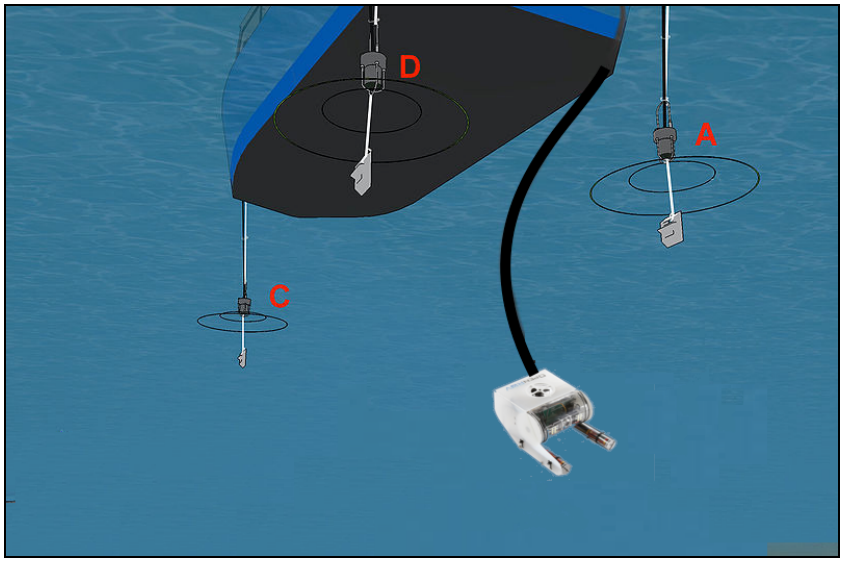
OpenROV Underwater Acoustic Location System Final Report

by Luis Sanchez, James Smith, Jason Shen
in conjunction with Jim Trezzo



1. Abstract

OpenROV is an underwater robotic platform developed to lower the cost of underwater exploration. With any form of exploration, the location of discoveries is very important data, but the OpenROV currently lacks a method of triangulation. This paper goes into the details of the digital signal processing algorithm portion of an acoustic location system that uses three surface mounted transducers that send out bursts of acoustic signals at different frequencies to triangulate an underwater rover. The algorithm determines the relative arrival time of each signal to the OpenROV. With this data as well as other sensors on the OpenROV, an accurate 3D location can be calculated on the computer onboard a surface vessel.

2. Introduction

The OpenROV platform is extremely useful to researchers and weekend explorers alike, but location triangulation is still a necessary feature yet to be implemented. If an OpenROV enthusiast uncovered a new shipwreck, or wanted to monitor marine life location would be much needed. Also, with the push for autonomizing the OpenROV platform, location data is required for such autonomous algorithms' success. The obvious need for a method of triangulation of an underwater robot is what led Jim Trezzo to propose the idea of an underwater acoustic location system for the platform. Jim has laid a tremendous amount of groundwork to allow us to contribute to his project and has helped us along the way

Triangulation on land is generally accomplished via GPS, but satellite signals cannot penetrate the water meaning we had to look elsewhere. The possibility of using optic waves was dismissed due to the limited range such a system would incur. The last option was using acoustics, so we've decided to implement triangulation via acoustic waves and a time difference of arrival algorithm. Acoustics are already in use in sonar systems and doppler velocity logs, so there is a wide range of research in the subject.

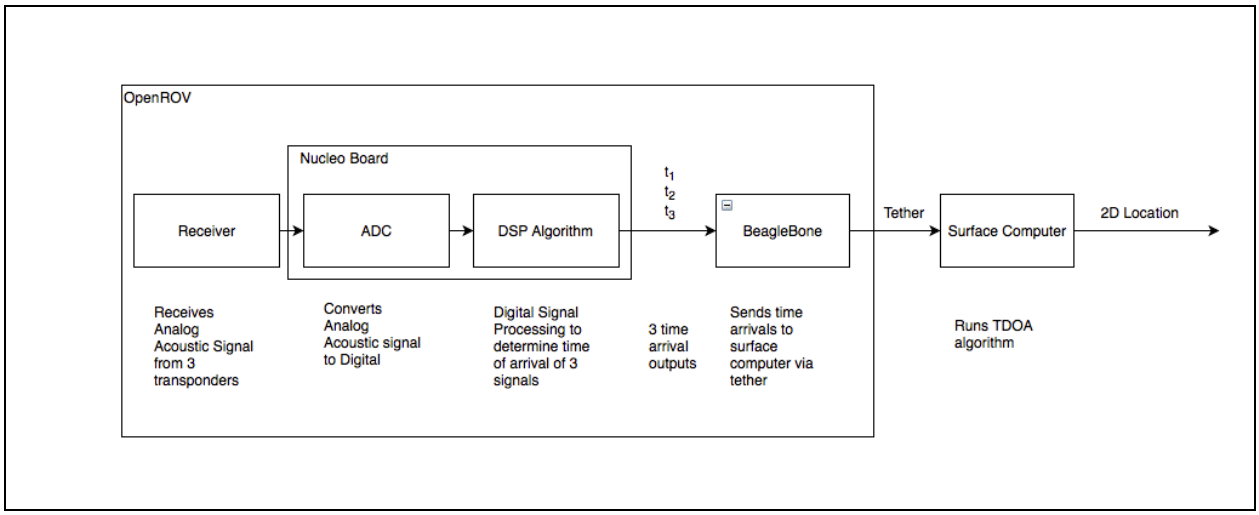
Similar, but more complex systems already exist on the market, but they are prohibitively expensive for the average OpenROV enthusiast. These systems upward of $10,000, and are overengineered for our purposes. This brings us to our main goal: create a simple, low-cost, albeit accurate acoustic location system for the OpenROV platform. With the current limited range of the OpenROV of 100 meters in mind, we've been able to cut costs by reducing the complexity of more expensive systems to provide a unique solution to the triangulation problem. This project aims to use off the shelf parts to hold to the open source mindset of the OpenROV, and allow enthusiasts to build similar systems of their own.

In summary, three transducers attached to a surface vessel will transmit sine waves at three distinct frequencies, so that when the signals arrive at the OpenROV, it can determine which signal came from which transducer. Upon receiving these signals, they are processed by the STM32F4 Nucleo board in the OpenROV's electronics bay, which was chosen for it's optimized DSP functionality as well as low cost. The output of this board is the time difference of arrival of each signal respective to the other two. This output is sent via the OpenROV's tether to the surface vessel where the TDOA algorithm is run to calculate the location of the rover with respect to the surface vessel.

We hope to achieve location updates 10 times per second, and this useful data can be displayed to the user or recorded for analysis down the line. The focus of this paper is on the DSP portion of the overall system, namely sampling the piezoelectric transducer onboard the OpenROV, and processing it to determining the time differences of each signal. Our work was in the following areas:

- Optimizing the many parameters of the overall system with regards to both signal generation and filtering
- Sampling acoustic signal output of receiver and converting to digital
- Processing digital signal and outputting time difference

3. Technical Material



Block Diagram of System

3.1. OpenROV

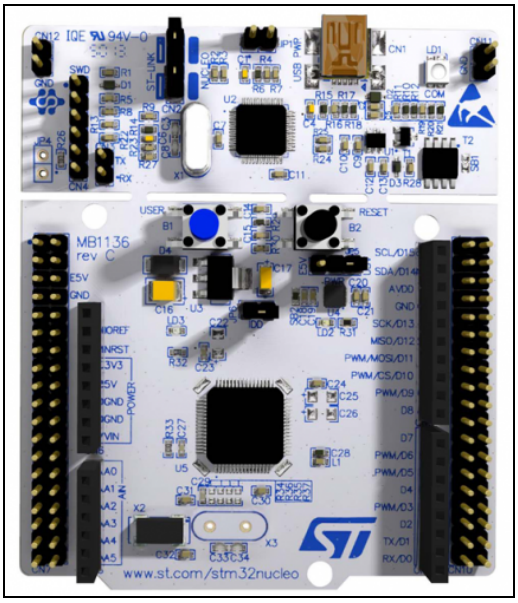
OpenROV Platform

OpenROV is the open source underwater robotic platform that our system is built to triangulate. The rover has 2 rear facing thrusters and one above facing thruster. It is battery powered and has a 2 hour runtime. It is tethered via a 100 meter ethernet cable for controls and communication with the pilot. All of the electronics are held in the electronics bay, which is where we will place any added components that the system needs.

3.2. Piezoelectric Transducers

The 3 transmitting transducers used in this project to send out the three signals from the surface vessel are piezoelectric transducers custom-built by Jim Trezzo. The receiver is identical to the transmitters, only it will be acting as an input, rather than an output. These transducers work best in the 30-40 kHz frequency range, so this is the range that we built our algorithm on, and the signal frequencies can be fine tuned to reduce error.

3.3. Nucleo Board (STM Nucleo-F446RE)



The board that we've used for this project is the STMicroelectronics NUCLEO-F446RE, which we will simply call the Nucleo board from this point on. This microcontroller is built around an ARM Cortex-M4 processor, and was chosen for it's DSP capabilities provided by ARM's Cortex Microcontrollers Software Interface Standard (CMSIS) libraries and for it's low cost ($10.12). Also integral to this project was the 12-bit ADC built in to the Nucleo Board. The board will be mounted within the OpenROV's electronics enclosure and connected to the piezoelectric receiver mounted atop the rover.

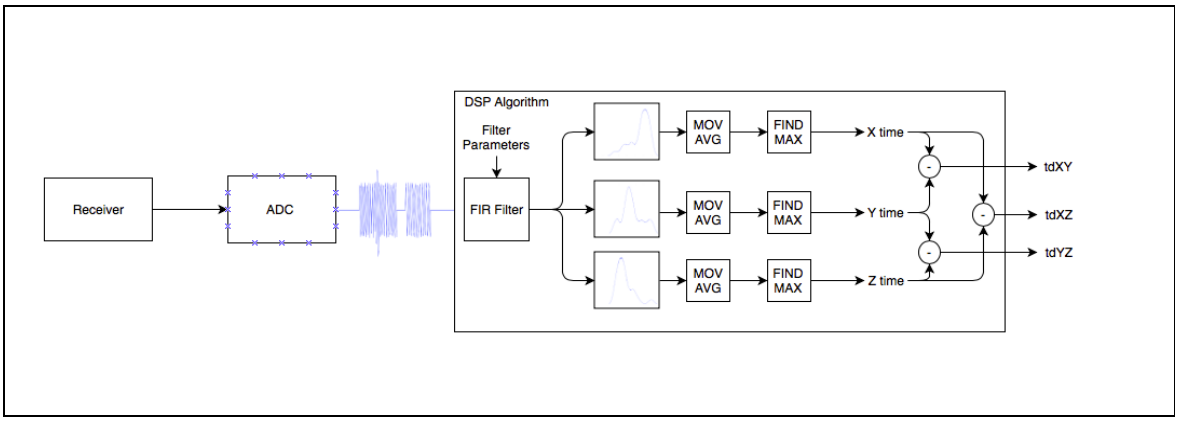
3.4. Cortex Microcontroller Software Interface Standard (CMSIS)

CMSIS is the library provided by ARM that gives access to useful digital signal processing tools such as filters and both basic and more complicated math functions. Since we needed to use an FIR filter to separate the varying frequency signals from the received signal, the CMSIS library gave us access to this functionality so we didn't need to write our own FIR filter.

3.5. Analog to Digital Converter (ADC)

The 12-bit analog to digital converter(ADC) embedded in the Nucleo Board will be used to convert the received acoustic signal onboard the OpenROV to a digital one that the DSP algorithm can process. First we tried using the MBED high level library to call a read function on the ADC. We found out this wasn't going to give us a high enough quality signal to be used for the filter. This also had a loop running in the program to keep calling a read to save into an array. This stops the program from being able to collect a signal and execute the filter code at the same time. We had to figure out how to use the HAL library for the microcontroller. This required having to set up the internal clocks on the microcontroller and then to use that to the ADC to continuously sample and save into the memory. In order to control the sample rate we would have to control the clock and calculate the rate the ADC will be filling the array. Also we had to set up direct memory access so the ADC was able to directly write into memory without the program explicitly call a read and choose the index in the array to save it in. Then we had to set up the interrupt handler for the ADC. When the specified array is filled up (by the ADC), it will then send an interrupt signal to have the main program to start executing the filter code. We were able to get this to work but we were not able to interface the boards together to send a test signal. We did not have time to setup a trigger to have the ADC automatically start collecting the signal when the source sends a trigger. Hopefully we can do this during the summer so we can see if the whole system actually works.

3.6. DSP Algorithm



DSP Block Diagram

This project focuses on determining the time difference of arrival of the three transmitted signals. The DSP Algorithm run on the Nucleo Board is not to be confused with the TDOA location algorithm, but rather it's output is the input to the TDOA algorithm. Next, the steps of the algorithm will be described in detail.

First, the DSP algorithm samples the received signal at a sampling rate of 266 Ksps and converts this analog signal to a digital one. Once the DMA buffer fills up, the FIR filter is run on the data in the buffer three times, once for each of the transmitted signals. The FIR filter coefficients corresponding to each frequency range are hard coded onto the board and are defined in the main.h source code file.

After the FIR stage of the algorithm, there will be a single peak (the impulse response) in the filtered data, and this peak represents the position in the received signal that the sine wave at the given frequency was

found to most likely be. The filtered data can be slightly noisy, so it is run through a moving average function to smooth the data. This moving average function is done on the data array itself to reduce the overhead of creating a new array due to the limited 128KB RAM present on the Nucleo Board.

After the smoothing stage, a peak detection function calculates the peak of the data, to estimate the location of the impulse response which will also be the signal's arrival position.

After the FIR filter, moving average and peak detection function have been run on each frequency range, the three time differences are placed on an output pin of the Nucleo Board, to be read by the OpenROV's Beaglebone microcontroller. From here, the data is sent up the tether and a time difference of arrival algorithm is run to determine the 2 dimensional location of the OpenROV. In conjunction with the depth sensor data onboard the OpenROV, 3D location can be determined from here and displayed to the pilot.

3.7. Filter Parameters

The filter was an important part of the project as it would affect the accuracy of the location calculation as well as the speed at which we could find the location. The filter order is the most important part of the FIR filter we started with from the python code provided. This code uses a similar function to that of Matlab to generate the coefficients for the FIR filter, given a frequency to filter out. Filter order determines the number of coefficients, which then determine how fine of a filter the FIR function will be. Python has a library that generates the numbers based on filter order and frequency. Finding the best meant going through a variety of these filter orders to get the best location at the end of the calculation. When the filter order value becomes too large, then the calculation to filter out certain frequencies becomes more time extensive as well as harder for the Nucleo to handle. A balance had to be made between these too.

3.8. Signal Parameters

The second part that needed to be optimized was the type of signals that would ideally be sent from the three transceivers on board the vessel. These signals had to be of different frequencies with 5,000 MHZ apart. The signals also could send a different number of sine cycles every time they all synchronized and sent their signals. The more cycles sent the least frequent we could complete the computation of the location. Another thing that would need to be balanced.

This ideal parameter finding was all done with scripts and in ideal scenarios which also calls into question the implication of how accurate results would be gathered with real world signals.

4. Milestones

	James	**Luis**	**Jason**
Week 4	Understand Filtering Techniques (FIR, fft, bandpass), Setup Github		
Week 5	Implement FIR bandpass filter in C	Begin testing of filter parameters	Implement Hello World on STM32, debug
Week 6	Setup ADC with potentiometer on Board 1	Optimize filter parameters in Python	Setup DAC to generate test signals on Board 2

Week 7	Create buffer to view last 10ms of signal	Create test signals, adding noise similar to what will be encountered underwater	Interface 2 boards together
Week 8	Run filtering test, fix bugs	Create test for TDOA algorithm	Run filtering test, fix bugs
Week 9	Implement TDOA Algorithm	Create test for TDOA algorithm	Socket.io communication setup
Week 10	Testing, Final Video/ Report work		

Initial Milestones (Green - Complete Red - Incomplete)

Above is a list of the initial milestones, which were adapted throughout the quarter as we were able to better determine how the project would pan out. Next, we will discuss our progress throughout the weeks of this quarter, and where we diverged from our original plan.

Week 4:

Understanding filtering techniques and Python Code (all) - This milestone isn't very demonstrable, but was still necessary for our progress. Originally we thought there would be a large variety of filtering techniques that we would have to sift through, but we found that FIR filters were the most widely used DSP filters, and it was also the method that Jim implemented in Python, so that lead us to use FIR filtering.

We also had to get familiar with Jim's Python's code in order to move on, so we spent a lot of time changing the parameters of his scripts and seeing how they affected the algorithms accuracy. Github was forked from Jim's.

Week 5:

Implement FIR Bandpass filter in C (Jason and James) - We implemented the FIR Bandpass filter using the filter coefficients generated from Jim's python script. We hard coded in a test signal and ran the FIR filter and compared the results to what the python filtered signal was. This milestone included implementing all the functions required for filtering: peak detection, moving average, absolute value, and calling FIR filter. The results are off, but we are still determining which filter parameters to use, at which point we'll retest. This code can be found on our Github under Software/CPrograms/main.cpp and main.h.

Implement Hello World on STM32 (Jason)- Get toolchain setup for STM32 and get example such as blinking an LED working on board. We started by using the mBed IDE, and it featured example code to

print “Hello World” into the console, but we had a lot of trouble actually getting the output due to our unfamiliarity with the board. In the end we used miniterm.py, which is a python based serial console. This was a good initial milestone to get us setup using the board and environment.

Begin Testing of Filter Parameters (Luis) - In order to test for the filters effect on the overall accuracy, all variables must remain constant while the filter order is varied. With variation of the filter order, I also explored the changes on the window for the pass bandwidth. The filter orders that were tested were N = 16, 20, 28, 32 , 60, 63, 64, 65, 69, 128, 256 with a constant pass bandwidth of 1000. The results pointed the optimal value being 63, when comparing across many different locations.

Week 6:

Setup ADC with potentiometer on Board 1 (James) - Implemented code to read from the onboard ADC using mBed function calls.. We used a potentiometer to test the functionality of ADC and our code. The ADC code turns an onboard LED on when the input signals amplitude passes a set value. The video link below shows the turning of potentiometer causing light to turn on as well as the output on the console of ADC readings. Using the mBed function calls ended up not being sufficient in the end because we couldn’t get an accurate nor large enough sampling rate. The new ADC implementation will be discussed below.

ADC Video

Setup DAC to generate Sinusoidal Signals (Jason) - Implemented code to generate sine waves using the onboard DAC on a second Nucleo board. The implementation seems to be a very low performance version and may need to be tweaks to be able to generate higher frequency signals with enough resolution. If that does not work we would probably have to use the function generator in the labs. The goal of the DAC is for us to test the if the microcontroller can receive, filter, and calculate the signals received by the ADC. This code can be found on the GitHub under dac.cpp

Optimize Different Parameters (Luis) - Number of cycles, threshold, transmits per second, in combination with the frequencies of the transceivers and the filter order were analyzed.

When varying **bandpass width**, we found that within many tests:

```
lvsanche@lvsanche-Lenovo-U310:~/git/acoustic-location-system/Parameter_Testing/test_3loc_filterOrder$ python readError.py
eorderB_250: 10.330000

eorderB_500: 8238.897747

eorderB_1250: 4.597331

eorderB_1000: 4.155030
```

Here the orderB_# is the cumulative sum of error that happens at the final calculation from the expected ROV location to the actual location.

Among others not shown, the best value is around 1000 after setting the Filter order to 63. Large amount of variance was noted however, meaning that the bandwidth might pick up unwanted noise in some instances. The rest of the variables were integrated into a larger script in order to save time and effort writing individual tests that might be different once all variables changed.

Week 7:

Script filter parameter optimizations (Luis) - Create script that generates filter parameters and determines error of different points in 3D space to optimize which parameters to use for final product. Large script is ran in order to find the best combination. This test is currently running, as I type this the script shown below has been running for 6 hours. Different locations of the ROV were also used in order to get a more comprehensive look at the performance and error being calculated from the TDOA algorithm given the parameters we examined. All this code can be found in the Github.

ADC Ring Buffer (James): Working demo or report on how the ADC buffer would work. This ended up being pretty straightforward, so we just coded it up and tested it. Also, in the end, we used the DMA buffer for holding the ADC samples, so the ring buffer we created was never used.

Interface Boards Together (Jason) - Send generated sinusoids from DAC on one board to ADC of other board. Signal should be able to be reconstructed after.

On this milestone we realized that both the mBed ADC and DAC function calls would not be suitable for our purposes, because we couldn't get enough precision on the sampling rate and it was also too slow. Jason started working more on this and ended up implementing ADC functionality that would write directly to DMA and then send an interrupt when the DMA buffer was full. This took much longer than expected due to our unfamiliarity with embedded systems and the lack of tutorials specific to our board.

Final Weeks:

Test Filter on Signal Generated by DAC - This proved much harder than we previously thought. Because of the amount of time we had spent on getting ADC working and getting our FIR filtering and smoothing bug free, we were unable to test on a DAC generated signal. Instead we spent this time debugging and testing on a hard coded signal, which worked out well. We tested on this hard coded signal in week 9. A bug that had been haunting us from the beginning of the FIR filter code was that James wasn't allocating space for the output array of the filter. This explained our random filter results that stumped us for so long, until Jason finally checked over the code. Upon fixing this very simple, yet nightmare of a bug, we generated a few different signals and tested the Filter code on each of them.

Because we were unable to test the filter running on an ADC sampled signal, it wasn't possible to execute the actual location determining TDOA algorithm. All in all, it was great to see our functions successfully filtering out the different frequency signals and determining a good estimate for the arrival time of signals.

Documentation and Presentations - We spent a large portion of the last couple weeks working on our final report and video, and our milestone report on the 8th week.

Other Issues - One of the issues we had was space issues on the board. Because we're not used to coding for embedded, where memory is so limited, we weren't as wary as we should've been about memory management. This caused us to alter the algorithm to perform filtering on a signal with fewer samples. As noted before, the mBed IDE proved to be very simple to use and had great functions built in, but they turned out to be too inefficient due to the high abstraction. We tried to port our code over to use different IDEs and debuggers, but we ran into constant trouble in doing so. In the end the filter code was done on mBed, because printing to console was much easier and the ADC code was done in VisualStudio, where it was easier to visualize the DMA buffer filling up and see the actual wave that was being sampled.

5. Conclusion

The main goal of this project was to triangulate an underwater robot from a surface vessel. Although we were unable to complete the project to it's entirety, much of our goals were complete. We were able to sample the Nucleo's onboard ADC at a high sample rate that makes the overall system feasible. We also wrote all the code for filtering the received signal, and optimized the parameters that control signal generation and filtering. The final goal that was left incomplete was the testing of our filter on ADC sampled code. This will just require spending some time to get the DAC working using interrupts similar to the ADC, but we're confident that our filter works correctly so upon implementation, testing should prove that filtering sampled signals will work similarly. Over the summer, we will continue to work on the project to see a functioning acoustic location system. It will be interesting to get in water tests on the OpenROV, so that we can truly benchmark the design and implementation.

6. References

Terms

Nucleo - the name for STM32F4 board
OpenROV - open source underwater robotics platform
CMSIS - Cortex Microcontroller Software Interface Standard (contains needed DSP libraries)
DSP - Digital Signal Processing
ADC - Analog to Digital Filter
FIR - Finite Impulse Response
DMA - Direct Memory Access
TDOA - Time difference of arrival, the location algorithm
Beaglebone - OpenROV's microcontroller running linux