsamp) - rockavg)>0.25):
                print '\nRock Error on node ',n,' sample ',i
            lastsamp = s


minrocks = 999
minsyncs = 999


newsyncs = []
newrocks = []


for n in range(nn):
    if (minrocks > len(rocks[n])):
        minrocks = len(rocks[n])

    if (minsyncs > len(syncs[n])):
        minsyncs = len(syncs[n])
print '\nHACKING OFF'
for n in range(nn):


    thisnodesrocks = []


    thisnodessyncs = []


    for i in range(minrocks):
        thisnodesrocks.append(rocks[n][i])


    for i in range(minsyncs):
        thisnodessyncs.append(syncs[n][i])


    newsyncs.append(thisnodessyncs)


    newrocks.append(thisnodesrocks)


    print len(rocks[n]) - len(thisnodesrocks)
    print len(syncs[n]) - len(thisnodessyncs)

syncs = newsyncs
```

```
rocks = newrocks


SyncedRockTimes = Sync.process(Nodes,SyncNode,rocks,syncs,333)


position = []


spots = len(SyncedRockTimes[0])


worsteSensor = []


for spot in range(0,spots):
    smallestError = 999
    worsteSensor.append(0)
    for x in range(0,4):
        for n in range(0,nn):
            Nodes[n].setDelay(SyncedRockTimes[n][spot])

        threeNodes = []
        for j in range(0,4):
            if j != x:
                threeNodes.append(Nodes[j])
        test1 = spatial.process(threeNodes,0,333,30,5)
        if test1.err < smallestError:
            smallestError = test1.err
            worsteSensor[spot] = x
            bestTest = test1
    if smallestError < 0.1:
        position.append(bestTest)
        #print bestTest


print '\nFinal DATA'
for test in position:
    print test


posx = []
posy = []
for pos in position:
    posx.append(pos.x)
```

```
        posy.append(pos.y)


    plt.scatter(posx, posy)
    plt.axis([-3, 10,-1, 5])
    plt.show()


    plt.hist(worsteSensor)
    plt.show()


run()
```