University of Southern Denmark

Bachelor Project

Robot Technology Spring 2016

Investigating bio-inspired object avoidance in a swarm of mobile robots

Niels Alexander Hvid
120293
nhvid13@student.sdu.dk

Rasmus Karnøe Stagsted
240394
rasta13@student.sdu.dk

Supervisors: Leon Bonde Larsen & John Hallam

Project period: February 1st, 2016 - June 1st, 2016

Preface

We would like to thank Tórur Andreassen for lending us, and helping with the setup of the pre-amplifiers and the USB-DUX-Fast which is used for analog-to-digital conversion of the microphones on the robots.

Abstract

Flocking behaviors seen in nature is the behavior elicited by a group of animals moving together. This behavior has some interesting aspects. For this reason, an implementation of the flocking behavior on mobile robots is attempted. This includes: figuring out which sensors can be implemented to obtain the necessary data; analyzing the requirements for the system; developing the robot; developing the software necessary to make the robots elicit the flocking behavior.

A simulation of the flocking algorithm is developed to get a feel of the performance as noise is introduced.

A small robot is developed that is able to obtain the necessary information required to elicit the flocking behavior. The position of other robots is obtained using four microphones and a multilateration algorithm. The microphones are sampled using the USB-DUX-Fast which communicates with the Raspberry Pi 3 through USB. The USB-DUX-Fast is currently not available in a form factor that allows mounting on small robots of this size, but development is in progress.

Software to interface with sensors and actuators and algorithms to replicate the flocking behavior seen in nature is developed and implemented in a pipe running on Linux.

Contents

1 Introduction

Seeing an almost fluid like swarm of birds flying over a field, forming new and impressive shapes almost every second, or a school of fish floating through the ocean as a giant bubble are some of natures many delights. This flocking behavior emerges when a group of animals move together as a flock where each individual appear to be part of a bigger unity.

Fish typically use this type of behavior as a defense against predation. It minimizes their risk of getting found by predators by more than a thousandth, and even if found, the chance of getting eaten is smaller. [4] Migratory birds utilize streamlines to reduce drag and increase efficiency when migrating, allowing the birds to fly further. [7]

In order for animals to elicit this behavior, they need to know the distance and heading of their close neighbors. Most animals use their eyes to obtain this information. However most fish use their lateral line to detect their nearest neighbors. [4] Since only the nearest neighbours affects behavior, complexity does not increase as the number of animals in the flock increases.

The behavior seen in a flock of birds, or a school of fish is surprisingly simple to implement. The flocking algorithm is said to follow just 3 basic rules: [6]

Alignment: Align with the neighbors, to attain a common heading.
Cohesion: Keep a short distance to the neighbors by steering towards their center of mass. (Long range attraction)
Separation: Separate from nearby neighbors to avoid collision. (Short range repulsion)

Implementing this style of behavior on mobile robots is interesting because the system involves no central controller. This means that time complexity does not increase with the number of robots added to the system. The algorithm could be used in systems where multiple robots work together to solve a task. In drones a collision is often fatal and implementing a flocking algorithm could reduce the risk of collisions for drones working together.

The goal of this project is to develop a platform consisting of a mobile robot with the necessary algorithms to elicit a flocking behavior. The robot will need to know the position and heading of nearby robots, therefore sensors capable of obtaining this information must be installed on the robot. Furthermore it is strived to develop a simulator as it makes it easier to develop and test the algorithms used for flocking.

2 Problem analysis

2.1 Flocking behavior

In the animal kingdom, flocking is said to follow 3 rules: Alignment, Cohesion and Separation. Since animals need to know the position and heading of their close neighbours to elicit this behavior, the robot will need sensors capable of obtaining the same information. Furthermore the robot also needs some actuators to move accordingly.

2.2 The platform

A platform is to be developed that features a CPU to run the algorithms and interface with the installed sensors and actuators. The platform must have some mounts for actuators, sensors, battery and circuit board(s). To make debugging easier, the platform should also feature LEDs for debugging. Mounting for the LEDs is also needed.

The platform should have differential drive to allow sharp turns as this makes it easier to avoid collision.

Since multiple platforms has to be purchased, keeping the price low is preferred.

2.3 The electronics

The electronics are divided in four parts: the sensors, the actuators, the Single Board Computer (SBC), and the battery.

Since the robot will be used by students later on, the hardware should be robust and not be damaged by rough use. This also include the electronics which should be protected in case the motors stall or too much power is drawn in any other way.

2.3.1 The sensors

The sensors on the robot need to obtain information about the position and heading of close neighbours. Preferably 360 degrees around the robot.

Using a camera might be the obvious choice as many animals rely on visual cues to obtain the same information. The camera(s) would need to be mounted in a way that gives clear view ahead of the robot, furthermore, since the robots need to know the position of each other, it is not enough with only 70-90 degrees field of view. The leading robot also needs to know at least some of its neighbours, therefore at least 180 degree field of view is necessary, preferably much more.

This can be solved in many different ways: One way is with a fish eye lens, this is a simple solution since it doesn't involve any moving parts in the sensory system, the disadvantage is the distorted image that may make it difficult to process. Another solution could be to have multiple cameras mounted on each robot, this does not distort the image in any way, but it may be difficult to handle the vast amount of data produced by two cameras. Mounting the camera on a servo motor, or mounting a mirror on a servo motor is also a viable solution, however, this is a complicated setup and a moving robot with a rotating camera trying to locate the position of other moving robots is a daunting challenge. Lastly it is possible to mount a spherical mirror above the camera, with this setup, the robot has large field of view. The distance to other robots would correlate with the distance from the center of the mirror. Any setup with mirrors mounted above the camera introduces the problem of occlusion, the mirror needs some structure to hold it in place, naturally this structure will partially block the field of view of the camera.

A simple sketch of possible camera setups can be seen in figure 1.

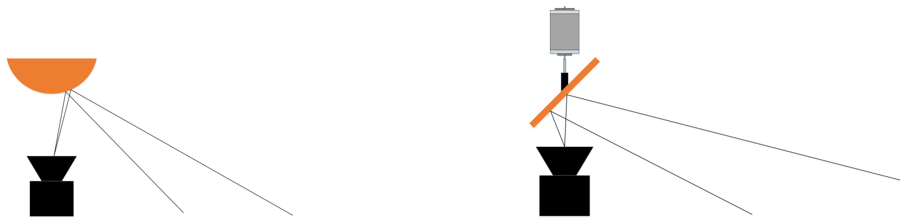
Figure 1: Camera with spherical and rotating mirror.

Using a camera to detect other robots also require the robots to have markers like LEDs mounted to make it easier to separate the robots from obstacles in the environment. Different colored LEDs could be used for the front and back to get information about the heading of other robots. However, this also produces problems with partially occluded robots. Much image processing would need to be done on each robot to get this information. Since image processing is computationally expensive and in order to get 360 degrees (or close to 360) peripheral vision would require either multiple cameras, rotating cameras or mirrors all of which unnecessarily increases the complexity of the project, it was decided to abandon the idea of cameras as sensoring mechanism.
Many sensors such as ultrasound and laser range scanners all share the problem that they can not easily distinguish between other robots and obstacles in the environment, laser range scanners in particular also comes at a high price, and the idea of basing a swarm of many robots off an expensive sensor is not appealing. Microphones on the other hand has the nice property that they record sound 360 degrees around them without any com-

plicated modifications and produce much less data, microphones are relatively cheap to buy, they are small and easy to mount. Each robot will need at least 3 microphones and a speaker or buzzer to produce a detectable sound. This setup is easier to install than a camera, it is less expensive, and requires less computational power. With 4 microphones mounted around the two axes of the robot (See figure 2), it is possible, with two second order equations, to calculate the position of the other robots (personal comunications with Leon Bonde Larsen). The orientation of the robots can be estimated with the change in position from the previous sound. For this reason, the decision to use sound to localize other robots was made, and will impact the specifications for the robot.

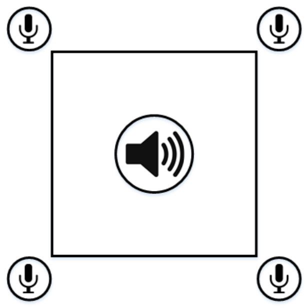

Figure 2: Sketch of the microphone and speaker placement on the robot.

2.3.2 The Actuators

The speed of the robot is not important, but being able to make sharp turns (turn on the spot) lowers the requirements for the range of the sensors as the robot will be able to avoid collision even with close encounters of other robots. Therefore a motor with high torque is preferred. The motors need to have encoders to measure the position and the speed of the motors.

2.3.3 Single Board Computer

Since the robot has to calculate the actuator actions based on some sensor input and some flocking rules, the robot needs a processing unit. For that purpose a single board computer (SBC) is a good solution. An SBC is a complete computer built on a small printed circuit board. A wide range of SBCs are available but not all SBCs fulfill the requirements.

The software on the SBC should be able to run fast enough to make the robots behave like a flock. The needed CPU clock speed can't be described mathematically since it depends on all the software running on the SBC, but it can be tested against a specific clock speed of the CPU. The memory requirement for the SBC is minimal since the only thing in need of 0long time storing (apart from the operating system) is the position of the robots within range which is limited since the sensor doesn't have infinite range.

The SBC needs four ADC channels or an interface to communicate with four external ADC channels to sample data from the four microphones. The ADC sample rate needs to be at least double the speakers frequency. Furthermore the SBC should have four PWM channels or an interface to communicate with an external PWM driver with at least four channels. These PWM channels are needed to regulate the power of the two motors.

2.3.4 The Battery

Powering the robot by battery simplifies the testing of the robots, no more than 10 minutes of run-time is necessary, but the longer they can run the better. The battery must be able to supply enough current to run all components on the robot. Any risk of overloading the battery must be dealt with.

2.4 Mathematics

Multilateration is the technique used to determine the position of an unknown emitter using the known position of multiple omnidirectional sensors. This can for example be used by radio receivers to locate the position of a radio transmitter, or by microphones to locate the position of a sound source. The transmitted signal is detected once on each sensor as the signal expands spherically (or circularly in 2D) and with constant propagation speed. With a synchronized clock on each sensor it is possible to measure the Time Difference Of Arrival (TDOA). Using the TDOA and the known position of each sensor it is possible to determine the position of the sound source.

One of the definitions of a hyperbola is that every point on the hyperbola has the same difference in distance to 2 points called the foci. The TDOA of a microphone pair represents exactly a difference in distance and thus forms a hyperbola. The sound source can be anywhere on this hyperbola. Therefore, forming a hyperbola from each microphone pair and looking for where they intersect tells us where the sound source is located (This approach was inspired by Leon Bonde Larsen). A hyperbola is centered around a point (h, k) each branch's closest point to the center is called the vertex, the vertex is a fixed distance a from the center. The foci of the hyperbola are "inside" each branch and are a fixed distance c from the center. b is the distance from the vertex to the asymptotes of the hyperbola. (see Figure 3).

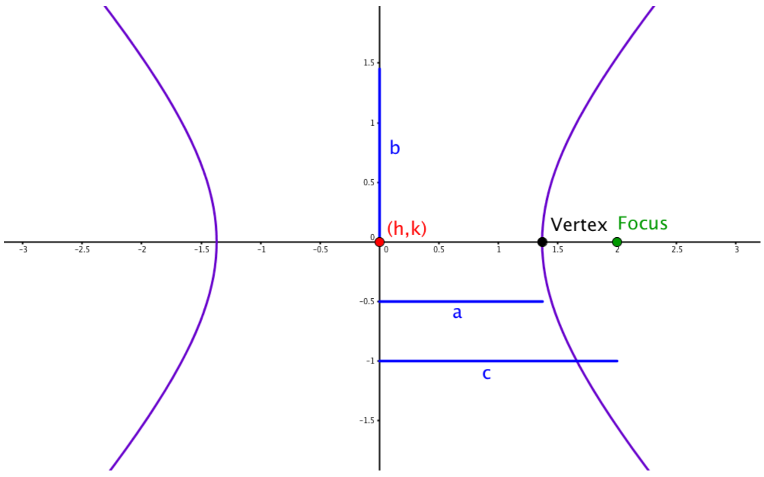

Figure 3: Hyperbola with the distances a, b and c as well as the center, vertex and foci.

The relationship between a, b and c (the proof of which is beyond the scope of this report) is:

$$b^2 = c^2 - a^2 \tag{1}$$

Which is identical to the Pythagorean theorem, however it is worth to note that this is not the Pythagorean theorem.

The equation for a hyperbola with a horizontal transverse axis (like the one shown in Figure 3) is:

$$\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1 \tag{2}$$

Which can be rewritten as:

$$b^2(x-h)^2 - a^2(y-k^2 = a^2b^2 \tag{3}$$

In the case where the microphones are mounted in such a way that each pair of microphones have either h or k to be 0. The solution can be greatly simplified. (See figure 4)

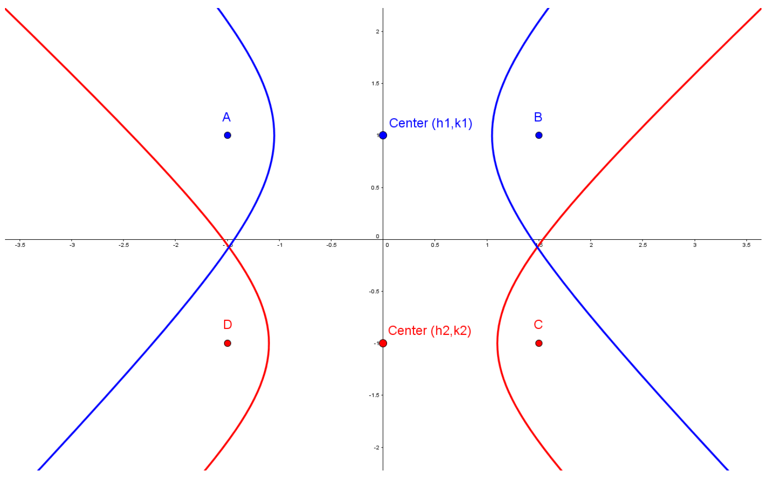

Figure 4: 4 microphones (A, B, C, D) with 2 hyperbolas where $h1 = h2 = 0$.

In the case where $h = 0$ we can express two equations for the hyperbolas as follows:

$$\begin{aligned} b_1^2x^2 - a_1^2y^2 + a_1^2k_1^2 - 2a_1^2k_1y - a_1^2b_1^2 = 0 \\ b_2^2x^2 - a_2^2y^2 + a_2^2k_2^2 - 2a_2^2k_2y - a_2^2b_2^2 = 0 \end{aligned} \tag{4}$$

multiplying the first equation with $1/b_1^2$ and the second equation with $1/b_2^2$ yields:

$$\begin{aligned} x^2 - \frac{a_1^2}{b_1^2}\cdot y^2 + \frac{a_1^2}{b_1^2}\cdot k_1^2 + \frac{-2a_1^2k_1}{b_1^2}\cdot y - a_1^2 = 0 \\ x^2 - \frac{a_2^2}{b_2^2}\cdot y^2 + \frac{a_2^2}{b_2^2}\cdot k_2^2 + \frac{-2a_2^2k_2}{b_2^2}\cdot y - a_2^2 = 0 \end{aligned} \tag{5}$$

Subtracting the second equation from the first equation cancels out all terms containing x, leaving a second order equation which can be solved to find the Y coordinate of the sound source. The final equation looks as follows:

$$\left(\frac{a_2^2}{b_2^2} - \frac{a_1^2}{b_1^2}\right)y^2 + \left(\frac{2a_1^2k_1}{b_1^2} - \frac{2a_2^2k_2}{b_2^2}\right)y + \left(\frac{a_2^2k_2^2}{b_2^2}\right) + a_2^2 - \left(\frac{a_1^2k_1^2}{b_1^2}\right) - a_1^2 = 0 \tag{6}$$

The same procedure is done with the other microphone pair in which $k = 0$ The resulting second order equation looks as follows:

$$\left(\frac{a_2^2}{b_2^2} - \frac{a_1^2}{b_1^2}\right)x^2 + \left(\frac{2a_1^2k_1}{b_1^2} - \frac{2a_2^2k_2}{b_2^2}\right)x + \left(\frac{a_2^2k_2^2}{b_2^2}\right) + a_2^2 - \left(\frac{a_1^2k_1^2}{b_1^2}\right) - a_1^2 = 0 \tag{7}$$

Equation 6 and 7 respectively are used to find the X and Y coordinate of the source.

At this point, it is worth to note that the further away the source is (along the axis), the smaller the TDOA will be. This is because the angular difference is smaller. This also means that the system becomes more stable if the distance between the microphones increases. This maximizes the angular difference and makes the system less sensitive to inaccuracies.

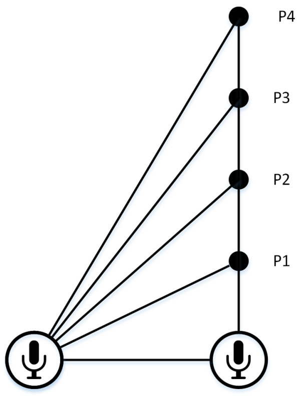

Figure 5: Two microphones and 4 points, P4 has much lower angular difference than P1.

As mentioned above, a low angular difference between the microphones means the system is less accurate, for this reason, sources located near the X and Y axis in figure 4 will also have a larger error given the same amount of noise compared to sources located far from the axis. This can be seen in Figure 6 which shows the error in red in a coordinate system around the robot. In the figure, each pixel represents a position in which a sound is emitted, the robot is located at the center. At each microphone a small error is added to simulate noise. The color of the pixel shows the magnitude of the error (darker red correlates with higher magnitude) of the multilateration of the calculation.

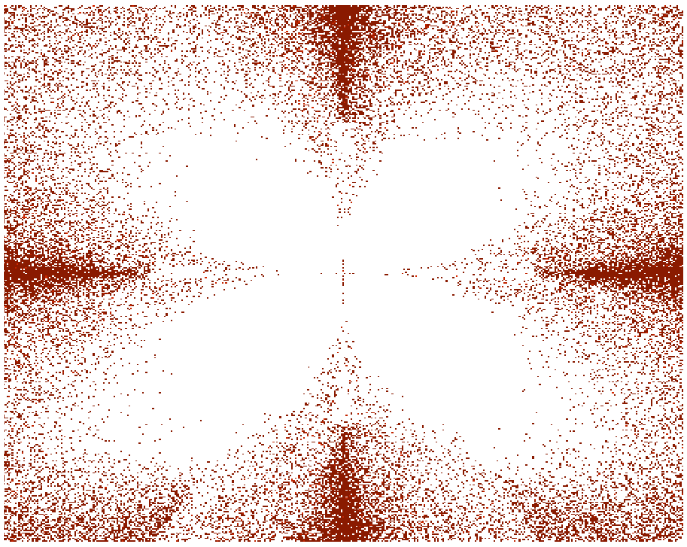

Figure 6: Error on the multilateration with noise, errors occur near the axis at smaller distances.

2.5 Software

Software to interface with the actuators and sensors of the robot needs to be developed. The software must run the flocking algorithms and follow the 3 basic rules of flocking.

Making the software modular is an advantage as this makes it much easier to develop small simple and well defined programs. Reusing part of the code is also much easier. The small programs should communicate through a pipe using standard input/output. Which makes it easy to test the individual programs since the output is known by a given input.

Another advantage by having the programs run in a pipe is, it becomes much easier to run simulations. Since only the first and last program in the pipe interfaces with the actual robot, replacing these two programs with simulations makes it easy to test all the other programs. This greatly reduces the debugging and development time.

The advantages of simulating the robots instead of programming directly to the physical robot, is; no need for having a robot for testing; no need for power management; a simulation can run much faster than real time; a simulation can ignore physical difficulties. Furthermore a simulation can be realistic enough to be implemented on a robot without a lot of work if the simulation is accounting for every important aspects of the physical world.

Apart from the robot, a simulation which simulates the flocking needs to be developed, this is used to see how the algorithm performs with inaccuracy in the localization.

2.6 Scalability

The system must be scalable, so the same hardware and the same software can run on many robots without changing the system. This means that the amount of data to be processed should be constant regardless of the number of robots. In the flocking behavior algorithm, each robot only needs to know the neighboring robots, so adding more robots to the system does not increase the computational complexity. The problem of scalability comes with the sensors, since the selected sensor is a microphone, the medium (air) is shared between all robots. This imposes the problem of congestion. As more robots are added to the system, the chance of data collision (Two robots making a sound at the same time) increases.

2.6.1 Sensor

At this time, it is difficult to calculate the scalability of the system because of the many unknown variables. Let us make some assumptions to get an estimate of the scalability.

If the sounds produced by the robots is very short (around $1ms$) then it does not make sense to first check if the medium is available (called Carrier Sense in the field of data communication). The communication method used by the robots share many similarities with the *pure Aloha* data communications method. To briefly summarize: Pure Aloha is the simple and elegant protocol in which, whenever a station has information to send to another station, it would do it immediately without any concern of collision, should an acknowledgement not be returned from the receiver, the station tries again after a random waiting time (called stepback time).

In this case, the robots also make their sound without concern of collision. However, in this case, the robot does not expect acknowledgement from the receiver. Instead, it is the receivers job to detect collision and in case of collision discard any information as this might be corrupt.

Let us assume the sound generated by the robots are $1ms$ in duration, and can be heard up to $1m$ away. This results in a total propagation time of $4ms$ (It takes roughly $3ms$ for sound to travel $1m$) Let us also assume that each robot needs to click two times every second for flocking behavior to function properly, then we can calculate the chance of a successful transmission with the following formula: [3]

$$Chance\ of\ success = \frac{G \cdot e^{-2 \cdot G}}{G} \cdot 100\% \tag{8}$$

In which G is the number of sounds generated by the system per frame length (in our case $4ms$) Obviously G is affected by both the number of robots, the number of times each robot must click every second, and how long the click lasts. The numerator is the throughput of the system, i.e. how effectively the medium is being used. The denominator represents the number of attempts to transmit a sound. If the assumptions are correct, the chance of successful transmission can be graphed as a function of the number of robots in the system, see figure 7

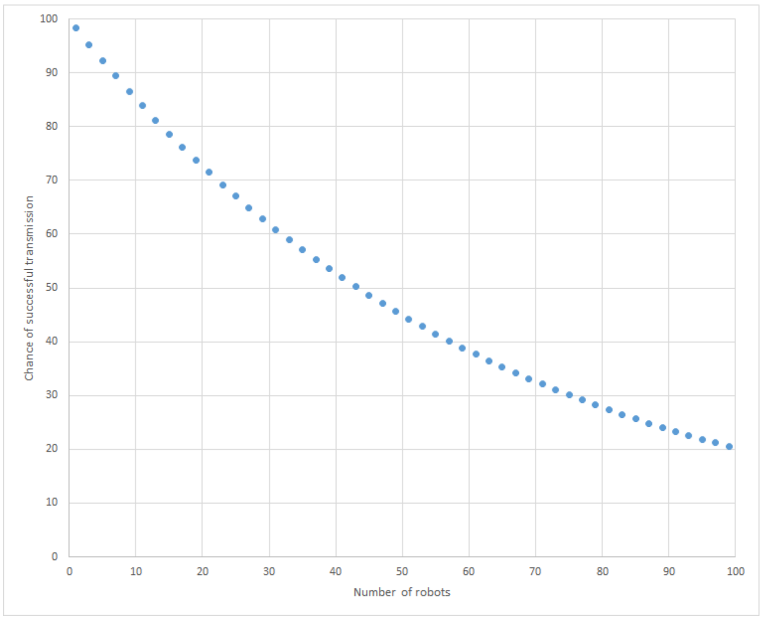

Figure 7: Chance of successfully transmitting a sound.

2.6.2 Programming

The user should be able to upload programs to all robots simultaneously regardless of the number of robots, to reduce uploading time. This should be implemented using the Wi-Fi connection to the SBC. Either a program or a script should be able to: connect to each robot; upload and compile the code and execute it from a remote client.

3 Requirement specification

With the analysis of the problem complete, the following requirements have been made and must be fulfilled when choosing solution for the project.

1. Platform
   (a) Mounts for motors, microphones, LEDs, battery and circuit boards.
   (b) Differential drive.
2. Electronics
   (a) Four microphones for localization.
   (b) Two motors for differential drive.
   (c) The critical components must be fused.
   (d) The battery must supply enough current to power the robot, and run at least 10 minutes.
3. Single Board Computer
   (a) Four PWM channels for the motors (either onboard or external).
   (b) ADC to sample the four microphones (either onboard or external).
   (c) Fast enough to run in near real-time.
4. Software
   (a) Module based programs.
   (b) Communication between programs through a pipe (standard input/output).
   (c) A simulation of the flocking algorithm to see how it performs given different inaccuracies.
   (d) Being able to program multiple robots at once.

4 Implementation

4.1 Hardware

Multiple robots have to be developed, therefore the complexity of building the robots was considered when designing them. The hardware development is divided into two major categories (platform and electronics) which are depending on each other.

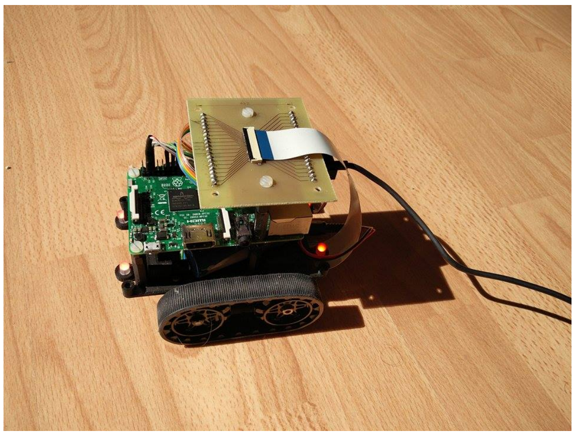

Figure 8: The robot platform

4.1.1 Platform

The Kilobot (shown in figure 9) is a robot designed specifically for swarming behaviors and with a unit price of just 15$ it is a very appealing platform to base the project on. The Kilobot has built-in infrared transmitters and receivers as well as ambilight sensors and LEDs. The infrared receiver is used to program the robots, this allows programming multiple robots at once, the infrared transmitter and receiver is also used for communication amongst the robots, the Kilobot can communicate with other robots up to $7cm$ away.

The kilobot moves $1cm/s$ and turns $45deg/s$ which means testing algorithms on these robots is going to take a lot of time. Furthermore the Kilobot has no method of getting the angle of the nearest neighbour, which is crucial for the flocking algorithm, also the Kilobots are very small, and mounting extra hardware will be very difficult. Therefore the Kilobot is discarded as an option for the platform.

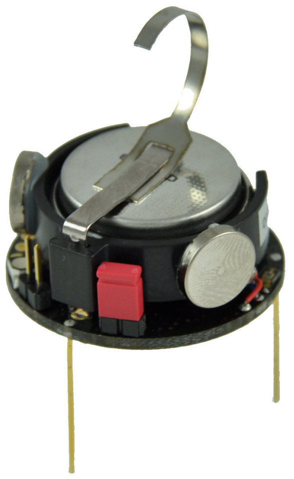

Figure 9: The Kilobot platform.

Another option is the Zumo platform by Pololu, the Zumo platform is a 100mm x 100mm robot platform originally designed for mini sumo competitions. The platform features caterpillar tracks which enables differential drive. The platform has built in mounts for DC-motors which complies with requirement 1a

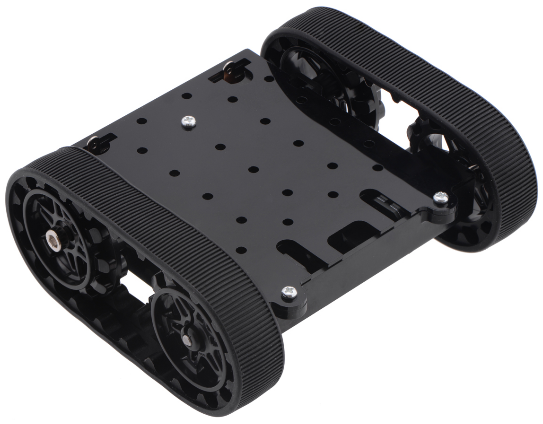

Figure 10: The Zumo platform.

For this project the Zumo robot platform (see figure 10) has been chosen. The dimensions of the Zumo chassis fits very well for a flock of 5-10 robots. The differentially controlled tracks, makes it very easy to turn. Each belt is controlled by a single DC motor. The mounts for motors on the Zumo platform fit a wide range of dc motors of different torques and gearings. Since the robots does not need to move fast, a motor with high torque is preferred to make sharp turns.

The price of the platform is 55$ which includes both the DC motors and encoders. The price is almost four times that of the Kilobot, but the Zumo robot is much better suited for this project.

The Zumo platform comes with an abundance of mounting holes on the top, however, the mounting holes does not fit the chosen SBC and lacks mounting for the microphones. For this reason, a new top plate was designed and 3D printed (see figure 11)

The chosen SBC (see 4.1.2.1) has mounting holes for mounting in embedded systems like this one, the top plate has mounting holes with the same dimensions which allows the SBC to be mounted with regular spacers.

The microphones used in this project (see 4.1.2.2) are very small. Simple holes in each corner easily holds the microphones firmly in place. This guarantees the same distance between each microphone on all the platforms. The microphones are placed at the corners of the robot to maximise the distance between them. And to reduce the shadowing effects by the chassis.

The top plate is designed to hold the chosen Li-ion battery in place. The battery fits nicely and wedges on the front makes it easy to slide a new battery in. The SBC is mounted on top with spacers meaning the battery is held securely in place.

Lastly mounts for the LEDs are also added.The LED mounts are placed near the corners of the robot.

All this complies with requirement 1a

The top plate has large holes in the middle which allows the wiring from both the microphone and the LEDs to get to the compartment underneath. The compartment was originally intended for AA batteries, but is now used for PCB (see 4.1.2.9).

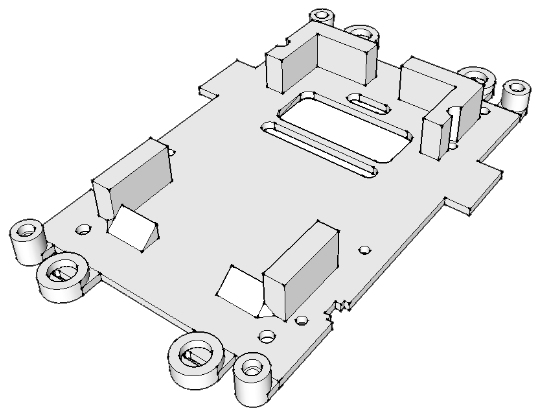

Figure 11: Model of the top plate.

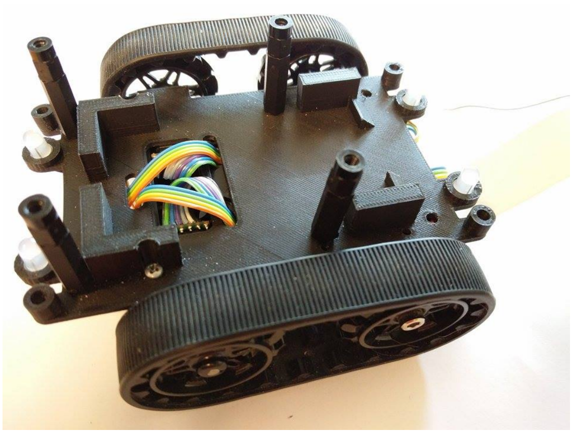

Figure 12: 3D printed plate on the robot.

4.1.2 Electronics

4.1.2.1 Single board computer

Name	CPU	Cores	Freq	Mem	PWM	ADC	Price
Banana Pi	ARM Cortex-A7	2	1GHz	1GB	1	4	40$
Raspberry Pi 2 B	ARM Cortex-A7	4	900MHz	1GB	1	0	40$
Raspberry Pi 3 B	ARM Cortex-A53	4	1.2GHz	1GB	1	0	40$

Table 1: A list of SBCs used in this project.

Some research has been made to find an SBC to fulfill the requirement specification. The Banana Pi was at first a good choice since it has four ADC channels and a $1Ghz$ dual core processor. But after a lot of work, there was still no communication to the ADC due to lack of documentation. Therefore the Raspberry Pi 2 were chosen which is in one of the most well documented series of SBCs. A month after project start, the new Raspberry pi 3 came out. Since the form factor is identical and almost all software to the Raspberry Pis is fully compatible with all versions of Raspberry Pis, we were able to switch out the Raspberry Pi 2 with the new Raspberry Pi 3. The reason for the switch to the Raspberry Pi 3 was to have built in Wi-Fi and a faster CPU without any change in price.

Raspberry Pi 3 has no ADC and only one PWM channel. Therefore a four channel PWM driver was made and an external ADC used.

4.1.2.2 Sensors

For localization of the robots a premade system made by Torúr Andreassen [2] is used. The setup includes four microphones, pre-amplifiers, filters and the USB-DUX-Fast ADC. This fulfills requirements 3b and 2a. The system is too big to be mounted on the robot.

But since a smaller form factor is under development it is waste of time to make our own system.

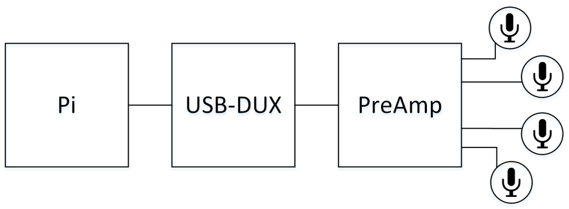

Figure 13: Simple block diagram of the sensor system.

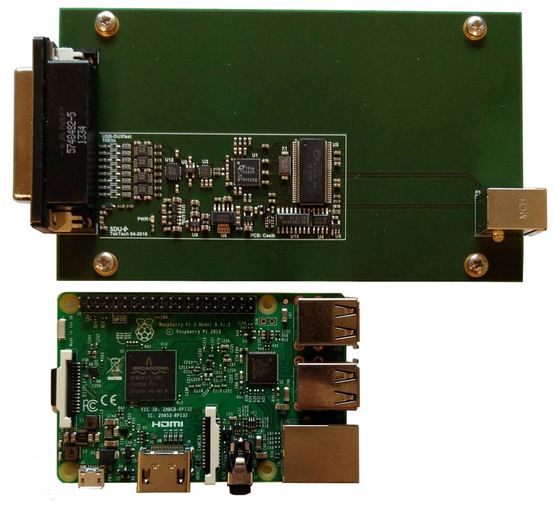

Figure 14: Picture of the small version of the USB-DUX-Fast and a Raspberry Pi 3 (The PCB size is the old size of the USB-DUX-Fast).

4.1.2.3 ADC

The USB-DUX-Fast is a 16-channel ADC with a 12-bit resolution. The USB-DUX-Fast samples with $78.125kHz$ at each channel which gives a total sample rate of $1.25MHz$. Since we only need four channels we are grouping four channels to each microphone to gain four times higher sample frequency. Then we end up with a sample rate of $312.5kHz$ at each microphone.

The SBC interfaces with the ADC through USB.

4.1.2.4 Sound generator

The idea was to use a piezo buzzer to generate a sound spike (impulse), which an onset detection algorithm could detect. The impulse needs a higher energy level than the noise in the environment but we were not able to make the buzzer loud enough to have a significantly higher energy level. Some data has been sampled with the sensor system (see section 4.1.2.2) of several snaps. The software is able to detect the spike on the prerecorded data, however a new way to generate spikes needs to be developed (see section 6).

4.1.2.5 PWM driver

Since the SBC does not have four PWM outputs a PWM driver is required in order to regulate the motors' power. PCA9685 is a 12-bit, 16-channels PWM driver with an I2C interface. Since the Raspberry Pi has an I2C interface the PCA9685 is a good fit as a PWM driver for this project. The driver has 16 channels and only four are used to control the motors. The other 12 channels are used to control the brightness of four RGB LEDs. This fulfill requrement 3a. The LEDs are mounted in each corner of the robot platform (see 4.1.1). Since the PWM driver is made for LEDs, there are internal current limit at $20mA$ at each channel, which makes no need for resistors to cap the current flow through the LEDs.

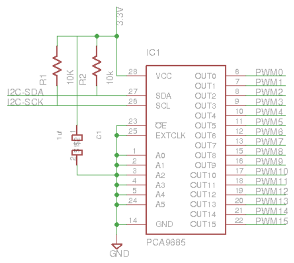

Figure 15: PCA9685 - I2C PWM driver.

The PWM driver supports up to 64 different hardware configured I2C addresses by a 6-bit address bus (A0 to A5). Furthermore the PWM driver has an internal clock of $25MHz$ and supports an external clock source of up to $50MHz$ which both can be scaled down in software.

$$\frac{25MHz}{2^{12}} \approx 6.1kHz \tag{9}$$

Since we are using the internal clock without scaling the frequency, the output is $6.1kHz$ (see equation 9). When a motor is running with a PWM frequency less the $20kHz$ it is detectable by the human ear. This can be solved by increasing the PWM frequency (see section 6).

For this project we are using phase correct PWM on all channels since some of them are controlling motors.

4.1.2.6 Full Bridge

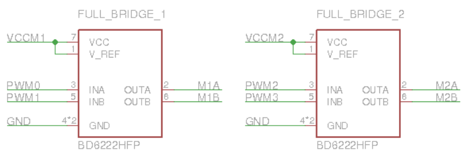

Figure 16: BD6222HFP - Full bridges.

In this project the full-bridge BD6222HFP is used. The full-bridge is rated to $18V$ and $2A$ which is suitable for the $1600mA$ motors running at $8.4V$ (see section 4.1.2.7). Furthermore the full-bridge have built in short circuit protection which prevents two mosfets at the same side of the motor to be turned on at the same time and damage the full-bridge.

Since the transistor in full-bridges is mosfet transistors the current can't be adjusted but only turned on and off. Therefore the full-bridge get its input from the PWM-driver to regulate the power of each motor.

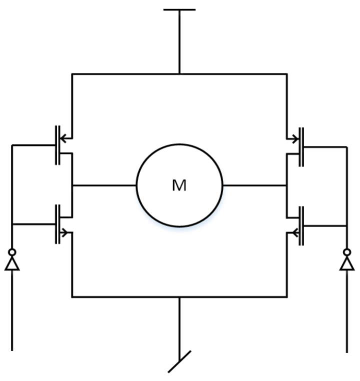

Figure 17: Full bridges.

4.1.2.7 Motor

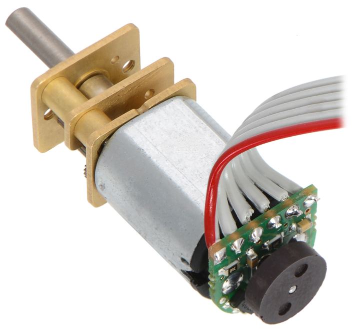

Figure 18: Motor and encoder for the Zumo platform.

The DC motors which are placed on the platform, have a gearing of $1 : 210$, to have a high enough torque to turn on the spot.

The motors are rated to sustain a muximum current flow of $1600mA$. The motors will draw exactly that when stalling at a voltage of $6V$, this was a design decision by Pololu to prevent users from burning the motors. Since we are using a battery with higher voltage we have to fuse the motors (see section 4.1.2.11) to prevent the motors from burning up.

The output axis of the motor is rated to turn $120rpm$ at $6V$. A test showed the motor turns $203rpm$ at $8.4V$ which is the maximum voltage we are supplying the motor with.

The speed of the motor could be useful to obtain, to regulate the duty cycle of the PWM signal. For that purpose encoder circuits are placed on the motor. The encoder is using 2-bit gray-code for the position which makes it easy to obtain change in position by a simple algorithm (see section 4.2.3.8).

Voltage	No load motor rpm	Max velocity	Max rotational velocity
$6V$	$134rpm$	$0.24\frac{m}{s}$	$4\frac{rad}{s}$
$7.4V$	$177rpm$	$0.31\frac{m}{s}$	$6.22\frac{rad}{s}$
$8.4V$	$203rpm$	$0.36\frac{m}{s}$	$6.83\frac{rad}{s}$

Table 2: Performance of the robot at different voltages.

Table 2 shows the maximum velocity is $0.36\frac{m}{s}$ this is a good speed to test these robots indoor. The rotational speed is $6.83\frac{rad}{s}$ which makes the robot very agile.

Now we know the maximum speed of the motor we can calculate the needed encoder update rate. The encoders have to update minimum four times per revolution of the motor because the encoders have four stages per round.

$$203Hz \cdot 4 = 812Hz$$

Each motor is connected to a full-bridge (see section 4.1.2.6) and the encoder is connected directly to the SBCs GPIO pins.

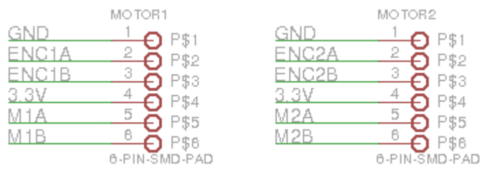

Figure 19: Motor and encoder.

4.1.2.8 Regulator

The electronics on the robot are using three different supply voltages. $8.4V$ for the motors, $5V$ for the SBC and $3.3V$ for the PWM driver. Therefore the voltage from the $8.4V$ battery have to be regulated down to $5V$ and $3.3V$.

The SBC does also have a $3.3V$ regulator which could be used instead of designing a new one. However the regulator on the SBC only allows sinking $250mA$. The $3.3V$ voltage source on the robot will supply the PWM driver which controls the full-bridge and 12 LEDs (4 RGB LED) which each pulls up to $25mA$ each resulting in a maximum current draw of $300mA$.

The conclusion is that the SBC does not supply enough current to the robot. Therefore two new voltage regulator circuits have been implemented.

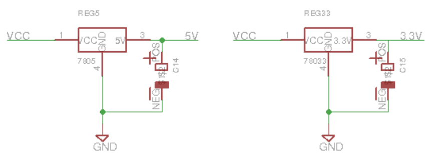

Figure 20: 7805 and 78033 regulators.

Each regulator provides $1.5A$ which is enough for the SBC and the PWM driver since the Raspberry Pi 3 is using less than $700mA$ when all four cores are running and the Wi-Fi module is active. The current draw from the USB-DUX-Fast and pre-amplifier print is not documented so to be sure the regulator does not burn out, the $5V$ regulator should be bigger (see section 6). Further more the $3.3V$ regulator could be removed if the PWM driver was supplied with $5V$ (see section 6).

4.1.2.9 PCB

The printed circuit board features some of the aforementioned electronic circuits in the chapter (PWM driver, full-bridges and regulators). The circuit board was made when the idea was based on the Banana Pi (see section 4.1.2.1) which resulted in the use of the 40-pin FFC header which could be connected directly to the Banana Pi. Since the SBC used now does not have a 40-pin FFC header with the same pinout, we are not able to plug the flat cable directly to the SBC. That means we needed to make an adapter print with 40-pin FFC header which is connected to the Raspberry Pi 3.

To make the robot nice and clean the PCB has to be less then $5.7cm$ x $6cm$ to fit in the battery compartment underneath the robot.

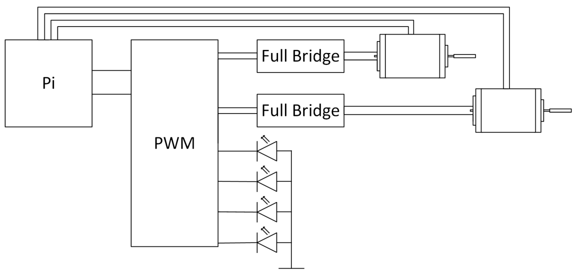

Figure 21: Block diagram of the PCB.

4.1.2.10 Battery

The Zumo chassis support four AA Batteries (see section 4.1.1) but since we want the robot to have a lot of torque the robot should have more than $6V$ for the motors. Also the discharge rate needs to be high enough to power everything on the robot.

The two motors are rated to $1600mA$ each, the four mounted LEDs have a combined maximum current consumption of $300mA$ and the SBC draws a maximum of $700mA$ [1]. This means that the battery at least should be able to supply $4200mA$.

We have chosen to use a Li-ion battery since it is rechargeable and usually stores more energy. The Ansmann 7.4V ($8.4V$ maximum) rechargeable lithium battery pack is a great choice for a battery, it fits nicely on the platform, can supply up to $5000mA$ and has a capacity of $2600mAh$ which give us a run time for at least 30 min which fulfill requirement 2d.

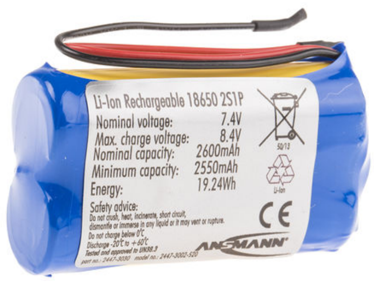
Li-Ion Rechargeable 18650 2S1P
Nominal voltage: 7.4V
Max. charge voltage: 8.4V
Nominal capacity: 2600mAh
Minimum capacity: 2550mAh
Energy: 19.24Wh
Safety advice:
CE
ANSMANN

Figure 22: Ansmann 7.4V Rechargeable lithiumionbatterypack.

4.1.2.11 Fuse

Since the robots are meant to be used by students, the hardware must be protected. Both motors and the battery has been fused to reduce the risk of burning them in case the motors stall. Since the motors is supplied with a maximum of $8.4V$ the motor will drain too much current when stalling.

$$\frac{6V}{1.6A} = 3.75\Omega \tag{10}$$

$$\frac{8.4V}{3.75\Omega} = 2.24A \tag{11}$$

Equation 10 show the resistance in the motor is 3.75Ω and equation 11 show the current in the motor at 8.4V the motor is pulling $2.24A$ which is too much to run continuously. Therefore we have fused both motors individually with a $1.5A$ fuse to be sure not to burn them out.

Each motor can pull $1.5A$ through the fuse, the PWM driver can pull $300mA$ and the SBC itself can pull up to $700mA$ which in all are $4A$. If more than $1A$ is drained from the USB port on the SBC while both the motors are running at maximum and all LEDs are on, the battery will be damaged since it will supply more than $5A$. Therefor the battery is fused with $4.5A$ (the three fuses fulfill requrement 2c).

The battery is connected to the barrel-jack connector which is fused by a $4.5A$ fuse and is then divided in three lines. A non fused (used for the voltage regulators) and two $1.5A$ fused lines for the two motors through the full bridges.

Since both the SBC and the PWM driver are drawing a variable amount of power we can't fuse the rest with a single fuse. It is possible to fuse them individually but there is not enough space left on the circuit board due to the limited dimensions of 5.7cm x 6cm (see 4.1.2.9). Therefore the SBC and the PWM driver are unfused.

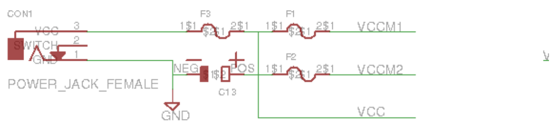

Figure 23: Power input.

4.2 Software

4.2.1 Operating system

The Raspberry Pi 3 is running Raspbian Jessie which is a Debian-based operating system designed for the Raspberry Pi series.

4.2.2 Flock simulation

A simulation of multiple robots was made to get a feeling of the flocking rules' individual importance. Since the units are SI units and the simulation can run faster than real time, the flocking rules can easily be tuned to fit the simulated robots and even the physical robots.

The simulation of the robot swarm is written in C++ using the graphical library SDL2.

The simulation is mainly based on a single class which contain the states of each robot and an update function to update the states of the robots.

The update function calculates the force of alignment, cohesion and separation from the settings the user has chosen, and then calculates the direction the robot wants to go by summarizing the three forces.

4.2.2.1 Alignment

The alignment force is based on the summarized directions of all robots within range.

The result is typically represented in one of two different ways. Either by an average velocity (done by dividing the vector by the number of robots in range), or as we did, by normalizing the velocity vector.

Average velocity:

$$F_a = \frac{\sum_{i=1}^{n} v_i}{n} \tag{12}$$

Unit vector:

$$F_a = \frac{\sum_{i=1}^{n} v_i}{||\sum_{i=1}^{n} v_i||} \tag{13}$$

Where F_a is the force of alignment, v is the velocity and n is the number of robots.

Algorithm 1 Alignment rule

```
1: function ALIGNMENT($robotsInRange$)
2:     velocitySum = (0, 0)
3:     for (robot : robotsInRange) do
4:         velocitySum += robot.velocity
5:     end for
6:     normalize velocitySum
7:     return velocitySum
8: end function
```

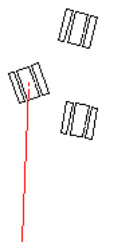

Figure 24: Alignment force.

4.2.2.2 Cohesion

The cohesion force is a force pointing towards the center of mass of the robots within range.

$$F_c = \frac{1}{M} \cdot \sum_{i=1}^{n} m_i r_i \tag{14}$$

Where F_c is the force of cohesion (or the center of mass), M is the total mass, n is the number of robots, m is the individual masses and r is the individual positions.

Since all the robots have the same mass we ended up with the equation:

$$F_c = \frac{\sum_{i=1}^{n} r_i}{n} \tag{15}$$

Algorithm 2 Cohesion rule

```
function COHESION($robotsInRange$)
    centerOfMass = (0, 0)
    number of robots = 0
    for (robot : robotsInRange) do
        centerOfMass += robot.position - this.position
    end for
    if (number of robots in range > 0) then
        centerOfMass /= numberOfRobots
        centerOfMass /= maxRange
    end if
    return centerOfMass
end function
```

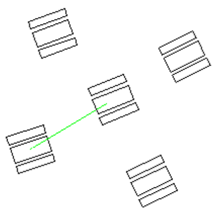

Figure 25: Cohesion force.

4.2.2.3 Separation

The separation force is a sum of the repulsion functions for each robot and object within range.

$$F_s = \sum_{i=1}^{n_r} \left(\frac{p_r - P}{||p_r - P||} \cdot (R - ||p_r - P||) \right) + \sum_{i=1}^{n_o} \left(\frac{p_o - P}{||p_o - P||} \cdot (R - ||p_o - P||) \right) \tag{16}$$

Where F_s is the force of separation, n_r is the number of robots, p_r is the individual robot position, P is the position of the robot itself, n_o is the number of objects and o_r is the position of the individual objects.

Algorithm 3 Separation rule

```
function SEPARATION(boids)
    forceVector = (0, 0)
    for (robot : robotsInRange) do
        directionVector = robot.position - this.position
        directionVector.scaleTo((maxRange - directionVector.getLength()))
        force += directionVector
    end for
    for (object : objectsInRange) do
        directionVector = object.position - this.position
        directionVector.scaleTo((maxRange - dir.getLength()))
        forceVector += directionVector
    end for
    forceVector *= 1
    return forceVector
end function
```

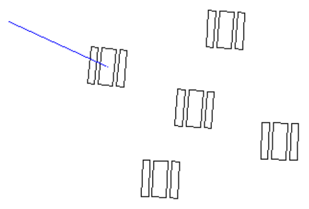
Figure 26: Separation force.

4.2.2.4 Border repulsion

A border repulsion function is implemented to keep the robots close to each other, even when not in a group.

Algorithm 4 Flocking force

```
function BORDER()
    forceVector = (0, 0)
    if (robot.pos.x < -borderLimit) then
        forceVector.x += pow(-pos.x - borderLimit, 1.5);
    end if
    if (robot.pos.y > -borderLimit) then
        forceVector.y += pow(-pos.y - borderLimit, 1.5);
    end if
    if (robot.pos.x > borderLimit) then
        forceVector.x -= pow( pos.x - borderLimit, 1.5);
    end if
    if (robot.pos.y > borderLimit) then
        forceVector.y -= pow( pos.y - borderLimit, 1.5);
    end if
    return force
end function
```

Figure 27: Border repulsion force.

4.2.2.5 Summarizing

The summarizing includes the coefficients for each rule:

$$F = F_a \cdot a + F_c \cdot b + F_s \cdot c \tag{17}$$

Where F is the force to the robot, a is a coefficient for the alignment force, b is a coefficient for the cohesion force and c is a coefficient for the separation force.

Algorithm 5 Flocking force

```
1: function CALCFLOCKINGFORCE(robotsInRange, objectsInRange)
2:     alignmentForce = alignment(robotsInRange) * alignmentFactor
3:     cohesionForce = cohesion(robotsInRange) * cohesionFactor
4:     separationForce = separation(robotsInRange, objectsInRange) * separationFactor
5:     borderForce = border() * borderFactor
6:     return alignmentForce + cohesionForce + separationForce + borderForce
7: end function
```

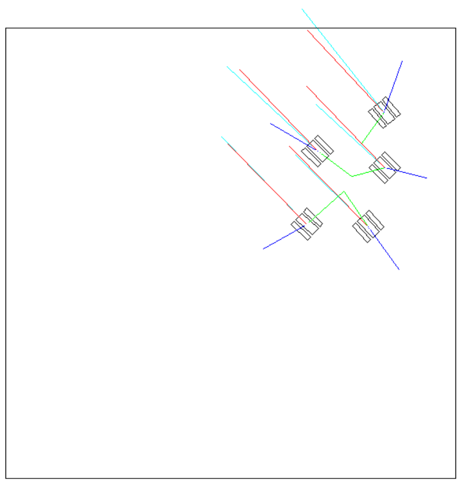

Figure 28: Alignment, cohesion, separation and the total force.

4.2.2.6 Integration

From the calculated force above, the velocity and angular velocity of the robot is calculated. The Euler integration rule is used to integrate the velocity and angular velocity to get the direction and position of the robots.

4.2.2.7 User interface

The user can interact with the program through an option window to change the parameters of the flocking behavior and to toggle the graphical display of the robots properties.

The coefficient for each of the three rules can be tweaked separately, the radius and repulsion of the border can be tuned. Furthermore a dynamic number of robots can operate with a random error with a maximum amplitude set by the user. Each force of the three rules, the total force, the velocity and the radius of each robot can be showed.

4.2.3 Robot

As specified in requirement 4a and 4b the programs on the robots should communicate in a pipe using standard input/output to send data to each other. A total of 8 programs have been made. (See Figure 29)

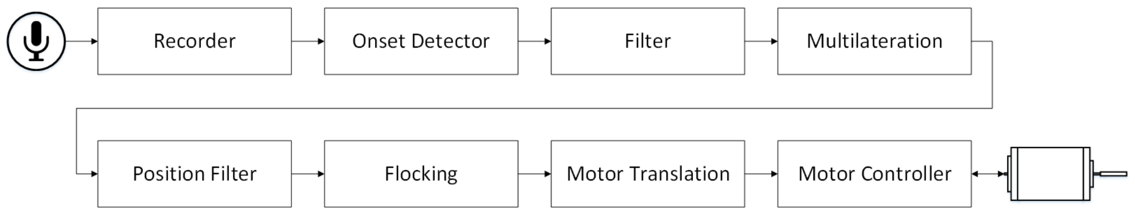

Figure 29: The pipe structure of the software.

The function of each program is described below:

4.2.3.1 Recorder

The recorder is not written yet but should read data from the USB-DUX-Fast and output the sample number and one sample value from each microphone in a comma separated ASCII string, every time the ADC has four samples.

<sample id>,<value1>,<value2>,<value3>,<value4>

Fx:

$$0, 0.68514, 0.53543, 0.35516, 0.86524$$

$$1, 0.65457, 0.49254, 0.43578, 0.90218$$

In case the rest of the pipe cannot keep up with the flow of data, the recorder can discard some of the samples.

We have used Matlab to convert a .wav-file (pre-recorded from the sensor system) to a comma separated file which we can pipe directly to the Onset detector.

4.2.3.2 Onset detector

The second program receives the data in four channels and determines when a click is recorded on each channel. This is done using an onset algorithm. The onset algorithm measures the energy level of the signal by differentiating the signal. When the energy level rises above a defined threshold the onset algorithm detects a click. A sleep time is implemented to eliminate the risk of detecting two onsets from the same click. When a click is detected, the microphone on which it was detected and the sample ID is passed to the next program.

4.2.3.3 Filter

The detection of an onset is very simple however the grouping of these onsets are very difficult. The third program tries to combine the individual onsets on each microphone passed from the previous program. This includes determining if there has been a collision. In case there has been a collision the data is discarded. If no collision is detected, the difference in arrival time is calculated based on the sample ID.

This is done by saving the last of each microphone onsets, and comparing the time of arrival of the newest and the oldest onset on the four microphones. If the difference between these two onsets are less than the the travel time of sound from one microphone to the diagonal microphone, the onsets are considered a set of same sound. The difference in time is then converted to difference in length using $340,29m/s$ as the speed of sound.

4.2.3.4 Multilateration

The fourth program receives differences in distance between the two microphones in the pair and the source. The differences are used to calculate the constants for the hyperbola which are used to calculate the position using two 2nd order equations.

The 2nd order equations are solved with the formular

$$x_1 = \frac{q}{A} \qquad x_2 = \frac{C}{q} \tag{18}$$

in which q is defined as:

$$q \equiv -\frac{1}{2}\left(b + sign(b)\sqrt{b^2 - 4ac}\right) \tag{19}$$

This method is more numerically as stable than the traditional method:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{20}$$

Becomes numerically unstable in case a or c (or both) is small because b and the discriminant will be close to identical [5].

The exact mathematical approach is covered in depth in section 2.4

In case the discriminant is negative, meaning there are no real roots to the 2nd order equation. The equation is solved as if the discriminant was 0 as this is the closest solution with real roots.

4.2.3.5 Position filter

A position filter is implemented to improve the estimate of the robot's position. The position filter gets cartesian coordinates and translates them to polar coordinates. The distance and the angle for each robot are separately going through a running average to smoothen out the deviation. When making the running average, the previously calculated position, closest to the newly calculated position, is assumed to be of the same robot. The number of robots are hard-coded, but could be implemented to be dynamic by clearing the buffer of the robots out of range. Lastly the angle and the distance are converted to a cartesian coordinate and passed to the flocking program together with the position difference (velocity) from last step.

4.2.3.6 Flocking

When the individual robots know where the surrounding robots are, the robots have the three flocking rules to apply. The robot calculates where to go according to the flocking rules algorithms described in section 4.2.2. The calculated force given by the algorithm is then written to standard output.

4.2.3.7 Motor translation

The Motor translation is a simple program that calculates the individual motor speeds based on the angle of the force coming from the flocking algorithms.

When the robot has a force pointing to the left, the right motor is at full speed and the left motor speed is linearly mapped by the angle between the force and the direction of the robot. And when the force is pointing to the right, the left motor is at full speed and the right motor speed is mapped.

Algorithm 6 Motor translation

```
1: force = input
2: angle = force.getAngle()
3: if (angle < 0) then
4:     left speed = 1
5:     right speed = map(angle, 0, pi, 1, -1)
6: else
7:     left speed = map(angle, 0, pi, 1, -1)
8:     right speed = 1
9: end if
```

4.2.3.8 Motorcontroller

The motor controller takes a target velocity for each motor and uses a PID controller to calculate the duty cycle for the PWM signals using the encoder inputs as feedback.

The inputs and outputs are implemented using wiringPi which is a commonly used IO-library for the Raspberry Pi series. WiringPi is written to many different programming languages etc Java, C#, Python and C which we are using.

The calculated PWM values are sent via I2C to the PWM driver to regulate the force of the motor, and the inputs are read from the GPIO pin at the Raspberry Pi.

The encoders are 2-bit gray code encoders which allow us to use a very cheap algorithm to update the position:

Algorithm 7 Sampling implementation

```
1: oldState = readEncoderState()
2: function UPDATEENCODER()
3:     newState = readEncoderState()
4:     if (newState != oldState) then
5:         if (MSB of newState != LSB of oldState) then
6:             position += 1
7:         else
8:             position -= 1
9:         end if
10:        oldState = newState
11:    end if
12: end function
```

This algorithm has to run at each motor at more than $812Hz$ (see 4.1.2.7).

4.2.3.9 Module based software

The best part of dividing the software in small programs is the ability to easily swap out or bypass part of the software. In our case we have recorded data from the sensor system and saved in a comma separated file using Matlab. The comma separated file can be piped directly into the second program in the pipe (OnsetDetector) without the use of the Recorder program, and we can get the output from fx the multilateration program, the flocking program, MotorTranslation program or any combination of these.

4.2.3.10 Programming

To simplify uploading and compiling of the code to the robots a bash script (inspired by Leon Bonde Larsen) was developed. The user enters the ip address of each robot and run it to upload all code to the robots, compile the code on the robots and execute the code if possible. The script is using SFTP to upload files from the computer to the robots, and SSH and TMUX is used to start the program on the robots. To eliminate the tedious task of typing the password every time a connection needs to be established, the ssh public key is copied to the robots. To reduce compiling time the code could be cross compiled at the user's pc and then uploade the executable files.

5 Validation

To validate the system performance, the robot has to localize the other robots precise enough for the flocking algorithm to work. Since we did not end up with more than one working robot, the system can not run in the real wold. One test (localization) was made on the physical robot and the other tests were made based on the simulation.

5.1 Processing speed

We were not able to validate the Raspberry Pi 3 as a fast enough candidate for the robot due to lack of finished hardware. However the most computationally expensive algorithm (the flocking algorithm) can run on a laptop at over $100kHz$ ($10\mu s$). So if we say the position refreshment rate should be $60Hz$ (as the simulation), the Raspberry Pi will have to run the algorithm within $\frac{1}{60}s$ which is about $20ms$. Therefore the Raspberry Pi 3 will most likely calculate the algorithm within the time step. Furthermore the Raspberry Pi 3 has four CPU cores which are all used due to the software being divided into eight programs.

5.2 Theoretical multilateration validation

To validate the multilateration program, some input data was calculated and fed into the program.

The sound source was tested at $0.5m$, $-0.5m$ and the input to the multilateration program was calculated to be

$$0.0393, 0.0753, -0.0435, -0.0710$$

Storing these values in a file and then using cat to pipe the data into multilateration resulted in the same coordinate $(0.5, -0.5)$ used to calculate the data.

It is important to keep in mind that this is ideal data with no noise, given the only noise is introduced by the sample rate then the output varies (see Table 3)

X	Y
0.451	-0.449
0.412	-0.411
0.398	-0.398
0.499	-0.497
0.554	-0.546

Table 3: Coordinates generated by the Multilateration program with input of 0.5, -0.5 and noise introduced by the sample rate

At the tested position and with the tested amount of noise, the position varies with up to $15cm$.

As discussed in section 2.4 the multilateration algorithm is more unstable near the axes, this is because the difference on 2 of the 4 microphone pairs are very low.

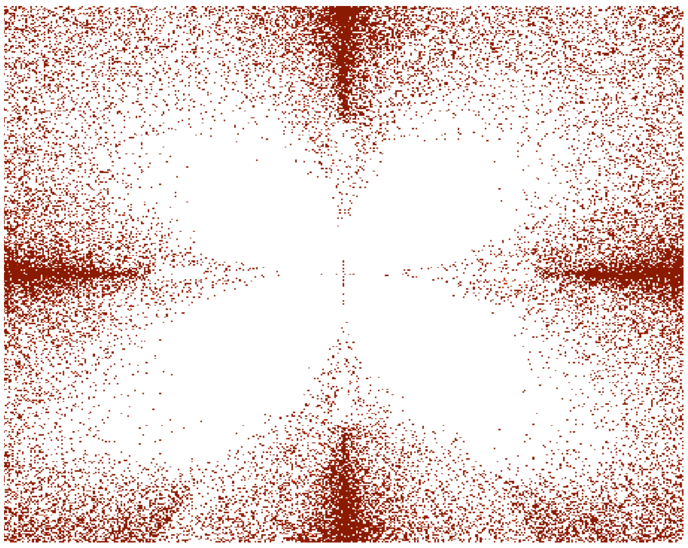

Figure 30: Error on the multilateration with noise, errors occur near the axis at smaller distances.

5.3 Practical validation

To validate the pipe, data was recorded on the microphones using the USB-DUX-Fast.

The data was recorded $38cm$ to the right of the robot, and $38cm$ behind the robot. Which means the data is recorded at a $45degree$ angle.

The .wav files were converted to .csv files using Matlab. The .csv files were piped into the onset detector using cat. Since the distance from each microphone, the speed of sound, and the samplerate is known, it is possible to calculate how many samples should pass from the initial onset until the other microphones detect an onset. The calculated values are compared to the actual values in table 4

Onset	1	2	3	4
Expected output	105	70	0	41
Actual output	106	71	0	40

Table 4: Table showing how many samples were recorded since the first onset was detected. Both calculated and actual values are shown.

In this test, the onset algorithm is within one sample of the expected values which is very good.

Next, the output from the onset program is piped into the filter and then into the multilateraion, the expected and actual output is shown in table 5

Multilateration	X	Y
Expected output	0.38	-0.38
Actual output	0.3835	-0.3975

Table 5: Table showing the expected X and Y values of the multilateration program and the actual output.

The expected and actual output varies with less than $2cm$ which is definitely a satisfying result.

5.4 Flocking validation

At the start of the program the robots have a random position and will have to unite by driving around until they encounter other robots. When a group of robots are driving together the three coefficients can be tweaked to fit the users needs. The alignment coefficient should be changed first for faster regrouping, and then the separation since the collision with objects depends on the separation force. Lastly the cohesion coefficient are tweaked to hold the group together.

The simulation is very visual due to the different effects such as the different forces from the rules.

Results show that the distance between the robots are more flexible when the separation and cohesion coefficients are small and more fixed when the coefficients are high. This is due to the gradients of the gravitational map in the workspace.

When a group of robot with almost fixed distance approach an obstacle the group is most likely to drive the same way around the obstacle. But when a group of robots with more flexible distance between them are approaching an obstacle they are likely to drive each way around and will have to unite again on the other side. To make the flock reunite after an obstacle, the robots will either need a larger search range so they will have contact even when bypassing the obstacle or increase the number of robots which makes the system act like a fluid.

When the error term is applied, the robots have a higher tendency to wobble. Since the error is linearly distributed in both angle and amplitude the average of the error will go towards zero when the number of iterations increase. When the integration has a step size of $\frac{1}{60}s$ and the robots' position is calculated for each step the error does not have a huge impact because the robot has little time to change it's direction. The system performs well even when the error is linearly distributed with a maximum amplitude of $1m$ and the system updates the positions at $60Hz$. With a position update rate at $5Hz$ and with a maximum error amplitude of $1m$ the robots still stay in a flock and do not collide with each other.

6 Conclusion

6.1 Multilateration

The multilateration is implemented on the robot and uses the 4 microphones to calculate the position of nearby robots using the click sounds they emit.

The implementation simplifies the task since the microphones are mounted symmetrically around the axes which results in two 2nd order equations. The equations are solved in a numerically stable way.

The system is more unstable along the axes of the coordinate system because the angular difference between two of the microphone pairs are lower.

Noise introduced by the sample rate does not impose a problem, this was tested with a position far from the axes. Positions close to the axes with noise may produce undesirable results.

6.2 Simulation

Like in a flock of birds, the simulated robots ended up acting like a part of a bigger unity. The robots stay together and are very likely to drive the same way.

In the beginning, each robot have a random position and need to find each other. If two robots encounter facing opposite directions, they are not likely to unite. But if the alignment coefficient is high or the robots are meeting with a sharp angle between the velocities they are very likely to unite.

The separation coefficient impact both the separation from obstacles and the separation from other robots. Therefore this variable is best to tune first and then tune the cohesion coefficient afterwards.

Changing the coefficients has a great impact on the behavior of the flock, but does not cause the robots to stop eliciting the flocking behavior.

6.3 Position logging

The position calculation update rate is important in this project since the motor speed is only calculated once for each position measuring. To minimize the cost of a low refresh rate of the robots position, the positions of each detected robots could be stored and the movement of the local robot being integrated up. A new speed for each motor could be calculated based on the new relative position of the other robots. Furthermore the other robots positions could be integrated up based on their velocities (this requires a good localization to determine the velocity of the surrounding robots).

6.4 Sensor system

The sensor system used in this project is too big to place on top of the robot platform. But the system is being redesigned to fit the form factor of a Raspberry Pi. That means in later work the USB-DUX-Fast and pre-amplifier print with filters could be stacked on the Raspberry Pi 3.

6.5 Battery charger

To reduce the complexity of charging the batteries, a battery charging circuit could be designed and implemented on the robot. This would make it possible to just plug a wall power supply into the robot to charge the battery or even drive the robot directly from the wall power supply. Furthermore a battery monitor circuit could be implemented to eliminate the risk of discharging the battery too much. This could be done by using a comparator to compare the battery voltage with the minimum voltage of a Li-Ion battery. When the battery voltage is below the minimum voltage of the battery the robot could light an LED, sound a buzzer or send a signal to the Raspberry Pi. The robot could alternatively be able to drive into a dock when the battery is discharged and then recharge itself when necessary. The dock could be sending out a specific frequency which the robots could recognize and drive to. The charging station could work like bumper carts, with a metal plate as floor and a metal plate as ceiling, a spring attached to the robot should touch the floor while another one touches the ceiling.

6.6 Buzzer

The sound produced by the piezo-buzzer is not loud enough to outplay the sound coming from the motors. Therefore another way to produce the sound must be developed to make the individual robots able to detect the other robots when driving.

6.7 Regulator

If the PWM controller is moved to the $5V$ line, the $3.3V$ regulator could be removed and a bigger $5V$ regulator should be replacing the existing one.

6.8 PWM driver

The PWM driver used in this project has a maximum clock speed of $25MHz$ which makes the motors produce a sound at $6.1kHz$. In a future project a PWM driver with higher frequency should be used. A PWM frequency above $20KHz$ which is the highest frequency detectable by the human ear should be used. The PWM driver should still have 16 channels to control both the motors and the RGB LEDs and have an interface matching the Rasperry Pis (I2C, SPI or UART).

6.9 Platform

The robot ended having a good velocity and rotational velocity for indoor use. The size of the robot is $10cm$ by $10cm$ meaning a flock of five to ten robots easily can drive around indoor.

7 References

[1] Raspberry pi faqs - frequently asked questions.

[2] Tórur Andreassen, Annemarie Surlykke, John Hallam, and David Brandt. Ultrasonic recording system without intrinsic limits. *The Journal of the Acoustical Society of America*, 133(6):4008–4018, 2013.

[3] Behrouz A. Forouzan. *Data Communications and Networking*. McGraw-Hill Education, 5 edition, 2013.

[4] Brian L. Partridge. The Structure and Function of Fish Schools, 1982.

[5] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes*. Cambridge University Press, 3 edition, 2007.

[6] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34, 1987.

[7] Ali E. Turgut, Hande Çelikkanat, Fatih Gökçe, and Erol Şahin. Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2(2-4):97–120, 2008.

8 Appendix

Directory structure of the electronic media:

Each code directory contains a source file and a Makefile except "includes" which is the library.

1. 3D Model
2. Code
    (a) Filter
    (b) Flocking
    (c) FlockingSimulation
    (d) includes
    (e) MotorController
    (f) MotorTranslation
    (g) Multilateration
    (h) OnsetDetector
    (i) PositionFilter
3. Datasheets
    (a) Ansman.pdf (Battery)
    (b) BD6222HFP.pdf (Full-Bridge)
    (c) PCA9685.pdf (PWM controller)
4. Report
    (a) Report.pdf
5. Schematics and PCB
    (a) PCB.brd (Eagle board file)
    (b) PCB.sch (Eagle schematic file)
    (c) PCB_brd.pdf (double sided print)
    (d) PCB_bottom_brd.pdf (bottom side)

    (e) PCB_sch.pdf (schematics)

    (f) PCB_top_brd.pdf (top side)

6. Videos

    (a) Robot video

    (b) Simulation video