Swarming Module

Project document

Version 1.1

June 9, 2016

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

List of Figures

List of Tables

Contents

1 Abstract

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"[**?**] . 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"[1].

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[1]. 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[**?**]. The cheapest sensor that would work on Mars would be a lidar[**?**]. 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[1]. 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[1].
- 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[1].
- Limited capabilities - The robots in a swarm should be relatively incapable or inefficient on their own with respect to the task at hand[1].
- 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[1].
- 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[1].

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[1].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 de 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 de 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 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.

5.1.1 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)[**?**], and Received signal strength (RSSI)[**?**]. 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.

5.2 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[**?**]. Because geometric functions are used to calculate the angle, there will always be two solutions to the equation (see figure 1 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 2.

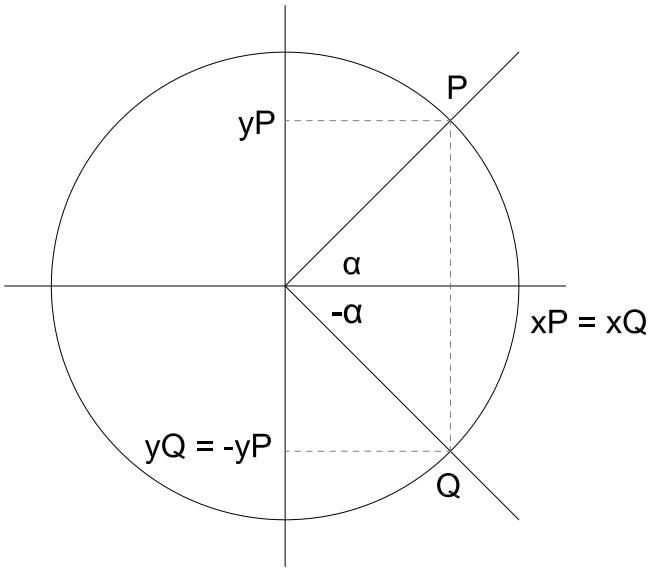

Figure 1: Unit circle where points P and Q are mirrored on the X-axes

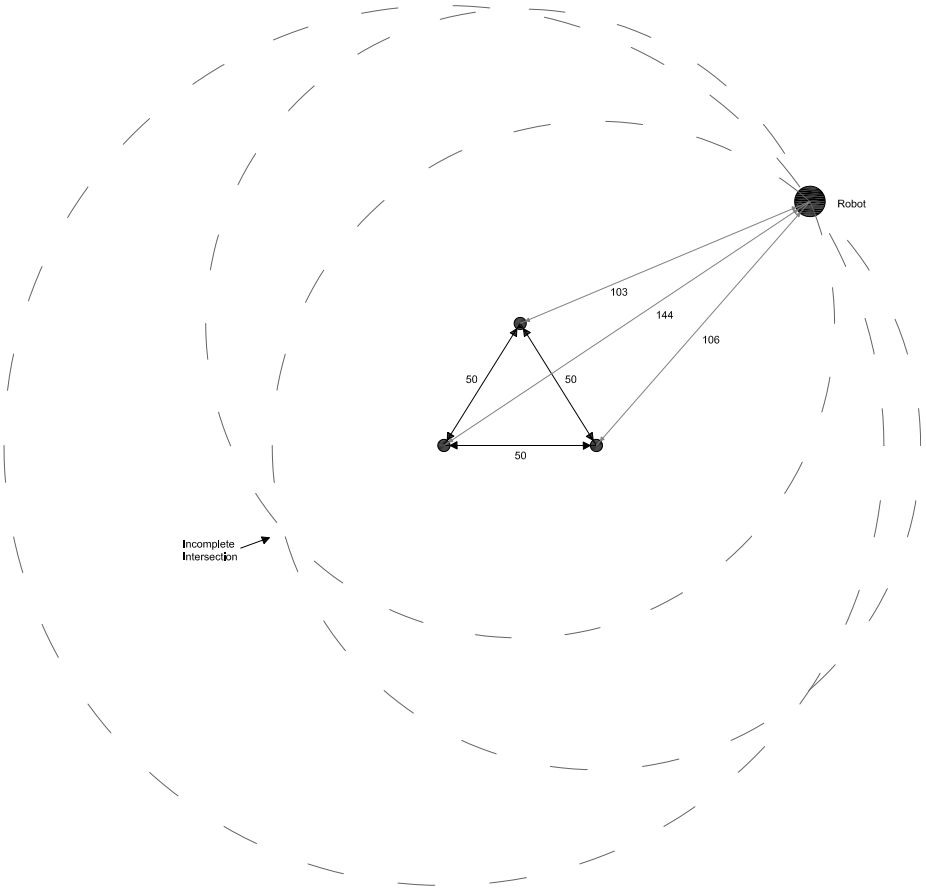

Figure 2: Angle determination using trigonometry

5.3 Demodulation methods

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. 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 de discussed in the next section.

5.3.1 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. To test the precision in time a test set-up was build. Two Xmega128a4u were used, one for the demodulator side and one for the modulator side see figure 3 . 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 could be identified by the demodulator.

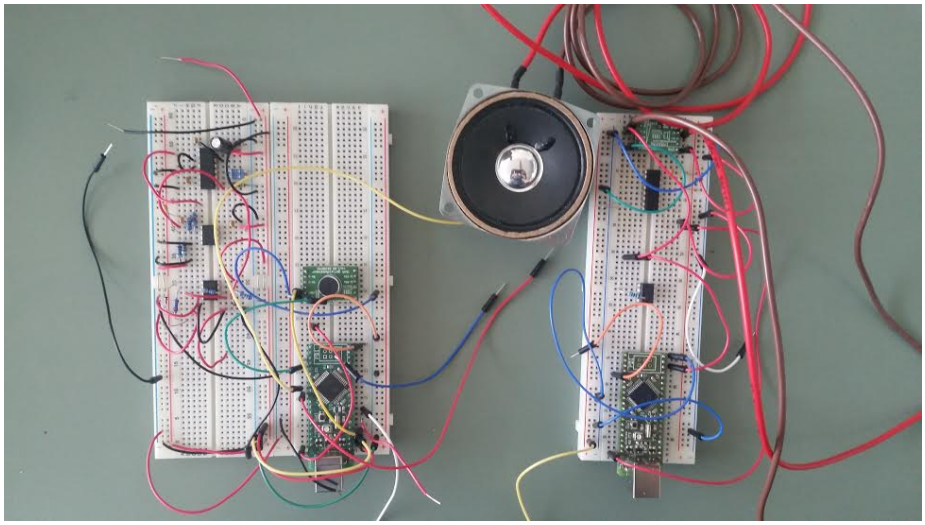

Figure 3: Test set-up, the right breadboard is de modulator with a speaker, the left breadboard is the demodulator with an microphone and the PLL

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 1024

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 1024. At close range the measured times seem the most stable. However some big deviations of around 500.000 clock pulses still exist. When these times are used to calculate the distance this would mean a deviation is distance of around 5 meters, which is unacceptable. The digital part of the demodulator is tested by connecting a frequency generator to it and observing the time deviation. The time deviation was at a maximum of 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 **??**[**?**].

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

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.

5.3.2 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 an microprocessor. The ADC of the Xmega contains an comparator. This can be used to detect how many times the incoming signal goes trough a certain threshold. From this information the frequency can be derived.

However after several tests and tweaks this method seems to be very unreliable in area's where there is some environmental noise. While testing this method we came to the conclusion that the results that were presented by the test-unit were to inconsistent. And therefore discarded this method.

The time delay measurement between the sent signal and the received signal gave in some circumstances promising results, however when there is just a little bit environmental noise the comparator of the Xmega picks up these sound waves disrupting the entire measurement process. This could be resolved by implementing a very narrow bandpass filter, however whereas a bandpass filter does not effect the frequency domain it does effect the time domain of a signal.

Also reflections play a huge part in the inconsistent measurements. We were able to measure up to 48 centimetres. Increasing the distance by a little bit resulted in very high values. Values that were not expected and had no relation with previous measurements, meaning that it was of no good use for a accurate distance determination.

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 sound 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.

```
#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();
```

```
45    stdout = &mystdout;
46
47    //timer TBV het meten van de frequentie
48    TCC0.CTRLA = TC_CLKSEL_DIV8_gc;          //
       prescaling 8
49    TCC0.PER = 0XFFFF;                  // maximal value
50
51    //timer TBV het meten van de overdrachtstijd
52    TCD0.CNT = 0;                    // Timer/counter D0
       value is 0
53    TCD0.CTRLA = TC_CLKSEL_DIV1_gc;          //
       prescaling 1
54    TCD0.PER = 0xffff;                  // maximal value
55    TCD0.INTCTRLA = TC_OVFINTLVL_HI_gc;        // high
       level overflow interrupt
56
57    //interrupt TBV het resetten van de timers bij
       startup
58    PORTB.INT0MASK = PIN0_bm | PIN1_bm;
59    PORTB.PIN0CTRL = PORT_ISC_RISING_gc;
60    PORTB.INTCTRL = PORT_INT0LVL_HI_gc;
61
62    PMIC.CTRL |= PMIC_LOLVLEN_bm | PMIC_HILVLEN_bm;
63    sei();
64
65    PORTC_DIRSET = PIN0_bm;
66    while (1) {
67      if (period > 5000 && measure >= 1000  &&
       start_measure == 1 /*&& period > 850*/){
68        start_measure = 0;
69        time = TCD0.CNT+(overflow*0xffff);
70        PORTC_OUTTGL = PIN0_bm;
71        printf("Time: %lu\n measure: %d\nmeasure2: %d\n"
       , time, measure, measure2);
72        cli();
73        measure = 0;
74        measure2 = 0;
75        sei();
76      }
77    }
78  }
```

Code 1: Implementation distance measurement using the Xmega comparator

5.4 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 was the point where the highest amplitude was achieved and the minimum distance of three meters was easily met.

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: [**?**] [**?**] [**?**] [**?**].

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.[1]
- 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 2 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 4 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 4. 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 4. 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.

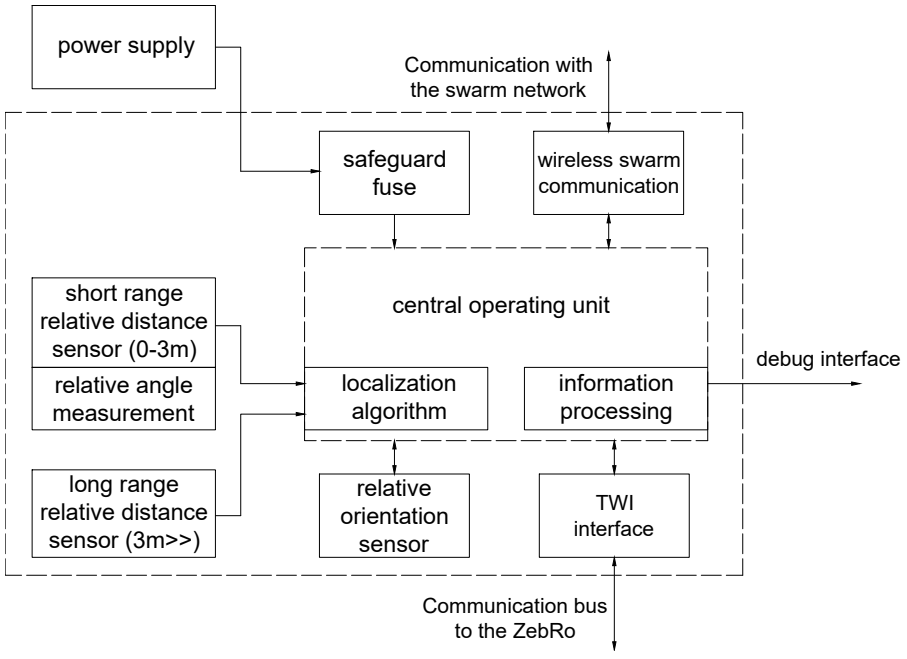

Figure 4: Overview of the swarm module design

7.2 Short range localisation sensor

The principle of the short range localisation sensor is displayed in figure 6 and 5. 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 **??** for a schematic representation of the modulator.

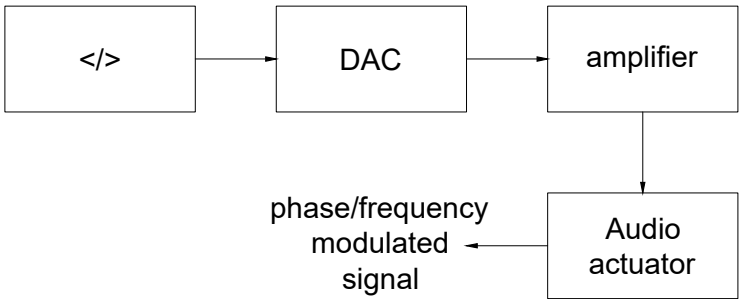

Figure 5: Modulator

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 6 for a schematic representation of the demodulator.

Table 6: Demodulation specifications

Module	Demodulator
Input	FSK Signal
Outputs	Reference point
Function	Demodulation, Reference point detection
Features	Noise insensitive, precision (time), Quick demodulation

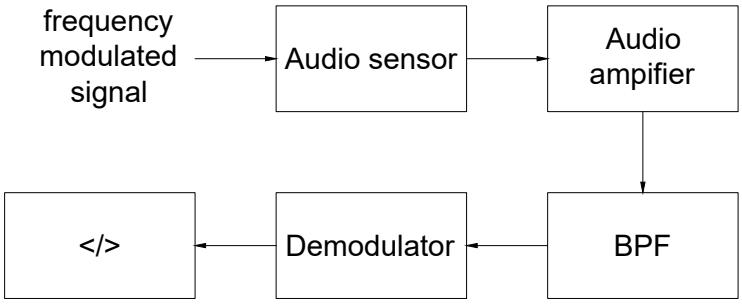

Figure 6: Demodulator

7.2.3 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.

7.2.4 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.

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 7.

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 de 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.

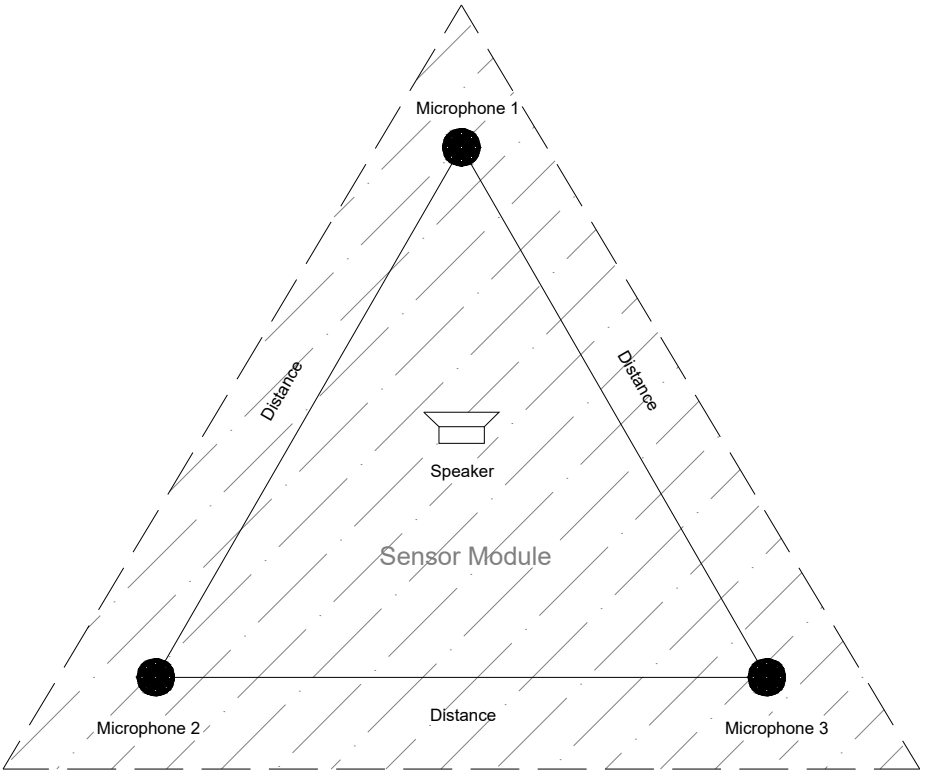

Figure 7: Localization Module

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 4 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 for a total of 9600 by several stages of amplifiers, so that the signal clips on the supply voltage. This ruins the shape of the signal. However we're only interested in the frequency which will remain the same. After that the signal is bandpass filtered, to get rid of most of the noise. After this the signal is fed into the ADC of the Xmega128a4u. The modulator and demodulator will now be discussed in detail.

8.1.2 Demodulator

The DAC of the Xmega128a4u is used to create a 4 kHz sine wave signal. The signal will be sent out for a pre-defined period of time (for example 10 periods) and then the signal will pause for a pre-defined period and continue for the same pre-defined period of time sent before (10 periods) see figure[**?**]. The pause will be the reference point within the signal. This reference point is to eliminate various uncertainties like the time it takes for the speaker to start resonating and to increase reliability. Because the chosen frequency lies within the speech range, the measurement can be easily disturbed by regular sounds. A reference point within this signal is difficult, maybe even impossible to reproduce in the form of environmental noise.

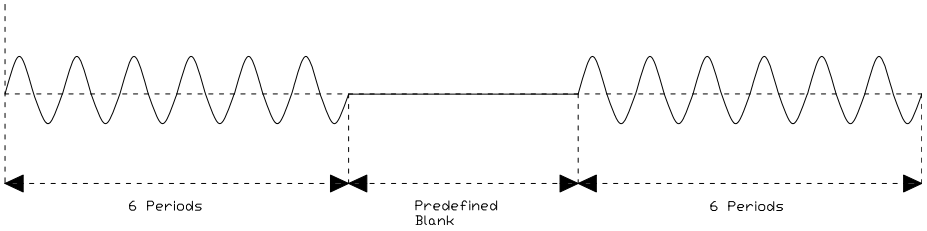

Figure 8: Example sent signal

De-modulating this signal will be done with the following block diagram see figure[**?**]. The sound is being captured by a microphone and this signal is being filtered with a bandpass filter to narrow the frequency range to limit the noise. The filtered signal is being amplified with a instrumentation amplifier to eliminate the common DC offset of the microphone. The analog comparator is a feature built in the xmega128a4u microcontroller it detects every rising edges of the signal. Measuring the time between every rising edge translates with a simple formula into a frequency.

By counting every rising edge it is also possible to track every period of the received signal. Which enables us to implement a reference point within the signal using the method mentioned above increasing reliability.

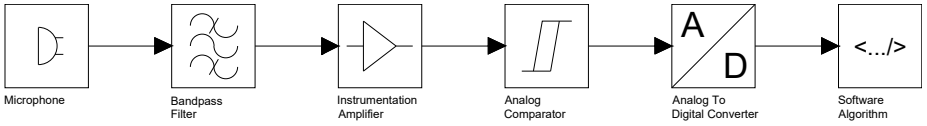

Figure 9: Demodulator block diagram

8.1.3 Modulator

The signal shown in figure[**?**] needs to be modulated to then be sent through a speaker. The signal is being generated within a microcontroller and sent to the speaker through 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 [**?**]. The "blank" spaces in the signal is done by stop sending for period equal to the pre-defined time for a blank.

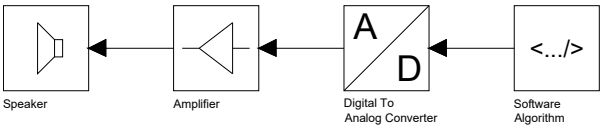

Figure 10: Modulator block diagram

8.1.4 Distance measurement

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. When it arrives it stops it's timer and takes off some pre-defined corrections based on tests done earlier. 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 **??**.

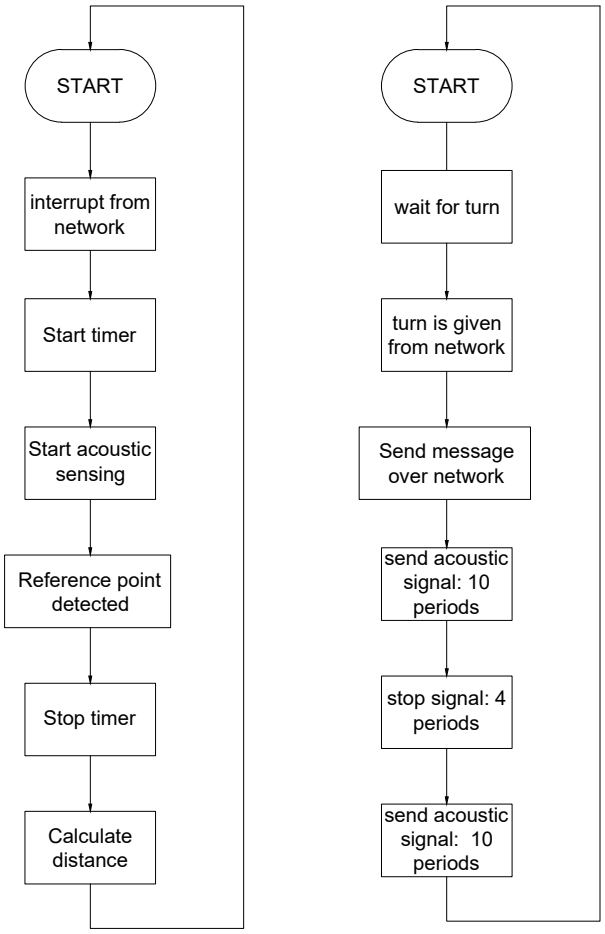

Figure 11: State diagrams for the distance measurement. The diagram on the right is for the receiving side, and the left diagram is for the sending side.

8.1.5 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 this is done. The following algorithm shows pseudo-code how the angle and relative location is derived from the three measured distances.

```
1 ISR(reference call){
2   start timer;
3 }
4 ISR(mic 1){
5   Time 1:= Timer;
6   recieved:=recieved+1;
7 }
8 ISR(mic 2){
```

```
9   Time 2:= Timer;
10  recieved:=recieved+1;
11 }
12 ISR(mic 3){
13  Time 3:= Timer;
14  recieved:=recieved+1;
15 }
16
17 While recieved = 3{
18  time_1:= time_1-overhead_time;  //remove overhead
     time
19  time_2:= time_2-overhead_time;  //remove overhead
     time
20  time_3:= time_3-overhead_time;  //remove overhead
     time
21  length_1:=time_1*speed_of_sound/clock_freq
22  length_2:=time_2*speed_of_sound/clock_freq
23  length_3:=time_3*speed_of_sound/clock_freq
24  angle_1:=acos((length_1^2+mic_distance^2-length_2^2)
     /(2*length_1*mic_distance));
25  angle_2:=acos((length_2^2+mic_distance^2-length_3^2)
     /(2*length_2*mic_distance));
26  angle_3:=acos((length_3^2+mic_distance^2-length_1^2)
     /(2*length_3*mic_distance));
27  read compas_angle;
28  angle_1:=angle_1+compas_angle;
29  angle_2:=angle_2+compas_angle;
30  angle_3:=angle_3+compas_angle;
31  x_1:=sin(angle_1)*length_1;
32  x_2:=sin(angle_2)*length_2;
33  x_3:=sin(angle_3)*length_3;
34  y_1:=cos(angle_1)*length_1;
35  y_2:=cos(angle_2)*length_2;
36  y_3:=cos(angle_3)*length_3;
37  x_center=x_1+center_distance_x;
38  y_center:=Y_1+center_distance_y;
39  recieved:=0;
40 }
```

Code 2: Algorithm to derive the relative location

8.1.6 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

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

swarming module. It also has a API with predefined functions which can be used to set up the wireless network.

8.1.7 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 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.

8.1.8 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.

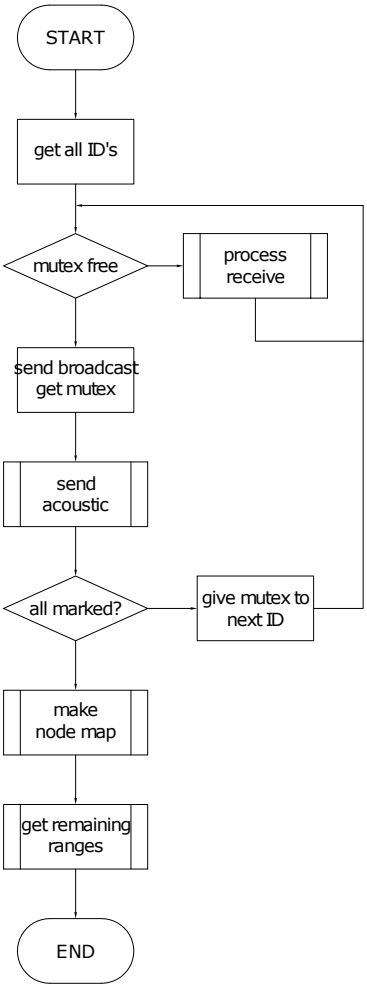

Figure 12: main algorithm for updating the swarm population including range requests

8.1.9 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 12. 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:

```
1 *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:

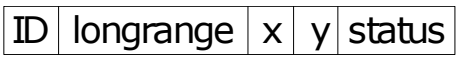

Figure 13: One population list frame

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.

```
1  void fillpopulationlist(char *message){
2    char *messagePointer;
3    char srcid[12];
4    char destid[12];
5    int distance[6];
6    uint8_t count, c;
7    c = 0;
8    count = 0;
9
10   memset(srcid, EOS, strlen(srcid));
11   messagePointer = message;
12
13   command(sbiv 1000);   // set broadcast interval
```

```
14    //Rx1: *RRN: AA1122334455,1F3CFF322133
        ,0,001843,04,-56
15
16    while(*messagePointer != ','){
17      //uart_putc(&uartC1, *messagePointer);
18      switch(c)
19      {
20      case 0:   //first part of broadcast is source ID
21        srcid[count] = *messagePointer;
22      case 1:   //second part is destination ID
23        destid[count] = *messagePointer;
24      case 3:   //third part is range
25        distance[count] = *messagePointer;
26        c = 0;
27      }
28      *messagePointer++;
29      count++;
30    }
31
32    if(*messagePointer == ','){c++;};
33    //next part of the broadcast message
34    if(Srcid =! myid){
35      Populationlist(srcid, longrange);
36    }
37    else if(destid =! myid){
38      Populationlist(destid, longrange);
39    }
40    //population list is filled
41  }
```

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.

8.1.10 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 13.1. since only one member can be the sending module the rest of the swarm will enter the "process receive" algorithm, figure 13.2. 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 (**??**) 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.

```
1  void order(){
```

```
2     if(all ids marked as done sending){
3       //END     //the entire populationlist is completed
4     }
5     else if(myid == marked as done sending || myid =!
       lowest, unmarked, id){
6         receiver();
7       }else{
8         sender();
9       }
10 }
```

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 12, 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.

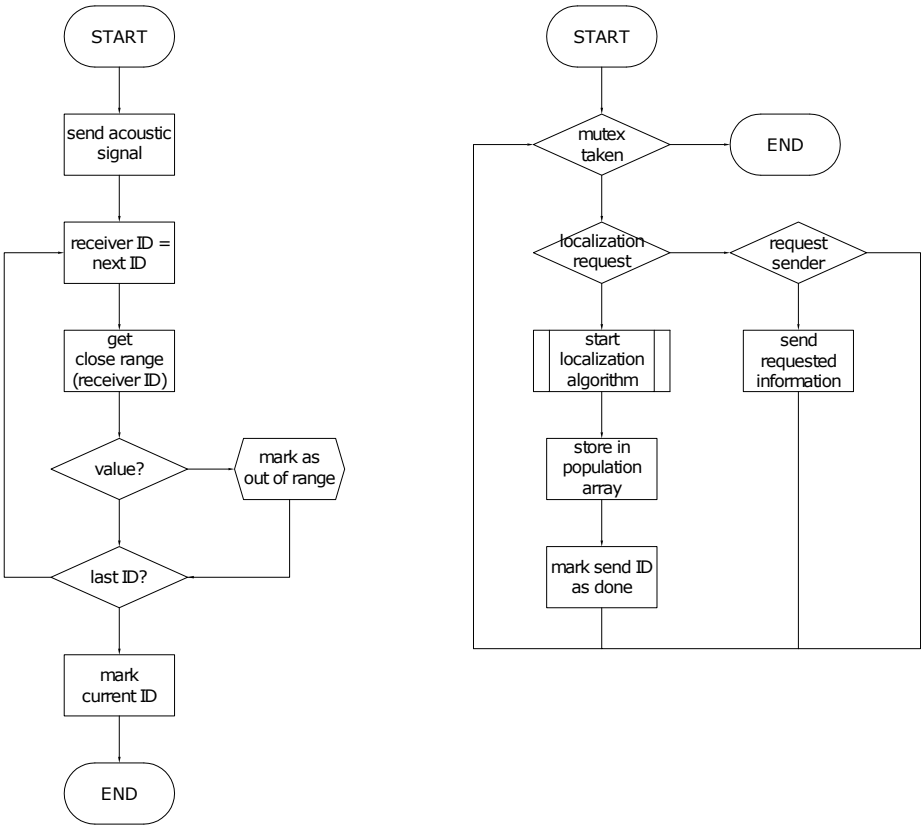

Figure 14: 1:"send acoustic" when the swarm module is in sensing state 2:"process receive", when the swarm module is in receiving state

```
1  void sender(){
2    char[12] recid;
3    char[RDATLEN] rrangedata;
4
5    PORTA.DIRSET = PIN0_bm;
6    //signal other microcontroller to send acoustic
       signal
7    Tx1: ("bdat 0 BDATLEN %c\r\n",myid);
8    PORTA.DIRCLR = PIN0_bm;
9    //broadcast start of acoustic sending
10
11   //Rx2: sends signal when sendingprocess is finished
12   if (Rx2 == "finished"){
13     command(gdat);      //read out transmitted data
14     //Rx1: "=RDATLEN,recid,rrangedata"
15     if(RDATLEN =! 0){
16       populationlist(recid,0,x,y);
17       //fills populationlist frame
18     }
```

```
    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.

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.

```
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 3: Validates the integrity of the received message.

```
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
31      //printf("SDAT\n");
32    }else if(strcmp(command, "AIR")  == 0){ // AIR
      Notification Message
33      //printf("AIR\n");
34    }else{
35      DebugPrint("No command\r\n");
36    }
37  }
```

Code 4: Determines the meaning of the message.

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

```
1  char * TranslateMessage (void){
2    char value[128];
3
4    memset(globalMessage, EOS, strlen(globalMessage));
5    memset(value, EOS, strlen(value));
6
7    value[0] = uart_getc(&uartC0);
8    strcpy(globalMessage, value);
9    while (value[0] != CR){
10     if (value[0] != CR){
11
12       strcat(globalMessage, value);
13     }
14     value[0] = uart_getc(&uartC0);
15   }
16   return globalMessage;
17 }
```

Code 5: Translates the received message. Converts characters to a single string

8.1.11 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).

```
1  typedef struct node {
2      char* data [DATASIZE];
3      struct node * next;
```

```
4 }node_t;
```

Code 6: 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.

```
1  void appendList (node_t *listHead, char* data[DATASIZE
       ]){
2      node_t *current = listHead;
3      while ((*current).next != NULL){
4          current = (*current).next;
5      }
6      (*current).next = (struct node *) malloc(sizeof(
       node_t));
7      if (new_node == NULL){
8          DebugPrint("No memory");
9      }
10     for (int item = 0; item < DATASIZE; ++item){
11         (*current).(*next).data[item] = data[item];
```

Code 7: Inserts a data array row to the end of the list

```
1  void insertList (node_t ** listHead, char*
       dataInternal [DATASIZE]){
2      node_t * new_node;
3      new_node = (struct node *) malloc(sizeof(node_t));
4      if (new_node == NULL){
5          DebugPrint("No memory");
6      }
7
```

```
8       for (int item = 0; item < DATASIZE; ++item){
9           (*new_node).data[item] = dataInternal[item];
10      }
11      (*new_node).next = *listHead;
12      *listHead = new_node;
13  }
```

Code 8: 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.

```
1   char* popList (node_t ** head){
2       node_t * next_node = NULL;
3
4       if (*head == NULL) {
5           return NULL;
6       }
7       next_node = (*head)->next;
8       for (int item = 0; item < DATASIZE; ++item){
9           returnArray[item] = (*head) -> data[item];
10      }
11
12      free(*head);
13      *head = next_node;
14
15      return returnArray;
```

Code 9: Reads a data array row at the beginning of the list and removes it afterwards

```
1   char* popListByNumber(node_t ** head, int indexNumber)
        {
2       node_t *current = *head, *temp_node = NULL;
3
4       if (indexNumber == 0){
5           return pop(head);
6       }
7
8       for (int item = 0; item < indexNumber-1; ++item){
9           if (current -> next == NULL){
10              return NULL;
11          }
12          current = (*current).next;
13      }
14
15      temp_node = (*current).next;
16
17      for (int item = 0; item < DATASIZE; ++item){
```

```
18          returnArray[item] = (*temp_node) -> data[item
        ];
19      }
20
21      (*current).next = (*temp_node).next;
22      free(temp_node);
23
24      return returnArray;
25  }
```

Code 10: Reads a data array row at the given index number and removes it afterwards

```
1   char* popListByValue(node_t ** head, char* value, int
        sizeOfList){
2       node_t *current = *head,*temp_node = NULL;
3
4       if ((*current).data[FIRSTDATAITEM] == value){
5           return pop(head);
6       }
7
8       for (int item = 1; item < sizeOfList; ++item){
9           temp_node = (*current).next;
10          printf("Item %d, data [%s]\n", item, (*
        temp_node).data[FIRSTDATAITEM]);
11          if((*temp_node).data[FIRSTDATAITEM] == value){
12              for (int itemb = 0; itemb < DATASIZE; ++
        itemb){
13                  returnArray[itemb] = *(*temp_node).
        data[itemb];
14              }
15              (*current).next = (*temp_node).next;
16              free(temp_node);
17
18              return returnArray;
19          }
20          current = current -> next;
21      }
22      return 0;
23  }
```

Code 11: Reads a data array row with the given first value of the current row and removes it afterwards

```
1   int sizeOfList(node_t *listHead){
2       int sizeOfList = 1;
3       node_t * current = listHead;
4
5       while ((*current).next != NULL){
6           current = (*current).next;
```

```
7           sizeOfList++;
8       }
9       return sizeOfList;
10  }
```

Code 12: 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.

```
1  Populationlist
2  Nr.      Data[0]  Data[1]  Data[2]  Data[3]  Data[4]
3  0        xxx      xxx      xxx      xxx      xxx
4  1        xxx      xxx      xxx      xxx      xxx
5  2        xxx      xxx      xxx      xxx      xxx
6  3        xxx      xxx      xxx      xxx      xxx
7  4        xxx      xxx      xxx      xxx      xxx
```

Code 13: Example of the population list with x's at the place of swarm data.

8.1.12 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 de communication is not defined yet, but needs to be fast enough to handle all the messages. The Swarmbee LE module will be used for the wireless communication. Its 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.

9 Results and Discussion

10 Conclusion

11 Recommendations

Table 10: 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 $\sim$ 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 $\sim$ 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).	

12 Bibliografie

References

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