AVS Signal Processing

Signal Processing for Acoustic Vector Sensors

Group K-1

Wouter Kayser, Lennart Noordsij and Olmar van Beurden
4360966, 4359720 and 4368150
Version: 1.1

Bachelor Graduation Project

TU Delft Delft University of Technology

Challenging the future

AVS Signal Processing
Signal Processing for Acoustic Vector Sensors

BACHELOR GRADUATION PROJECT

Group K-1
Wouter Kayser, Lennart Noordsij and Olmar van Beurden
4360966, 4359720 and 4368150

July 14, 2017

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Delft University of Technology

EXECUTIVE SUMMARY

The goal of the project is to design a system to detect and classify events from sound pressure and particle velocity data acquired by acoustic vector sensors. When an event is detected it has to be classified, and sent to the main station.

This system is implemented in MATLAB. A Short Term Averaging / Long Term Averaging algorithm has been used to determine a detection threshold. After detection the event is classified by obtaining relevant frequency data. By means of simulation, the optimal parameters have been calculated to get the best results, looking at false positive rate, false negative rate, time resolution, and frequency resolution.

The project has laid a basis with a solid platform for simulation. Before a final product is ready, many parts need to be implemented. These steps are mostly documented in the discussion further in this thesis.

Contents

List of Abbreviations

AVS	Acoustic Vector Sensor
BAP	Bachelor Graduation Project
DFSWT	Digital Fourier Square Wave Transform
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
FIFO	First In First Out
FPGA	Field Programmable Gate Array
Pmod	Plug-in Module
RMS	Root Mean Square
SNR	Signal to Noise Ratio
STA / LTA	Short Time Averaging over Long Time Averaging
UART	Universal Asynchronous Receiver/Transmitter

Introduction

This thesis will describe the results and process of the development of the DSP module for an Acoustic Vector Sensor (AVS) array system that will be used to detect and localize audio sources. An AVS is capable of measuring both sound pressure and particle velocity. The assignment was commissioned by Microflown, the company creating the AVS. The main goal of the project was to reduce the data transmission through bandwidth limited wireless channel. The coming sub-chapter will describe the problem and also the framework in which the project falls. Furthermore a short overview of what the DSP module will be doing, and how the most important functions are implemented is given. Then a state-of-the-art analysis of relevant literature is done, explaining the theoretical background of the project.

After these introductions the requirements will be laid out and there will be discussion on why these are important. Then the design and its performance will be discussed. Finally, there will be recommendations on how to improve and expand the system, as well as a conclusion on the overall findings. A link to the code can be found in the Appendix.

1.1 Problem Statement

The goal of this Bachelor Graduation Project is to design a system for sound localization using an array of Acoustic Vector Sensors. This is done using a predefined framework, in the way it has been discussed with the supervisor.

In figure 1-1 the framework of the project is shown. At stage 1, the AVS array is shown. All of the sensors measure the sound pressure, as well as the X and Y acoustic velocity of the sound wave. The sensors are made by the company Microflown. These are state of the art and often only one at a time is used. However this time an array is used to gather more data in order to improve on the resolution of localization.

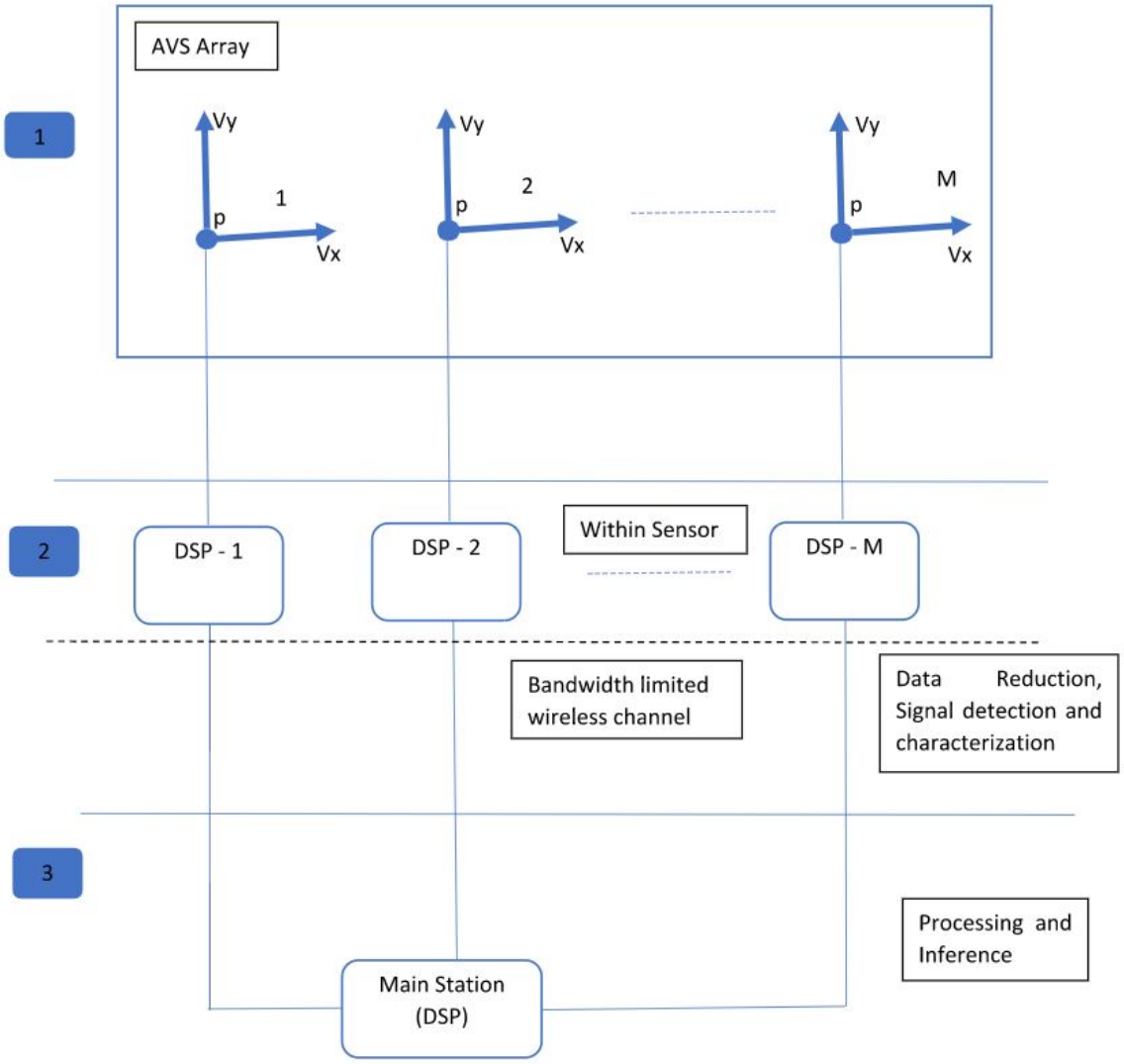

Figure 1-1: The framework of the project

In the second stage, the sensor data is processed by a DSP module. Every AVS has a DSP module to do on site calculations, which minimizes the amount of data to be sent to the main station, while making sure that all the important information is transmitted. This is important because the bandwidth between the DSPs and the main station is limited. Besides detection, the DSP also characterizes the events.
The third stage is the main station. The main station receives the data sent from the DSPs and processes it. This means the main station uses an algorithm to accurately determine the position of detected sound sources. The main station is also in control of the sensor network, and will be what the operator interacts with.

To realize this project, three components have to be designed. These parts are the DSP module, the wireless link, and the main station. Another group will work on the main station and the necessary algorithms. The wireless link will be put aside for the moment. This thesis will focus on the design process of the DSP module.

1.2 System Overview

Looking at the system overview of the DSP module, the first thing needed is communication with the sensor. After this step some sort of buffer or storage needs to be implemented to store the incoming data. This is needed to give time to process the results and to later also send the samples recorded before an event is detected. The next step is event detection. In this thesis we will be using an Short Time Averaging / Long Time Averaging (STA/LTA) algorithm, as this gives an adaptive trigger level and can be easily extended when needed. This algorithm will be further described in section 1.3.3.
After this the signal needs to be characterized so that the main station has more metrics available. The primary metric will be the main frequency component of the signal. In order to properly detect the frequency, the noise has to be filtered to additive white Gaussian noise. The event classification and frequency estimation has to be gathered and transmitted together with the raw sensor data.
For transmission, another module has to be designed in order to to set up a wireless link with the main station. This module can also receive information from the main station to update the DSP module with new parameters or functions.

This all will be replicated per AVS so every sensor has its own processing module, which will reduce the amount of data that needs to be sent around.

1.3 State of the art analysis

This section contains information about AVS and event detection. This information is gathered from relevant papers, theses and the developer of the AVS, Microflown Technologies [1].

1.3.1 Acoustic Vector Sensor

Acoustic emissions can provide valuable information about the location of sound sources. Sound localization is usually done with multiple pressure sensors (microphones). Since pressure sensors are scalar and do not have intrinsic directivity they need to be placed in an array to obtain the data required for localization. The Nyquist spatial sampling theorem states that the distance between the

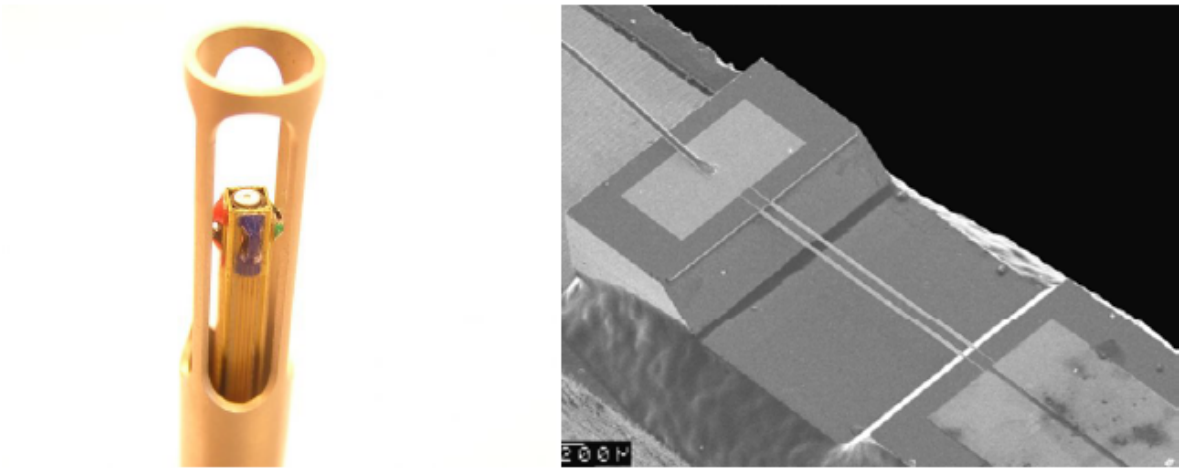

Figure 1-2: An Acoustic Vector Sensor (left) and an acoustic particle velocity sensor (right) [2]

microphones needs to be at least half of the wavelength. This means that there is a maximum wavelength which can be detected by an array with a set distance between the microphones. Equation 1-1 indicates that the distance between pressure sensors is inversely proportional to the frequency of the sound source you want to locate.

$$d = \frac{\lambda}{2} = \frac{c}{2f} \tag{1-1}$$

Where c is the speed of sound, and f the signal frequency. This means that for low frequency sources, the array length needs to be multiple meters. The Acoustic Vector Sensor provides a solution for this problem [1]. Figure 1-2 shows an AVS.

1.3.1.1 Principle

At any point in space, a sound field can be completely described by sound pressure (scalar) and acoustic particle velocity (vector). The first one being measurable by a normal pressure sensor, the velocity of a sound field only recently became directly measurable with the invention of the Microflown Acoustic Vector Sensor (AVS).
An AVS integrates a sound pressure transducer with three orthogonally placed particle velocity sensors (Microflown sensors) to measure all three components of the acoustic particle velocity. Each of the particle velocity sensors consist of two extremely sensitive heated wires. When air flows across the wires, the first wire cools down a little and due to heat transfer the air picks up some heat. Because the heated air now cools down the second wire, this wire cools down less than the first wire. This results in both wires having different temperatures, and therefore different resistances, causing a voltage difference proportional to the particle velocity. The result is also directional, since reversing the airflow will result in a reversed temperature difference. Combining three of these velocity sensors gives you the ability to measure the x, y and z direction of incoming sound waves. This means that the AVS has intrinsic directivity [1].

1.3.1.2 Characteristics

The acoustic vector sensor has a broad-banded frequency range from 0.1 Hz to 20 kHz, with a 360-degree field of view. Because of the multi-directional field of view the AVS can detect multiple sound sources simultaneously, even in complex situations and environments [3][4].

1.3.1.3 Performance

Since the AVS is for sound localization purposes, it can best be compared with an array of pressure sensors, as these have the same purpose. If you compare a single AVS with a single pressure sensor, you can draw the conclusion that the AVS will perform better in localization, as it has intrinsic directivity. Also, the intrinsic directivity means that the frequency resolution of the AVS is not spatially dependent [5].
Since an AVS consists of either 3 or 4 sensors, a pressure sensor and either x and y, or x, y, and z direction particle velocity sensors, it also requires 3 or 4 data channels. As stated in Kitchens thesis [6], when comparing an AVS with an array of 4 regular pressure sensors it will either perform similar or worse in terms of directivity and resolution. This means that an AVS is not inherently better in every case, and the value should be decided based on the application. The spatial independence and omni-directionality make AVS (array) implementations most useful for limited space applications and tracking of moving sources.
Another advantage of the AVS is that it is able to locate two separate sources. An N-element 3D AVS array is theoretically able to identify 4N-2 sources [7]. This is much better than the pressure sensor arrays, which are typically able to identify up to $\frac{N}{2}$ sources [8]. In the case of a priori information on the source signal, a single AVS has been able to detect up to 6 different speech sources [9].
Table 1-1 gives a short summary of the properties of the discussed localization setups.

	AVS	AVS Array	Pressure Sensor Array
Directivity	Intrinsic	Intrinsic and DOA	DOA
Amount of detectable sources	2	4N-2	$\frac{N}{2}$
Amount of data channels	4	4N	N
Frequency spatially dependent	No	No	Yes

Table 1-1: Properties of the different localization methods

1.3.2 Event detection and classification

The main use of AVSs is audio surveillance. Since sending raw data over requires a lot of bandwidth, it is more efficient to detect events at the sensor instead of at the main station. Therefore the AVS should be able to detect events, to determine when to send data to the main station. An event can be any type of sound data that you want to detect.

1.3.2.1 General Event detection and classification in wireless sensor networks

To detect events directly at the sensor, the data should be processed there. Only local data is available, though for improvement of accuracy, local exchange of sensor information can be used. Event detection is done by pattern recognition, which is the process of classifying raw data. In wireless sensor networks this is done by:

- Sampling: data is gathered and optionally preprocessed to eliminate background noise.
- Feature extraction: extract features from the data, with the goal to reduce dimensionality while preserving characteristic information.
- Classification: feature vectors are classified using either a priori knowledge from a training session or by statistical means alone.

Digitally a continuous signal cannot be saved, since it consists of infinite points. Therefore sampling needs to be done. Sampling is taking measurements with a certain frequency. From these points, the original signal can be reconstructed, if at least the Nyquist rate is reached: $F_s > 2BW$ [10]. That means that complete reconstruction is possible if the sampling rate F_s is at least twice the bandwidth of the sampled signal. Since detection starts from 0 Hz, the bandwidth is equal to the maximum frequency in the sampled signal.

Simple feature extraction can be determining the minimum, maximum, or mean energy of the data. More advanced feature extraction is however possible. For audio signals, relevant features to extract can be the volume distribution, in which the RMS value of a frame is calculated to estimate the volume. Another useful audio feature is the pitch contour, which can be used to differentiate between different types of speech. Also extracting frequency features can be useful to detect events. By calculating the Fourier transform of the audio signal, frequency features show. With this also the bandwidth and frequency centroid can be extracted [11].

Classification is done by classification algorithms. When training is used for classification, ideally only the nodes matching the relative position of the fixed node during the training should classify the event as belonging to one of the trained classes, while all other nodes reject the event [12]. When training data is not available then still classification can be done after feature extraction. Then manually certain features have to be assigned to certain events to classify the incoming data.

1.3.2.2 Audio surveillance event detection

Audio surveillance is not widely used, but has many advantages over video surveillance, for example:

- Less bandwidth, or memory storage is needed
- A single sensor can be omni-directional
- Events can be detected even when obstacles are present along the direct path. In localization this might pose issues however, due to reflections, reverberations, and echoes.
- Gunshots and shouts for example have little video counterpart.

- It can be used to detect certain events over 40 km away, something video surveillance does not even come close to [4].

Audio surveillance also has disadvantages, since audio signals of several sources tend to overlap, and mutipath propagation can result in echo and reverberation effects. Also the SNR is typically lower than in video surveillance.

The simplest form of event detection in audio sensing systems is by using the signal energy. If it surpasses a certain threshold this indicates an event. This however poses many problems as noise can be dynamic. A variable threshold can solve this problem, but periodically occurring noise can still trigger an event, while ideally this would be still classified as background noise. More advanced detection and classification models can overcome this problem [13].

1.3.2.3 Event classification

It can often be hard to classify events, since it is difficult to foresee all the different sounds that could be present in an environment. Also foreground sounds may have very different characteristics. Different sounds may need very different methods to correctly classify [13]. A lot of audio features can be used for classification. The simplest one, signal energy, has already been mentioned, but many others are available. Temporal, spectral, perceptual, spectral distribution, and correlation-based audio features can be distinguished. Out of these features a feature vector can be built. To reduce computational complexity it is desirable to keep this vector small. What features to use in the feature vector depends on the different events that should be distinguishable. [14].

1.3.3 STA/LTA

A method of threshold detection is by using a Short Term Averaging / Long Term Averaging (STA/LTA) algorithm. The STA/LTA algorithm is an algorithm that is based around detecting events, independent of background noise. The long term average measures the average energy value over a certain longer period. The short term average measures the average energy over a much shorter period of time. This is illustrated in figure 1-3, where the average energy of samples 4801 to 24000 is the LTA, and the average energy of samples 1 to 4800 is the STA. Each time a new sample is measured, sample 24000 is deleted, and each sample shifts left in the figure. Now a new STA and LTA are calculated. A pre-defined threshold factor exists, and if $STA > LTA * factor_1$ an event is triggered. This event stays triggered until $STA < LTA * factor_2$, in which $factor_1$ can be different from $factor_2$.

Figure 1-3: The STA and LTA windows

When an event is triggered two things can happen to the threshold. It can still update the long term averaging, so the threshold will update during an event, or it can be frozen during an event. Hybrids of the two are also possible. If the threshold is updated during an event, this might de-trigger the event prematurely. If the trigger is frozen however, a change in background noise during the event might make the system stuck in the event detected state for too long.

Another parameter that can be used in combination with this algorithm is a trigger number. Once the threshold has been surpassed, the systems will go in a pre-trigger state, until the threshold has been surpassed an amount of times equal to the trigger number in a row. Only then it will detect an event. This parameter can greatly decrease the amount of false positives, while keeping the threshold low. Therefore events with less amplitude can be detected, without many false positives.

1.3.3.1 Uses of STA/LTA

The STA/LTA algorithm is used in systems with dynamic background noise. Since this system adapts to the background, it automatically filters out this dynamic noise. The algorithm is widely used in seismic activity measuring systems, to only measure seismic activity that is wanted, and disregard seismic background noise [15]. It can however also be used in acoustic surveillance systems to compensate for electrical noise in the system, and acoustic background noise from the outside, like wind [16].

Programme of requirements

2.1 Introduction

The product that will be made has one primary function, which is the detection of acoustic events, and sending the accompanying sample data to the main station. The main priority with this, is to conserve the needed bandwidth. This means that the system to be designed has to be able to detect events based on acoustic data with certain accuracy, and as little mistakes as possible. Also the system requirements depend on the expected input, which comes from the AVS sensors, and output expected by the main station, which will be used there to use for localization.

Since we do not actually have the sensors for use in testing, it is important that the system designed can adapt to whatever input there is. If the amount of data channels, or the sample rate is changed, it is important that the system can be adapted to the new situation. It is therefore required to keep the system dynamic, and have all parameters, including the STA/LTA parameters fully tunable.

The system must be able to detect all events, but more importantly it should not miss as little as possible, with for now a focus laying on sinusoidal and impulsive signals. Therefore a requirement for the false negative rate, and false positive rate is needed. The system should be able to detect events correctly with an as low as possible SNR.

2.2 Functional requirements

The introduction above leads to the following concrete requirements:

1. The system must be adaptive to its surrounding noise to operate in beforehand unknown environments. That means that the threshold, and other parameters, can be set beforehand, and do not have to be changed by hand based on location the system is deployed, without conceding performance. This is to minimize the amount of operator time needed for tuning after every change in environment.

2. The system must be controllable from the main station. By wireless link, the system must be able to reset, and parameters should be able to be adapted, without actually being close to the system. This would be to change the type of events that should be detected, as not everyone would be interested in the same events.

3. The system must be fully tunable to adapt to any input, and needs from the main station. Meaning that if the sampling frequency changes, or the amount of input channels changes, the system can adapt by changing a single parameter and not the whole code.

4. The system must be prototyped onto hardware, and function according to the performance requirements. Since the system should be able to work on its own, without a computer capable of running MATLAB present, the DSP must be made into hardware.

2.3 System Requirements

5. The DSP software must be able to detect events for SNRs from 0 dB and up. Detecting means that at the given SNR it abides with the given false positive and negative requirements.

6. At an SNR of 0 dB the system must detect at least 90 % of the events, so a false negative rate of maximum 10%. At 3 dB SNR, the system should detect 100% of the events.

7. The false positive rate must be 1% at most, meaning a maximum of 1 false detection per 100 seconds. Otherwise there is an unnecessary usage of bandwidth to transmit signal data of just noise.

8. The time resolution should be 0.1 seconds, meaning the system should be able to detect the start and stop of an event with an accuracy of 0.1 s.

9. The software must be able to estimate the frequency components present in an event with a precision of 20 Hz. This requirement is needed to both classify events correctly, and for the main station to localize events more precise.

System Design

3.1 Sensor Data

The first part of the system is the sensor array, which outputs the data to be processed. In order to be able to efficiently test the processing algorithms, a sensor data simulator is designed. This is required to test the system response to certain events, without having to record the sample data in a real situation (e.g. loud explosions, specific frequencies). Another benefit is when we want to test the accuracy of certain algorithms. For this we would need to generate hundreds of datasets, taking a lot of time when done manually, while the simulator can create this data in seconds.

Figure 3-1 shows the sensor data simulator. It consists of 4 components. The following subsections will explain the function of the components.

All these components are also encapsulated in functions that allow arbitrarily amounts of events and arrays of AVSs. This means that the method to create a very simple setup, can also be used for complexer systems.

3.1.1 Event generator

The event generator is the base of the entire simulator and is used to generate the waveforms that are created by the event. It can adjust the amplitude, delay, and duration of all the events, and also the generator can ramp each signal up and down, using a trapezoid window.

The current system can create six different signals. The first type is a rectangular pulse and can be used to do a simple test. However this pulse will not be a signal found in the real world. The second type is a cosine wave, with this we can model sinusoidal sources like drones and many other objects. The third type is a white noise source, which is mostly used to compare with some of the earlier test data received from the supervisor. The fourth type is the usage of an external recording, with which we

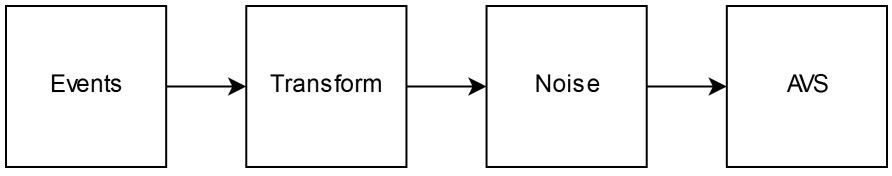

Figure 3-1: A block scheme of the sensor data simulator

can simulate the detection of more complicated audio fragments. This could be any other interesting signal for which the parameters need tuning. The fifth type of signal is a sine wave with changing frequency, which can for example be used to simulate Doppler shift. The last type of signal is another type of impulse and is one period of a triangle wave. This is closer to many impulsive noise sources than a blockshaped pulse.

3.1.2 Event transformer

This is a simple function that generates the necessary delays and calculates the angle of the event with respect to the X-axis. At the end of the function all events are transformed and summed. This is mostly here to model the transmission through air and can later be extended to also take into account more advanced propagation effects. Such as different absorptions for different frequencies and maybe to add different multi path signals.

This module can be extended to include the possibility for moving sources, as this is currently missing and will be very interesting for the main station.

3.1.3 Noise generator

This module will generate the amount of noise to reach the specified SNR. This noise is currently based on measurements made with the AVSs. This means that the noise power envelope should match the real world. There is also an option to generate white noise, but this type of noise is rare in the real world.

The noise is currently uncorrelated with the noise at the different AVS. In the real world, there is correlation between the noise[17]. However, the problem statement was to look into uncorrelated noise. Noise correlation would be an interesting feature to implement in the simulator.

3.1.4 AVS receiver

The last section will model the behavior of sensor itself. In this section a few approximations are made, as the datasheets of the AVS sensors were not available. This module takes into account what the rotation of the AVS is and calculates the acoustic velocity data. Then it also scales the data with gains separately for the pressure and acoustic velocity. The last part is the quantization noise created when sampling with a certain bit depth. This module could be extended to better simulate the AVS when the datasheets are available.

3.1.5 Other features

In the previous sections the processing chain has been discussed. However there are many other functions written to help out when setting up a simulation. These all were necessary to validate and compare the data that came from the simulation and the results from the MATLAB and FPGA implementation of the DSP module.

To set up arrays of sensors a special function has been made that set all of their specifications and the right distances from each other. They also had the possibility of turning the sensor, so that multiple angles could be simulated.
Another function will predict from the original settings for the system the expected moments when the event will start and end. These predictions can then be compared with the results that come from the simulated DSP module. From this the false positive rate and false negative rate can be calculated.

Apart from these functions another few have been made to visualize all of the data. There is one to plot all the location and relevant information of the sensors and the events. Next to this there are also functions to show the output from the DSP modules and overlay appropriate data. These all can also be found with the rest of the code.

3.2 Signal processing techniques

As explained in the system overview, the system needs to be able to detect signals and classify them. This section will discuss which signal processing techniques are implemented, why those techniques are chosen and how the performance of signal processing can be optimized.

3.2.1 STA/LTA

The first step is event detection. In order to detect events, the system has to differentiate between pure background noise and real events. Since the noise power might vary in time, a system with a static threshold can not be used. This leaves the options of a dynamic threshold, or event specific detection.

Although event specific detection will give the best detection results in most cases, it is not the optimal solution in this case. Most signals give the best detection results when using a cross correlation. The problem with the cross correlation is that it is impossible to implement it without a priori information on the signal type. Furthermore, specifically detecting a sine wave can be done with a Fourier Transform. However, continuous Fourier Transforms are very computationally expensive, and this costs a lot of energy. This is a major problem in a remote, battery operated system, because usually maintenance is expensive. A last option could also be cyclo stationary processing, which would allow the system do detect even more advanced signals.

All these considerations lead to a system with a dynamic threshold, more specifically a Short Time Averaging / Long Time Averaging approach. The basics of this method have been explained in the theoretical background section. The main advantages are detection of different kinds of events, adaptivity to varying noise levels, and very tunable parameters. A downside is that the approach is based on comparing signal power to noise power, which means the performance will be very dependent on the SNR. Figure 3-2 contains a block scheme of the system.

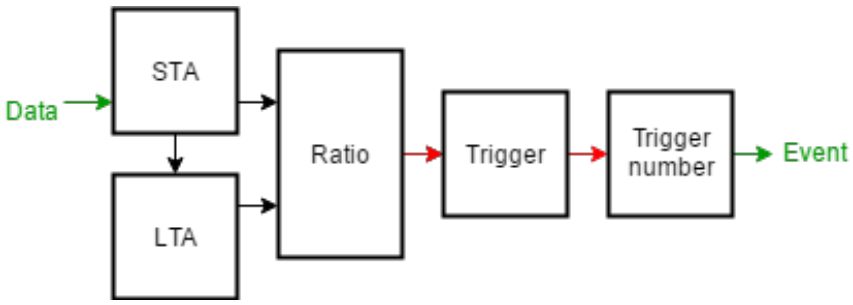

Figure 3-2: A block scheme of the STA/LTA system

3.2.1.1 Averaging window sizes

The window size is an important parameter. It influences the time resolution, as well as the reliability of detection.
In order to get an average which fits within the PoR, the window size has to be longer than the duration of a single wave of the signal which has to be detected. The signal length in samples is expressed in equation 3-1,

$$l_w = \frac{f_s}{f_{sig}} \tag{3-1}$$

where f_s is the sample rate of the AVS and f_{sig} is the minimum frequency that will be averaged properly.
Besides the frequency requirement, the window size also influences the detection of a short time pulse. If the window is much larger than the length of the pulse, its average energy over the whole window will be low. This means short pulses with low amplitude are hard to detect in a system with a large short window size.
Furthermore, the ratio between the short and long window needs to be sufficiently large, as the long window has to be an even more accurate average. However, the samples have to be stored in order to compute the mean. This means that a large window also requires a large memory size.
All these factors need to be taken into account when determining the window size.

3.2.1.2 Threshold

As stated in the theory section, an event is triggered if $STA > LTA * factor_1$. If this factor is a large number, the signal power needs to be much higher than the noise power in order to be detected. This leads to false negatives. Increasing the factor means increasing the required SNR for detection.
On the contrary, when this factor is lowered a small deviation in signal power due to either noise variance or an increasing noise level will be detected as a real signal more often. This leads to false positives. The noise variance is also dependent on the window size, as a small window will give large deviations due to inaccurate averaging.
For this threshold two different values can be chosen that need to be exceeded to start or end an event. This will act as a Schmitt trigger. The effect will be that the event detection will not rapidly turn off and on when the ratio is at the threshold. The result of this is that events will be less split across multiple detections.
Because we wanted to detect both short and long events, the decision was made to implement the system using a frozen threshold, as a non-frozen threshold would prevent longer events from being detected and transmitted to the main station until the very end. This means that once the threshold is exceeded, only the STA updates, while the LTA doesn't, until $STA < LTA * factor_2$.

3.2.1.3 Trigger count

The downside of the threshold is that a single detection of a value above the threshold does not have to be an event. If a detection is caused by noise variance, it will only last one or a few samples. Whereas if it is a real event, the signal power will systematically be higher than the noise power. This means that event detection can be based on multiple concurrent detections instead of a single detection. This idea is implemented with a trigger count, which counts the amount of times that the STA power is detected to be more than a factor larger than the LTA power.

In summary: if $STA > LTA * factor_1$, then $triggerNumber = triggerNumber + 1$, if this is the first time the system goes into pre-trigger state, and the LTA is frozen. If $triggerNumber = triggerCount$ an event is detected.

Increasing the trigger count will decrease the chance of a false positive, while not influencing the chance of a false negative, as long as the event lasts longer than the trigger number.
Another option would be to have the AVS in an array communicate with each other upon detecting an event. This would give an advantage in terms of joint event detection, since the chance of a false positive is significantly lower when multiple AVS detect an event at the same time. The problem,

however, is that processing the AVS data has to be done separately as there is no hardware to establish communication. Also, the chance of a false positive can be reduced to practically zero with the implementation of the trigger number.

3.2.2 Noise whitening

After the inspection of the measurement data, it was found that the noise that the AVS receives is not white. For better operation in later stages of the algorithm, e.g. frequency estimation, the noise needed to be white. This meant that a filter had to be designed to whiten the noise. The disadvantage to this is that the received signal also gets transformed, however most of the attenuation was found at lower frequencies. This is not a big disadvantage as most low frequency events contain more energy. The filtered signal will also not be used to determine the location or direction as part of the data would have been lost.
Creating the filter has been achieved by making a recording without events. This recording than has been transformed into the frequency domain and taken the inverse of off it. The next step was to convert this signal into dBs and fit a multiple order function to. Which gave a smooth approximation of the inverse noise envelope. Using the filterBuilder tool built-in in MATLAB, this then has been transformed into a filter which could be applied to the received data. The result can be found in figure 3-3
This filter should be custom made for all the sensors and their environment, as the filter is built for a specific noise spectrum. Since different environments and sensors will have a different noise spectrum, they require a custom made filter. This can be automated at start up, when no events are detected, and thus only noise is measured.

3.2.3 Fourier Transform

To classify events it is important to gain as many characteristics as possible. Spectral density information is very useful when it comes to acoustic data. An obvious way to gain insight in the spectral characteristics of a signal is by using the FFT. This technique is used to determine the available frequency components of any input.
The main issue with the FFT is its computational complexity. Although it does give insight into signal characteristics, it is very unnecessary to continuously apply the FFT to the input when there is no detection. Therefore, the FFT is only computed after an event is detected.
The frequency accuracy is directly impacted by the amount of samples the FFT is applied to. An N time samples FFT divides the frequency spectrum into N bins. The maximum deviation between the measured frequency and the real signal frequency can be calculated with equation 3-2.

$$f_{error} = \frac{f_s}{2 \cdot N} \tag{3-2}$$

This means that for larger N, the frequency estimation is more accurate. The downside is that it is less time accurate. This is not a problem for constant frequency sources, but can introduce issues in time varying frequency sources as the frequency over the sample will be averaged.

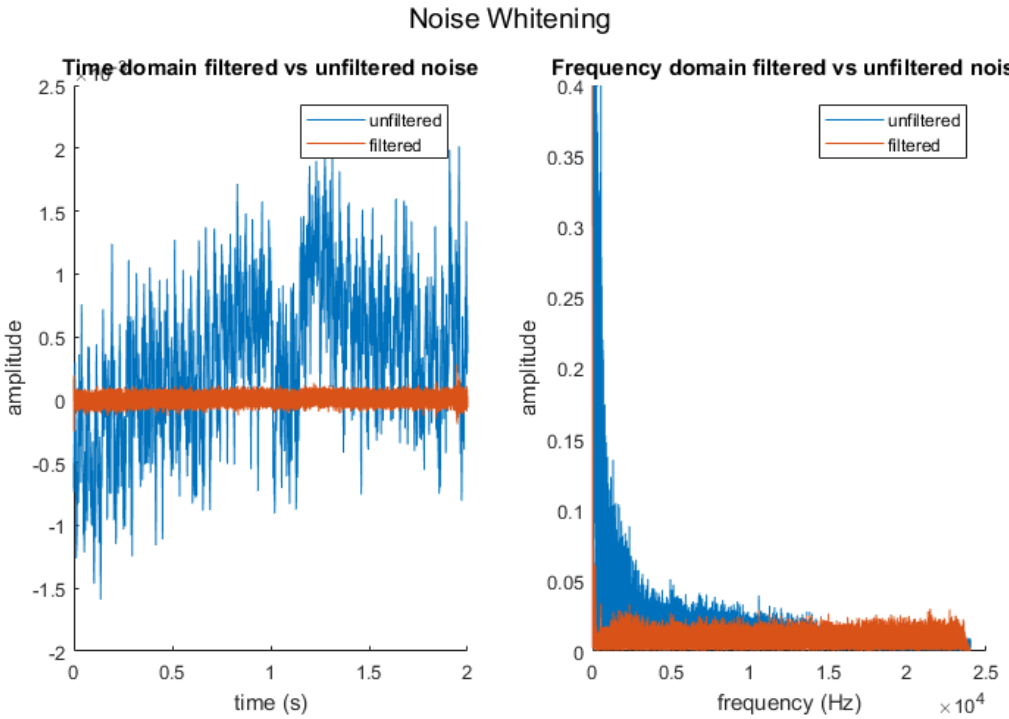

Figure 3-3: Noise whitening filter

3.3 Parameter tuning

In order to optimize the parameters and verify if they meet the requirements, a simulation program has been designed. The parameters to be optimized for event detection are the window sizes, the STA/LTA threshold factors and the trigger number. For frequency detection these are the frequency threshold factor and the amount of samples per FFT.

3.3.1 Event detection parameters

The impact of increasing the event detection parameters has been listed in table 3-1.

	Bigger Window Size	**Higher Threshold factor**	**Higher Trigger number**
Pros	More accurate averages Lower false positive rate	Lower false positive rate	Lower false positive rate
Cons	More memory required Less sensitive to short time signals	Higher false negative rate	Less sensitive to short time signals

Table 3-1: Effect of increasing STA/LTA parameters

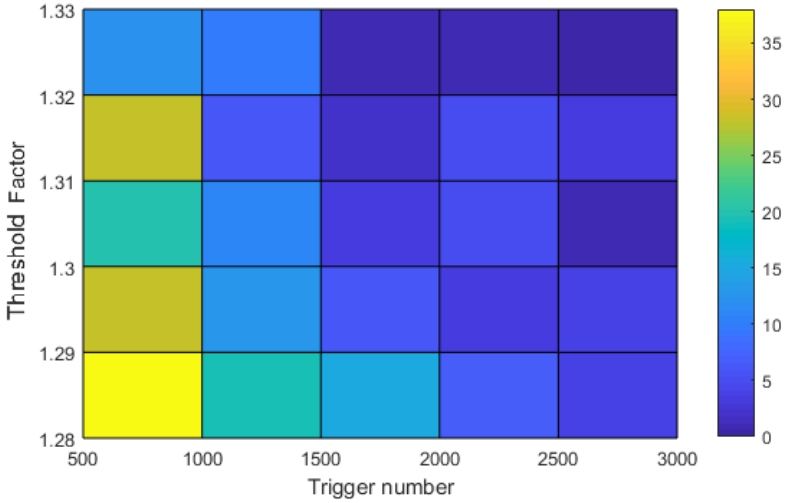

Figure 3-4: False positives per 1000 seconds

The optimization of these parameters has been done in 2 steps. First, the window size has been determined according to the time and frequency requirements. Then, the trigger number and threshold factor are balanced in order to meet the false positives requirement, the time resolution and the detection SNR requirement. As the detection is not frequency dependent, this has been done with a test signal on a set frequency. Proving that the chosen parameters are sufficient for every frequency will be done in the verification section.
In order to make sure that the long time window can average the signal reliably, the long window has been set to 24.000 samples, which is equal to half a second at the sampling frequency of 48 kHz. Equation 3-1 shows that any signals of 2 Hz or higher will be averaged properly, which will be where most of the noise will be. The short window is set to 4.800 samples, which is 0.1 seconds. This means it will be difficult to detect low power signals shorter than 0.1 seconds. Since the time resolution requirement is an accuracy of 0.1 s, this is within the requirements.
A simulation tool has been designed to find how the false positive requirement can be met. It counts the amount of detections in a simulated noise signal, using the noise profile from the real AVS data. These detections are always false positives, as there are no events in the simulation. Figure 3-4 gives a simulation of 1000 seconds of noise for varying factor and trigger number. It is clear that in order to minimize the false positive rate, the trigger number and threshold factor should be as large as possible. In figure 3-5, the event detection rate is shown for different SNR and factor. The event is a sinusoid of 5 kHz that lasts for 1 second. The event counts as detected if the start time and stop time of the detection are within 0.1 seconds of the real event start and stop times. The detection rate is determined as the average result of a Monte Carlo simulation with 1000 trials. The trigger number is set to 2500 in this simulation.
In order to maximize the detection rate, the threshold factor should be as low as possible. For a factor of 1.3, the detection rate at an SNR of 0 dB is higher than 90%. This meets the requirements.

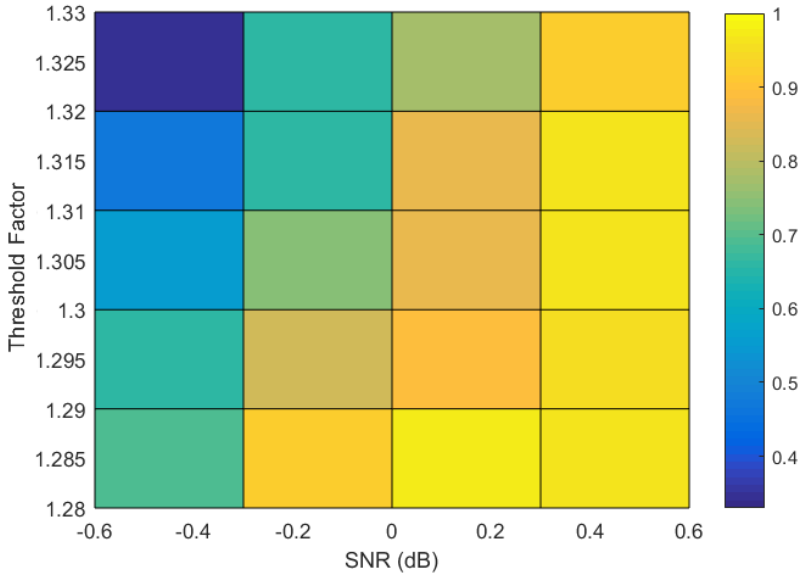

Figure 3-5: Detection rate of a 5 kHz sinusoid

3.3.2 FFT parameters

Setting the FFT sample size has been done based on the required frequency resolution. The requirement is that the frequency is determined with an accuracy of 20 Hz. Equation 3-2 with $f_s = 48$ kHz and $f_{error} = 20$ Hz leads to $N_{min} = 1200$. As most FFT hardware implementations only work in powers of 2, the sample size is determined to be 2048 samples. This meets the frequency accuracy requirement as this leads to an accuracy of ± 12 Hz.
To determine which frequencies are present in the signal, the power of each frequency bin is compared to the average power per frequency bin. If the power is above a threshold factor times the average power per frequency bin, the frequency is detected.

Increasing this threshold factor will reduce the detection of lower peaks, e.g. noise. However, if the frequency power is very low, the factor needs to be very low in order to detect the signal. Also, when the signal type is an impulse, there should not be any frequencies detected. In order to detect both strong and weak signals properly, the threshold needs to be adaptive.
This is depicted in figure 3-6. Note that the colors do not have any meaning. A point in the graph means a frequency that has been detected. The detected signal is a linear frequency sweep from 1 kHz to 2 kHz.The first plot has a threshold factor of 70 and the second plot a factor of 10. In the first plot, the low power start from 1 to 1.2 kHz is not detected optimally. This means the power is not high enough to pass the threshold factor. In the second plot, the frequency power is so large that the low threshold factor causes a lot of detections in frequencies close to the real frequency, which makes the frequency estimation very broad.
The last plot uses a factor adaptive to the signal power. The detection start with a factor of 70. If no frequency peaks are found, the factor is lowered by 10. This means the factor will change from 70 to 10 until a frequency has been detected. In this way, the most dominant frequency is always found, without having to lower the factor to the point where the detection broadens the estimated frequency. In this way a frequency can be estimated with the required accuracy.

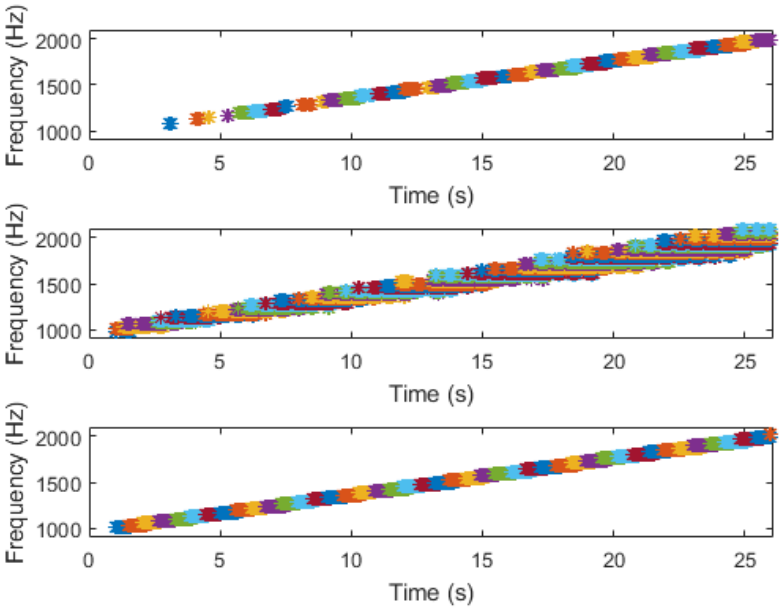

Figure 3-6: Frequency with frequency factor: 70 (top), 10 (middle), and an adaptive factor (bottom)

3.3.3 Conclusion

The design of the system, using all of the sub systems explained in the previous sections, can be seen in figure 3-7. A green arrow indicates an in- or output, a black arrow indicates a data stream and a red arrow indicates a control signal or parameter. The transmission part has not been implemented yet. The final values for all the parameters are given in table 3-2. In the verification section the performance of these parameters will be evaluated for multiple different signals in order to determine if the system meets the requirements.

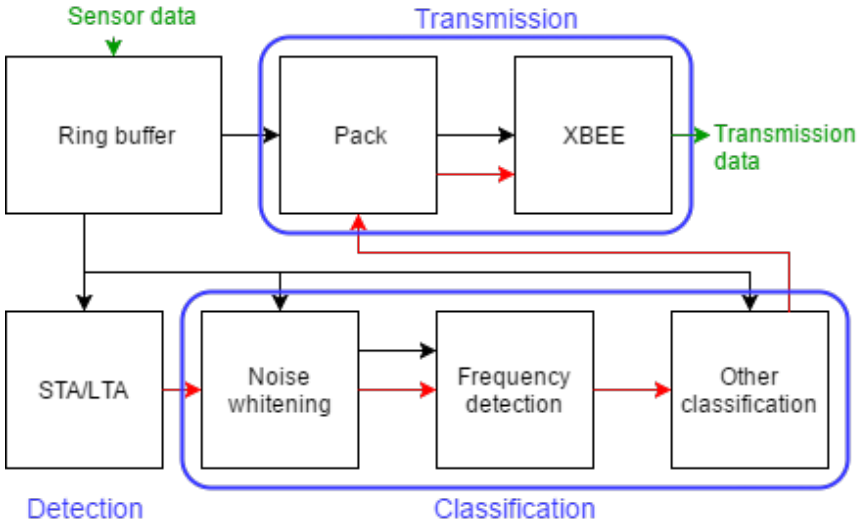

Figure 3-7: An overview of the complete DSP system

Sample rate	48 kHz
Long window	24000 samples
Short window	4800 samples
Threshold factor	1.30
Trigger number	2500
FFT sample size	2048
Frequency factor	70, 60, 50, 40, 30, 20, 10

Table 3-2: The values for the parameters chosen after simulation

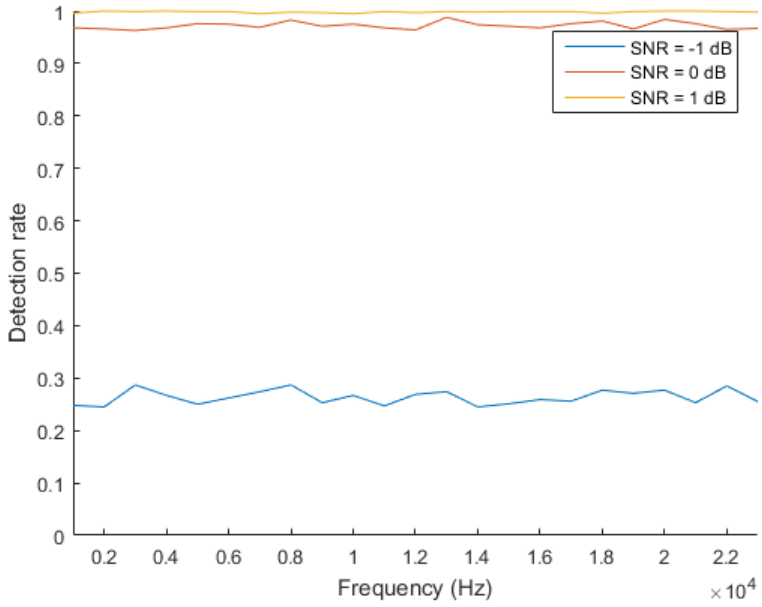

Figure 3-8: Detection of sinusoidal signals with different frequencies

3.4 Design verification

The values calculated in the parameter tuning section have been optimized for the simulated data. This section contains an analysis to verify that the system meets the detection requirements for all frequencies and pulses. This verification is aimed at the event detection, as we have not yet developed a tool to measure the classification performance. Also a number of real measurements are used to verify that these are also detected. The parameters used are the ones given in table 3-2.

3.4.1 Sinusoids

In the parameter tuning section, the parameters are chosen based on a single frequency as the detection varies only slightly with changing frequency. Figure 3-8 gives a plot of the detection rate for varying frequencies in the range from 1 kHz to 23 kHz. The detection rate is the average detections in 1000 trials. The requirement of a detection rate of 90% or higher is met for all these frequencies with the given parameters.
Figure 3-9 shows the detection rate for frequencies between 100 Hz and 1 kHz. For very low frequencies, the detection becomes more difficult. The requirement of 90% is met for frequencies of 360 Hz and higher. With increasing SNR, even these lower frequencies will approach a detection rate of 100%.
In conclusion, the system meets the 0 dB requirement for frequencies of 360 Hz and above. The 100% detection rate at 1 dB is also valid for these frequencies, which means the requirement of 100% detection at 3 dB is also met.

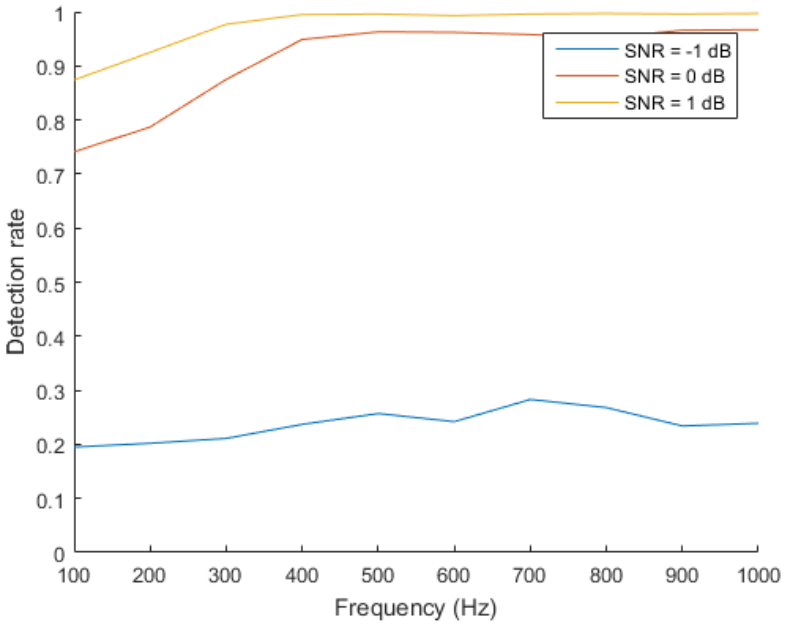

Figure 3-9: Detection of sinusoidal signals with low frequencies

3.4.2 Impulse signals

In figure 3-10 the detection rate of impulse signals with different duration is given. As the required time resolution is 0.1 seconds, the simulation only takes into account pulses with this duration or longer. The 90% detection rate requirement is not met at an SNR of 0 dB, it is met for pulses of at least 0.14 seconds long at an SNR of 1 dB. The 100% detection rate is met for pulses of 0.12 seconds and longer at 3 dB, as the detection rate on a pulse of 0.1 seconds is 99.5%.

3.4.3 Measured signals

In figure 3-11 impulse sounds created by clapping hands have been used as input for the system. The crosses above zero indicate the start of a detected event, while the crosses below zero indicate the end of a detected event. As can be seen the system has detected every impulse in the data. Between seconds 3 and 5 of the data four impulses have been detected as one event because they are close together. This is not a problem, because all the event data will be sent to the main station due to this.

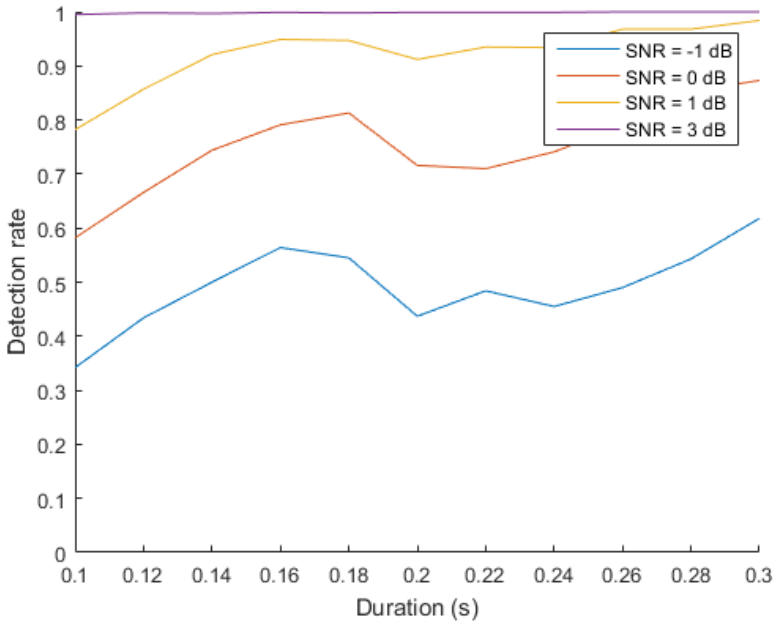

Figure 3-10: Detection of impulse signals with various duration

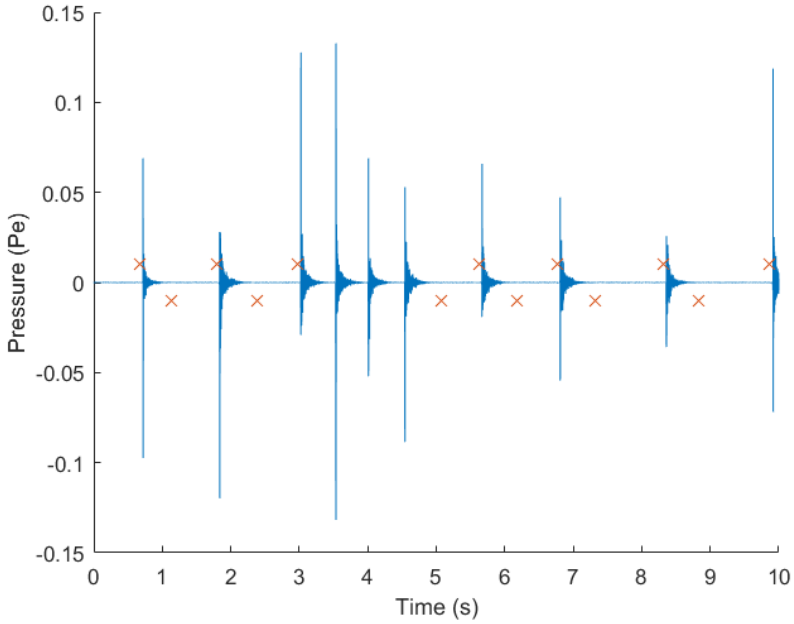

Figure 3-11: Impulse sounds

In figure 3-12 a frequency sweep between 2 kHz and 3 kHz can be seen. The colors in this figure have no meaning. A point means a frequency has been detected. The top figure shows the inputted data. It is visible that the sweep is detected completely, and as one event. However one other event that is detected can be seen in the figure. The amplitude of this event is small, but when zooming in it is a

pulse. The cause of this event is a mouse click. Even though it might be very small in amplitude, the system does detect this event.
The point of figure 3-12 is to verify the frequency detection with a measured signal. As can be seen during the sweep, the frequency is linearly increased. This corresponds to the linear frequency sweep. Therefore it verifies that the frequency detection does work on this measured signal, although no extensive performance against SNR analysis has been done. The accuracy is within the specified 20 Hz, as can be seen in the Parameter tuning section.

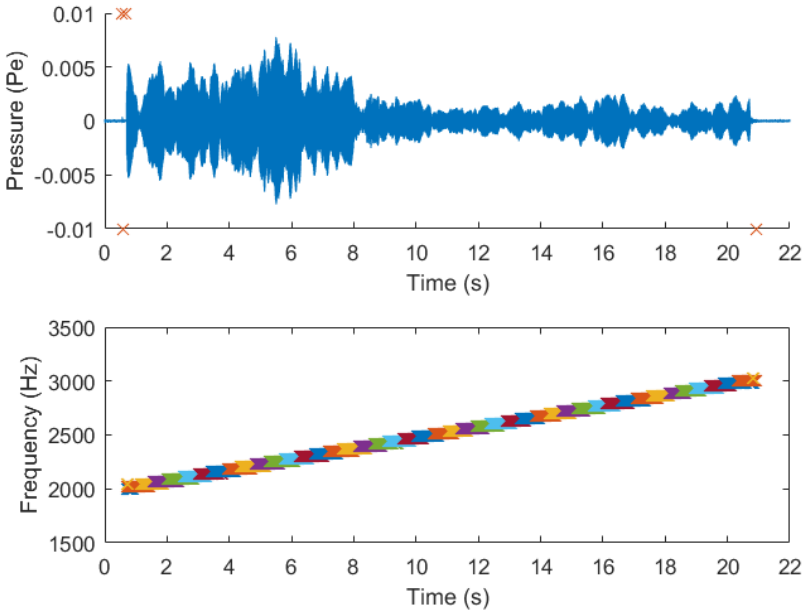

Figure 3-12: Frequency sweep between 2 and 3 kHz

Hardware implementation

4.1 Hardware implementation

In this section our effort to prototype our system is explained. For this an FPGA was used. For the project we wanted to use a DSP. However a DSP was not readily available. An FPGA on the other hand was, and a DSP can be programmed onto an FPGA.

4.1.1 Verilog translation

Our MATLAB code had to be translated to Verilog. A complete overview of the system can be seen in figure 4-1.

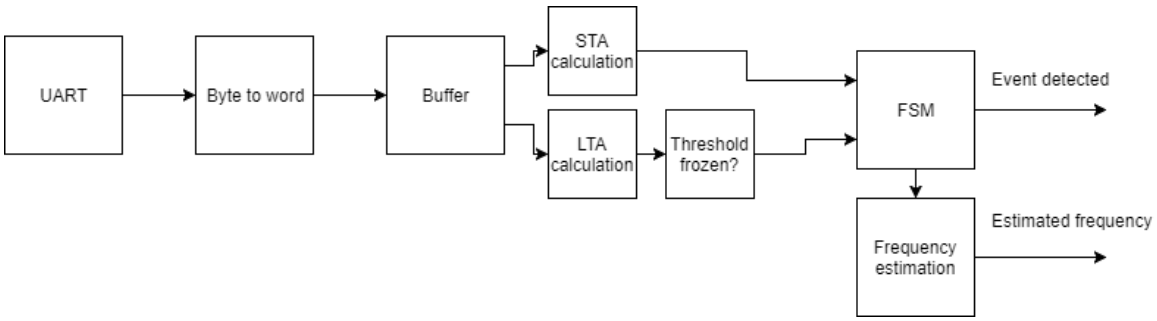

Figure 4-1: Overview of the FPGA implementation

Words containing sound data from MATLAB are sent to the FPGA. This connection simulates the AVS that would otherwise be directly connected to the FPGA. First a buffer was made. Every time a new word has been received by the FPGA, the buffer shifts all its data right by the wordsize, 16 bits in our case, and stores the new data in the first 16 bits of the buffer. The buffer decides what the first and last word belonging to both the STA and LTA are and outputs these.

Next the connection to MATLAB has been made. This was done by using an 8-bit UART connection. For the source of the UART code, see appendix 2.

Because the UART used is an 8 data bit UART, and the word size is 16 bits, there has to be a module added that concatenates the data bits to 16 bit words. Each time the UART module receives a new byte, this module updates the current word. Each two bytes received form a new word, which are sent to the buffer. A PMod USBUART chip has been used to make communication possible.

Every time the buffer is updated, the STA and LTA will also update. For both the STA and LTA update, the same module is used. In the Top Level module these two are distinguishable. The difference between the two are the inputs. These come directly from the buffer and are either the first and last words belonging to the STA window, or the first and last words belonging to the LTA window.

The STA and LTA are calculated by adding the first word squared to the STA or LTA, and subtracting the last word squared.

To be able to freeze the LTA, a next module decides if for the calculation of the threshold either the current LTA is passed on, or if it should be frozen, the previous value is remembered instead.

Both the STA and the LTA, either current or an older version, are passed on to the FSM module. This module can be seen in figure 4-2, and decides if there is an event or not, and only starts to work when the buffer is completely filled up. If it is not, then there is no correct calculation yet of the STA or LTA, and therefore trying to detect an event makes no sense.

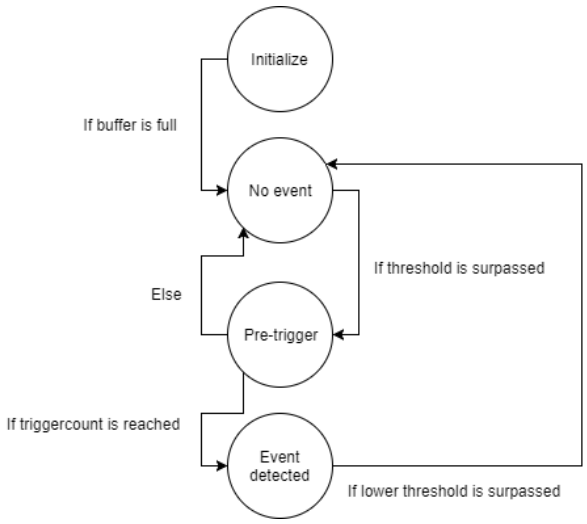

Figure 4-2: Overview of the FSM

This module goes into no event state, and then compares if $STA \times LTASize > threshold \times factor_1 \times STASize$. Since the STA and LTA are not yet actually averaged before, this is where that is compensated for, by multiplying the one side with the LTA size, and the other with the STA size, so actually the averages are compared. The reason why it is only done here, and why a multiplication is used as opposed to a division, is because in hardware translation the system rounds when getting decimal numbers. Therefore multiplying instead of dividing results in more precise detection.

If the threshold in the FSM is surpassed, the system goes into pre-trigger state. If the threshold is surpassed by triggercount times in a row, the system goes into event detected state. If not, the system goes back to no event state.

In event detected state, the system stays there until $STA \times LTASize < threshold \times factor_2 \times STASize$. Then the system goes back to no event state.

When the system is in pre-trigger state or event detected state, the LTA update is frozen.

4.1.2 MATLAB integration

To test the written Verilog code, a .dat file has been created through MATLAB, in which every signal that can be created with MATLAB can be tested through ModelSim. Therefore the ModelSim code can be tested quickly with a wide variety of signals, and added noise.
The FPGA has a UART connection to receive sensor data. For now the sensor data is simulated using MATLAB. MATLAB then has a serial connection that through the Pmod USBUART chip is sent to the FPGA. Via MATLAB both actual measured data, and self generated data can be send. This way the FPGA can be tested to see if the hardware actually works.

4.1.3 FPGA verification

To verify if the FPGA implementation works the event detected output has been put on a led. Then data has been simulated and sent with MATLAB, and by checking if the led was either on or off when expected, the system has been verified. The system has been tested with sinusoidal signals with a duration between 0.1 and 0.5 seconds, with a frequency of 20 Hz, 500 Hz, and 2000 Hz. The values used were 1.5 for the threshold factor and 50 for the trigger number, while the STA size and LTA size were 400 and 2000 respectively. The STA size and LTA size were chosen arbitrarily, while the threshold factor and the trigger number were calculated with the MATLAB script as optimal values for these window sizes.

The MATLAB script was run 100 times, with a signal duration of 1 s, for each possible combination of a duration of either 0.1 s, 0.3 s, and 0.5, s and a frequency of either 20 Hz, 500 Hz, and 2000 Hz. Each time it was checked if the system detected the event, and for the correct duration.

The duration of the signals did not influence the tests. The system gives the same results for events of 0.1 s, as for 0.3, and 0.5 s. Also the 20 Hz, and the 500 Hz tests gave the same results. Only the 2000 Hz test gave different results, which were consistent between the durations.
The results were that the system detects all tested signals down to -1.5 dB. For the signals of 500 Hz and 20 Hz, a 100% detection rate was achieved down to -2.5 dB. For the 2000 Hz signal the detection rate went down to 52% at -2 dB. For the other two frequencies the detection rate went down to 40% on -3 dB.
The false positive rate was 0.

The conclusion can be drawn that for the tested signals, with the indicated values the FPGA works above the indicated detection rate requirements. At this moment however there is not a way yet to efficiently test the FPGA. Therefore it cannot be guaranteed that the implementation works sufficiently for each signal.

4.1.4 FPGA frequency estimation

To classify information about a detected event on the FPGA frequency estimation is needed. This can be done like in MATLAB using the FFT, where all frequency components in the received signal can be sent to the main station. However an FFT module requires a lot of logic gates and computational power. Therefore another transformation can also be used, called the Digital Fourier Square Wave Transform (DFSWT).

4.1.4.1 DFSWT

After looking into using a less computationally complex system to find the main frequency components, two papers were found [18] and [19] on the subject of DFSWT. This is a novel approach of determining the frequency components in a signal. It is based on the WalshHadamard transform.
The main disadvantage of this transformation, is its reduced performance when the main component has an offset from the frequency off the bin. To better counter this problem a longer sample will be needed to process and additional averaging or processing to be performed.

The transform itself look a lot like the standard Discrete Fourier Transform. However in stead of the complex sinusoidal waveforms perfectly symmetrical square waves. Here is also a phase offset introduced that makes sure that the sinusoidal waves do not have zero-crossings. This solves the problem of checking for zeros that the standard Walsh-Hadamard transform has no standard solution for. Of interest is also the complex part, which is the same as the real part but 90 degrees phase shifted. The formula for this can be seen at (4-1), where the sgn function is defined in (4-2).

$$Z_s(k) = \frac{1}{N}\sum_{n=0}^{N-1} z(n) S_N^{nk}$$
$$S_N^{nk} = \operatorname{sgn}\left[\cos\left(\frac{k2\pi n}{N}\right) + \phi_0\right] + j\operatorname{sgn}\left[\sin\left(\frac{k2\pi n}{N}\right) + \phi_0\right] = I_S^T + jQ_S^T \tag{4-1}$$

$$\operatorname{sgn}(x) = \begin{cases} 1, if\ x > 0\ or\ x = 0^+, \\ -1, if\ x < 0\ or\ x = 0^-. \end{cases} \tag{4-2}$$

This can also be defined without sinusoidal signals. Which allows it to be implemented easier in FPGAs and reduces the overall complexity of the calculations. First in (4-3) the output data is redefined in real and complex parts. Then the two sets of symmetrical square waves are described in (4-4) and (4-5).

$$X(k) = \sum_{n=0}^{N-1} z(n) I_S^T$$
$$Y(k) = \sum_{n=0}^{N-1} z(n) Q_S^T \tag{4-3}$$

$$Is = \operatorname{sgn}\left(\frac{N-1}{2} - nk \bmod N\right),\ \text{for}\ \begin{cases} k = 0, \ldots, N/2, \\ n = 0, \ldots, N-1, \\ and\ N\ even. \end{cases} \tag{4-4}$$

$$Qs = \text{sgn}\left[\frac{N-1}{2} - \left(nk + \frac{N}{4}\right) \bmod N\right], \text{ for } \begin{cases} k = 1, \ldots, N/2, \\ n = 0, \ldots, N-1, \\ and\ N\ even. \end{cases}$$

$$Q_0 = 0, \text{ for } \begin{cases} k = 0, \\ n = 0, \ldots, N-1, \\ and\ N\ even. \end{cases} \tag{4-5}$$

This can also be easily implemented in a FPGA by using a more real time approach, where each sample the actions for each bin can be derived using a sawtooth waveform module. Then at the end of the module an accumulator can find the energy in the last section of the signal for each frequency bin.

Discussion

5.1 Communication with the main station

The transmission of the data through the wireless link is a point of interest, as the problem statement includes minimizing the amount of data to be transmitted. Although the scope of this thesis is on the signal processing, some steps have been taken in order to help reduce the data to be transmitted. The system is designed to be compatible with a number of data reduction techniques. This section will discuss a number of techniques to explore when designing the communications system.

5.1.1 Parameter adjustment from main station

One of the envisioned use cases has the AVS array at a remote location and one main station to view the results. This might make it hard to update the values and algorithms that are used by the DSP module. A valuable addition to the system then might be software and parameter updates over the air. The result of this is that the parameters can still be tuned after a long time, when more is known about the system. To implement this the DSP modules should all be also capable of receiving data and have a more advanced packet decoding module. On an FPGA this would also add a lot more complexity to update parameters during run time.

5.1.2 Request based data transmission

One of the requirements was to conserve the bandwidth that is used to send all the data to the main station. One of the ways to reduce data traffic is not transmitting. This is why a potential solution could be to first send all the characteristics of the event. Then the main station can determine if the event is needed and then request the sample data for the event. This would not change much on the DSP module except for more long term storage. Another advantage is that the main station could only request the signals with a high enough SNR. The result of this is that the main station could potentially start earlier with processing with the best available data. This technique could also be extended to facilitate the resending of data, when a transmission went wrong.

5.1.3 Data aggregation

Data aggregation means combining multiple data packets into one packet containing the useful data. This reduces the required bandwidth from the DSP to the main station, as well as the amount of channels required [20]. Data aggregation is widely deployed in sensor networks as it is one of the most effective ways to reduce bandwidth requirements [21]. In order to apply data aggregation, multiple data streams or packets have to be collected in a single sink [22]. This would suggest using a single substation which communicates with the main station for one or multiple arrays.

5.1.4 Packets

The data the sensor generates is three data streams of 48 kHz, with a word size of 16 bits. Including start and stop bits, this results in a receiving bitrate of 2.88 Mbit/s. Since the sending module needs to send both the received data and more during an event, the sending bitrate needs to be higher. The best way to divide this is to have the module send all recent data, that is generated sensor data and frequency data, and on top of that, send the other data that needs to be sent, which is the acoustic data before the event was triggered. After all the remaining data has been sent, the room for this data can be left empty or, if both the FPGA and the main station have this implemented, the bitrate can then be lowered.

5.1.5 Communications substation

As mentioned above, the communications substation can reduce the amount of data to be transmitted as well as the amount of channels required. Furthermore, implementing a means of local communication could allow communication between sensors in order to apply joint event detection as explained in the signal processing techniques section. Which would improve the diversity and accuracy of the event detection, as multiple algorithms can be used. The substation can also be used as a cache for software updates for the DSP modules. This all requires another module on location that acts as the master of all the sensors.

Conclusion

6.1 Conclusions

The goal of the project was to design a signal processing module for an AVS, in order to give the main station some basic information on signal classification. Also, it should reduce the required bandwidth to transfer the sensor data.
The first step is event detection. Data without an event will not be classified or transmitted, which leads to a massive decrease in required bandwidth in a low event environment. The event detection is able to achieve detection rates of over 90% at an SNR of 0 dB, up to 100% at SNR values of 2dB or higher. The false positive rate is lower than 1%, meaning less than 1 event every 100 seconds.
The data classification estimates the frequencies that are present in the signal. This estimation is accurate up to $\pm$ 12 Hz. The design of the system allows implementation of detection modules for other signal characteristics.
All of the detection and classification modules have adjustable parameters, which means they can be adjusted to their surroundings quickly.

6.2 Recommendations

Based on the results acquired in this thesis, we do the following recommendations:

Get AVS data in an as early stage as possible, to at use the true structure of the sensor output, as well as gain more realistic insight into the noise the sensor outputs, and event data outputted by the sensor.

Get the AVS sensors to use in measurements, testing, and system verification. In this thesis a lot of sensor data used is simulated, while using the actual data of the AVS could give more realistic results, and therefore a more realistic set of parameters usable for detection in the DSP.

Since the system is designed to work with far field data, it is recommendable that far field data is obtained. In this project we could only obtain near field data, for the only setup data was recorded with, was in a small room, resulting in all data being measured at a distance of about 1 meter only. Obtaining far field data could actually verify if the systems works as well in far field.

Future research is required into using arrays of data, the effect of different array setups, and the effect that local data transfer between sensors within an array could have, on their accuracy to the detect and classify events.

Also local sharing of data could prevent the same data being sent to the main station multiple times, and with that save bandwidth. Furthermore more extensive research in how to efficiently send data from the sensors to the main station is required.

Appendix

1. Code used for the project

All of the code discussed and all other code used in this project can be found on the following GitHub page: http://www.GitHub.com/Werser/AVS-Simulation.

The measurement data from the sensors is not available, as MicroFlown has a strong preference for not making the data available to the public.

2. UART source

UART module has been used from page: https://www.nandland.com/vhdl/modules/module-uart-serial-port-rs232.html

Bibliography

[1] Microflown AVISA, "Acoustic Vector Sensor," accessed 26-4-2017. [Online]. Available: http://microflown-avisa.com/acoustic-vector-sensor

[2] N. R. Krishnaprasad, "Acoustic vector sensor based source localization," Master's thesis, Delft University of Technology, 2016.

[3] B. Behrmann, "Acoustic Vector Sensor; Possibilities to enrich Unmanned Air Systems," accessed 26-4-2017. [Online]. Available: http://www.bciaerospace.com/turin/images/stories/conferences/microflown%20avisa.pdf

[4] Microflown AVISA, "FAQ," accessed 26-4-2017. [Online]. Available: http://microflown-avisa.com/faq/#A4

[5] D. F. C. W. Jing and D. P. Cabo, "Sound source localisation using a single acoustic vector sensor and multichannel microphone phased arrays," in *Internoise*, 2014.

[6] J. P. Kitchens, "Acoustic vector-sensor array processing," 2010. [Online]. Available: http://www.rle.mit.edu/dspg/documents/kitchens_phd_eecs_2010.pdf

[7] J. W. Wind, E. Tijs, and H.-E. de Bree, "Source localization using acoustic vector sensors: a music approach," in *Proceedings of NOVEM 2009 - Noise and Vibration: Emerging Methods*. Southampton, UK: Institute of Sound and Vibration Research, ISVR, April 2009. [Online]. Available: http://doc.utwente.nl/75423/

[8] M. Wax and I. Ziskind, "On unique localization of multiple sources by passive sensor arrays," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 7, pp. 996–1000, Jul 1989.

[9] S. Zhao and D. L. Jones, "Underdetermined 2d doa estimation using acoustic vector sensor," in *2014 9th IEEE Conference on Industrial Electronics and Applications*, June 2014, pp. 1087–1090.

[10] H. J. Landau, "Sampling, data transmission, and the nyquist rate," *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1701–1706, Oct 1967.

[11] Z. Liu, J. Huang, Y. Wang, and T. Chen, "Audio feature extraction and analysis for scene classification," in *Proceedings of First Signal Processing Society Workshop on Multimedia Signal Processing*, Jun 1997, pp. 343–348.

[12] G. Wittenburg, N. Dziengel, C. Wartenburger, and J. Schiller, "A system for distributed event detection in wireless sensor networks," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, ser. IPSN '10. New York, NY, USA: ACM, 2010, pp. 94–104. [Online]. Available: http://doi.acm.org/10.1145/1791212.1791225

[13] M. Crocco, M. Cristani, A. Trucco, and V. Murino, "Audio surveillance: A systematic review," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 52:1–52:46, Feb. 2016. [Online]. Available: http://doi.acm.org/10.1145/2871183

[14] G. Valenzise, L. Gerosa, M. Tagliasacchi, F. Antonacci, and A. Sarti, "Scream and gunshot detection and localization for audio-surveillance systems," in *2007 IEEE Conference on Advanced Video and Signal Based Surveillance*, Sept 2007, pp. 21–26.

[15] A. Trnkoczy, "Understanding & setting sta/lta trigger algorithm parameters for the k2," *Appl Note*, vol. 41, pp. 16–20, 1998.

[16] J. Kotus, K. Łopatka, and A. Czyżewski, *Detection and Localization of Selected Acoustic Events in 3D Acoustic Field for Smart Surveillance Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 55–63. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-21512-4_7

[17] M. Hawkes and A. Nehorai, "Acoustic vector-sensor correlations in ambient noise," *IEEE Journal of Oceanic Engineering*, vol. 26, no. 3, pp. 337–347, Jul 2001.

[18] E. Cabal-Yepez, T. Carozzi, R. de J. Romero-Troncoso, M. Gough, and N. Huber, "Fpga-based system for frequency detection of the main periodic component in time series information," *Digital Signal Processing*, vol. 18, no. 6, pp. 1029 – 1044, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1051200408000547

[19] E. Cabal-Yepez, H. Miranda-Vidales, A. Garcia-Perez, J. M. Lozano-Garcia, R. Alvarez-Salas, and A. L. Martinez-Herrera, "Harmonic component estimation through dfswt for active power filter applications," in *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, Nov 2013, pp. 810–815.

[20] R. Dunlap, "In-network aggregation in wireless sensor networks," Georgia Institute of Technology, Tech. Rep.

[21] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 14, no. 2, pp. 70–87, April 2007.

[22] L. Krishnamachari, D. Estrin, and S. Wicker, "The impact of data aggregation in wireless sensor networks," in *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*, 2002, pp. 575–578.