Swarming Module

Project document

Version 1.2

June 27, 2016

M. van Wilgenburg, W. Mukhtar, E. van Splunter, M. Siekerman, T. Zaal and M. Visser

List of Figures

1 Oscilloscope measurement of the received acoustic signal at a distance of 1 meter. 11
2 Unit circle where points P and Q are mirrored on the X-axes . . 13
3 Angle determination using trigonometry 14
4 Example receiving signal PLL 15
5 PLL demodulator schematic 16
6 Test set-up demodulating with the PLL 16
7 The test results of the PLL plotted against the expected results . 17
8 Block representation of the frequency measurement using a Xmega [?] 18
9 Example modulated signal 18
10 Comparator test set-up 21
11 Measured results and expected results of comparator based measurements 21
12 Time delay between ***f*** (sent) and ***g*** (received) 22
13 Auto correlation process of a cycle of a sine wave 23
14 Algorithm simple correlation 25
15 Expected results from the correlation algorithm 27
16 Test set-up for cross-correlation 27
17 Test set-up for cross-correlation 28
18 Measured results from the correlation algorithm 28
19 Deviation between measured and expected results 29
20 Graph of expected values versus measured values 29
21 Histogram 80 cm measurement 30
22 Histogram 90 cm measurement 30
23 Histogram 100 cm measurement 30
24 Histogram filter algorithm 31
25 Results after filtering 33
26 Graph of results after filtering 34
27 Overview of the swarm module design 38
28 Modulator 39
29 Demodulator 40
30 Localization Module 42
31 Modulator block diagram 43
32 Block representation of the receiving side 45
33 State diagrams for the distance measurement. The diagram on the right is for the receiving side, and the left diagram is for the sending side. 46
34 main algorithm for updating the swarm population including range requests 51
35 One population list frame 52
36 1:"send acoustic" when the swarm module is in sensing state 2:"process receive", when the swarm module is in receiving state 55
37 Arrangement of the swarmbee measurement for 3 to 100 meters. 65
38 Table of the swarmbee measurement for 3 to 100 meters. Values of zero are incomplete measurements of the swarmbee module. J or N indicate if there is a connection between the swarmbee's. . . 65

39 Graph of the swarmbee measurement for 3 to 100 meters. 66
40 Putty log of the filled population list 67

List of Tables

1 Measured times in clock pulses, pre scaler is set to 1 16
2 Swarming localization specifications 36
3 Specifications number of units 36
4 Wireless communication module 37
5 Central operating unit 37
6 Demodulation specifications 40
7 Wireless communication module 40
8 Central control unit 41
9 Central operating unit 50
10 Swarming localization specifications 63
11 Wireless communication module 64
12 Swarmbee LE module 64
13 Swarmbee LE module 68

Contents

1 Abstract **5**

2 Introduction **6**
2.1 Project methodology 6

3 Project Definition **7**
3.1 Problem analysis 7
3.2 Context analysis 7
3.3 Problem definition 8
3.4 Goal 8
3.4.1 Swarming module 9

4 Research-question **9**
4.1 Sub-questions 9

5 Research **10**
5.1 Acoustic Sensing 10
5.1.1 Signal analysis 10
5.2 Localization 11
5.2.1 Relative distance 11
5.3 Relative angle 12
5.4 Signal processing 14
5.4.1 Analog Phase locked loop 15
5.4.2 Conclusion demodulating with the PLL 17
5.4.3 Demodulating using the comparator of a Microprocessor . 17
5.4.4 Conclusion demodulating using the comparator 21
5.4.5 Cross-correlation 22
5.4.6 Cross-correlation implementation 23
5.4.7 Simplified cross-correlation implementation 25
5.4.8 Cross-correlation results 27
5.4.9 Filtering cross-correlation 31
5.4.10 Cross-correlation results with filtering 33
5.4.11 Conclusion cross-correlation 34
5.5 Swarm communication network 35
5.5.1 Network criteria 35
5.5.2 Recommendation and conclusion 35

6 Specifications **36**
6.1 Wireless communication 36
6.2 Central operating unit 37

7 Functional specification **37**
7.1 Swarming module 38
7.2 Short range localisation sensor 38
7.2.1 Modulation 39
7.2.2 Demodulation 39
7.3 Wireless communication 40
7.3.1 Central control unit 41

8 Design **41**
8.1 Short range localization sensor 41
8.1.1 Acoustic actuator sensor 41
8.1.2 Acoustic sender 42
8.1.3 Acoustic receiver 45
8.1.4 Angle measurement and relative location 48
8.2 Swarm communication implementation 49
8.2.1 Central operating unit 49
8.2.2 Communication 50
8.2.3 Swarm population 51
8.2.4 Acoustic sensing send/ receive 53
8.2.5 Population list 59
8.2.6 Communication test plan 63
8.2.7 Test results 65
8.2.8 Wireless communication 67

9 Results and Discussion **68**

10 Conclusion **69**
10.1 Swarm communication 69
10.2 relative localization 69

11 Recommendations **70**

12 References **72**

1 Abstract

This document describes the technical aspects of the so called "Swarming module". This is a module made to provide relative localization to other units in a swarm of robots. A swarm of robots must meet the following criteria to qualify as a swarm: Autonomous, Large numbers, Limited capabilities(per unit), Scalability and robustness, and distributed coordination. The swarming module this project set out to make provides distributed coordination because it provides communication and localisation for a unit in a swarm. The criteria scalability and robustness and large numbers are important for the specification of the module. The project set out to find a robust and relatively simple implementation that was also scalable. It was decided to choose acoustic signals as means for localization for short range, due to their relatively low propagation speed compared to radio waves. To provide communication and long range localisation the Swarm-bee module was used, which allows for a scalable communication network. Due to many setbacks, the project did not succeed in implementing a finished product but a lot of useful research and tests where done so that some substantiated recommendations can be made for future work.

2 Introduction

This document describes the technical aspect of the "Swarming Module". This module is developed by students of the Amsterdam University of Applied sciences in collaboration with the Delft University of Technology. This project is a part of a running research program that is looking into the benefits of swarming compared to "standard" approaches, which would be one bigger and more complex robot doing all the work. Finally this program looks at the applications swarming might have on Mars.

This project contributes to the program by developing a so called "Swarming Module". This module will allow units in a swarm to determine the relative location to one another. When the units know their relative location, multiple complex tasks can be achieved like: path finding, payload transfer from one unit to another, autonomous recharging, and much more.

2.1 Project methodology

We will use the V-model to give structure to the design approach. The V-model uses multiple phases to make this project successful. The first three phases are:

- User requirement specification
- Functional specification
- Design specification

At the end of these phases the design stage has ended with all corresponding documents. For each stage that is mentioned above a test phase is present. This is to justify the specifications and decisions that have been made. The test plan can be found the qualification document. When all these stages have been carried out correctly a well tested product will be the result. In the first phase the wishes of the customer will be mapped. From these a research framework will be set up, including a research questions and multiple sub questions. These questions needs to be researched and written down. This will be done in a separate research document. From this document specifications will be made with well supported arguments explaining why. And test conditions for these specifications If everything is complete the next stage will begin: "the Functional specification". From the specification that have been set up the functional specifications will be made. The design is orthogonalised so that is easier to design the separate functions. At the end the separate functions can be added together to make one big flowchart. Again at the end of the stage new test conditions should be made. In the third stage: "the Design specifications" the previous specifications and functional designs will be implemented. All design specifications should be as detailed as possible. For example the dimensions of the components need to be written down. If this phase is completed the next step is the "system build". During the system build the functions of the product will be build separately so that every group member can continue to work. When all separate functions are build they will be tested. If all separate blocks work fine, they can be added to each other so that one product will remain. Thereafter the complete system will be tested. As noted before the test plan will be written down in the qualification document.

3 Project Definition

In this section the project will be further defined. The problem and context will be analysed. The problem will then be defined and goals for the project will be set.

3.1 Problem analysis

Researcher have always been inspired by nature. When they looked at "social" insects like ants they discovered "swarming"[1] . The behaviour of one ant on its own seems illogical, but together they solve problems of great importance for the entire colony. These ants make us of the so called "trail laying" and "trail following" principle. Every ants lays a trail of pheromones, when a few ants walk back and forth to a food source. The one walking the shortest route will lay a more concentrated trail. The other ants will get attracted by the strongest trail, this way a positive feedback loop is created, which will make every ant walk the shortest route if given enough time. This is one example where relatively simple units, can achieve complex goals like path finding because they work together in a swarm. This principle is called "swarm intelligence"[2].

Swarming can have alot of up sides compared to the "classical" approach. A few are: quicker solution time, lower unit complexity and a greater fault tolerance[2]. When for example one of the units breaks down, then will the other units still be able to complete the task. When this happens to one, more complex unit, this wont be the case. For these reasons its interesting to researching the applications of swarming.

3.2 Context analysis

The Delft university of technology started a program to research the applications of swarming. In this program multiple universities work together to make this research possible. The idea of this program is that each project group contributes a small bit to reach the end result, which will be a working swarm of robots. The technology developed should be modular, so it can be easily used on other platforms.
The programs focus lies on researching the applications of swarming on Mars. Its preferable that the technologies developed also work on Mars, but in some cases other technologies can be used to cut the cost. For example, for a simple proximity sensor on earth, an ultrasonic sensor would do just fine. But in the Mars atmosphere the ultrasonic waves get heavily dampened to the point where the sensor just wont work[3]. The cheapest sensor that would work on Mars would be a lidar[4]. While the average ultrasonic sensor costs around $2e$ the cheapest lidar costs atleast $100e$. In this project the aim is to build one unit for around $200e$, just one lidar sensor would be half of the robots budget. To proof the concept of swarming the robots don't need to be "Mars proof", so costs will be cut where possible.

3.3 Problem definition

Swarming intelligent systems are typically made up of simple agents(robots) interacting locally with one another and their environment. The group of individuals acting in such manner is referred to as a swarm[2]. For a group of robots to qualify as a swarm-robotics the following criteria have to be met:

- Autonomy - It is required that the individuals that make up the swarm-robotic system are autonomous robots. They are able to physically interact with the environment and affect it[2].
- Large number - A large number of units is required as well, so the cooperative behaviour (and swarm intelligence) may occur. The minimum number is hard to define and justify. The swarm-robotic system can be made of few homogeneous groups of robots consisted of large number of units. Highly heterogeneous robot groups tend to fall outside swarm robotics[2].
- Limited capabilities - The robots in a swarm should be relatively incapable or inefficient on their own with respect to the task at hand[2].
- Scalability and robustness - A swarm-robotic system needs to be scalable and robust. Adding the new units will improve the performance of the overall system and on the other hand, loosing some units will not cause the catastrophic failure[2].
- Distributed coordination - The robots in a swarm should only have local and limited sensing and communication abilities. The coordination between the robots is distributed. The use of a global channel for the coordination would influence the autonomy of the units[2].

These criteria are a good indication as to what makes a system swarm-robotic. But should not be used to determine whether a system is swarm-robotic or not. This is because some criteria are still somewhat vague[2].Looking at these criteria we chose to define the problem into two sub-problems: communication and (relative) localization. The criteria Large numbers, Scalability and robustness are important for the communication. Distributed coordination will be provided by the localization part. The communication will provide a network which will keep track of the number of units in the swarm, and does not rely on one node to function. This network is used by the localization part to send critical information needed to calculate the distance and angle. A swarm cant be depended on one unit or a "beacon" to determine the actual location. Because of this the localization will always be relative to other swarming modules.

3.4 Goal

As previously discussed the problem is now divided in two sub-problems, that together form the Swarming module. The goal is to create multiple functioning Swarming modules, so that they can be properly demonstrated. How many units are needed to properly demonstrate swarming, will later be defined.

3.4.1 Swarming module

Distributed coordination is one of the swarming criteria. To achieve this, some form of (relative) localization is needed. This should keep units from moving to close or to far from each other. And could also be used to accomplish certain goals like; mapping, assembly of structures or inspections [?]. Communication is needed to share information about the environment and every units position. Requirements here are that the communication should not be depended on one host and should be scalable. This is so communication is not cut off when one of the units breaks down. The scale-ability is important so that units can be added or removed from the swarm [?]. Because this project is part of a running program the modules made should be modular so they can be used on future projects. Summed up, the swarming module has the following characteristics.

- (Relative) Localization
- Communication
- Scalable
- Modular

4 Research-question

The main question of this research is as following: *"How can communication and relative localization be achieved in a swarm or robots?"*. To give an answer to this question there are multiple sub-questions to research first.

4.1 Sub-questions

The following questions need to be answered to come to a good conclusion to our research:

- "What is swarming?"
  - "What is the definition of swarming?"
  - "How many robots are needed to create a swarm?"
  - "How do the units communicate within the swarm?"
  - "How do robots in the swarm know their location?"
- "Swarming communication"
  - "What software protocol should be used?"
  - "What hardware protocol should be used?"
  - "What is the minimal required communication speed?"
  - "What hardware is needed to implement the communication?"
- "Communication between modules"
  - "What software protocol should be used?"
  - "What hardware protocol should be used?"
  - "What is the minimal required communication speed?"
  - "What hardware is needed to implement the communication?"

5 Research

This section will a summary of the findings during the research phase. For the full research done during this project we refer to the "Research Document". The research is split up in the following different subjects: Localization, Hardware communication protocol, Software communication protocol.

5.1 Acoustic Sensing

In this section the behaviour of acoustic signals will be researched. This was done in real life experiments using a microphone, speaker and scope to analyse the received signals. During experiments the following points where analysed:

- Sound has to spread in every direction (omnidirectional).
- The received acoustic signal has to be well defined within a minimal distance of 3 meters.
- What frequency shows the best results.

More details about the experiment and the set-up are shown in section 4 of the Research document. The first experiment had the speakers facing the microphone directly, this was clearly the best situation, higher frequencies showed higher amplitude and the signal was well defined. However the most ideal situation would be for one speaker to be omnidirectional instead of having to use multiple speakers.
Another test was done with the speaker and microphone both facing upwards. It was observed that higher frequency sound (around 5 kHz) becomes more directional. The best results were achieved around 3 - 4 kHz, this is the frequency range where the highest amplitude was achieved and the minimum distance of three meters was easily met.

5.1.1 Signal analysis

The received signal was analysed using a oscilloscope to gain a better understanding of how the signal should be processed and what the time deviation is. In figure 1 the received signal is displayed at a distance of 1 meter.

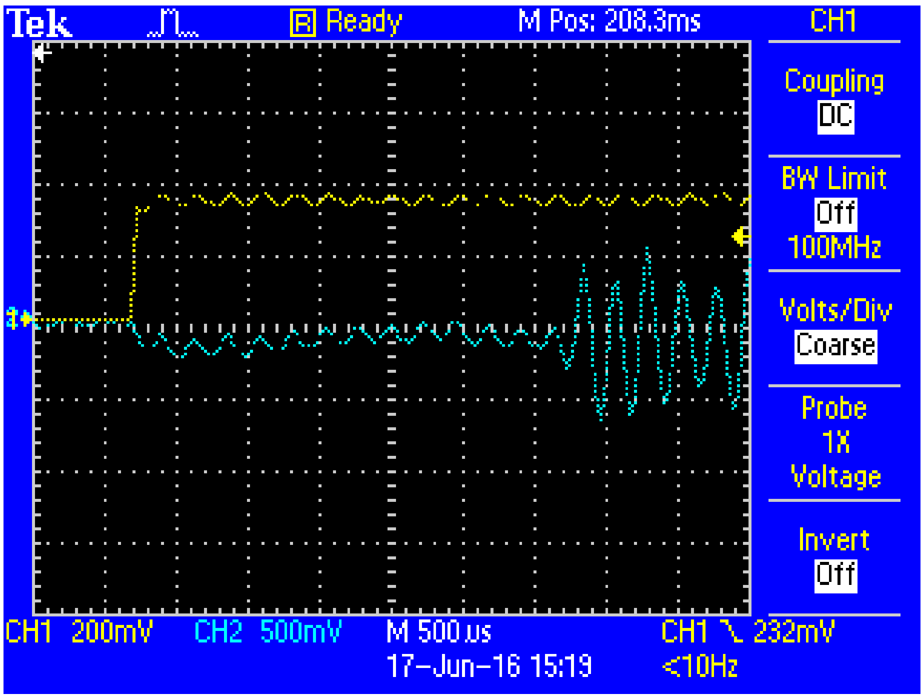


Figure 1: Oscilloscope measurement of the received acoustic signal at a distance of 1 meter.

The rising edge shows when the signal is send. The signal is then received a few moments later. Its observed that the acoustic signal arrives 3 ms later. Multiplying this with the speed of sound, a corresponding distance of roughly 1 meter was found. This supports the theory of distance measurement using acoustic signals. The received acoustic signal shows some distortion in amplitude. But the frequency is still well defined. This shows that approaches that process the received signal should be focussed on the frequency of the signal.

5.2 Localization

This section will cover the following subjects: Relative distance measurement, and Relative angle measurement. With these two variables the relative location of other units can be derived.

5.2.1 Relative distance

Before any calculations can be done to derive the angle of other units a distance must first be determined. The first choice that has to be made is what type of signal is used to do this. The most commonly used type of signal to measure distance are radio waves, which propagate with the speed of light. Hardware to measure these signals over short distances, would require clock-speeds of in the GHz, hence be expensive and unstable. Therefore the choice is made to use acoustic signals to measure the distance. This drastically lowers the requirements for the hardware that detects the signal.

Now that this is established there are still two ways to implement a distance measurement. These are: Time of flight (TOF)[5], and Received signal strength (RSSI)[6]. During experiments (found in section 4.3 of the Research document) it was found that the received amplitude of the acoustic signal was not proportional to the distance. It was also found that the signal was well defined over the specified distance of three meters. Because of this the signal can be detected and used for TOF.

A system is proposed that combines the two signals (radio and acoustic) in one system. Swarming modules each get their place in time to send their acoustic signal. At the moment the module starts sending the acoustic signal, it sends a message over the radio communication channel saying "I'm going to send my signal", the radio signal arrives close to instantly compared to the speed of the acoustic signal. At this point all the other modules start their timers, and stop them when the acoustic signal arrives a few moments later. The distance can now be calculated to multiply the time with the speed of sound.

5.3 Relative angle

Determining the relative orientation with respect to each other can be done in various ways. Some methods involved, have larger limitations than others. In general angular measurements are done using goniometric equations. It was found that calculating the angle with a single measuring point on every swarming module has one big problem[7]. Because geometric functions are used to calculate the angle, there will always be two solutions to the equation (see figure 2 and section 3.1.1 of the Research document). To solve this the units would have to move and recalculate again to get the right answer. This would limit the units in their movement and functionality and is non-desirable.

$$Because\ Xp = Xq,\ cos(-\alpha) = cos(\alpha)\ applies \tag{1}$$

$$Because\ Yq = -Yp,\ sin(-\alpha) = -sin(\alpha)\ applies \tag{2}$$

To solve this problem a system is proposed with three or more acoustic receivers with a predefined distance between them. Using the difference in time and the predefined distance, the angle can be calculated with only one solution. This method is shown in figure 3.

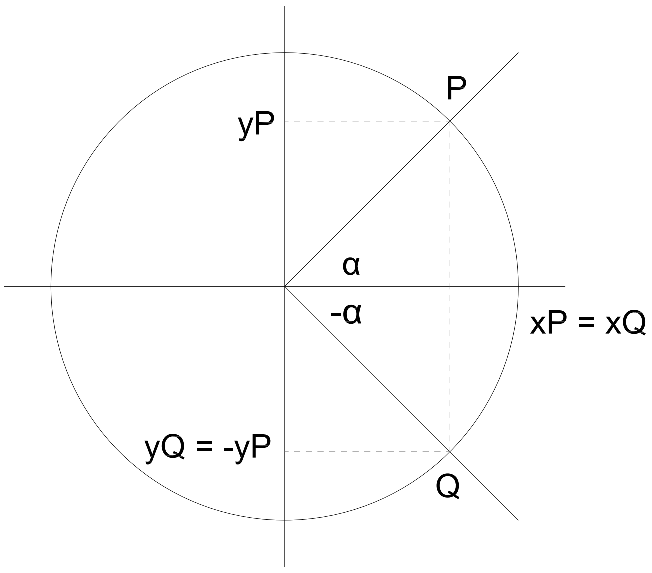


Figure 2: Unit circle where points P and Q are mirrored on the X-axes

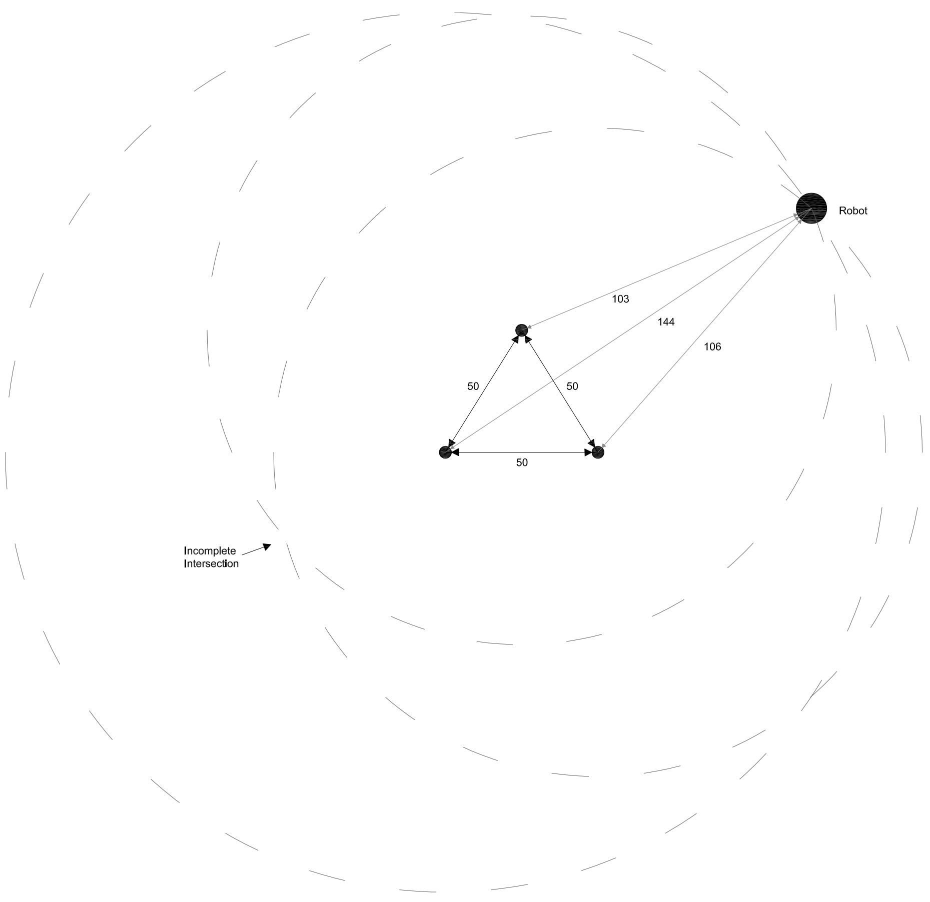


Figure 3: Angle determination using trigonometry

5.4 Signal processing

As talked about before the signal will be modulated and demodulated to introduce a reference point in the signal. This reference point is used to determine the exact time the signal has travelled. Also the incoming signal from the microphone will suffer from noise and interference. Because of this the demodulator should be noise resistant. Also its part of the specification of the swarming module that it should be a robust system. Therefore the use of high clock micro-controllers and complicated software are not preferred.

Project members have experience with frequency demodulation using an analog PLL, and were confident this could be easily tested relatively quick. The PLL can be configured for high noise insensitivity using phase comparator one of the 4046CD chip, this is stated in the PLL designer guide /citeplldisgn. In section 9.3 of the research document it was stated that in theory, the time delay created by the PLL should be constant. This is because the delay is created by the RC time of the filter, which is a constant. A constant delay it easily compensated and will not limit the system. The PLL approach will discussed in the next section.

5.4.1 Analog Phase locked loop

In this experiment a 4046CD PLL is used to demodulate an acoustic signal. Time measurements are done to analyse the behaviour of the PLL, and test its potential to be used for this project. A window comparator translates the VCO-in voltage into binary code. The comparator was used instead of the ADC of the microprocessor to reduce overhead created by the sampling of the ADC. The timer on the micro-controller is now triggered by a binary '1' created by the comparator see figure 4.

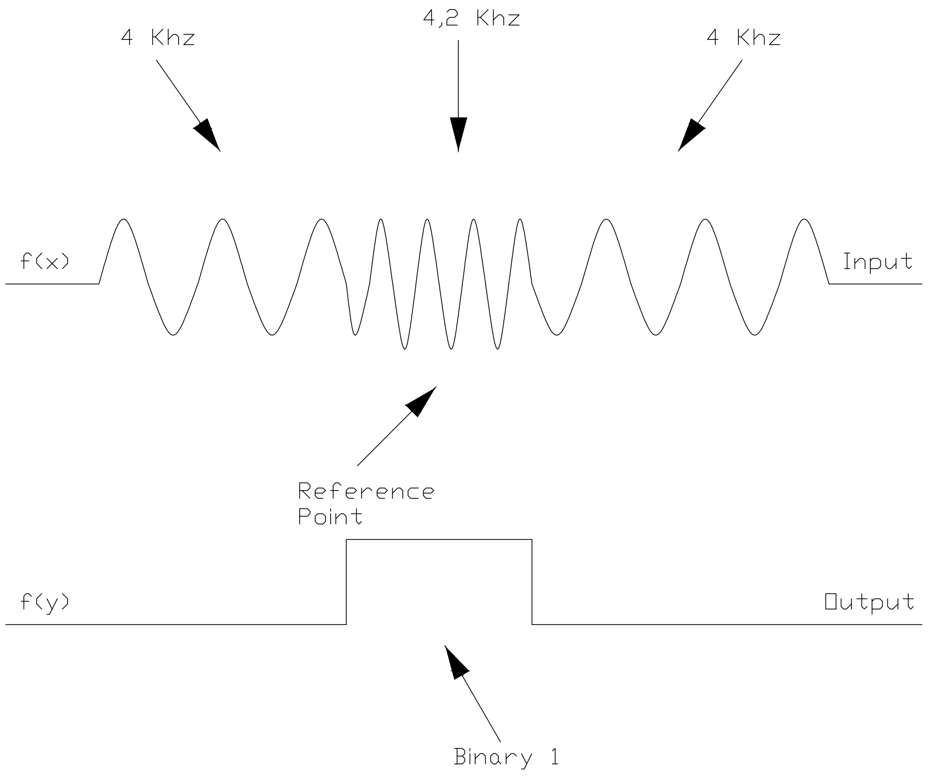


Figure 4: Example receiving signal PLL

To test the precision in time a test set-up was build. Two ATXmega128A4U were used, one for the demodulator side and one for the modulator side see figure 6. The PLL demodulator is build according to the schematic shown in figure 5.

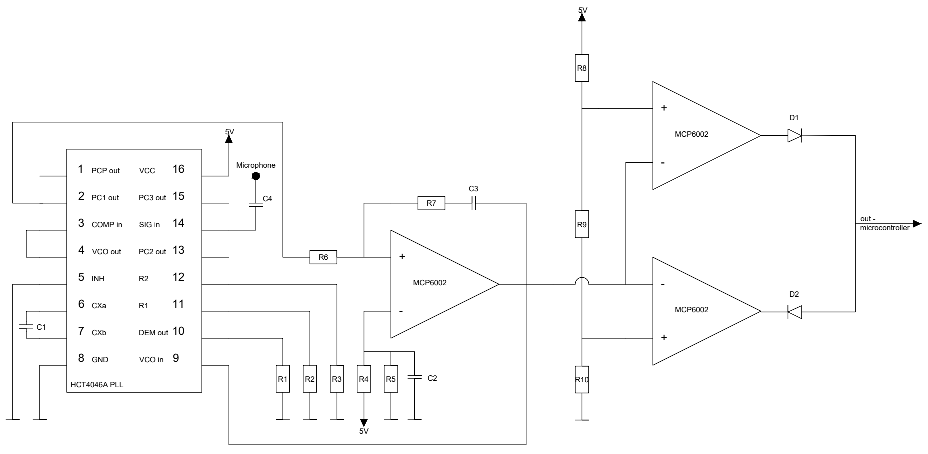


Figure 5: PLL demodulator schematic

During tests the PLL was able to lock on the acoustic signal from a maximum distance of 4 meters which is well within the specification of 3 meters. The PLL was also able to demodulate the FSK signal at this range. So now the reference point can be identified by the demodulator.

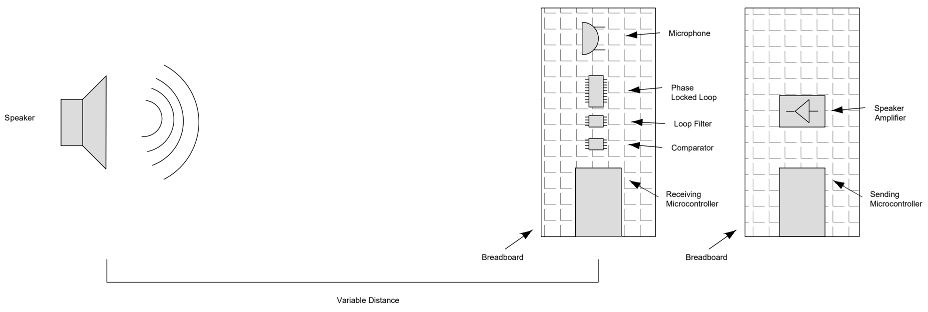


Figure 6: Test set-up demodulating with the PLL

Measure point	1	2	3	4	5	6	7	8	9	10
0,1m	3115769	3011495	3048980	3048133	3113623	3049715	3112041	3114898	3041495	3115769
1m	3112100	3012334	3102004	3159832	3068347	3123548	3110398	3119478	3062395	3119325
3m	3130596	3105236	3120056	3125069	3140686	3101498	3130295	3049821	3101956	3120549

Table 1: Measured times in clock pulses, pre scaler is set to 1

The modulator and the demodulator are connected with a wire. The modulator sends a bit when it starts to send the acoustic signal. This bit interrupts the demodulator which starts a timer. When the reference point in the acoustic signal is detected by the PLL the window comparator sends a binary '1' to the micro-controller which stops the timer. The timer values were then printed in a terminal and are documented in table 1 The measured times are printed in clock pulses with a pre-scaler of 1. The test results are plotted in figure 7.

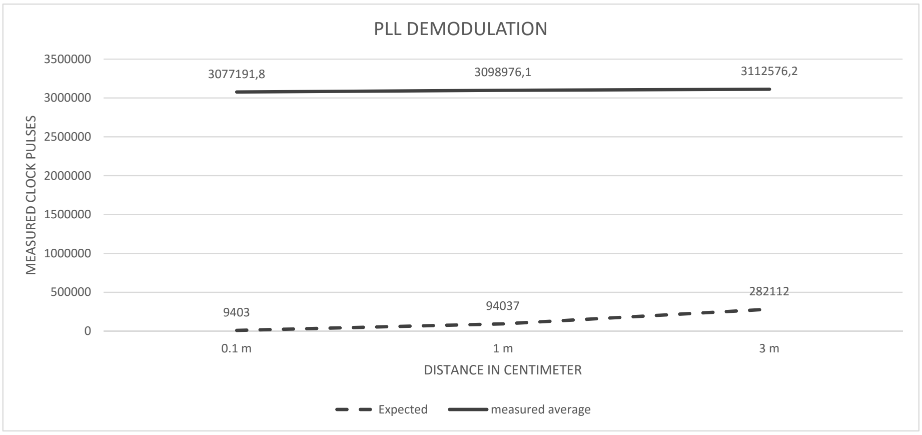


Figure 7: The test results of the PLL plotted against the expected results

5.4.2 Conclusion demodulating with the PLL

At close range the measured times seem the most stable. However some big deviations of around 500.000 clock pulses still exist. This translates in a deviations close to 5 meters, which is unacceptable. The digital/micro-controller side of the demodulator is tested by connecting a frequency generator to it and observing the time deviation. The time deviation was at its maximum 200 clock pulses, which is acceptable. It was observed that, when in lock the VCO-in voltage would be at a constant a stable level of 2,5V. When the frequency jump was made from 4kHz to 4,4kHz the VCO-in voltage would always rise to the same voltage of 2,78V. This means that the loop filter always has to integrate same voltage hop. The transfer function of the filter is given by equation 3[8].

$$H(p) = \frac{1 + p\tau_2}{\tau_1\tau_c(p^2 + p\frac{\tau_2}{\tau 1 \tau 2} + \frac{1}{\tau_1\tau_c})} \tag{3}$$

The transfer function shows that delays can only be caused by the time constants τ_1, τ_2, τ_3. This eliminates the PLL-loop as the source of the deviation in delays. The comparator is the only possible source left. It was determined that the deviation in time is created by the deviation in rise and fall time of the window comparator. In conclusion it's not the PLL itself but that causes the problems but the analog to digital conversion circuit. The deviation in time observed is far too large to function as an implementation for the demodulator. The choice for not implementing the PLL with the use of an ADC was because, the only reason for implementing the PLL in analog was to enable the use of a low performance platform. Also it was observed that almost every PLL is typically used for very high frequencies up to 100 MHz and even more. Our implementation however comes close to the frequency limitation of the PLL, which could cause instability. And therefore a new method to measure time delay in the sent signal must be developed.

5.4.3 Demodulating using the comparator of a Microprocessor

Its been established that the analog solution does not meet the criteria. The other option is to process the signal digitally, with a microprocessor. The available micro-controller hardware during this project is Xmega based. As stated in section 5.1.1 the signal processing should be frequency based. The AVR application notes describe a straight forward way to measuring frequency using the Xemga's comparator [?]. This method is illustrated in figure 8. This method is based on counting the rising edges using the comparator, thus counting the periods. The corresponding frequency can then be derived.

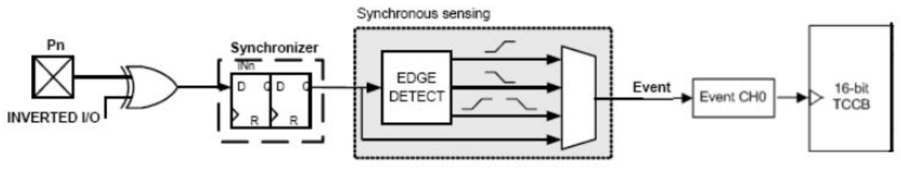


Figure 8: Block representation of the frequency measurement using a Xmega [?]

In the code below the measurement implementation using the comparator of the Xmega is shown. Where on line 6 to 14 an interrupt service routine is used to start the measurement and set all timers to 0. This happens when the sending unit starts sending the acoustic signal. The idea is to send a signal of 4 kHz and after a predefined amount of periods the signal stops (blank space). This blank space acts as the reference point within the signal see figure 9.

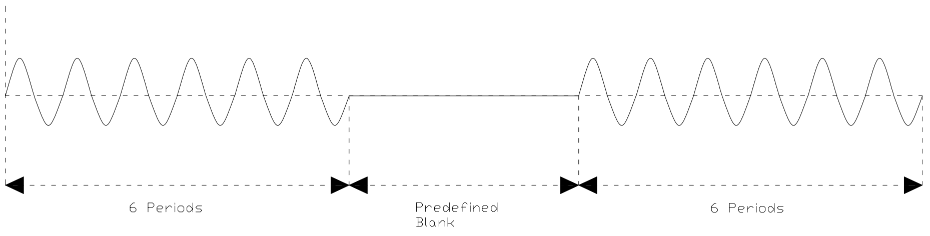


Figure 9: Example modulated signal

On line 22 to 38 the interrupt service routine of the comparator is used to detect every rising edge that the Xmega micro controller detects. In this interrupt the time between every edge that is detected is also measured, translating into a frequency.

On line 66 to 78 a continuous while loop has been implemented to transmit the measured frequency and the total time from the point the measurement has started till the point the measurement has stopped. The if statement checks if the frequency is of a value way below the 4 kHz and if the start-measurement flag has been set to 1, the measure value has to be equal or more than 1000 to make sure that atleast 1000 equal repeated periods have been detected.

```c
#include "main.h"

int measure2 = 0;

ISR(PORTB_INT0_vect)
{
  PORTC_OUTTGL = PIN2_bm;
  TCD0.CNT = 0;
  overflow = 0;
  start_measure = 1;
  period = 0;
  measure2 = 0;
  measure = 0;
}

ISR(TCD0_OVF_vect)
{
  overflow = overflow+1;
}

ISR(ACA_AC0_vect)
{
  if (ACA.STATUS & AC_AC0STATE_bm){
    count++;
    if (count == 0){
      TCC0.CNT = 0;
    }
    else if (count == 1){
      period = TCC0.CNT;
      count = 0;
      TCC0.CNT = 0;
      measure2++;
      if ( period < 1000 && period > 900){
        measure++;
      }
    }
  }
}

int main(void)
{
  SystemClock_init();               // initialize 32
   MHz clock
  uart_init();                      // initialize UART
  init_ac();
  stdout = &mystdout;

  //timer TBV het meten van de frequentie
  TCC0.CTRLA = TC_CLKSEL_DIV8_gc;         //
  prescaling 8
  TCC0.PER = 0XFFFF;                 // maximal value

  //timer TBV het meten van de overdrachtstijd
  TCD0.CNT = 0;                   // Timer/counter D0 value is 0
  TCD0.CTRLA = TC_CLKSEL_DIV1_gc;          // prescaling 1
  TCD0.PER = 0xffff;                 // maximal value
  TCD0.INTCTRLA = TC_OVFINTLVL_HI_gc;        // high level overflow interrupt

  //interrupt TBV het resetten van de timers bij startup
  PORTB.INT0MASK = PIN0_bm | PIN1_bm;
  PORTB.PIN0CTRL = PORT_ISC_RISING_gc;
  PORTB.INTCTRL = PORT_INT0LVL_HI_gc;

  PMIC.CTRL |= PMIC_LOLVLEN_bm | PMIC_HILVLEN_bm;
  sei();

  PORTC_DIRSET = PIN0_bm;
  while (1) {
    if (period > 5000 && measure >= 1000  && start_measure == 1 /*&& period > 850*/){
      start_measure = 0;
      time = TCD0.CNT+(overflow*0xffff);
      PORTC_OUTTGL = PIN0_bm;
      printf("Time: %lu\n measure: %d\nmeasure2: %d\n", time, measure, measure2);
      cli();
      measure = 0;
      measure2 = 0;
      sei();
    }
  }
}
```

Code 1: Implementation distance measurement using the Xmega comparator

Tests were done at different distances, with 10 measurements at every distance, for the test set-up that has been built see figure10.

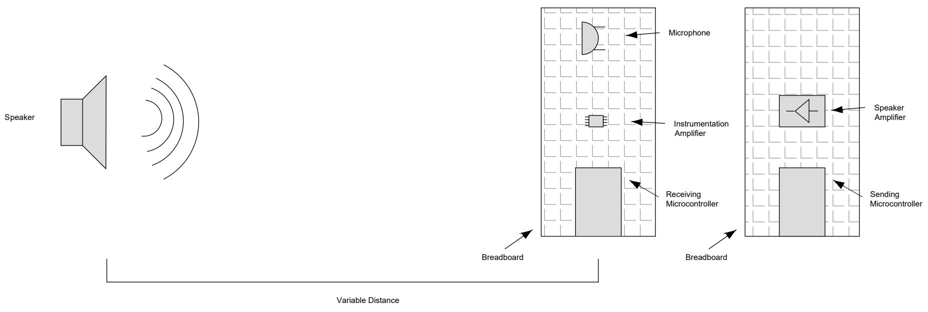


Figure 10: Comparator test set-up

The measurements are plotted in a graph with the expected values are shown in figure 11.

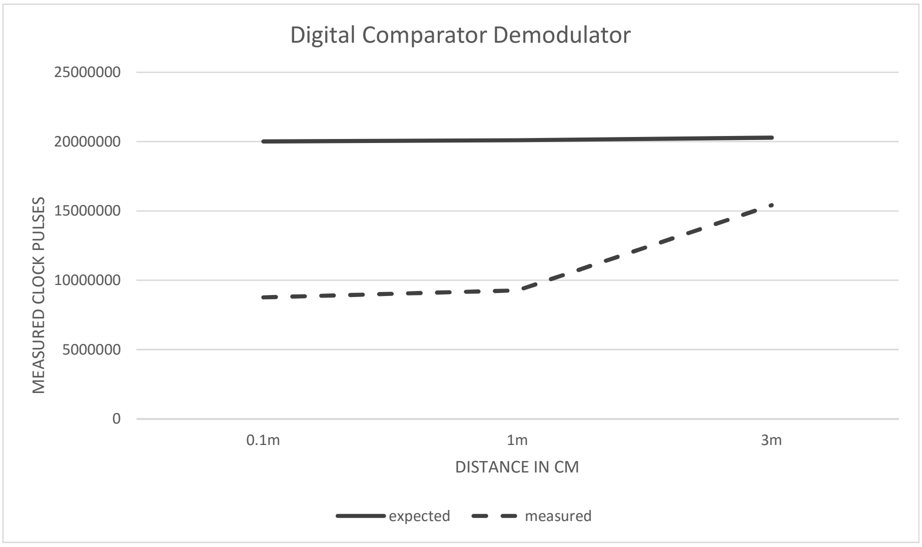


Figure 11: Measured results and expected results of comparator based measurements

5.4.4 Conclusion demodulating using the comparator

The time delay measurement between the sent signal and the received signal gave in some circumstances promising results, however when there is just a small amount environmental noise the comparator of the Xmega picks up these sound waves disrupting the entire measurement process. No stable measurements could be accomplished using this method due to noise signals, passing through the comparators threshold generating miscounts. Due to these miscounts the large variations in measured values are generated as seen in the graph shown in figure 11. Because of these large variations the measurements are unusable for our purpose and a new method has to be chosen. A method that is resistant to noise and is able to determine the time delay in the received signal. A more noise insensitive method called cross-correlation will be discussed in the next section.

5.4.5 Cross-correlation

In the previous section it was observed that digitally processing the signal showed potential. However the previous approach, suffered heavily from noise interference. Because the signal that is received is pre-defined and known a "light" version of a technique called cross correlation can be implemented. Cross correlation looks at the correlating points in the two signal arrays, the known signal and the received signal. Cross correlation has great noise insensitivity because, the noise in the received signal is uncorrelated to the signal that's already known [**?**].

$$R(\tau) = \int_{-\infty}^{+\infty} x(t)y(t+\tau)dt \tag{4}$$

Where:
x(t) and y(t) is a function of time;
τ is the time delay;
R is the cross correlation, which is a function of the time delay τ

Using correlation we can determine the distance as follows, consider two functions $\boldsymbol{f}$ and $\boldsymbol{g}$ which only differ from each other by an unknown shift along the x-axis. Using cross-correlation we can find how much $\boldsymbol{g}$ must be shifted along the x-axis to make it perfectly identical to $\boldsymbol{f}$. Essentially the formula slides the function $\boldsymbol{g}$ along the x-axis calculating the integral at each position. When the function are identical at each other (match), the value of ***(f*** $\times$ ***g)*** is the highest number. this is because when the peaks of the signal are aligned, they make a very large contribution to the integral which should result in the highest value.

Doing so also allows us to extract the time delay of the signal τ see figure 12, which in our case means the delay between sending the acoustic signal and receiving it.

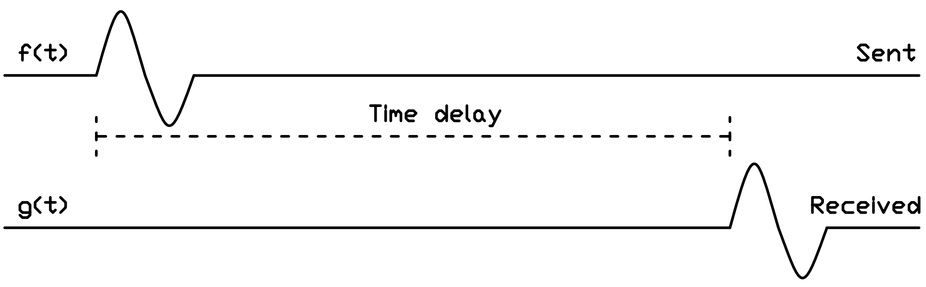


Figure 12: Time delay between $\boldsymbol{f}$ (sent) and $\boldsymbol{g}$ (received)

As described earlier if we shift the function $\boldsymbol{g}$ over the x-axis to find the point where the two signals correlate the most. Doing so the correlation process will produce several peaks. The peak where both signals correlate the most, should be the highest as shown in figure 13. While shifting the receiving function over the x-axis it is necessary to keep track of the amount of shifts required to reach the highest point. And how many periods shifts (time shifts) required to align both signals, doing so the time delay τ can be acquired. Multiplying this with the velocity propagation of the speed of sound should give us an distance.

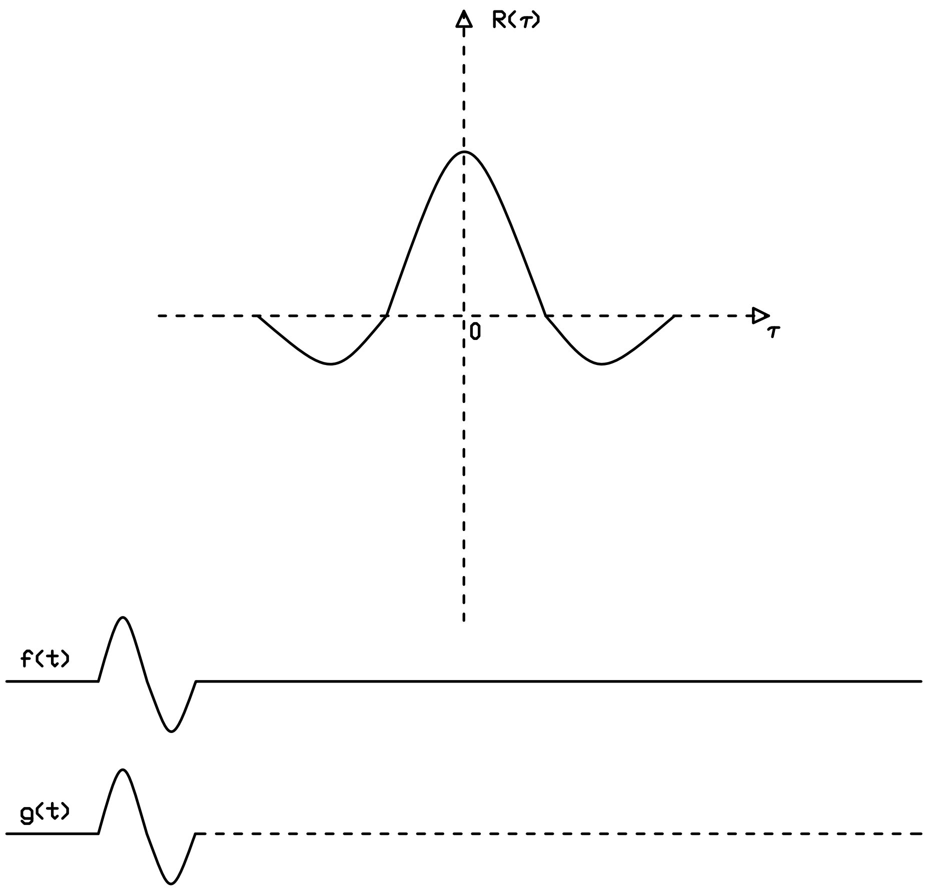


Figure 13: Auto correlation process of a cycle of a sine wave

5.4.6 Cross-correlation implementation

The implementation has been done in two different ways. One of them being implementing it according to the literature where a lot of floating point calculations has to be done. So we have analysed the original cross-correlation method and developed a light-weight simplified version.

This simplified has to however compromise on accuracy, but gains processing speed. Where the original cross-correlation takes up to 5 minutes in an ATXmega256A3U 32MHz Microcontroller the simplified version takes up to max 2 seconds. This increase in processing time is very crucial because when several robots move around in a swarm and it takes up to 5 minutes to generate a distance, the distance is already outdated long before it will ever be useful.

In the original cross-correlation algorithm and source code, the accuracy comes from denominator which ensures that the amplitude of the received signal has no influence on the total measurement overall. For the original cross-correlation see equation 5. The implementation of the software is shown in code

2.

$$R = \frac{\sum_{i}^{n}[(x(i) - mx) * (y(i-d) - my)]}{\sqrt{\sum_{i}^{n}(x(i) - mx)^2} * \sqrt{\sum_{i}^{n}(y(i-d) - my)^2}} \quad (5)$$

```
/************************************************
Cross Correlation
AutoCorrelation -- 2D Pattern Identification
Written by Paul Bourke
August 1996
************************************************/

int i,j
double mx,my,sx,sy,sxy,denom,r;

/* Calculate the mean of the two series x[], y[] */
mx = 0;
my = 0;
for (i=0;i<n;i++) {
   mx += x[i];
   my += y[i];
}
mx /= n;
my /= n;

/* Calculate the denominator */
sx = 0;
sy = 0;
for (i=0;i<n;i++) {
   sx += (x[i] - mx) * (x[i] - mx);
   sy += (y[i] - my) * (y[i] - my);
}
denom = sqrt(sx*sy);

/* Calculate the correlation series */
for (delay=-maxdelay;delay<maxdelay;delay++) {
   sxy = 0;
   for (i=0;i<n;i++) {
      j = i + delay;
      if (j < 0 || j >= n)
         continue;
      else
         sxy += (x[i] - mx) * (y[j] - my);
      /* Or should it be (?)
      if (j < 0 || j >= n)
         sxy += (x[i] - mx) * (-my);
```

```
            else
                sxy += (x[i] - mx) * (y[j] - my);
            */
        }
        r = sxy / denom;

        /* r is the correlation coefficient at "delay"
    */

    }
```

Code 2: Implementation Cross-correlation

5.4.7 Simplified cross-correlation implementation

For the simplified version of correlation where the processing time has been significantly reduced several parts of the formula has been removed. The new formula is listed below in equation 6. The algorithm based on this formula shifts the sampled signal by one step every time till the highest value of the summation has been acquired see figure 14. At the point where the summation gives the highest output the signals match the most. When this point has been reached the designed function implemented in c code returns the amount of shifts required in the sample array to reach this value.

$$R = \sum_{i}^{n}[x(i) * y(i - d)] \tag{6}$$

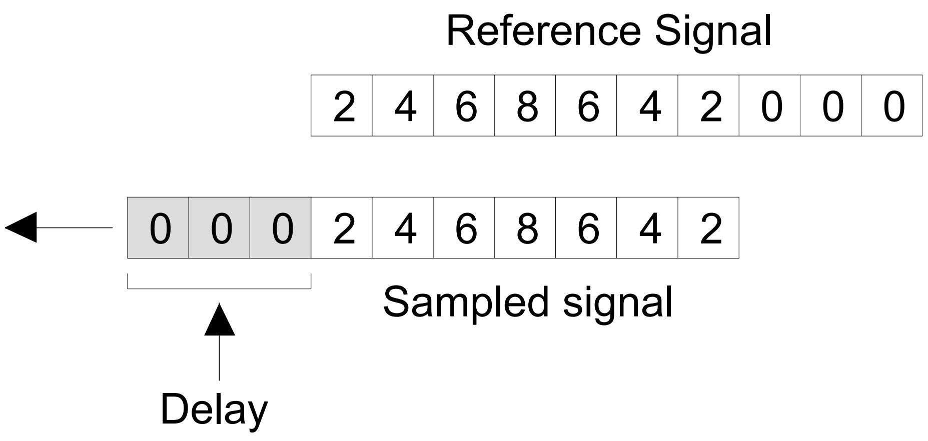


Figure 14: Algorithm simple correlation

The implementation of the source code in c is listed in below see code 3. This is a c file of the function that shifts the sampled array to align it with the reference signal. At the point where the sampled signal and the reference signal match the most the value of the time shift will be returned. And the distance can be calculated with a simple formula.

```c
/**********************************************
 * Simple Correlation.c
 * Created: 14-6-2016 11:01:46
 * Author: Zwermen 2
 **********************************************/

#include <avr/io.h>
#include <stdio.h>
#include "correlation.h"

uint16_t correlation (int16_t adc_arraylength, int16_t
    raw_arraylength, const uint16_t rawdata[],
   volatile int16_t Filtereddata[]){
  volatile int i = 0, k = 0, timeshift = 0;;
  volatile int32_t sum = 0, sum_old = 0, sum_1 = 0,
   sum_2 = 0;

  for(i = 0; i < adc_arraylength-1; i++){
    sum = 0;
    for(k = 0; k < raw_arraylength; k++){
      sum_1 = rawdata[k];

      if((k+i) > adc_arraylength){
        sum_2 = 0;
      }
      else{
        sum_2 = Filtereddata[k+i];
      }

      sum += (sum_1)*(sum_2);
    }

    if (sum >= sum_old){
      timeshift = i;
      sum_old = sum;
    }
  }
  if(i >= adc_arraylength-1){
    return timeshift;
    for(int j = 0; j >= adc_arraylength; j++){
      adc_arraylength[j] = 0;
    }
  }
  return 0;
}
```

Code 3: Implementation Simplified version of cross-correlation

5.4.8 Cross-correlation results

After implementing the algorithm mentioned above, several tests can be performed. With the analog to digital converter of the ATXmega256A3u the received signal from the microphone will be sampled with a sample frequency of 5µ seconds. This means every shift that is required in the sampled array equals a delay of 5µ seconds. With this in mind, the results we expect to see are listed in the table shown in figure 15. Where the first row shows the distance, the second row shows the amount of travel time required for the distance shown in row one and the last row shows the expected shifts required. We do however expect to see some deviation in the measured results relative to the expected results due to overhead time in the processing process.

Expected Results		
Results:	Time of flight (s)	Time shifts
10 cm	0,000301568	60,31
20 cm	0,000603136	120,63
30 cm	0,000904704	180,94
40 cm	0,001206273	241,25
50 cm	0,001507841	301,57
60 cm	0,001809409	361,88
70 cm	0,002110977	422,20
80 cm	0,002412545	482,51
90 cm	0,002714113	542,82
100 cm	0,003015682	603,14

Figure 15: Expected results from the correlation algorithm

For the several tests that has been done the test set-up shown in figure 16 has been used. Where two microcontrollers, one ATXmega128A4U is used for sending the acoustic signal through the speaker and a ATXmega256A3U was used for receiving the signal with a microphone.

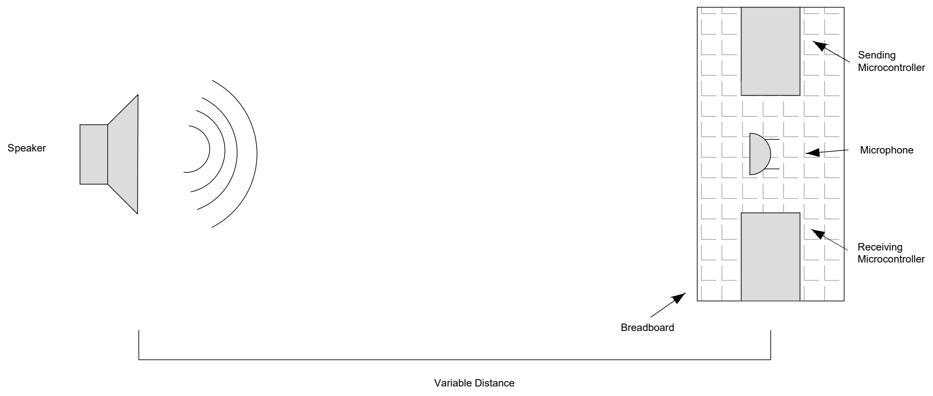


Figure 16: Test set-up for cross-correlation

The test set-up is connected according to the following schematic see figure 17. Where two different microcontrollers the Xmega128A4U and the Xmega256A3U are utilized. The various connections to the microphone and speaker are shown in the schematic. This enables others to reassemble the test set-up.

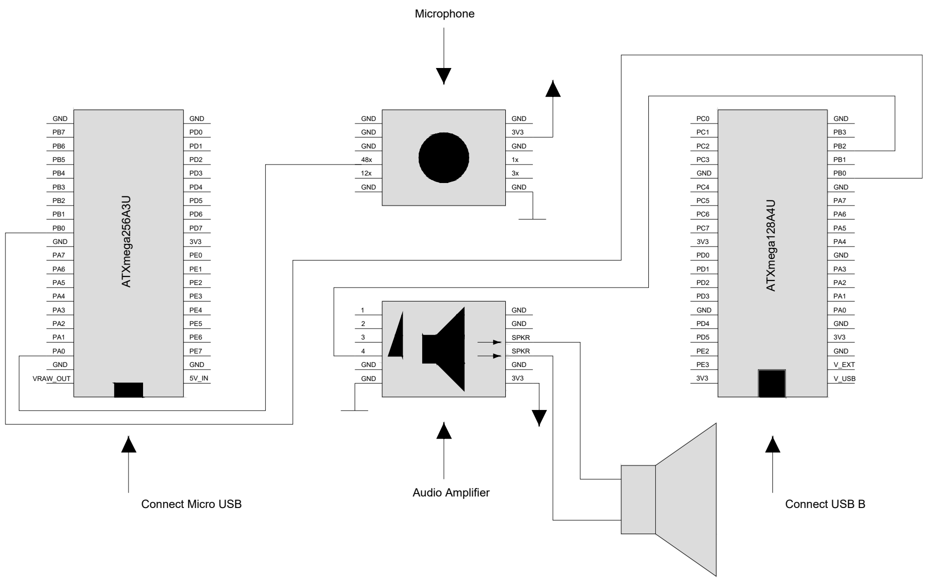


Figure 17: Test set-up for cross-correlation

In the table shown in figure 18 the measured results are displayed using the simplified version of cross-correlation. Per distance there are a total of seven measurements done to check for consistency in the values presented. As seen in the results at 80, 90 & 100 cm there are several faulty values measured. However most of the values measured are correct with a certain overhead time and correction.

Measured Results										
Results:		1	2	3	4	5	6	7	Measured Length	
10	cm	79	80	81	81	81	81	81	0,13	m
20	cm	133	133	133	134	133	133	133	0,22	m
30	cm	183	183	184	183	184	183	183	0,30	m
40	cm	236	236	235	236	236	235	236	0,39	m
50	cm	290	290	289	289	289	289	290	0,48	m
60	cm	341	340	341	340	341	340	341	0,57	m
70	cm	394	393	393	393	394	394	393	0,65	m
80	cm	525	525	526	525	568	525	525	0,87	m
90	cm	614	615	614	1566	615	614	615	1,02	m
100	cm	658	657	657	658	1035	659	824	1,09	m

Figure 18: Measured results from the correlation algorithm

In the table shown below in figure 19 the deviation between the expected results and the measured results are shown. As seen at close range the deviation is quite large however the further we get the more precise the measurement seems to be.

Deviation Results	
Results:	
10 cm	74,46%
20 cm	90,70%
30 cm	98,87%
40 cm	102,23%
50 cm	103,99%
60 cm	106,12%
70 cm	107,43%
80 cm	108,81%
90 cm	88,26%
100 cm	91,66%

Figure 19: Deviation between measured and expected results

In the figure 20 a graph is shown with the expected values (actual distance) and the measured distance.

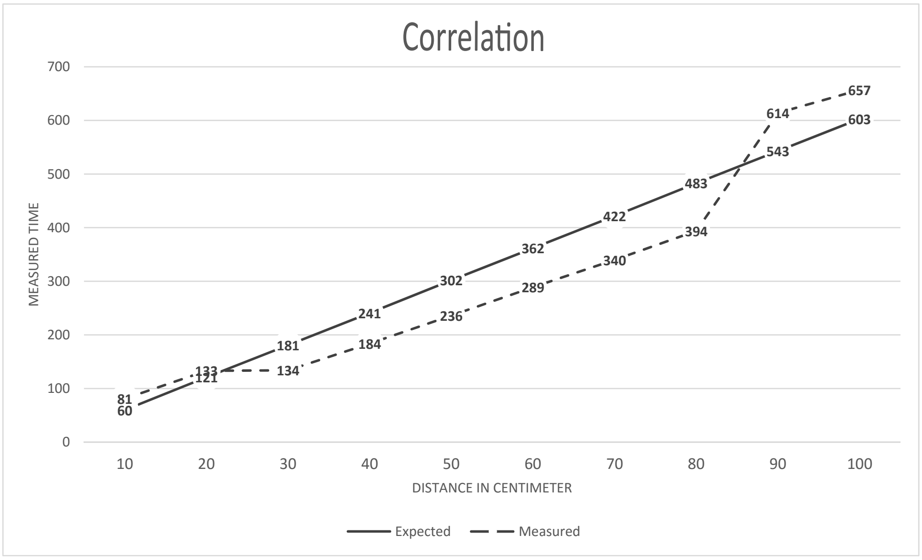


Figure 20: Graph of expected values versus measured values

However the faulty values measured still needs to be resolved because the values are in some cases very high resulting in large distances. An method to resolve this is to send multiple times and measure this signal multiple times. Doing so the histogram could be created which should display the most common values and the least common values. Choosing the most common value should increase overall stability and reduce faulty measurements.

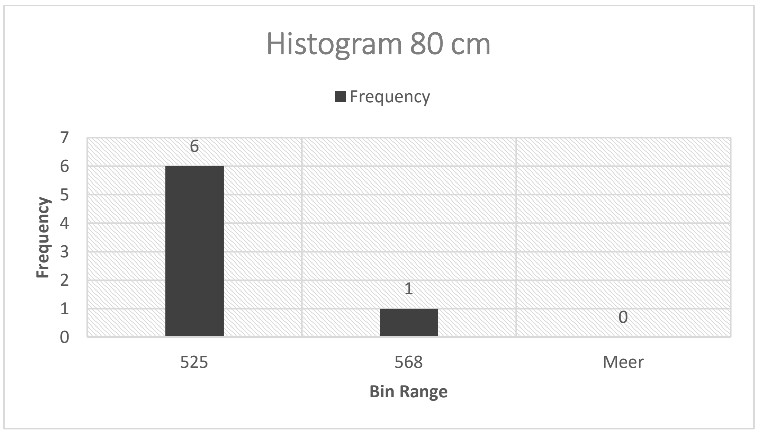


Figure 21: Histogram 80 cm measurement

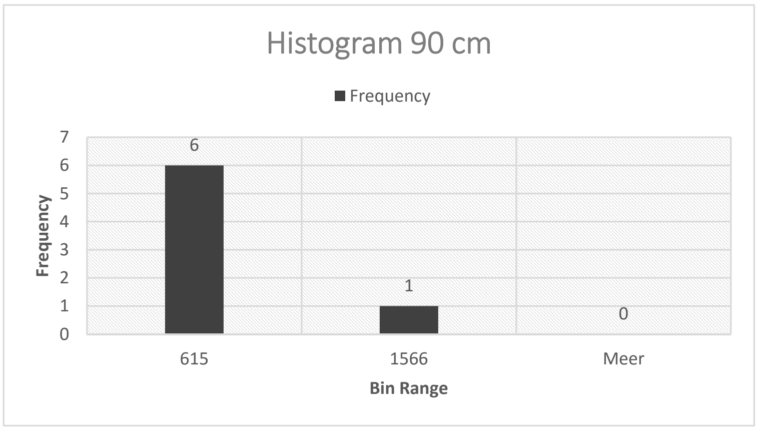


Figure 22: Histogram 90 cm measurement

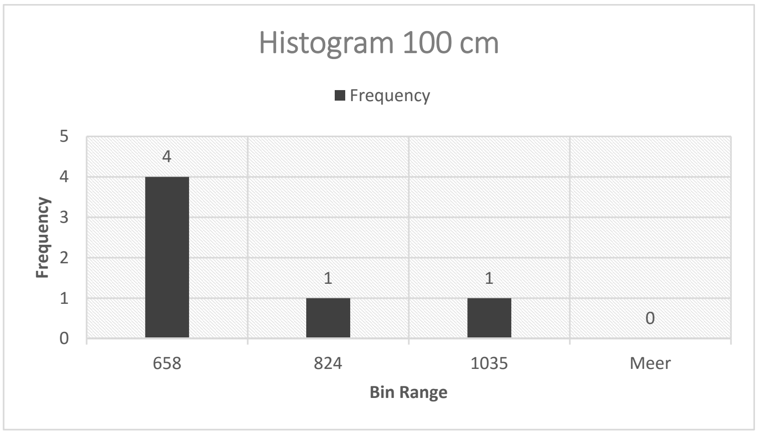


Figure 23: Histogram 100 cm measurement

In the figures 21, 22 & 23 the histograms of the measurements with some faulty values are displayed. Implementing a method to send multiple signals and also receiving each signal independently, with the use of some filter function that chooses the most common value the output should be stable and non fluctuating. If this has been accomplished a method can be implemented to increase overall accuracy with the use of some sort of analog or digital filter that filters out other frequencies.

5.4.9 Filtering cross-correlation

As mentioned and showed in the results above, there is a chance that a measurement can be faulty. This could be resolved by implementing some sort of histogram filter. The idea is to receive multiple measurements and determine the most common time delay value. This is done by sending out 10 short acoustic sound signals.

At the receiving end the software determines the time delay for each acoustic signal and puts this in an array. To filter out the highest values out of the array (which are faulty), the array must first be sorted from highest to lowest value. Afterwards the two top values (highest) and one bottom value (lowest) could be removed and the rest is averaged, this algorithm is shown figure 24.

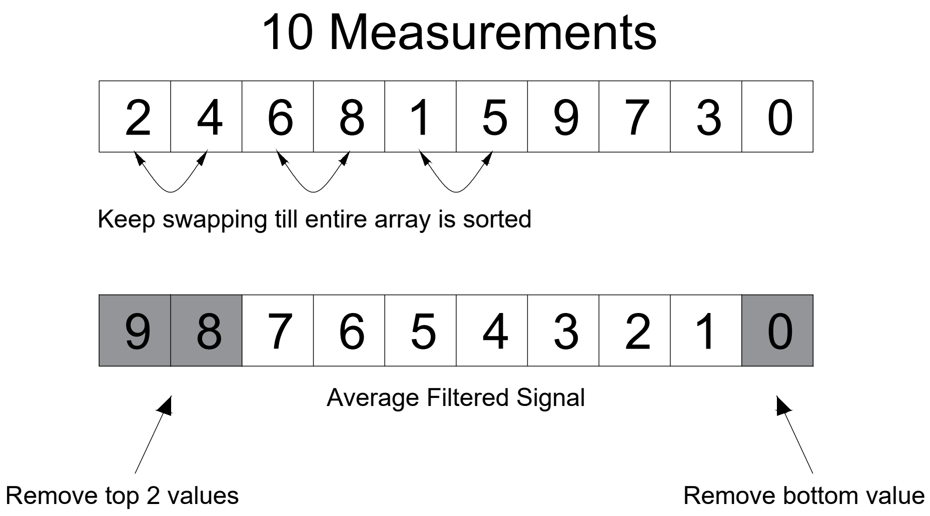


Figure 24: Histogram filter algorithm

This algoritm is implemented in the code 4 shown below. Where the array is sorted using the function called "Quicksort" and where the function called "Filter" removes the top two values and the bottom value and averages the leftover values in the array.

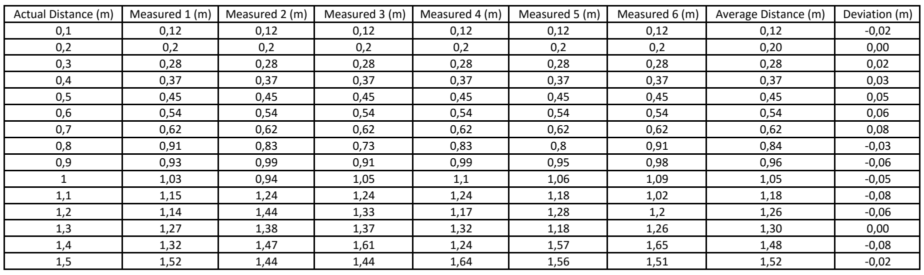
```c
/***************************************************
Histogram Filter Function
Cross Correlation
For distance measurements
Created: 20-06-2016
Author: Zwermen 2
***************************************************/

#include <avr/io.h>
#include <avr/pgmspace.h>
#include <stddef.h>
#include <stdio.h>

#include "quicksort.h"


void swap(volatile int16_t a[], uint16_t i, uint16_t j
   ){
  uint16_t tmp;

  tmp  = a[i];
  a[i] = a[j];
  a[j] = tmp;
}
void quickSort(volatile int16_t a[], int16_t left,
   int16_t right) {
  int16_t i;
  int16_t j;
  int16_t v;

  if(right > left) {
    v = a[right];
    i = left -1;
    j = right;
    while(1){
      while (a[++i] < v);
      while (a[--j] > v);
      if ( i >= j)
      {
        break;
      }
      swap(a,i,j);
    }
    swap(a,i,right);
    quickSort(a,left,i-1);
    quickSort(a, i+1, right);
  }
}

```

```c
uint16_t filter(volatile int16_t nums[], uint16_t size
    ) {
  uint16_t k, samples = 0;
  uint32_t sum = 0;

  for(k = (size / 10); k < (size - (size / 5)); k++) {
    sum += nums[k];
    samples++;
  }

  return (uint16_t)(sum / (uint32_t)samples);
}
```

Code 4: Implementation of histogram filter

5.4.10 Cross-correlation results with filtering

After implementing the filter function a new set of measurements can be taken to see how much improvement has gained from implementing such a filter. Our expectation is to atleast observe some sort of improvement towards accuracy. And also gain some distance in contrast to the previous results where only one pulse has been sent. In the table shown in figure 25.



Actual Distance (m)	Measured 1 (m)	Measured 2 (m)	Measured 3 (m)	Measured 4 (m)	Measured 5 (m)	Measured 6 (m)	Average Distance (m)	Deviation (m)
0,1	0,12	0,12	0,12	0,12	0,12	0,12	0,12	-0,02
0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,20	0,00
0,3	0,28	0,28	0,28	0,28	0,28	0,28	0,28	0,02
0,4	0,37	0,37	0,37	0,37	0,37	0,37	0,37	0,03
0,5	0,45	0,45	0,45	0,45	0,45	0,45	0,45	0,05
0,6	0,54	0,54	0,54	0,54	0,54	0,54	0,54	0,06
0,7	0,62	0,62	0,62	0,62	0,62	0,62	0,62	0,08
0,8	0,91	0,83	0,73	0,83	0,8	0,91	0,84	-0,03
0,9	0,93	0,99	0,91	0,99	0,95	0,98	0,96	-0,06
1	1,03	0,94	1,05	1,1	1,06	1,09	1,05	-0,05
1,1	1,15	1,24	1,24	1,24	1,18	1,02	1,18	-0,08
1,2	1,14	1,44	1,33	1,17	1,28	1,2	1,26	-0,06
1,3	1,27	1,38	1,37	1,32	1,18	1,26	1,30	0,00
1,4	1,32	1,47	1,61	1,24	1,57	1,65	1,48	-0,08
1,5	1,52	1,44	1,44	1,64	1,56	1,51	1,52	-0,02

Figure 25: Results after filtering

As observed from the measured results shown in figure 25, the maximum observed deviation at this moment is +/- 0,08 meters. Whereas before the maximum observed deviation was =/- 0,12 meters, an improvement of 0,04 meters. Also an increase of 50 centimetres in gained distance has been observed whereas before only accurate measurements could be done up to 1 meter. The final results are displayed in a graph shown in figure 26.

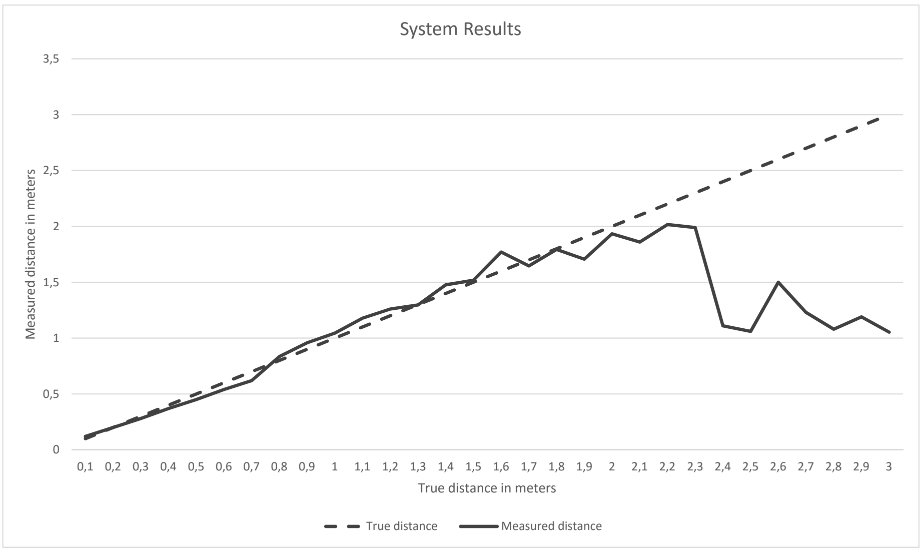


Figure 26: Graph of results after filtering

5.4.11 Conclusion cross-correlation

In the graph shown in figure 26 the expected values are set versus the measured values with the implementation of a ”histogram filter” that chooses the most common value of the series of samples.

As also seen in the graph is that the measured results do not vary as much as they did in the graph displayed in figure 20, where the results did vary quite a bit relative to the expected values.

However there is still a maximum deviation of +/- 0,08 meters observed which must be reduced to enable a localization algorithm where three microphones are used which are spread 10 centimetres apart. To overcome this a bandpass filter of some sort might increase accuracy, however at this point this is highly speculative and requires further research in this specific field. Something that could also contribute to accuracy is implementing the full cross-correlation algorithm developed by Paul Bourke instead of the lightweight cross-correlation algorithm developed and implemented by us. This implementation however needs some tweaks to dramatically increase the processing time which was as mentioned earlier up to 5 minutes with a ATXmega256A3U 32MHz microcontroller. Another approach could be to use a more powerful platform like for example a ARM Cortex processor.

Even though the results are not within the specifications (+/- 9%), cross-correlation seems the be the most promising method to distinguish signals and determine the time delay. The results gathered with the implementation of a simplified version of the correlation algorithm seems to be very promising, therefore we think that an implementation of the full cross-correlation algorithm will result in stable and accurate values with some overhead time due to processing, which can be compensated for.

5.5 Swarm communication network

This chapter is a summarizing of chapter 'Swarm communication' in the research document of this project. The is a global research of (wireless) swarm communication networks. This research was done to determine the needed protocol(s), algorithms and the use of hardware for creating swarm networks. This chapter cites the conclusions and recommendations from the research document. Some of the important references are: [9] [10] [11] [12].

5.5.1 Network criteria

There are several requirements to realise a robot swarm. Chapter 'Swarm communication' in the research document and some items in chapter 'Swarming' in this document name these requirements.

Each member of the population needs to communicate with the rest of the swarm. A communication network is needed to make the members of the population interact with each other.

The requirements relevant for swarm communication:

- Scalability of the swarm - It is important that the population size of the swarm is dynamic, because of the scalability of the swarm.
- Distributed communication - The use of a global channel for the coordination would influence the autonomy of the units.[2]
- Wireless communication - To give the members full freedom of movement it is useful for the communication to be wireless.
- Autonomy - A swarm doesn't have a master. Each member needs to be capable of maintaining the network. These networks also must be able to reroute around nodes that have been lost.

Each item has additional sub criteria, this and additional information can be found in the research document.

5.5.2 Recommendation and conclusion

The results of the sub-study in the research document have resulted in recommendations for the implementation of the swarming module. An enumeration of the most important recommendations is given below:

- From all of the available network topologies, a partially connected mesh network satisfies the desired network requirements.
- With the use of routing protocols the network can be rerouted around nodes that have been lost. The most suitable protocol for re-routing in this situation is greedy routing or simular.
- Wireless communication is not only useful, it must be a requirement. Due to the mobility of the swarm members, wired communication is feasible.
- As stated in the research, UWB, Wi-Fi and ZigBee protocols are all suitable to create a wireless mesh swarm network. Six potential protocols have been investigated and eventually four of them are compared in the conclusion. The comparison is based on transmitting range, scalability, data coding efficiency, protocol complexity, data distribution per square

meter and power consumption. The results from the comparison of the protocols don't differ much from each other, only Bluetooth stands out.

The other additional recommendations and conclusions can be found in the research document.

6 Specifications

Before the features of the Swarming module are specified, swarming itself should be specified further. This discussed in section 1 of the research document. By creating a set of realistic scenarios. From this specifications like: minimal distance, maximal distance, maximum number of units, and precision are derived. These specifications are summarised in Table 10 and Table 3.

	Close range (0 - 3 m)	**Long range (3 - 25 m)**
Dead zone distance	0,015 meters	2 meters
Deviation (distance)	+/- 9%	+/- 1 meter
Angle Resolution	45%	-
Update frequency	40 Hz	1 Hz

Table 2: Swarming localization specifications

	minimum	**maximum**	**maximum (close range)**	**Preferred (demonstration)**
Number of Units	3	∞	130	6 to 12

Table 3: Specifications number of units

The choice is made to divide the the specifications for localization into two categories: Close range, and long range. This choice was made because at close range more precision precision is needed to prevent the robots from crashing into each other. While at long range the localization will be used not to loose the swarm. The maximum number at close range is derived from the theoretical number of units that can physically fit in a 3 meter radius. And the preferred number of units for demonstration purpose is used to set a goal for the project. For further clarification how these specifications are determined see chapter one of the research document.

6.1 Wireless communication

In Table 11 a more detailed version of the specifications are written down. From these specifications the module will be implemented as is shown in the next section.

Table 4: Wireless communication module

Module	Wireless communication module
Input	5 VDC Swarm data RF signal
Outputs	RF signal Swarm data
Function	Processes information to and from other swarm modules. Transmission speed is not defined yet. The transmission range must be around 100 meters or more.

6.2 Central operating unit

The central operating unit processes data from the sensors and swarm network. This unit uses the external wireless swarm communication, the local communication bus and the debug interface as an communication channel. The local communication bus uses a TWI (Two Wire Interface) protocol and is used for the communication with the robots peripherals i.e. actuators. The debug interface of the central operating unit uses an UART protocol to communicate with external systems for debug purposes.

Table 5: Central operating unit

Module	Central operating unit
Input	3,3 VDC Data streams (TWI, UART)
Outputs	Data streams (TWI, UART)
Function	Calculates the relative position to other swarming modules. Distributes local and external data streams. Analyses and uses sensor information.

7 Functional specification

This chapter describes the functional design of the swarming module. An overview of the swarming module is given in figure 27. The module is designed in such a way that it is modular. This module can be placed on an object (i.e. a robot) and can be used by the object to send and receive relevant information about other in-range swarming modules.

This chapter discusses the following subjects:

- Swarming module
- Power supply and safety
- Central operating unit
- Wireless communication module
- Relative orientation sensor
- Orientation sensor
- Localisation sensor

7.1 Swarming module

The overall design of the swarming module is shown in figure 27. Each block of the overall design is described in the following figures. The central operating unit is at the hearth off the swarming module, and takes care of the localization algorithm.

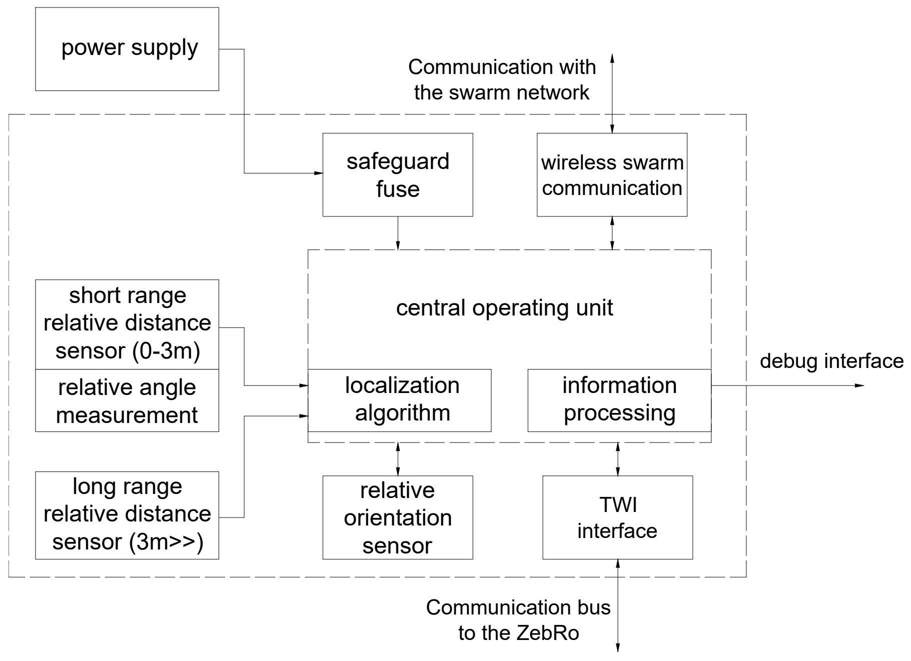

power supply
Communication with the swarm network
safeguard fuse
wireless swarm communication
short range relative distance sensor (0-3m)
relative angle measurement
central operating unit
localization algorithm
information processing
debug interface
long range relative distance sensor (3m>>)
relative orientation sensor
TWI interface
Communication bus to the ZebRo

Figure 27: Overview of the swarm module design

7.2 Short range localisation sensor

The principle of the short range localisation sensor is displayed in figure 29 and 28. As mentioned earlier the short relative distance measurement will be done acoustically. With acoustic sensing it is required to let the speaker sway to the right oscillation frequency, reaching this oscillation frequency takes time since the speaker won't oscillate at the right frequency instantly. This complicates the measurement a little bit, since just detecting the to be measured signal won't always translate into an exact distance due to the time the speakers take to sway into oscillation.

A solution to this problem is to create a fixed reference point within the sent signal. Creating this reference point after a fixed time enables the speaker to sway into oscillation making sure that no faulty measurements occur do to this "swaying time". When the reference point has been detected by the receiver an accurate distance can be determined by subtracting the fixed "sway time".

Without the implementation of this fixed reference point it is possible for a receiver to miss a few periods of the signal, due to the speaker still swaying into oscillation and the time of swaying being unknown. For example take an acoustic signal (Speed of sound 340.29 m/s) with a frequency of 4000Hz, one period of this signal is 0.00025s. Missing one period of this signal translates into a distance error of 8.5cm. Since the sway time of a speaker can change over time due to mechanical stresses a fixed time stamp seems the most suitable solution for the long term. The best way to create this reference point it still to be determined. This will be done by either frequency shift keying or phase shift modulation. Some real like testing will have to determine the best method.

7.2.1 Modulation

The modulation method is going to be determined by how well the reference point can be recognized by the demodulator. In the research phase, test were done with an speaker and a microphone. This showed that frequency and phase were well defined over distance but amplitude was not. Because of this the available methods for modulation of the acoustic signal are frequency modulation or phase modulation. The most important feature of the modulator must be consistency, meaning that the reference point should be send at exactly the same point in the signal every time. This is so that no deviation will occur when compensating for the missed periods before the reference point. In theory this can be done by both of the methods.

See figure 28 for a schematic representation of the modulator.

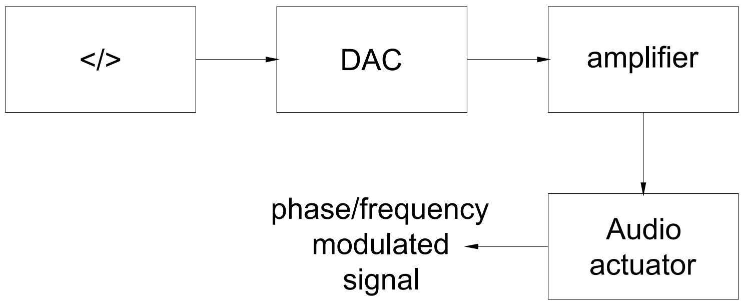


Figure 28: Modulator

7.2.2 Demodulation

A demodulator will be needed to retrieve the digital signal send from one swarm-module to another. One swarming module will send an acoustic signal to the other modules, this signal will then be picked up by an sensor and will need to be processed properly before it can be used to determine the time it took the acoustic signal to travel from one swarming-module to the other. The signal retrieved by the acoustic sensor will also pick up a lot of noise and other sounds from the environment. Also, when further away the signal might be low in amplitude. To increase the amplitude and lower the noise, the signal will be amplified and filtered. These modifications to the signal will improve the signal to noise ratio drastically but a lot of noise might still remain. Therefore the demodulator itself must be insensitive to noise. The main purpose is the demodulator is to identify the reference point. And more importantly, determine the point in time the reference point is spotted. The time determined will be used to calculate the distance. Hence, the precision in which the demodulator determines the reference point in time effects the precision of the distance measurement. Only one swarming module will be sending his acoustic signal at any moment. When there are 10 units in its specified range and a preferred update frequency of 40Hz (see section 1). This gives every swarming-module only a small window in time to send their signal and for the other to receive it. When a frequency of 4kHz is chosen for the signal, every swarming-module will have 10 periods to send their signal. There for the demodulator must be able to lock onto the signal and determine the reference point within a few periods of the acoustic signal. See figure 29 for a schematic representation of the demodulator.

Table 6: Demodulation specifications

Module	Demodulator
Input	4 kHz signal
Outputs	Time delay
Function	Demodulation, Time delay determination
Features	Noise insensitive, precision (time), Quick demodulation

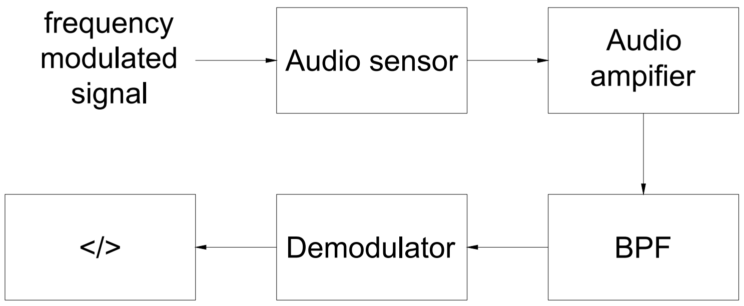


Figure 29: Demodulator

7.3 Wireless communication

The specifications of the wireless communication are discussed in our framework, but will be shortly summarized.

Table 7: Wireless communication module

Module	Wireless communication module
Input	Communication with the system Communication with the other robots in the swarm
Output	Communication with the system Communication with the other robots in the swarm
Functions	establishes the communication between the modules swarm Obtains information regarding the distance between the different robots

To communicate between different units a communication network should be set up with a common protocol . In addition, the protocol must be able to provide additional information such as localization. See Table 7.

7.3.1 Central control unit

The central control unit handles all communication between the various modules that are present on the robot. See Table 8

Table 8: Central control unit

Module	Central control unit
Input	All information from the system
Output	All control of the modules in the robot
Functions	All the communication comes together and controls the inputs and outputs

8 Design

Now that specifications of the functional blocks are defined, a implementation for these blocks will be found, using the information from the research phase. Just like section Functional specifications every block will be discussed independently. How all of the functional blocks fit together will be discussed in the section for the central processing unit.

8.1 Short range localization sensor

The short range localization consist of multiple smaller parts: Acoustic actuator/sensor, modulation/demodulation, distance measurement, and angle algorithm. The code shown in this section is written for a Xmega256a4u.

8.1.1 Acoustic actuator sensor

As stated earlier a short range localization will be implemented using acoustic wave signals. The sensor will be implemented using microphones and an actuator will be implemented using a speaker.

For Trigonometry angle determination the module must atleast poses three microphones spread in a triangle where all sides have the same distance as shown in figure 30.

For the distance between the different microphones a length of 10 cm is chosen because with the velocity propagation of sound being 343.2 m/s (at 20°C Celsius) and the underlying distance between the microphones being 10 cm using an 32 MHz micro-controller should grant enough overhead time to process the three different times being detected by the microphones.

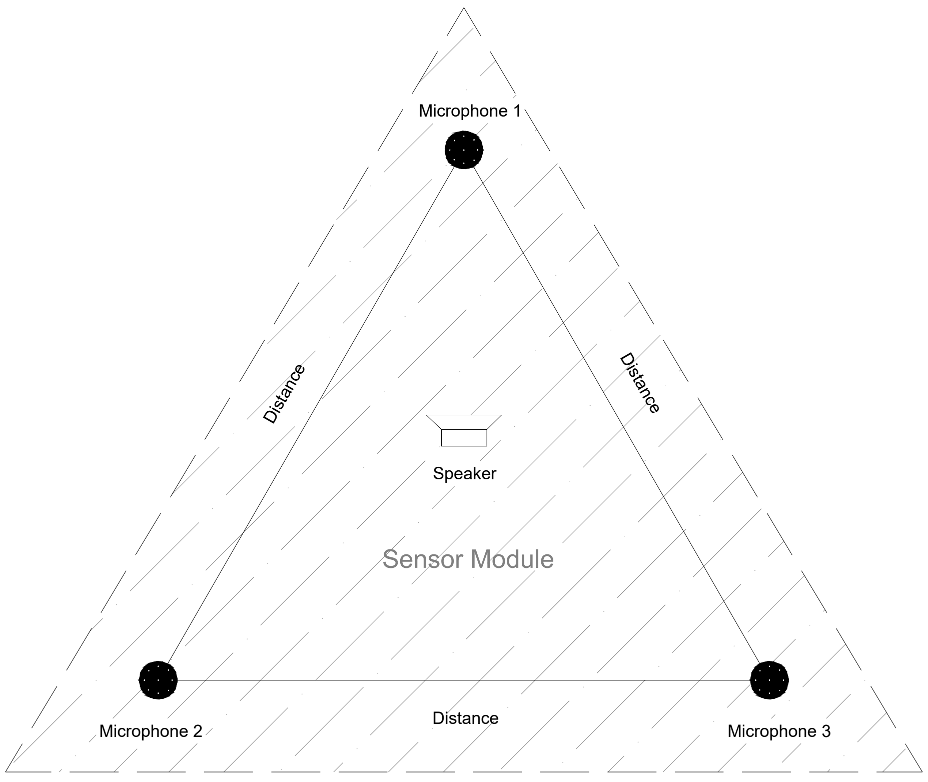


Figure 30: Localization Module

The acoustic signal is send by a speaker facing upward. A metal plate is placed a 3 mm above the speaker, tests during the research phase showed this greatly improved the omnidirectional spread of the signal. The signal that's send it chosen to have a frequency of 3,8 kHz which has the best omnidirectional and range properties (section 4 of the research document).

The signal received by the microphone, translates in just a few millivolts. This signal is amplified 48 times. Then the signal is fed into the ADC of the Xmega256a4u. The sender code and receiver code will now be discussed in detail.

8.1.2 Acoustic sender

The signal is being generated within a micro-controller and sent to the speaker using a digital to analog converter. This signal is first being amplified to make sure it is loud enough. The block diagram for modulating this signal is shown in figure 31. The signal is set at 3764 Hz, it consists of an array using 50 samples per period. The sender will send out a series of 10 "beeps", every beep consists of 1250 samples of the 3764 Hz signal. For every beep the sending module sends a interrupt signal to the receivers. The mean is taken of the array samples of the five sampled signals. The code for the acoustic sender is shown below.

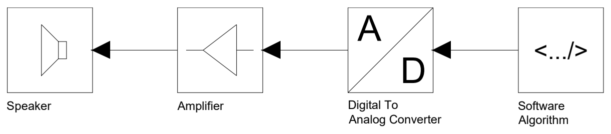


Figure 31: Modulator block diagram

```
/***************************************************
 * Sinus Generator
 * For Distance Measurement
 * Created: 14-6-2016 11:01:46
 * Author: Zwermen 2
****************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include "clk.h"
#include "UART.c"

#define F_CPU 32000000UL

uint16_t sin4kHz[50] =
   {4,36,100,195,319,470,646,844,1061,1294,1538,1791,
2048,2304,2557,2801,3034,3251,3449,3625,3776,3900,
3995,4059,4091,4091,4059,3995,3900,3776,3625,3449,
3251,3034,2801,2557,2304,2048,1791,1538,1294,1061,
844,646,470,319,195,100,36,4};

int SEND_COUNT = 0;
int MAX_IDX = 50;
volatile int count = 0;
uint8_t idxsin = 0;

ISR (TCC0_CCC_vect)
{                          // Timer overflow, put next
   sample into DAC
  PORTB_OUTSET = PIN0_bm;
  MAX_IDX = 50;
  DACB.CH0DATA = sin4kHz[idxsin];
  idxsin++;
  if (idxsin >= MAX_IDX) { idxsin = 0; }

}

ISR (TCD1_CCA_vect)                  // period timer
{
  if (count <= 250 && SEND_COUNT < 10){
    PORTB_OUTSET = PIN0_bm;
```

```c
    count++;
    SEND = 1;
  }

  if (250 < count && count <= 1000 && SEND_COUNT < 10)
   {
    PORTB_OUTCLR = PIN0_bm;
    count++;
    SEND = 0;
  }

  if (count > 4000 && SEND_COUNT < 10){
    count = 0;
    SEND_COUNT++;
    DACB.CH0DATA = 0;
  }

  if (SEND_COUNT == 10){
    DACB.CH0DATA = 0;
  }
}

int main(void)
{
  SystemClock_init();
  TCC0.CNT = 0;                    //Reset timer 0
  TCC0.PER = 160;                  //188235Hz
  TCC0.CTRLA = TC_CLKSEL_DIV1_gc;          //Prescaler
  TCC0.INTCTRLB = TC_CCCINTLVL_LO_gc;        //
   TCC0_CCC_vect, Compare Match

  TCD1.CNT = 0;                    //Reset timer
  TCD1.PER = 0x2134;                 //3764Hz
  TCD1.CTRLA = TC_CLKSEL_DIV1_gc;          //Prescaler
  TCD1.INTCTRLB = TC_CCAINTLVL_LO_gc;        //
   TCD0_CCA_VECT, Compare Match

  PMIC.CTRL = PMIC_LOLVLEN_bm;
  sei();

  DACB.CTRLB = 0x00;                 //Single channel
   operation PB2 only
  DACB.CTRLC = 0x08;                 //Vref = Analog
   Supply Voltage
  DACB.CTRLA = 0x04;                 //CH0EN = Enable
   Channel 0
  DACB.CTRLA |= 0x01;                //ENABLE = Start
   the DAC

  uart_init();
  stdout = &mystdout;

  PORTB_DIRSET = PIN0_bm;

  while(1)
```

Code 5: Acoustic sender code

In the next sections the workings of the receiving end will be explained.

8.1.3 Acoustic receiver

The signal send by the acoustic sender is received by the microphone of the receiving modules. As discussed in the research section, a simple version of cross-correlation is used to determine the TOF (time of flight) of the acoustic signal. The receiving micro-controller has a copy of the array that is used to create the signal in the sending micro-controller. When the receiver, receives an interrupt from the sender it will start to sample data from the ADC that's connected to the microphone. The sampling time is 50 times per period, this is exactly the same as the sending side. Ten individual beeps are sampled. The mean is taken from the five individual samples before the cross-correlation algorithm is performed to determine the distance. For further explanation on the simplified version of cross correlation used see section 5.4.7. A Block representation is given of the receiving side shown in figure.

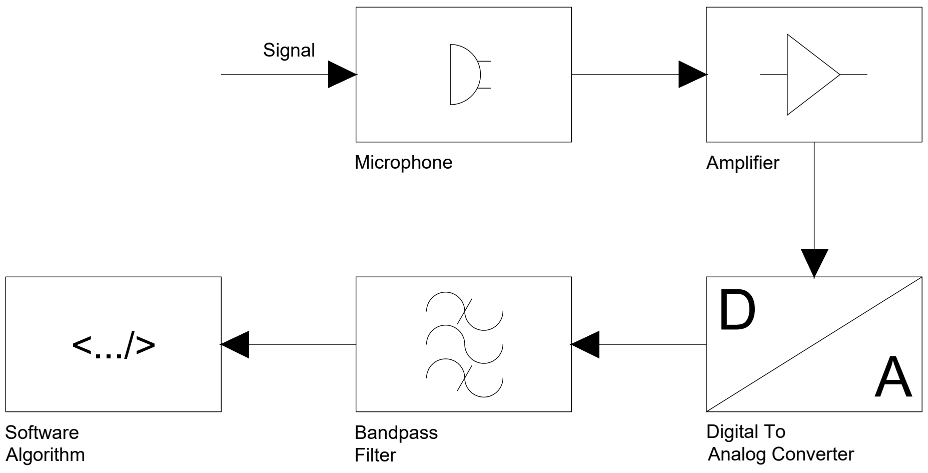


Figure 32: Block representation of the receiving side

The distance is derived from the time it takes the acoustic signal to travel from module A to module B. Suppose that module A is about to send its signal. Just before it starts sending the acoustic signal, it sends a message over the radio communication, which is almost instant compared to the speed of the sound waves. Module B starts its timer and wait for the signal to arrive. The distance can now be derived multiplying the speed of sound with the measured time. The state diagram for the distance measurement is shown in 33.

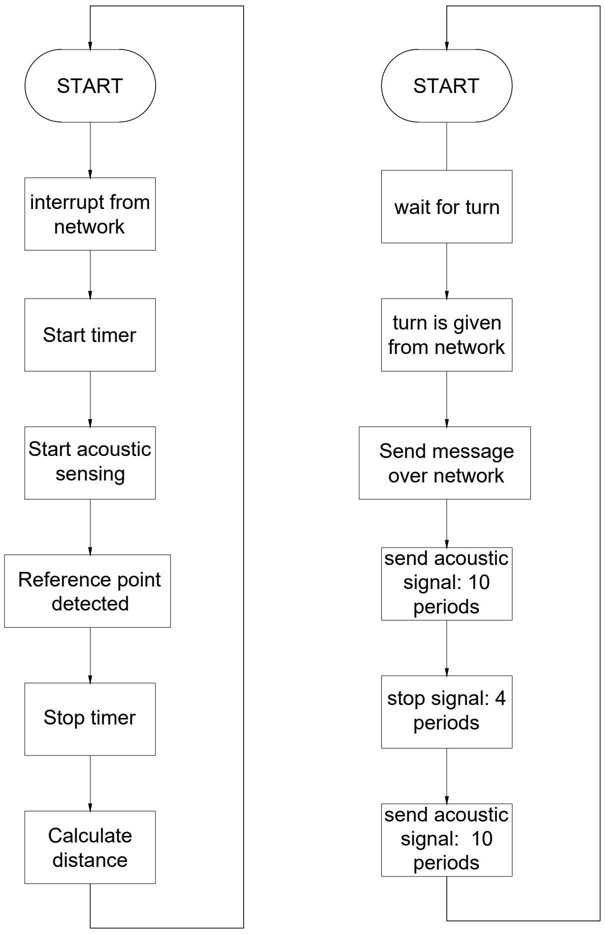


Figure 33: State diagrams for the distance measurement. The diagram on the right is for the receiving side, and the left diagram is for the sending side.

This state diagram is translated in C code shown in 6. The correlation function can be found in the research section.

```
/***************************************************
Implementation
Simplified Cross Correlation
For distance measurements
Created: 20-06-2016
Author: Zwermen 2
***************************************************/

#include "main.h"

void Config32MHzClock(void);

```

```
int main(void) {

  PORTB.INT0MASK = PIN0_bm;
  PORTB.PIN0CTRL = PORT_ISC_RISING_gc;
  PORTB.INTCTRL  = PORT_INT0LVL_HI_gc;

  PORTC_DIRSET = PIN0_bm; //Blue LED
  PORTF_DIRSET = PIN0_bm; //Green LED
  PORTF_DIRSET = PIN1_bm; //Red LED

  SystemClock_init();   // init 32MHz clock
  ADC_init();         // init ADC
  uart_init();        // init uart
  adc_timer_init();   // ADC timer init
  stdout = &mystdout;

  PMIC.CTRL |= PMIC_LOLVLEN_bm | PMIC_HILVLEN_bm;
  sei();

while(1){
    if(measured >= MEASURES){
      measured = 0;
      quickSort(time_delay, 0, MEASURES - 1);
      time_shift = filter(time_delay, MEASURES);
      result = factor * (float) time_shift;
      int z = result*100;

      printf("Measured Distance:%dcm\n", z);

      for(int y = 0; y <= MEASURES; y++){
        time_delay[y] = 0;
      }
    }
  }
}

ISR(PORTB_INT0_vect)
{
  if(measure == 0){
    ADCdata[0] = ADC_read();
    samples = 1;
    PORTC_OUTTGL = PIN0_bm;
    TCD0.CNT = 0;
    measure = 1;
  }
}

ISR(TCD0_OVF_vect)
{
  if(measure == 1){
    ADCdata[samples] = ADC_read();
    samples++;

    if(samples > ADCLENGTH){
      cli();
      time_delay[measured] = correlation (ADCLENGTH,
   lenraw, rawdata, ADCdata);
      measured++;
      measure = 0;

      for(int i = 0; i <= ADCLENGTH; i++){
        ADCdata[i] = 0;
      }
      sei();
    }
  }
}
```

Code 6: Acoustic receiver and correlation code

8.1.4 Angle measurement and relative location

The Swarming module has three microphones placed with known distances between them. The relative angle is derived from the time measured with each microphone, and the distance between the microphones. In section 3.2 there's more information on how the algorithms needed. The following algorithm shows pseudo-code how the angle and relative location is derived from the three measured distances. No implementation of this pseude-code was achieved during this project. A reliable distance measurement is needed to derive the angle using this method. Small deviations in the distance can cause far bigger deviations in the angle, due to the relatively small distance between the microphones. The distance measurement achieved during this project, will still need further tweaking before it can be used for localization.

```
ISR(reference call){
  start timer;
}
ISR(mic 1){
  Time 1:= Timer;
  recieved:=recieved+1;
}
ISR(mic 2){
  Time 2:= Timer;
  recieved:=recieved+1;
}
ISR(mic 3){
  Time 3:= Timer;
  recieved:=recieved+1;
}

While recieved = 3{
  time_1:= time_1-overhead_time;  //remove overhead
   time
  time_2:= time_2-overhead_time;  //remove overhead
   time
  time_3:= time_3-overhead_time;  //remove overhead
   time
  length_1:=time_1*speed_of_sound/clock_freq
  length_2:=time_2*speed_of_sound/clock_freq
  length_3:=time_3*speed_of_sound/clock_freq
  angle_1:=acos((length_1^2+mic_distance^2-length_2^2)
   /(2*length_1*mic_distance));
  angle_2:=acos((length_2^2+mic_distance^2-length_3^2)
   /(2*length_2*mic_distance));
  angle_3:=acos((length_3^2+mic_distance^2-length_1^2)
   /(2*length_3*mic_distance));
  read compas_angle;
  angle_1:=angle_1+compas_angle;
  angle_2:=angle_2+compas_angle;
  angle_3:=angle_3+compas_angle;
  x_1:=sin(angle_1)*length_1;
  x_2:=sin(angle_2)*length_2;
  x_3:=sin(angle_3)*length_3;
  y_1:=cos(angle_1)*length_1;
  y_2:=cos(angle_2)*length_2;
  y_3:=cos(angle_3)*length_3;
  x_center=x_1+center_distance_x;
  y_center:=Y_1+center_distance_y;
  recieved:=0;
}
```

Code 7: Algorithm to derive the relative location

8.2 Swarm communication implementation

During the research, we came across some plug and play wireless swarm communication implementations. One of these implementations is the SwarmBEE LE module from Nanotron. The SwarmBEE LE module meets the given network criteria and most of the recommendations. The module uses a radio frequency 2,4 GHz signal as a communication channel. The module is not only a communication module, but can also be used to determine relative distances to each swarming module. It also has a API with predefined functions which can be used to set up the wireless network.

8.2.1 Central operating unit

In Table 9 the basic specifications of the AtXmega128A4U are shown. This is the microcontroller used in as the central operating unit. The microcontroller

Table 9: Central operating unit

Parameter	**Value**
Flash (kBytes):	128 kBytes
Pin Count:	44
Max. Operating Freq. (MHz):	32 MHz
CPU:	8-bit AVR
Max I/O pins:	34
USB Interface:	Device
SPI:	7
TWI (I2C):	2
UART:	5
ADC Channels:	12
ADC Resolution (bits):	12
ADC Speed (ksps):	2000
DAC Channels:	2
SRAM (kBytes):	8
EEPROM (Bytes)	2048
Operating Voltage (Vcc):	1.6 to 3.6

from Atmel complies with all the specifications. There are also newer micro controllers available, but are not yet implemented for this project, since the older versions can handle the workload and work fine. There are two I^2C connections available on the microcontroller which are needed to connect the module to the robot. Also the multiple ADC and DAC connections can come in handy if any sensors or actuators need to be added to the module. Overall the AtXmega128A4U fits all the specification that are required, and there is room to expand the functionality when it is necessary. An overview of all the specification can be found on the website of Atmel.

The Central operating unit handles multiple functions to make the group of robots a swarm. Those functions will be explained in this section. There are four major functions that are implemented in the microcontroller. Controlling communication that goes in and out, storage and maintain data, calculating the population of the swarm and send and receive message that are used for localization. The communication will be mainly about the communication with the Swarmbee module. The connection to the robot is just as important but will not be thoroughly discussed in this document. First of all we will discuss the implementation of the communication.

8.2.2 Communication

The central operating unit needs to send request and receive data from the Swarmbee module. This information is transmitted using the UART protocol. The communication is mainly used for sending API commands to the Swarmbee module. The UART connecting uses a baudrate of 115200 bps since this is the transmission speed of the Swarmbee module. The Xmega uses two pins, RX (PC2), TX PC3) to achieve a wired connection with the Swarmbee. The module needs to be able to communicate with a robot platform. To communicate with this platform the module uses I^2C communication. This should be standard for every robot, so that the module can easily be placed on any type of robot platform. For debugging pin C7 en C8 are reserved. The debugging connecting can be established with a terminal such as Putty.

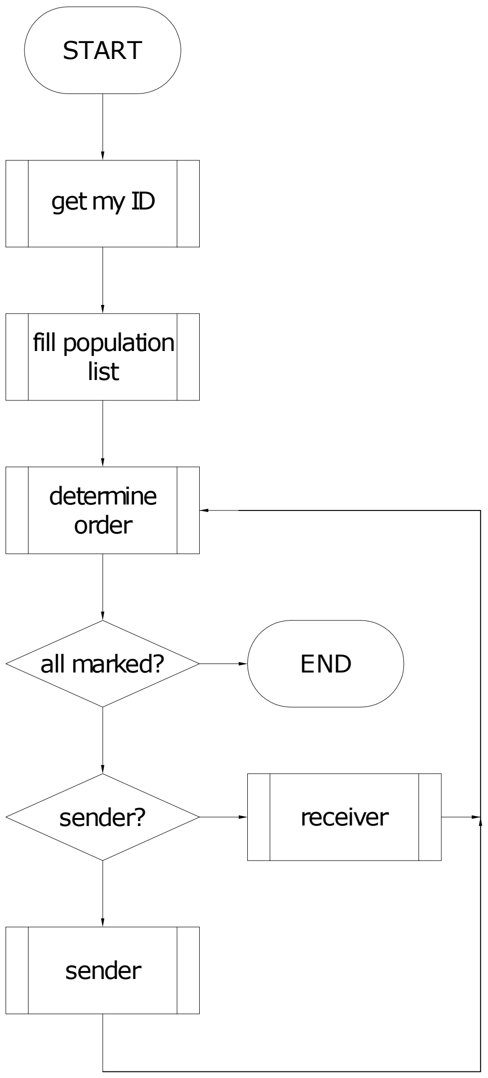


Figure 34: main algorithm for updating the swarm population including range requests

8.2.3 Swarm population

To update the swarm population all modules need to receive the ID's from other modules in range of the Swarmbee module. This is the first step in the flowchart of the main algorithm shown in figure 34. The swarmbee automatically assigns a unique ID to all nodes. A list of all nodes and the corresponding ranges can be requested from the swarm by a API broadcast message "*RRN". This broadcast message can be requested at a certain interval which depends on the demand of this information. The broadcast interval can be set via a command; ”sbiv 1000”, where sbiv stands for ”set broadcast interval” and the time is set at 1000ms, or 1 second. When the swarmbee receives a ranging request its response will look somewhat like this:

```
*RRN:1F3123123133,1F3CFF322133,0,001843,04,-56
```

Where the first two hexadecimal values represent the source ID (sending module), and the destination ID (receiving module). This is followed up by an error code and the range relative in centimetres to the destination node. The end of the message contains a custom value which can be set by a ”notification configuration”-request. For example, this value can represent different sensor values like; x,y,z acceleration, the measured RSSI value or the current battery level.

To fill the population list with including ranges first, the module needs to know its own ID. This can be received by a command, ”GNID”, or ”get node ID”. A volatile variable ”myid” will be set to this string.

Now that the node knows its own id the population list can be filled with information. A frame in the list will look like this:

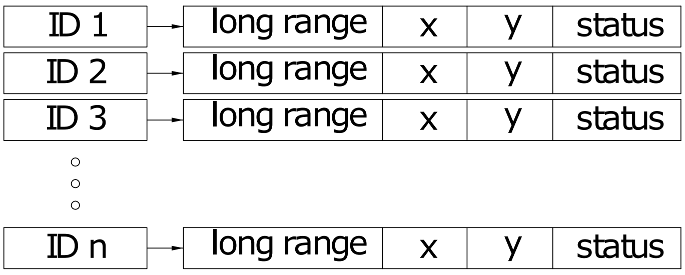


Figure 35: One population list frame

Where the ID and the longrange can be received from the swarmbee module, the x and y coordinates are from the acoustic sensing algorithm and the status is set by the main algorithm.

The following code describes how this function will be executed.

```
void fillpopulationlist(char *message){
    char* myNodeId = "343556767898";
    char* messagePointer;
    char* databaseRow [DATASIZE];
    uint8_t count = 0;
    row_t* rowx;
    char* NodeID;

    rowx = (struct row *) malloc(sizeof(row_t));
    messagePointer = message;

    while(*messagePointer != ','){
        rowx -> id[count] = *messagePointer;
        *messagePointer++;
        count++;
16      }
17      *messagePointer++;
18      rowx -> id[count] = EOS;
19      count = 0;
20
21      while(*messagePointer != ','){
22          rowx -> targetID[count] = *messagePointer;
23          *messagePointer++;
24          count++;
25      }
26      *messagePointer++;
27      rowx -> targetID[count] = EOS;
28      count = 0;
29
30      while(*messagePointer != ','){ //skip "0,"
31          *messagePointer++;
32      }
33      *messagePointer++;
34
35      while(*messagePointer != EOS ){
36          rowx -> longRange[count] = *messagePointer;
37          *messagePointer++;
38          count++;
39      }
40      rowx -> longRange[count] = EOS;
```

This function will be executed when the received message is a ranging request notification, because this is the message that will be the argument of this function. At line 16 a while loop starts filling: srcid, destid and distance with the corresponding information from the ranging request notification. A comma indicates that a new value starts. At line 34 and 37 the received id's are compared to its own id so that the right id will be set with the range. The id's and longranges are now filled in the population list.

8.2.4 Acoustic sensing send/ receive

After all node ID's are known the first swarm unit in queue will get the mutual exclusion to be the only sender in the acoustic ranging process, it will enter the "send acoustic"-process, figure 36.1. since only one member can be the sending module the rest of the swarm will enter the "process receive" algorithm, figure 36.2. When a node is done sending it will set a mark and send a broadcast to the rest of swarm. The status bit in the frame (35) will be set as one, now all members know that this ID is done. The "order"-function(pseudocode) will keep checking who is the lowest in queue. When all id's status bits are marked the complete algorithm is completed.

```
void order(){
  if(all ids marked as done sending){
    //END     //the entire populationlist is completed
  }
  else if(myid == marked as done sending || myid =!
   lowest, unmarked, id){
      receiver();
    }else{
      sender();
    }
}
```

When the swarm module is in sending mode it will send out a sound signal and retrieve the ranging results form the receiving modules one by one. The receiving modules will start the localization algorithm when a request from the sender is received. When there is no result a node will be marked as "out of range". When a sender retrieved all ranges this node gets marked as 'done'. The main algorithm, figure 34, will now give the mutex to the next swarm member until all units are marked. When all nodes are marked, each module will create its own node map. When there are any missing ranges these gaps will be filled by the (long) ranges received by the swarm bee in the ranging request broadcast message.

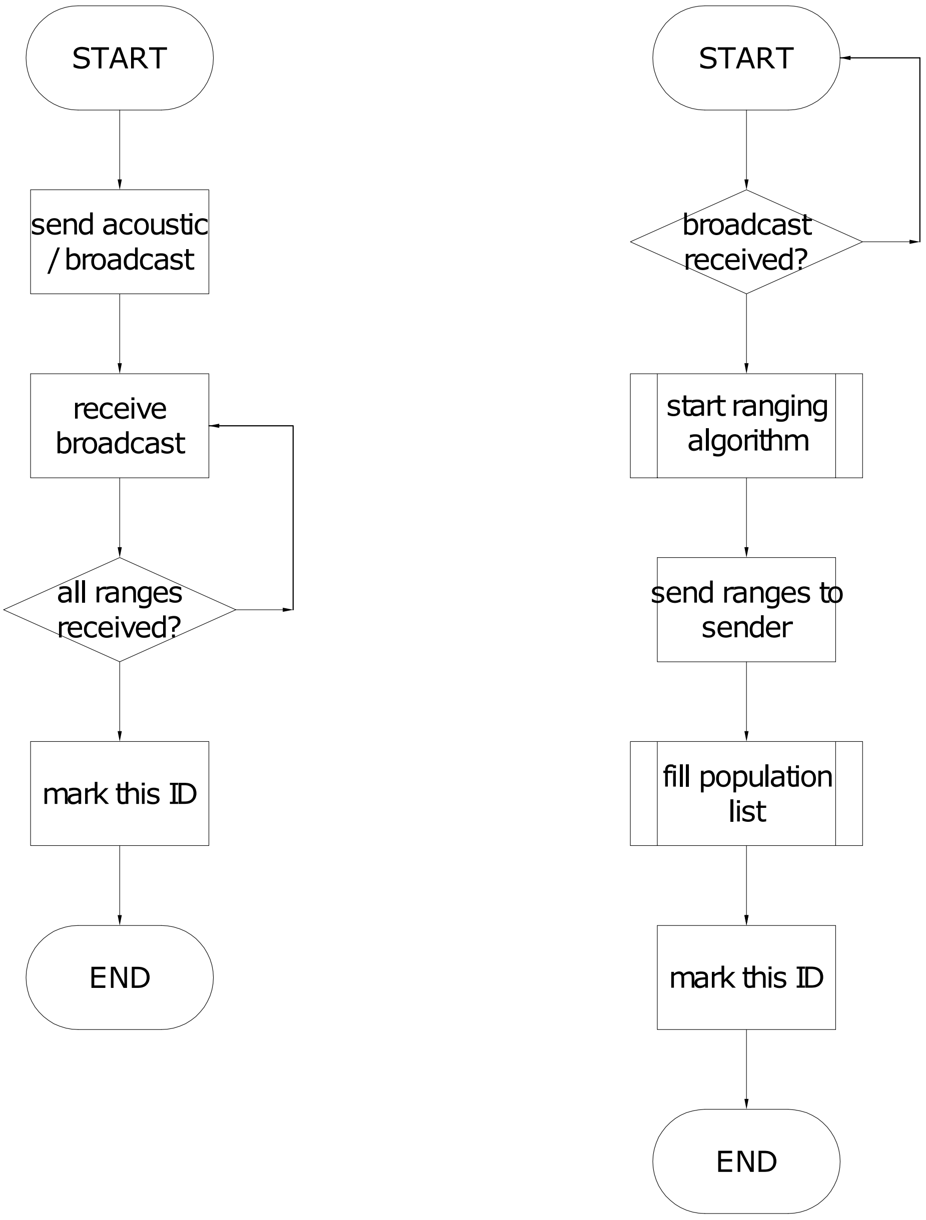


Figure 36: 1:"send acoustic" when the swarm module is in sensing state 2:"process receive", when the swarm module is in receiving state

```
void sender(){
  char[12] recid;
  char[RDATLEN] rrangedata;

  PORTA.DIRSET = PIN0_bm;
  //signal other microcontroller to send acoustic
   signal
  Tx1: ("bdat 0 BDATLEN %c\r\n",myid);
  PORTA.DIRCLR = PIN0_bm;
  //broadcast start of acoustic sending

  //Rx2: sends signal when sendingprocess is finished
  if (Rx2 == "finished"){
    command(gdat);      //read out transmitted data
    //Rx1: "=RDATLEN,recid,rrangedata"
    if(RDATLEN =! 0){
      populationlist(recid,0,x,y);
      //fills populationlist frame
    }
    else{
      //Populationlist: mark myid as done sending
      command("bdat 0 BDATLEN %c\r\n",mark);
      //done sending mark
      order();
    }
  }
}
```

```
void receiver(){
  char[12] sendid;
  char[TDATLEN] trangedata;

  if(Rx1 == done sending mark){
    //Populationlist: mark sendid as done
    order();
  }

  //Rx1: = "AA1122334455"
  sendid = Rx1;
  if(Rx == AA1122334455 ){  //when broadcast message
   has been received
    PORTA.DIRSET = PIN1_bm;
    //signal other microcontroller sending has started
    PORTA.DIRCLR = PIN1_bm;
  //Rx2: acoustic ranging results
  rangedata = Rx2;
  //TDATLEN = bytes data
  Tx1: ("sdat 1 %c TDATLEN %c\r\n",sendid, trangedata)
   ;   //send ranging information to sender
  }
}
```

The communication module uses two UART connection. One for the communication with the swarmbee module and one for debug purposes. Two functions have been made to keep those data channels separated from eachother. F The validatemessage compares the received message with a defined type of message length. The type of messsage is defined by another function, "DetermineCommandType". This function separates the first characters of the message (the command characters) from the rest. Based on the first separated command characters it executes the intented function.

```c
uint8_t ValidateMessage (char *message, uint8_t command){
    uint8_t messageLength;
    messageLength = strlen(message) - 2;

    switch (command) {
        case 1: // command RRN
            if (messageLength == RRN_LENGTH){
                return true;
            }
        case 2: // command XX
            if (messageLength == RRN_LENGTH){
                return false;
            }
        case 3: // command XX
            if (messageLength == RRN_LENGTH){
                return false;
            }
        default:
            return false;
    }
}
```

Code 8: Validates the integrity of the received message.

```c
void DetermineCommandtype (char *message){
  char *messagePointer;
  char command[4];
  uint8_t count = 0;

  memset(command, EOS, strlen(command));
  messagePointer = message;

  //DebugPrint(message);

  *messagePointer++;

  while(*messagePointer != COMMAND_END){
    *messagePointer++;
    if(*messagePointer == ( COMMAND_END)){ break;}
    command[count] = *messagePointer;
    count++;
  }
  command[count] = EOS;

  *messagePointer++;

  if    (strcmp(command, "*RRN") == 0){ // Data
   Notification Message
    //RRN_function(messagePointer);
    fillpopulationlist(messagePointer);
  }else if(strcmp(command, "DNO")  == 0){ // Node ID
   Notification Message

  }else if(strcmp(command, "NIN")  == 0){ // Ranging
   Result Notification Message
    //printf("NIN\n");
  }else if(strcmp(command, "SDAT") == 0){ // SDAT
   Notification Messages
    //printf("SDAT\n");
  }else if(strcmp(command, "AIR")  == 0){ // AIR
   Notification Message
    //printf("AIR\n");
  }else{
    DebugPrint("No command\r\n");
  }
}
```

Code 9: Determines the meaning of the message.

The translateMessage function receives byte for byte from the swarmbee modules and forms the intended messages.

```
/**
 * Translates the received message converts characters
 * to a single string
 * @param   value received value
 * @return  message pointer to the translated message
 */
char * TranslateMessage (void){
  char value[128];

  memset(globalMessage, EOS, strlen(globalMessage));
  memset(value, EOS, strlen(value));

  value[0] = uart_getc(&uartC0);
  strcpy(globalMessage, value);
  while (value[0] != CR){
    if (value[0] != CR){

      strcat(globalMessage, value);
    }
    value[0] = uart_getc(&uartC0);
  }
  return globalMessage;
}
```

Code 10: Translates the received message. Converts characters to a single string

8.2.5 Population list

As the specifications stated, the swarm population must be dynamic. This means that the gathered data must be stored in a dynamic list. Just as the other parts of the code, this is implemented in C code.

The dynamic list is based on the linked list principle, where the next item is linked to the previous item. Each 'item' consists of a one dimensional array with string values. This principle is equivalent to an two dimensional array, but makes it a lot more complex.

A structure 'node' is defined and consits of a pointer to next node and the data located at the current item adress. The first item that gets inserted starts at the head (beginning of the adress range).

```c
typedef struct node {
    char* data [DATASIZE];
    struct node * next;
}node_t;
```

Code 11: Linked list node structure.

Not every function writen is used in the final code for the swarmbee. For future use and the development and debug purposes, these functions are still usefull. The following enumeration summarizes the purpose of each function:

- SizeOfList - Amount of modules in the list
- PrintHeaderList - Used for debug purposes
- PrintList - Used for debug purposes
- AppendList - Inserts a data array row to the end of the list
- InsertList - Inserts a data array row at the beginning of the list
- PopListByNumber - Reads a data array row at the given index number and removes it afterwards
- PopListByValue - Reads a data array row with the given first value of the current row and removes it afterwards
- PopList - Reads a data array row at the beginning of the list and removes it afterwards
- ViewListByNumber - Reads the data array row at the given index number

The following is the implementation of the insert and append function. In the final code only the insert function is implemented. This is done to create a first in first out buffer, this is explained later with the pop functions.

```c
void append (node_t *listHead, char* data[DATASIZE]){
    node_t *current = listHead;
    while (current -> next != NULL){
        current = current -> next;
    }
    current -> next = (struct node *) malloc(sizeof(
   node_t));
    for (int item = 0; item < DATASIZE; ++item){
        current -> next -> data[item] = data[item];
    }
    current -> next -> next = NULL;
}
```

Code 12: Inserts a data array row to the end of the list

```c
void insert (node_t ** listHead, char* dataInternal [
   DATASIZE]){
    node_t * new_node;
    size_t size = sizeof(node_t);

    new_node = (struct node *) malloc(sizeof(node_t));
    if (new_node == NULL){
        DebugPrint("No memory");
    }

    memset(new_node, 0, sizeof(node_t));

    for (int item = 0; item < DATASIZE; ++item){
        new_node -> data[item] = dataInternal[item];
    }

    new_node -> next = *listHead;
    *listHead = new_node;
}
```

Code 13: Inserts a data array row at the beginning of the list

The code for the pop functions can be found in the following three code snippets. The pop by number is implemented because of the chronologic order. It retrieves the total amount of rows in the list and begins to pop at the last and finishes at the first row.

```c
char* pop (node_t ** head){
    node_t * next_node = NULL;

    if (*head == NULL) {
        return NULL;
    }
    next_node = (*head)->next;
    for (int item = 0; item < DATASIZE; ++item){
        returnArray[item] = (*head) -> data[item];
    }

    free(*head);
    *head = next_node;

    return *returnArray;
}
```

Code 14: Reads a data array row at the beginning of the list and removes it afterwards

```c
char* popListByNumber(node_t ** head, int indexNumber)
    {
    node_t *current = *head, *temp_node = NULL;

    if (indexNumber == 0){
        return pop(head);
    }

    for (int item = 0; item < indexNumber-1; ++item){
        if (current -> next == NULL){
            return NULL;
        }
        current = current -> next;
    }

    temp_node = current -> next;

    for (int item = 0; item < DATASIZE; ++item){
        returnArray[item] = temp_node -> data[item];
    }

    current -> next = temp_node -> next;
    free(temp_node);

    return *returnArray;
}
```

Code 15: Reads a data array row at the given index number and removes it afterwards

```c
char* popListByValue(node_t ** head, char* value, int
    sizeOfList){
    node_t *current = *head,*temp_node = NULL;

    if (strcmp(current -> data[FIRSTDATAITEM], value)
    ==0){
        return pop(head);
    }

    for (int item = 1; item < sizeOfList; ++item){
        temp_node = current -> next;
        if(strcmp(temp_node -> data[FIRSTDATAITEM],
    value)==0){
            for (int itemb = 0; itemb < DATASIZE; ++
    itemb){
                returnArray[itemb] = *temp_node ->
    data[itemb];
            }
            current -> next = temp_node -> next;
            free(temp_node);

            return *returnArray;
        }
        current = current -> next;
    }
    return NULL;
}
```

Code 16: Reads a data array row with the given first value of the current row and removes it afterwards

```
char* viewListByValue(node_t ** head, char* value, int
     sizeOfList){
    node_t *current = *head,*temp_node = NULL;

    if (strcmp(current -> data[FIRSTDATAITEM], value)
   ==0){
        return pop(head);
    }

    for (int item = 1; item < sizeOfList; ++item){
        temp_node = current -> next;
        if(strcmp(temp_node -> data[FIRSTDATAITEM],
   value)==0){
            for (int itemb = 0; itemb < DATASIZE; ++
   itemb){
                returnArray[itemb] = *temp_node ->
   data[itemb];
            }
            current -> next = temp_node -> next;

            return *returnArray;
        }
        current = current -> next;
    }
    return NULL;
}
```

Code 17: Reads a data array row with the given first value of the current row

```
int sizeOfList(node_t *listHead){
    int sizeOfList = 1;
    node_t * current = listHead;

    while (current -> next != NULL){
        current = current -> next;
        sizeOfList++;
    }
    return sizeOfList;
}
```

Code 18: Amount of modules in the list

An example of the debug interface output. There is not data (displayed with x's) in the this example. Each 'Data[]' represents an value for example, the first one is the node ID.

```
Populationlist
Nr.      Data[0] Data[1] Data[2] Data[3] Data[4]
0        xxx     xxx     xxx     xxx     xxx
1        xxx     xxx     xxx     xxx     xxx
2        xxx     xxx     xxx     xxx     xxx
3        xxx     xxx     xxx     xxx     xxx
4        xxx     xxx     xxx     xxx     xxx
```

Code 19: Example of the population list with x's at the place of swarm data.

8.2.6 Communication test plan

Testing is a critical part of a project. It proofs whether a product suits the specifications that where made in an earlier stage. If it does not fit the specifications, the results needs to be documented well and can be given to a next project group. So that they will know this solution is not the right one and needs modification to fit the specifications. In this document the test plan and results of the Swarmbee will be discussed. The main tasks of the Swarmbee module are to determine the distance between multiple Swarmbee's, and to send and receive data from swarm module to swarm module. First of all the specifications of the Swarmbee will be shown again, then we will discuss our test plan, followed by the results and conclusion. Figure 37 shows the measurement arrangement.

In an earlier stage of this project, specifications where made for the long distance measurement and the data speed between the swarm modules. These where as following:

	Close range (0 - 3 m)	**Long range (3 - 25 m)**
Dead zone distance	0,015 meters	2 meters
Deviation (distance)	+/- 9%	+/- 1 meter
Angle Resolution	45%	-
Update frequency	40 Hz	1 Hz

Table 10: Swarming localization specifications

Table 11: Wireless communication module

Module	Wireless communication module
Input	5 VDC Swarm data RF signal
Outputs	RF signal Swarm data
Function	Processes information to and from other swarm modules. Transmission speed is not defined yet. The transmission range must be around 100 meters or more.

From these specifications we chose the Swarmbee module that had the following specification:

Table 12: Swarmbee LE module

Parameter	**Value**
Frequency range	ISM band 2.4 GHz (2.4 - 2.4835 GHz)
Modulation	Chirp Spread Spectrum (CSS)
Transmission modes	80 MHz, 1 Mbps or 250 Kbps (80/1 or 80/4 mode)
TOA capture accuracy	<1 ns (better than 30 cm)
Typical air time per ranging cycle	1.8 ms
RF output power	configurable - 22 t0 16 dBm
RF sensitivity	-89 dBm typ. @80/1 mode -95 dBm typ. @80/4 mode
RF interface	50 Ohm RF port (for external antenna)
Host interface (UART)	500bps ~ 2 Mbps
Power supply	3 - 5.5 V
Max. supply voltage ripple	20 mVpp
Active power consumption*	120 mA during transmission, 60 mA during receive in 80/1 mode
Power consumption in sleep mode*	5.5 mA (transceiver disabled, all peripherals on)
Power consumption in snooze mode*	4.5 microA (transceiver disabled, all peripherals off, wake-up by timer)
Power consumption in nap mode**	4.5 ~ 600 microA (transceiver disabled, all peripherals off, wake-up by interrupt)
Power consumption in deep-sleep mode*	<1 microA (device completely disabled)
Operating temperature range	-30 - 85 °C
Dimensions	40 mm x 24 mm x 3.5 mm
Weight	7 g
*Power consumption in all modes is measured at 20°C, 3.3 V.	
**Power consumption in nap mode depends on interrupt sources (GPIO pins or MEMS or both).	

For testing the long distance measurement two different situations where tested. One test in doors, and another outdoors. We measure the distances relative to node 1, which is connected to a computer. From the computer we can see the measured distance from node 2 or 3 to node 1. We measure the following distances in order: 3 meter (minimal distance), 5 meter, 10 meter, and from here every 10 meter until 60 meters. Further then this distance can not be measured since the building is not long enough to measure the distance in a straight line. Outdoors we do the same thing only here we measure until a 100 meters. Every distances is measured three times per node. So we have 6 measurement points per distance outdoor and 6 measurement points indoor. These measurements will be written down in Excel and will be analysed. In figure 37 is a simplification of how the measurement was done.

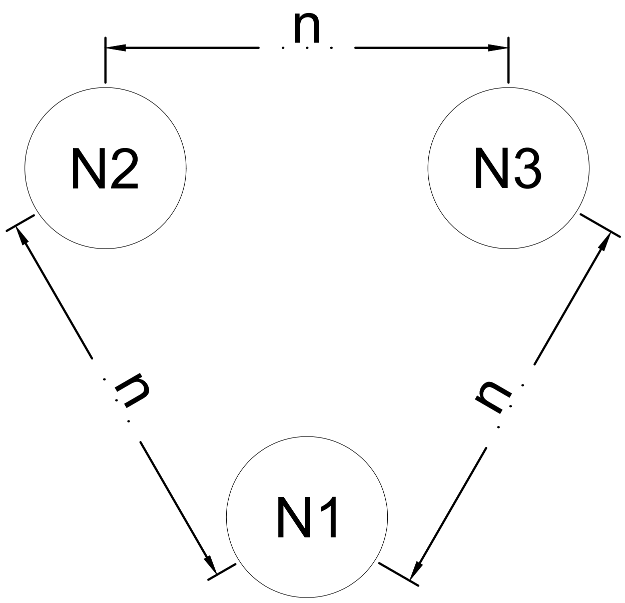


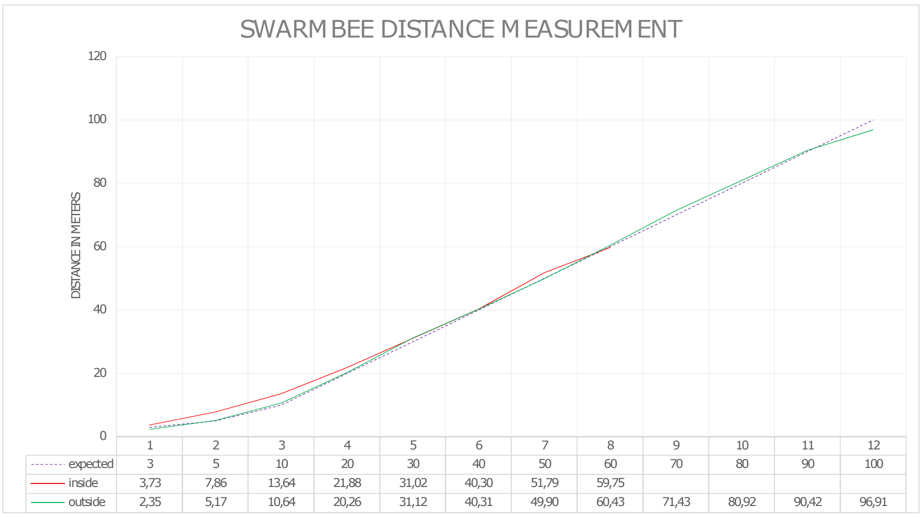
Figure 37: Arrangement of the swarmbee measurement for 3 to 100 meters.

8.2.7 Test results

Take note that the long-range distance measurements are received less frequent as the distance increases. The maximum distance in an average room is about 60-65 meters, the connection between the swarmbeeś is instable or lost when this range is passed. As shown in figure 38, some values indicate a distance of 0.0 meters. This means that the swarmbee was not able to calculate a relative distance despite the connection was established. If there is a connection with or without a distance calculation, the connection is ”J”. Without a distance calculation the connection is 'N'.

		poging	afstand (m)	connectie	afstand (m)	connectie	afstand (m)	connectie	afstand (m)	connectie	afstand (m)	connectie	afstand (m)	connectie	afstand (m)	connectie	afstand (m)	connectie	afstand (m)	connectie	afstand (m)	connectie	afstand (m)	connectie	afstand (m)	connectie
	afstand (m)		3	J/N	5	J/N	10	J/N	20	J/N	30	J/N	40	J/N	50	J/N	60	J/N	70	J/N	80	J/N	90	J/N	100	J/N
Binnen	1 naar 2	1	1.44	J	9.13	J	15.34	J	24.34	J	35.72	J	39.83	J	51.05	J	59.73	J	0.00	N	0.00	N	0.00	N	0.00	N
		2	1.53	J	7.81	J	14.92	J	23.42	J	30.06	J	39.64	J	52.57	J	62.04	J	0.00	N	0.00	N	0.00	N	0.00	N
		3	1.34	J	8.61	J	12.28	J	23.07	J	0.00	N	40.64	J	53.31	J	59.78	J	0.00	N	0.00	N	0.00	N	0.00	N
	1 naar 3	1	0.00	J	6.88	J	13.31	J	19.03	J	29.04	J	40.35	J	50.18	J	59.47	J	0.00	N	0.00	N	0.00	N	0.00	N
		2	7.61	J	7.27	J	13.63	J	19.37	J	30.07	J	40.71	J	51.62		58.88	J	0.00	N	0.00	N	0.00	N	0.00	N
		3	6.71	J	7.44	J	12.35	J	22.07	J	30.19	J	40.64	J	52.01	J	58.61	J	0.00	N	0.00	N	0.00	N	0.00	N
Buiten	1 naar 2	1	2.5	J	4.99	J	10.69	J	20.37	J	31.20	J	40.50	J	51.01	J	60.52	J	70.84	J	80,7	J	90.63	J	96.96	J
		2	2.47	J	5.09	J	10.60	J	19.96	J	30.73	J	0.00	J	49.99	J	60.40	J	71.83	J	0.00	J	90.25	J	96.89	J
		3	2.12	J	5.40	J	10.67	J	20.37	J	31.43	J	40.22	J	49.41	J	60.96	J	71.97	J	81.30	J	90.76	J	96.71	J
	1 naar 3	1	2.22	J	5.09	J	10.61	J	20.25	J	31.20	J	40.21	J	49.51	J	60.38	J	71.53	J	81.20	J	90.52	J	0.00	J
		2	1.99	J	5.18	J	10.32	J	20.14	J	31.01	J	40.11	J	49.87	J	60.22	J	70.87	J	80.50	J	91.04	J	96.71	J
		3	2.77	J	5.24	J	10.92	J	20.45	J	31.17	J	40.50	J	49.59	J	60.07	J	71.52	J	80.90	J	89.32	J	97.30	J

Figure 38: Table of the swarmbee measurement for 3 to 100 meters. Values of zero are incomplete measurements of the swarmbee module. J or N indicate if there is a connection between the swarmbee's.

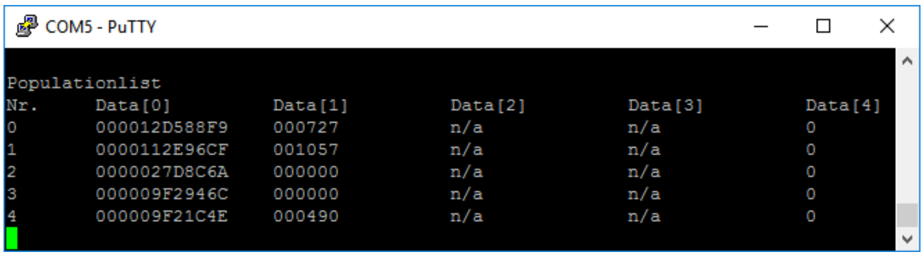


	1	2	3	4	5	6	7	8	9	10	11	12
expected	3	5	10	20	30	40	50	60	70	80	90	100
inside	3,73	7,86	13,64	21,88	31,02	40,30	51,79	59,75				
outside	2,35	5,17	10,64	20,26	31,12	40,31	49,90	60,43	71,43	80,92	90,42	96,91

Figure 39: Graph of the swarmbee measurement for 3 to 100 meters.

The results indoor are a bit disappointing. The deviations are on most measurement a lot larger then 1 meter that was specified. Even though the specifications of the Swarmbee state that the capture accuracy is better than 30 cm. The test results show that this statement is not true for the indoor distance measurement. Also the distance measurement should be updated more then a few times per second. We specified one time per second is acceptable, but most of the time it takes about 20-30 seconds for node 1 to recognize and then it could take up to a minute to send a distance measurement result. The further away the module the longer it takes to refresh the distance but if the node sends a measurement it tends to be more accurate then closer measurements. Objects or people moving in the area between the measurement influenced the connection a lot. The measured distance deviated 5 meters with object passing trough. Outdoors there was less of an obstruction and the measurements were a lot more accurate. Outside most of the measurements where within the limits of a deviation of 1 meter. But again not within 30 cm as is stated in the specifications of the Swarmbee. Outdoors the measurement was more stable, even when people where passing trough. The only downside was that the longer the distance the longer the node send his measured distances. When we tested the 100 meter, the measurement could take up to 1 to 2 minutes. This time the test used only three swarm modules and one of them used as a relative location. It would be interesting to know how a larger swarm will affect the update frequency. From this we can say that in some specific situations this module can fit some of our specifications, but there was no situation in which all of our specifications are met. This does not say that the specifications can not be reached with this module. There is a possibility to use an external antenna. We advise to use an external antenna and to redo this test to see if the measurements improve.

When a ranging request of 5 nodes has filled the dynamic array, the populationlist will look as follows:



```
Populationlist
Nr.     Data[0]         Data[1]         Data[2]         Data[3]         Data[4]
0       000012D588F9    000727          n/a             n/a             0
1       0000112E96CF    001057          n/a             n/a             0
2       0000027D8C6A    000000          n/a             n/a             0
3       000009F2946C    000000          n/a             n/a             0
4       000009F21C4E    000490          n/a             n/a             0
```

Figure 40: Putty log of the filled population list

8.2.8 Wireless communication

For the wireless communication module the most important function is that it can transmit information over atleast 100 meters. Since the specification states that the swarm must be detected within this range. The speed of the communication is not defined yet, but needs to be fast enough to handle all the messages. We estimated the speed that is necessary to handle all communication. We estimated that the maximum size of a message is 1kB. Since the long distance communication had an update frequency of 1Hz, the minimum communication speed that is required is 8kpbs. The Swarmbee LE module will be used for the wireless communication. The Swarmbee LE module will be used for the wireless communication. It uses the ISM band, 2.4 GHz to transmit and receive the data. The maximum range of the module relies on the environment in which it is used. Under ideal conditions the Swarmbee can reach a distance of 1200 meters. It depends on how many obstacles, reflections and interference there is to disturb the signal. An experiment from Nanotron showed that until a range of 150 meters the ranging success rate is about 100% with a transmission speed of 155 kbps. Experiments we have done ourself show that the antenna on the device does not respond very well inside a building. Test outdoors should still be done, but there is a possibility that an external antenna is needed to reach these distances. The actual speed is also not tested yet, but could be done in the future. Transmitting data between the swarm modules can be configured with a speed of 250 Kbps or 1 Mbps. Tests will show if this will be enough. This module fits all the specifications and is suitable for the use of wireless communication.

Table 13: Swarmbee LE module

Parameter	**Value**
Frequency range	ISM band 2.4 GHz (2.4 - 2.4835 GHz)
Modulation	Chirp Spread Spectrum (CSS)
Transmission modes	80 MHz, 1 Mbps or 250 Kbps (80/1 or 80/4 mode)
TOA capture accuracy	<1 ns (better than 30 cm)
Typical air time per ranging cycle	1.8 ms
RF output power	configurable - 22 t0 16 dBm
RF sensitivity	-89 dBm typ. @80/1 mode -95 dBm typ. @80/4 mode
RF interface	50 Ohm RF port (for external antenna)
Host interface (UART)	500bps ~ 2 Mbps
Power supply	3 - 5.5 V
Max. supply voltage ripple	20 mVpp
Active power consumption*	120 mA during transmission, 60 mA during receive in 80/1 mode
Power consumption in sleep mode*	5.5 mA (transceiver disabled, all peripherals on)
Power consumption in snooze mode*	4.5 microA (transceiver disabled, all peripherals off, wake-up by timer)
Power consumption in nap mode**	4.5 ~ 600 microA (transceiver disabled, all peripherals off, wake-up by interrupt)
Power consumption in deep-sleep mode*	<1 microA (device completely disabled)
Operating temperature range	-30 - 85 °C
Dimensions	40 mm x 24 mm x 3.5 mm
Weight	7 g
*Power consumption in all modes is measured at 20°C, 3.3 V.	
**Power consumption in nap mode depends on interrupt sources (GPIO pins or MEMS or both).	

9 Results and Discussion

It is essential that the communication is reliable and can handle the workload. If the communication is not working properly the whole system will fail. In the previous sections all the design choices are written down and explained. The result is a working swarm module that can be controlled in the way you want the swarm to work. All incoming and outgoing messages are handled reliable and can be easily accessed. The communication uses UART to handle internal communication with micro-controller and uses I^2C to handle the communication to the platform where the module will be installed. An added feature is that the communication now keeps track of the population. This was not a feature on the Swarmbee yet, but now the tracking is accurate and reliable. The Swarmbee modules are not tested to work like an actual swarm yet, but the only reason is that because there is not any swarm objective implemented. If the instructions are send to the swarm modules these can be send to the central operating unit of the robot where the robot can give a fitting response. The module can also determine the distance between each swarming module. This should work fine but there are some minor issues , which can be worked on in the future. The test with these feature where mainly done inside, but should work fine outdoors. Also the standard antenna is used, but a more powerful antenna could be easily implemented.

Three implementations of relative localisation are proposed. Two of which did not meet the expected specifications. The first method using the analog PLL was supported by thorough research but the theory did not meet the experiments in practice. The problem was found to be the comparator circuit which would cause differences in the delay caused by the circuit. This made the circuit unfit for our cause. It was then decided to use digital signal processing to find the reference point in the acoustic signal. This first method used a comparator of the micro-controllers ADC. Without noise this method worked well enough to measure the signals frequency. But with just small amounts of noise like people talking the frequency could not be measured. Attempts were made to reduce the noise using a small banded filter but this did not help enough to make the measurements constant. The third attempt is were the project stands now and uses correlation to determine the time delay in the signal. Using correlation a distance measurement within the specifications was reached up to 1,5 meters.

10 Conclusion

The conclusion is divided into two subsections, Swarm communication and Relative localization. As described in project method the V-model was used to give the project structure. As part of the V-model specifications defined in every phase should be tested. For this the qualification document is created and will be included in the same folder as this document. However no actual testing could be done because there is no finished product to test.

10.1 Swarm communication

The following research questions for swarming communication were drafted:

- ”What software protocol should be used?”
- ”How do the units communicate within the swarm?”
- ”What is the minimal required communication speed?”
- ”What hardware is needed to implement the communication?”

During this project all of the questions were answered. A unit can communiaite with the swarming via the Central operating unit, which in turn communicates with the Swarmbee module that maintains the swarm network. The Swarmbee module is used to set up the swarm network. This uses a 2,4Ghz protocol. The software used determine the swarm population is self written and explained in the design section of this document. The minimal required speed is estimated around 8kbps, which is easily reached by the Swarmbee module. With all the questions answered the swarm communication is mostly done. Practice tests still need to determine whether or not the system works properly.

10.2 relative localization

The important research questions for the relative localization are:

- ”What is the definition of swarming?”
- ”How do robots in the swarm know their location?”

Swarming was first defined to determine the specifications for the relative localization. It was found that the angle measurement did not have to be real precise for the localization to be useful for the swarm. Algorithms were drafted to determine the location of other units using three microphones on predefines locations on the module itself. To implement actual localization, a reliable distance measurement first had to be implemented. A distance measurement was implemented that met the specifications to a distance of 1,5 meter. However due to many setbacks while trying to implement this, no time was left to implement a way to measure the relative angle.

11 Recommendations

The progress of this project is not as far as we had hoped for, and there are many things that can be improved to make the module as complete as possible. As far for the communication part, most of the functions are completed, except for a few. If there was more time we would have tried an implementation for the swarm to test if the modules would really work in a swarm, and then swarming behaviour could be confirmed. This is something that should be done in the next phase of this project. As said before the communication with the acoustic sensor is not established yet. This is a vital part for the module in order to make it work as a whole. This is definitely a must in a next phase, but it could not fit our timetable in this phase, and we had prioritize our tasks. The distance determination of the Swarmbee module is not tested thoroughly and is critical if you want a reliable Swarm. If the distances are not correct, it could have sever consequences. There are still some tasks that need to be done. The tasks just mentioned should be done first in a next stage of this project.

We still recommend using acoustic signals to determine the distance. We found that analog solutions cause to many deviations in time to create a usable implementation. Digital signal processing seems the best way to deal with the problem. The properties of the signal that was to be received are already known. Because of this we propose the use of convolution (discussed in chapter 5.3). Convolution allows for noise-heavy signals to be detected and used.

There are a few points of interest to improve the code. These points might make the code quicker and more reliable. Two problems that were encountered with the current micro-controller, that is used for this project, are its limited memory and processing power. One method to overcome these obstacles is by adding up the values read from the ADC, this means not every read value needs its own space in the array. With the current ten measurements of 1250 samples each that would mean saving up to 16.875 bytes when using 16-bit integers in said array. Since the total data memory of the ATXMega128A3U is only 16KB this solution would save a lot of memory in the micro-controller. Using the previous solution also means the correlation-algorithm will get an average outcome from the ten measurements. A second point of interest to make the measurement more reliable while not slowing down the measuring process too much, is by using Paul Bourke's cross-correlation algorithm. But instead of using floating point calculations, which requires a lot of processing power, we recommend using fixed point calculations. The micro-controller can handle these calculations much better than floating point calculations, improving the speed drastically. Using this method it is assumed that the measurement will be more reliable. A determination has to be made what kind of precision is needed from the distance measurement to implement a angle measurement with a resolution of 45deg. We recommend using a simulation program like Mathlab to run the full algorithm using stored values from the ADC, to determine if the specifications can be met using this algorithm. Once this is proven the algorithm can be ported to the micro-controller. Another way to overcome the problem of the limited amount of memory is by connecting extra external memory, like an SD-card for example. The current micro-controller used has an external slot to connect an SD-card, if this can be configured the amount of data will not be a problem any more. The only thing that should be taken into account is that reading and writing to the SD-card might not be fast enough for all the variables used. When correlation does not meet the specifications the recommendation is trying to implement 'code 1' with a frequency based approach, but instead of using the analog comparator using the ADC this might make it less susceptible to noise.

12 References

References

[1] Wikipedia. Swarm robotica. [Online]. Available: https://en.wikipedia.org/wiki/Swarm_robotics

[2] *Swarm Intelligence and Its Applications in Swarm Robotics*, D.A. Aleksandar Jevtic, 1 2015, vertrouwlijk.

[3] A. D. Hanford. Computer simulations of progagation of sound on mars. [Online]. Available: http://acoustics.org/pressroom/httpdocs/151st/Hanford.html

[4] *Lidar on the Phoenix mission to Mars*, James Whiteway, 6 2008, vertrouwlijk.

[5] *Radio Frequency Time-of-Flight Distance Measurement for Low-Cost Wireless Sensor Localization*, University of California And San Diego, 3 2011.

[6] *Is RSSI a Reliable Parameter in Sensor Localization Algorithms – An Experimental Study*, State University of New York at Buffalo, 1 2010.

[7] *Distributed Range-Based Relative Localization of Robot Swarms*, Harvard University, 1 2014.

[8] *Analoge signaalbewerkings-techniek*, prof.dr.ir. J. Davidse, 5 1991.

[9] L. E. F. Jr., *Principles of electronic communication systems.* Avenue of the Americas, NY: McGraw-Hill, 2008.

[10] M. K. V. R. V. S. G. H. T. Braun, A. Kassler, "Multi-hop wireless networks."

[11] Wikipedia. Mesh networking. [Online]. Available: https://en.wikipedia.org/wiki/Mesh_networking

[12] *Position-Based Routing in Ad Hoc Networks*, University of Ottowa and Universidad Nacional Autonoma de Mexico, 6 2002.