NTNU – Trondheim
Norwegian University of
Science and Technology

Path Integration in a Swarm of Robots

Anders Søbstad Rye

Master of Science in Computer Science
Submission date: Januar 2014
Supervisor: Pauline Haddow, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Anders Søbstad Rye

Path Integration in a Swarm of Robots

Master's Thesis, January 2014

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

In this report I propose a method of navigation for differentially wheeled robots inspired by path integration in certain social insects like bees and ants. It is a very simple method, intended for use in low-tech robots with very limited hardware, such as swarm robots. Path integration is essentially dead reckoning as used by animals, calculating the relative position based on the movements made since the last known position. It is a tried and true method of navigation that also has significant flaws, especially in that inaccuracies accumulate and magnify over time. In this report I want to examine whether communication and information sharing between robots in a swarm can alleviate some of the drawbacks, and make it a viable method for navigation for swarm robots over relatively short distances.

Sammendrag

I denne rapporten foreslår jeg en metode for navigasjon for roboter inspirert av måten noen sosiale insekter som bier og maur navigerer. Det er en veldig enkel metode, ment for enkle roboter som har veldig begrenset med instrumenter tilgjenglelig, som svermroboter. Metoden bygger påen metode som essensielt er bestikkregning som brukt av dyr, det vil si åberegne sin relative posisjon basert påbevegelser gjort siden sist kjente posisjon. Det er en velkjent metode for navigasjon brukt av mennesker i århundrene før moderne navigasjonssystemer. Det har ogsåstore ulemper, spesielt med hvordan unøyaktigheter akkumulerer og forstørres over tid. I denne rapporten vil jeg undersøke om kommunikasjon og informasjonsdeling mellom roboter i en sverm kan hjelpe med åminske noen av disse problemene, og gjøre det til en brukbar metode for navigasjon for svermroboter over relativt korte avstander.

Preface

This report is my master's thesis, which concludes my 5-year studies at the Norwegian University of Science and Technology (NTNU). It is based upon my specialization project, in which i was part of a group that designed and built, from the ground up, a robot intended for use in swarm projects. In this project, I will use that robot to do swarm related experiments.

I would like to thank my supervisor Pauline C. Haddow, for her invaluable support and feedback throughout this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Swarm intelligence is the study of the collective behavior that emerges from the actions of a swarm of decentralized and self organizing individuals. The core principle of swarm intelligence is that from simple behavior on an individual level there can emerge advanced behavior on a collective level. This is how ants, termites and bees can build intricate nests and hives without any central entity governing their actions, or with any of the individuals having any concept of their place in the larger picture. They follow simple rules, modified by locally available information, and from that emerges relatively complex behavior.

Applying the principles of swarm intelligence to robots is called swarm robotics. Like its counterpart in nature, swarm robot systems consists of many (usually small) individuals, each running a simple program, following simple rules, from which a meaningful behavior emerges at the swarm level, rather than the individual level.

The main appeals of using a swarm robot system is the following:

- **Fault tolerance**. A robotic swarm is highly fault tolerant by nature, since the swarm consists of many identical individuals, should one fail, there will still be a number of robots left that can perform the same task. Naturally, this also means there is no single point of failure. Even if you were to remove half of the robots in a swarm, it would still probably be able to solve its task, though most likely at a lower efficiency.

- **Cost**. The nature of swarm intelligence, that complex collective behavior can emerge from simple individuals means that the robots themselves can be made very simple
- **Scalability** Scaling the system to tackle larger problems can be as simple as adding more robots to the system

In a robot system like this, where cost and complexity needs to be kept to a minimum, there is little room for relatively advanced hardware such as positioning systems or cameras. Instead, we must rely on simple hardware, simple software, and the power in numbers. Navigation in a robot of such limited hardware is naturally then quite tricky. In an essentially "blind" robot, in the sense of very limited sensory capabilities, options are very limited.

Dead reckoning is a method of calculating one's position by estimating the distance traveled from a previous, known position. Usually by measuring speed and heading, and calculating the distance traveled, and in what directions. It is how sailors navigated the seas for hundreds of years before modern positioning systems.

In animal navigation, dead reckoning is usually known as path integration. Several animals, like geese, mice, and ants have been shown to use path integration in order to find their way home. The instances I will focus on are certain species of desert ants who use the sun as a compass and measures their distance traveled in order to find their way back to the nest, as well as how bees not only can find their way back to a place they have been, but also communicate this location to other bees through dance.

In this report I take inspiration from these social insects, and design a simple, low-tech method of navigation suitable for a swarm of robots. I have no ambitions of making a system that is pinpoint accurate. In fact, my aim is to make a method that is at the very least better than no navigation, i.e. random search. In which case it only needs to be accurate enough to point the robots in the general direction of their target, and in that way narrowing down the area in which they'll need to search. I wish to examine the possibility of using such a system in a cooperative task, in this case an object retrieval task.

1.2 Goals and Research Questions

The questions I want to answer in this report are the following:

- Can a robot navigate sufficiently accurate over short distances using methods inspired by path integration in insects?

- Can the accuracy and/or efficiency be improved by using communication between many robots in a swarm?

In the above question, "sufficiently accurate" means accurate enough to solve the task at hand. E.g. if the task is to move a certain distance, then return home, sufficiently accurate would be to come within the radius necessary to detect home, or at the very least, narrow down the search area.

By "efficiency" I mean compared to a random search for the target, do the robots use less time searching for the target on average.

1.3 Thesis Structure

In this report I will first define and explain the concepts material I will be using in my report in chapter 2. In chapter 3 I will discuss the architecture and implementation of my system. In chapter 4 I will describe the experiments I want to perform and their results, and finally, in chapter 5, I will discuss and evaluate the results of these experiments.

Chapter 2

Background Theory and Motivation

2.1 The ChIRP robot

The ChIRP robot[1][2] is a robot that was designed and built here at NTNU by students, including myself, as a specialization project during the fall of 2012. It was designed to be a lower-cost alternative to the robots used here for swarm experiments and education.

It is a small, puck-shaped robot using designed to be as cheap as possible, while still being useful for use in swarm research. The ChIRP robot was designed with the following five core requirements in mind:

- **Low cost**. A low cost is very important when dealing with swarm robots. With swarm experiments requiring many tens or even hundreds of robots, using more fleshed out robots can quickly become prohibitively expensive.
- **Low complexity**. Similarly to the above, when dealing with so many robots, it is important that each one is easy to build, assemble, use and maintain. The ChIRP robot can be soldered and assembled by hand, and has a simple library that gives access to its basic functionality.
- **Small Size**. The robot needs to me reasonably sized to allow many of them to work in the same space.

- **Basic capabilities built in**. In it's most basic configuration (no attachments), the robot
- **Expandability** . The core functionality of the robot can be enough for many kinds of experiments. However, we wanted the robot to be as flexible as possible. The robot therefore need some way to attach additional hardware.

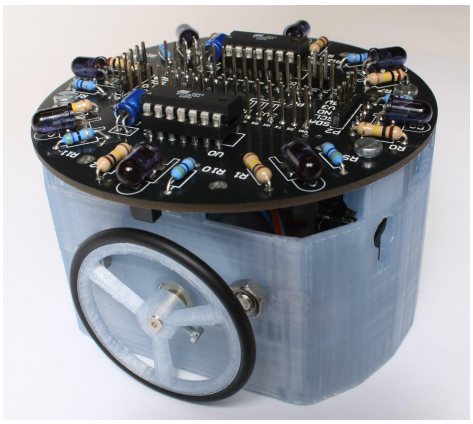

Figure 2.1: The ChIRP Robot

The Robot is made up of several parts. There is a sensor board (the top board, as seen in figure 2.1) which contains the eight pairs of infrared emitters and detectors. These are used for obstacle detection, infrared detection, and inter-robot communication. Attached to the sensor board is a motor board, containing the hardware that controls the motors. The motors are very standard geared stepper motors, with a resolution of 512 steps per revolution. Finally, there's the battery and the battery charger/booster circuit, which powers the robot. All the components are held together by a plastic, 3D-printed chassis.

There are in total four micro-controllers on the robot. The main micro-controller, the one executing the main program controlling the robot, is an Arduino Micro (which I will refer to as "Arduino" throughout this report). The rest of the micro-controllers are ATtiny84s (which i will refer to as "ATtiny"). Two of the ATtinys control one half of the sensors each. They are responsible for measuring distances, communication, and everything else that the sensors need to do. The last ATtiny is on the motor board, controlling both the motors. All of the micro-controllers are attached to the same I^2C bus. In this way, the Arduino can communicate with and control the ATtinys, by fetching sensor data and setting the wheel speeds,

for example.

To meet the first two requirement, the low cost and complexity, we designed the robot to require as simple components as possible. There are no advanced hardware like cameras for example, and thus no need for a sophisticated processor. The Arduino is a relatively low cost microprocessor, that, while not very powerful, is very flexible in its applications. Fortunately, in swarm robotics, there are no requirements for large computations, so these components does the job well.

Finally, to meet the expandability requirement, we designed several ways to attach additional hardware to the robot. Two sets of pins on the top expose the I^2C bus, allowing attachment of other I^2C compatible hardware, such as bluetooth, an accelerometer, a compass, etc. In addition, the pins of the main Arduino micro-controller Are also exposed on the top board, allowing attachment of simpler hardware that can interface directly with the micro-controller, such as LEDs, ambient light sensors, etc. The robot also has tabs on along the sides of the chassis, which allows attachment of larger components, like for example a gripper arm.

To make the robot as easy to use as possible, there is a complete library of code for each of the micro-controllers that allow the user access to the basic features of the robot, as well as an interface to ease the development of additions to the robot. This library, as well as other resources, are available at the ChIRP web site[3].

The end result is a robot that, while very simple, can perform the basic requirements for many swarm research projects just as well as its more advanced and expensive counterparts.

2.2 Path integration

Many animals have the ability to navigate very accurately, even over huge distances. Migratory birds regularly migrate thousands of kilometers to and from their breeding grounds. All without any maps or instruments. Animals use many different techniques in order to navigate, such as the earths magnetic field, landmarks, the sun and even odour.

I will focus primarily on how bees and desert ants navigate. Bees and ants can use both the sun and the polarization of the light in the sky as a compass[5]. It is not completely understood exactly how desert ants measure their distance traveled[6],

but flying bees seem to measure their distance by using optical flow[7](that is, they see how fast they are moving compared to the ground).

With these tools to measure both heading and distance traveled, these ants and bees use a form of vector navigation to find their way back to the nest[4]. The insect then use a kind of accumulator to record their position relative to the nest. This can be modeled like a coordinate system with the origin at the nest. One model of the accumulator has the insects continually update their position relative to the nest as they travel. To return to the nest, they would then have to travel towards the origin of the coordinate system again.

This is the model I will base my implementation on, as it is the most straight forward one. Other models allow certain insects to return by the same path they came by "resetting" their position and "flipping" their compass, but this model is not that useful in my case.

2.3 Other methods of navigation in swarm robotics

There have been many different approaches to the problem of navigation in a swarm of robots.

One of them is based on the concept of trophallaxis[8]. Trophallaxis in insects is mouth-to-mouth feeding. Given an area with a "nest" and a food source, robots can use this technique to find a path from the nest to the food source. It works in the following way: each robot keeps track of its "energy" level. When a robot finds the food source, it will fill up its energy level and then try to find the way back. When the robot travels it expends energy, meaning the further away from the food source, the less energy it will have left.

When the robot meets another, it can then "feed" the other robot, sharing its energy. A robot can use this to navigate towards the food source, if that robot then finds another robot with a higher energy level, it will know it is going in the right direction. This essentially creates an energy gradient in the arena, and robots can find the target by following the rising levels of energy, and then back by following the falling energy levels.

Another method I looked at was having the robots themselves form the path between two points (the "nest" and "prey")[9]. When a robot finds the nest, the robot waits there. When another robot finds the first robot, it will "attach" to it, by waiting next to it at the end of its sensory range. Other robots can then attach to the outer robot, and a chain is formed.

In the beginning, several different chains will extend from the nest in random directions. Robots at the end of a chain has a chance to abort and resume wandering, to avoid that a chain ends up in a dead end. Eventually one of the chains will hit the prey, and a chain between nest and prey is formed. The other robots can then follow the chain to the prey and back.

For the box pushing task, my work will build primarily on a master's thesis by NTNU students Jannik Berg and Camilla Haukenes Karud from 2011[10], but I've also looked at earlier work by Gross and Dorigo[11] as well as Kube and Bonabeau[12].

2.4 Motivation

The obvious, and biggest drawback with dead reckoning is how errors will accumulate while traveling, causing the accuracy to rapidly decline. What I want to examine in this report is whether dead reckoning can be used to navigate simple robots accurately enough over relatively short range. And also whether the inaccuracies inherent in dead reckoning navigation can be alleviating by communicating between robots. I don't expect to be able to navigate over long ranges like this, but for example using a swarm of robots to explore or search in an area, and the return home, like the desert ant does. Or sharing location information with other robots, limiting their search area, like the bee does.

There are three main reasons why I think adding communications between robots in this task may improve the system, which I want to examine in this report:

- **Spread of information**. I hope to see landmark information spread within the swarm faster than it would if each robot were to be required to discover each landmark for itself
- **Failure detection**. If a robot receives conflicting reports from several other robots, it can assume that it is off course, and can then reset and restart
- **Error correction**. If the robot receives other information that deviates from its own, but agree with each other, the robot can possibly use this to correct its own information.

Chapter 3

Architecture

In this project, I will use the ChIRP robot described in 2.1. I will use the ChIRP library that I helped develop in an earlier project as a base. I will be using an earlier revision of the software (revision 109), as it is the one I am most familiar with, and because the current revisions are still under active development.

In order to simplify the implementation of the experiments below, I will implement a number of atomic sub-tasks, which can be combined and used to define the behaviors needed for each of the experiments. These sub-tasks are described in detail in this section.

3.1 Maintaining vector

In order to determine determine the direction and distance to the target or home relative to itself, each robot needs to maintain a vector (i.e. both distance and angle) pointing to the target. Whenever the robot moves or turns, it needs to recalculate this vector.

In order to simplify these calculations while the robot needs to maintain a vector, it will only move in discrete movements, i.e. it will either turn or move, not both at simultaneously.

3.1.1 Implementation

In practice, this is not too complicated. It works by keeping track of positions relative to the robot's starting position and heading. It is essentially a coordinate system with it's origin at the robots starting position, with its X-axis pointing in the direction the robot is initially facing.

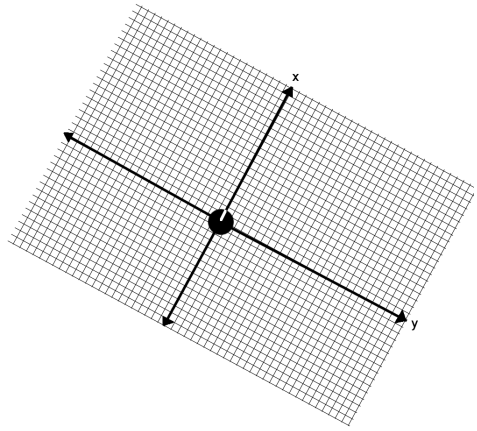

Figure 3.1: The coordinate system with origin at the robot's starting point, and the x-axis pointing in the direction of the starting heading.

Every time the robot has moved, it will update its position using based on its current heading and the distance it traveled. This is achieves using basic trigonometry:

$$x = x + StepsTravelled * cos(CurrentHeading)$$
$$y = y + StepsTravelled * sin(CurrentHeading)$$

Additionally, every time the robot turns, it simply updates the heading with the amount it has turned. So, if the robot moves 1000 steps directly ahead from its starting position, its new internal position is then $[1000, 0]$. If the robot then turns 90 degrees and moves 500 more steps, its internal position will be $[1000, 500]$.

To make the robot accurately rotate so that its internal heading matches the real one, I measured the wheel diameter and the distance between the middle of the wheels. To make the wheel measurement as accurate as possible , I measured the wheels at several points and used the average, in case of uneven wheels. I used

this to calculate the number of steps needed to rotate the robot fully:

$$StepsPerTurn = StepsPerRevolution * \frac{RobotCircumference}{WheelCircumference}$$

where StepsPerTurn is defined in the motor specifications. In addition, to improve accuracy by reducing wheel slippage both when turning and moving, the motors accelerate and decelerate gradually every time the motors change speeds.

If the robot detects an obstacle it will stop and poll the motor drivers for the number of steps actually taken, and use that to update the position before changing direction away from the obstacle and moving on.

Since the robot will only move in straight lines, and turn while in place, no more advanced calculations are needed, and the only thing the robot needs to keep track of is its current heading and position.

With this very basic framework at the bottom, I can then use this to record positions of landmarks (nest, food source, etc) and calculate the path from the current position to any of the recorded landmarks.

Of course, the robot in its basic configuration has no sensors to detect external influences on it, such as a compass or accelerometer. This means that this system is dependent on that each of the robots movements are initiated by itself. If the robot is displaced or rotated significantly by an outside force, by colliding with another robot, for example, any positioning, heading and landmark data is instantly rendered useless.

Two things are therefore vitally important for a system like this to work:

- Obstacle avoidance
  The robot needs to be able to detect obstacles and especially other robots well before making contact with them.
- Failure detection
  The robot needs to be able to determine whether it has come off course, either by arriving at a target position without finding a target, or by receiving conflicting data from several other robots, at which point it will reset position, heading and landmark data and start from scratch.

3.2 Searching

The searching phase is simply a random walk. From it's starting position, the robot will alternate between moving a random distance and turning a random distance.

3.2.1 Implementation

As it is implemented now, the robot simply turns in a random direction by a random factor in the range between 60 and 180 degrees. It then moves 1000 steps, before it repeats the process.

Alternatively, but not implemented, if the robot is not maintaining a vector at the time (before it has found any landmarks), the random walk can be achieved by simply randomly altering the ratio between the wheel speeds, giving the robot a smooth random walk. Without having to stop before turning, this is probably more efficient.

3.3 Communicating (sending)

In order to achieve communication between the robots using the built in IR-LEDs, I wanted a simple but sturdy communications protocol with very little overhead and other caveats. This because the robots preferably need to communicate while on the move with other robots that might also be moving, so communication needs to be swift, with little overhead.

While it is possible to transmit data explicitly over IR, which I did consider, it requires a relatively complex communications protocol. In our previous project with these robots, we did briefly experiment with data transmission between robots. Robots need to discover each other, then stop and align to ensure a steady signal, then establish a connection using a simplified TCP-like protocol. Messages need to be error-checked and acknowledged or retransmitted, before the robots can disconnect and continue.

In my case I want to be able to continually broadcast a simple message to any robots that may or may not be nearby. In this use case, the data transmission protocol above is very much overkill. Instead, I will use a much simpler method.

Communication between robots is achieved by simply pulsing the IR-LEDs. By varying the pulse duration and duty cycle of the pulse wave, the robot can communicate different messages to the recipient robot. To achieve this, different pulse widths correspond to different, pre-programmed messages in the recipient. For example, a pulse width of 40ms of can correspond to message A, while a pulse width of 50ms can correspond to message B, and so on. In addition, the time between pulses can be used in the exact same manner, so two messages can be communicated at the same time, as seen in fig 3.2.

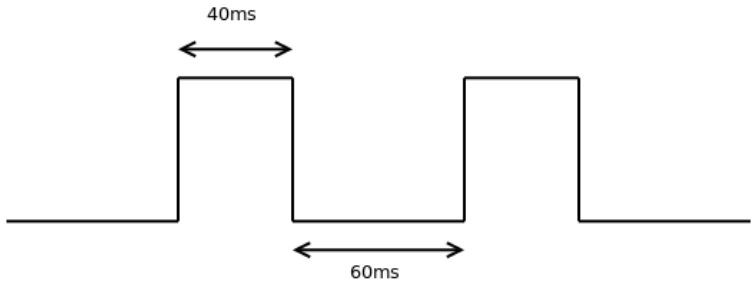

Figure 3.2: A pulse wave with a pulse duration of 40ms, and time between pulses of 60ms

Essentially, what this does is allow the sending of two integers, in the form of the duration of the HIGH and LOW phases of the pulse, very quickly between two robots. These two integers must then be interpreted by the receiving robot. In these experiments, I want to use this method to communicate very rough "coordinates" between robots. A basic example is the following: A robot has found some landmark and wants to share its location with other robots in the swarm. The robot then determines which IR-LED is currently pointing towards the landmark and starts pulsing that LED using a pulse-with that is pre-programmed to mean "towards". Also, the LED on the opposite side pulses with a different pulse-width, sending the message "away from". This can be seen in figure 3.3.

Of course, the direction alone is not enough if the recipient robot wants hold and maintain this new vector. It also needs the distance, otherwise the direction will be displaced whenever the robot moves (figure 3.4). This is where the second transmitted number comes in. Instead of having certain ranges of pulse-withs correspond with certain messages, in this case I want to use the raw number itself to roughly communicate the distance to the target. If each millisecond of pulse-with for example corresponds with 100 steps of distance, it will give the estimate a resolution of roughly 2.6cm.

When deciding on the resolution of the distance, it is important to keep in that since there is a number to transmit directly corresponds with the pulse-with of the signal, it will naturally take longer to transmit a larger number. In my case, the arena is smaller than about 1.7m meters at its widest point, so the pulse-width will never be more that roughly 65ms. Adding 40-60ms for the other part of the message, it means that in the worst case, it will take roughly 120ms to transmit the message.

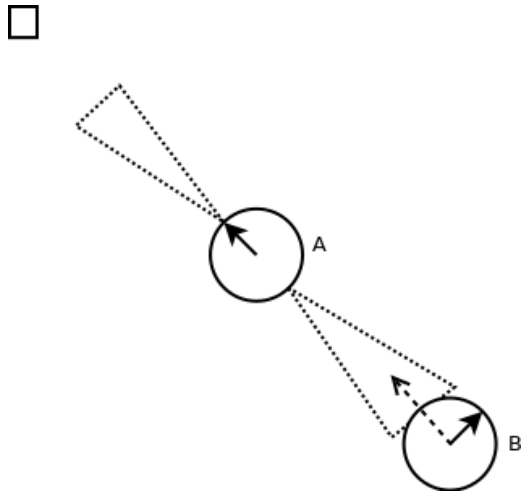

Figure 3.3: Black arrows denote the robot's heading. Robot A has found a landmark (square) and is pulsing its IR-LEDs both towards and away from the landmark at using different pulse-widths. Robot B receives the "away from"-signal through its left receiver, and can using that calculate the direction to the landmark (dashed line arrow).

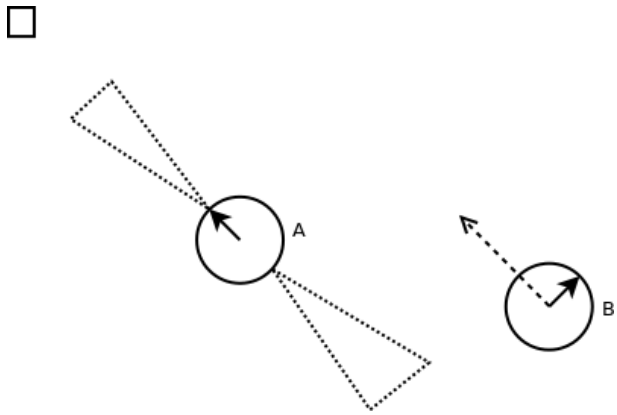

Figure 3.4: Robot B moves without having any distance data. Vector cannot be corrected and remains pointing in the same direction relative to the robot's heading, but no longer pointing towards the target.

3.3.1 Implementation

As it is now, only the most basic functionality is implemented for testing purposes. Only the ATtinys that control the sensors have been programmed to emit a hard-coded type of signal as described above.

What's missing is the interface between the Arduino main processor running the main program and the sensor controllers. Which would allow any program to change and control the message being sent through each of the eight LEDs.

3.4 Communicating (receiving)

This task takes care of measuring the pulse width of incoming IR-light in all directions, filtering, interpreting and translating the results. This part is split into two parts, one running on the main Arduino itself (collecting and using the results) and one on the sensor board ATtinys (measuring, filtering and storing the results).

3.4.1 Measure

Measuring the pulse is done by the sensor-driver ATtinys. They continually measure incoming light in each of the I-receivers. The incoming signal is interpreted as either HIGH or LOW depending on if the light received is over a certain threshold. The last state for each of the receivers are recorded, and if the state differs the next time the light is measured (i.e. a change has occurred), one of several things happens depending on the change, which can be seen in figure 3.5.

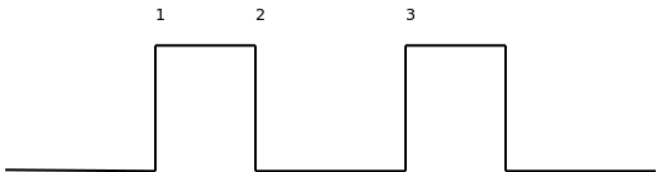

Figure 3.5: When the pulse changes from LOW to HIGH (1), a time stamp is recorded. When the pulse changes from HIGH to LOW (2) the time difference is calculated and stored. Additionally another time stamp is stored for calculating the duration of the LOW pulse. When the pulse changes from LOW to HIGH again (3), that time difference is also stored, and the cycle begins from the top.

Pulse width range (ms)	Message
25 - 34	towards
35 - 44	away from

Table 3.1: Pre programmed messages example

The last measured times are stored for each of the IR-receivers, and can then be retrieved by the main program running on the Arduino.

3.4.2 Filter

In order to reduce the impact of noise from other IR sources (other robots, primarily), some sort of filtering should be implemented. I will use a running median to eliminate outlying measurements. This means I will store the last 3-5 measurements and then calculate the median each time the Arduino requests the results. To help calculate the running median I am using a slightly modified public domain library[13].

Of course, this also means several measurements are required before the Arduino can fetch the results, adding a slight delay. As long as the length of transmissions are kept relatively short, and the number of value stored to calculate the median is not too much, this will still keep the transmission time in the sub-500ms range.

3.4.3 Interpret

When the Arduino itself has received the results from the ATtinys, it needs to interpret them in order to use them. To do this, the program needs a list of pre-programmed messages, and pulse-widths that correspond to them. To deal with inaccuracies in the measurements, each message should have a range of pulse-widths that correspond to them. A simple example can be seen in table 3.1

If for example a robot detects a signal in its front facing IR-detector, and measures it to be 41ms HIGH time and 15ms LOW time. It can then look up in the table and see that it is a signal denoting that it's going away from the target. Since it receives the signal in it's front facing IR-detector, it can conclude that the target lies in front of it. From the 15ms LOW time, it can calculate a distance of 1500 steps (about 39cm) to the target. See figure 3.6.

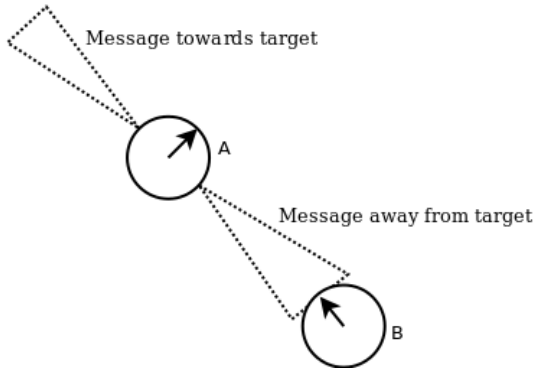

Figure 3.6: Directions of messages

3.4.4 Translate

Of course, I need to take into account the heading of the robot when it receives the message. If the robot is pointing towards the signal, like in the example in figure 3.6, no translation is required. However if the robot is pointing in any other direction when it receives the signal, it needs to translate it, so it makes sense in its coordinate system. If, for example the recipient robot receives the signal through its left IR-detector, like in figure 3.7, it will know that the sending robot's heading is 90 degrees to the left compared to its own heading. the received vector must therefore be translated 90 degrees to the right before it is stored.

3.4.5 Implementation

On the ATtinys, I have implemented the measuring and filtering systems as described above. The actual measuring is performed repeatedly in the loop section of the program. Filtering is performed when the Arduino requests the data. For filtering I used a running median using 4 stored values.

On the Arduino side I have only implemented requesting and outputting the data from the ATtinys for testing purposes, for reasons I will go into more detail with in section 4.3.

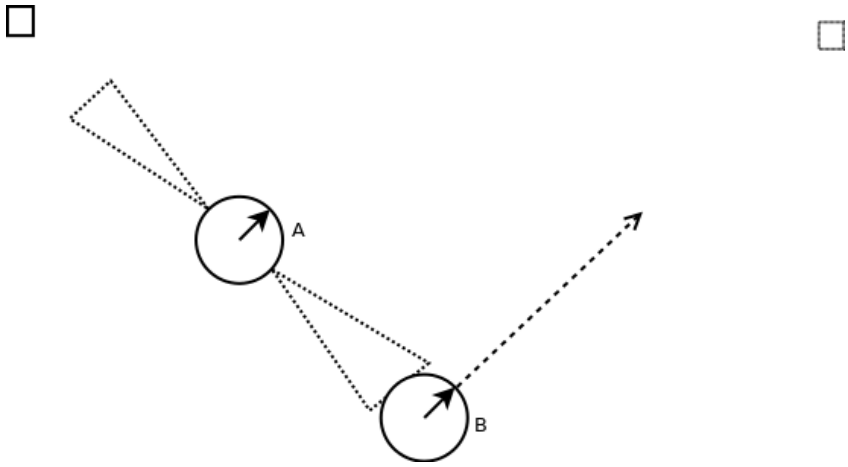

Figure 3.7: Without translation, the robot will think the target is in front of it.

3.5 Avoiding collisions

In order to maintain an accurate vector, avoiding collisions with obstacles or other robots is critical. To detect obstacles, the robots will periodically use their IR-sensors. Obstacle detection works by simply flashing the lights and measuring the amount of light that is reflected back. If the returned light is above a certain threshold, an obstacle is detected.

Something which is important to keep in mind is that the sensor board and IR-LEDs that are needed to perform this obstacle detection is also needed for the communication. Since I want the robots to move while doing communication, It will also need to be running this obstacle detection during communication.

While it would probably be possible to run the obstacle detection in "parallel", and filter out the extra noise caused by the extra flashes on the recipient side, there is a more elegant solution. Since the obstacle detection only requires that the LEDs are on periodically, I can measure the light returned during a pulse while communicating instead. So while the pulse is high, measure any returned light to detect obstacles.

Unfortunately, detecting other robots is not quite that simple. Because of the semi-transparent chassis of the robots, they do not reflect light very well. To detect other robots, the robots can instead try to detect the signals emitted by the other robots since they will also periodically emit light to detect obstacles.

Either that or attach reflective surfaces to the robots.

3.5.1 Implementation

I have only implemented the basic obstacle avoidance, simply turning in a random direction whenever an obstacle detected. But I have not implemented the more advanced features, like obstacle detection during communication.

3.6 Waiting

In some of the experiments I have considered having the robots wait in place for signals from other robots before they continue. This behaviour in itself is rather simple, however it is important to avoid stagnation, so I do not end up in situations where all robots are waiting.

To avoid stagnation I will implement a rule that the robots will not wait indefinitely. They will have a random chance to abort waiting and search for the target every few seconds. This chance can increase over time. This can be used to adjust how many robots should be out searching while the others wait, by adjusting the chance to abort waiting.

This sub-task has not been implemented.

3.7 Approaching the box

In one of my experiments I plan to use a square IR-light emitting box, which the robots will need to approach from the right angle. This box is described in more detail in chapter 4.

If the box has been detected, but the robot is on the wrong side of the box, it will need to find a way to move around and approach it from the right angle. One way to do this is circling around the box at the "edge" of the emitted light, by attempting to keep a distance to the box while moving around it in a counter-clockwise direction.

Can use code similar to line following robots, i.e. continually measuring the light from the box while correcting its direction. To maintain the vector while doing this, the robot should make periodic measurements and discrete movements.

This sub-task has not been implemented.

3.8 Pushing

Related to the previous task, when the robot is aligned correctly in relation to the box, the robot should push. This sub-task also needs stagnation avoidance, and can use a similar technique as was used in the waiting sub-task.

This sub-task has not been implemented.

Chapter 4

Experiments and Results

In this section, I will detail each of the experiments I wish to perform, and how I intend to perform them.

In the following experiments, I will be using one or more of the following components:

- **ChIRP Robot**. This is the ChIRP robot as described in section 2.1.
- **Light shield**. An optional add on to the ChIRp robot that attaches to the top of the robot and contains 8 Light Emitting Diodes as well as 8 light sensors that can detect differences in ambient light. For example from an overhead lamp.
- **Arena**. 121.5 cm by 121.5 cm square walled arena. It has a smooth surface surrounded by gray walls
- **Food source or "box"**. 25cm by 25cm square box made of clear plastic. Inside are 12 IR-diodes, 3 on each side, pointing outwards through the box. The diodes emit a constant light that can be sensed by nearby robots. Depending on the detection threshold in the robots, the range of detection is around 30cm from each wall of the box.
- **Target**. This is either a smaller IR-light emitting target that the robots can sense using their IR-receivers, or an overhead light source that the robots can detect using the light shield. These marks points of interest in the arena, such as nest, food source or similar.

4.1 Experimental Plan

4.1.1 Accuracy, accumulated errors (blind)

Using:

- 1 robot
- Arena

This experiment will measure the accuracy of a single robot running blind (i.e. using only its own distance measures, and obstacle detectors to avoid obstacles). The robot will run a set distance, maneuvering a set amount of times along the way, before attempting to return to its starting point.

The intention of this experiment is to measure the errors accumulated along the way, by measuring its distance from the starting point when it finishes.

By changing the following variables I can determine how each influence the final result:

- **Total distance**. The distance traveled by the robot.
- **Total turn distance**. The total distance turned by the robot.
- **Number of maneuvers**. The number of times the robot stops and turns while traveling

4.1.2 Communication

Using:

- 2 robots

This experiment will test the performance of the simple communications protocol described in section 3.3. Primarily to examine the following properties of the protocol:

- **Reliability of interpretation of incoming signals**. The robot needs to somewhat reliably differentiate between the incoming frequencies in order to use the information. This means I need to measure the following:
- **Accuracy**. The amount of times the interpreted message matches the sent message.

- **Number of different messages**. How does the number of different messages (or total width of the pulses bands) impact accuracy
- **Tolerance for noise**. How does noise from external sources or other robots impact accuracy.
- **Errors during communication**. How accurate is the actual information transferred. What is the worst case scenario in difference between sent angle vs received angle.

4.1.3 Communication + Navigation

Using:

- 2+ robots
- Arena
- 2 Targets

Robots will start within one target, then randomly traverse the arena in search of the other target. When a robot finds the target, it will attempt to travel back home while broadcasting the rough direction of the target to any robot that might wander into it's range. On arrival at the home node, it will continue back and forth between the targets while broadcasting.

Alternatively, some robots can wait at the "nest" (home node) while some robots search the area for the target. They will wait there until one of the scout robots return, analogous to how bees communicate possible nest locations with other bees.

The main object of this test is to compare the efficiency of the system with and without using communication. Thus several runs will have to be made in each case, and the efficiency of the system will be measured in the following ways.

- **Average time to find the target**. My hypothesis is that this time will decrease with the use of communication, as it will in theory limit the search area of the robots.
- **Travel time between the targets**.
- **Failure rate**. I.e time spent in a "failed" state, which means having lost the targets and/or time spend with erroneous information.

4.1.4 Cooperative box pushing

Using:

- 2+ robots
- Arena
- Box
- Target

In this experiment the robots will use all of the defined sub-tasks. The object of the test is to use the navigation to solve a cooperative task, which in this case is to move an object (box) to a target location (nest). The box is too heavy for one robot to move, and thus requires cooperation.

In this experiment all robots start at the "nest". In its simplest incarnation, all robots randomly traverse the arena in search of the box. When a robot finds the box, it will navigate to the side of the box opposite of the target and attempt to push it towards the target. Like in the previous experiment, a variation includes one where most of the robots will wait at the nest, occasionally sending out "scouts" to find the box and report back. Stagnation detection is important here to avoid waiting or pushing indefinitely.

In order to cope with the many inaccuracies, this experiment can also be attempted to be done iteratively by the robots, in the sense that they will periodically abort and reset. So if the robots are pushing the box in the wrong direction, or even the right direction, they will abort and restart. My hypothesis is that this may allow the robots to make corrections if off course, and gradually push the box towards the target.

Again, the main object of the experiment is to compare the system with and without using communication. This also requires measures of efficiency of the total system, which can be done in the following ways:

- **Time to achieve the goal**. Which in this case is to push the box to the target location.
- **Time to find the box**.
- **Failure rate**. As with the previous experiment.

4.2 Experimental Results

4.2.1 Accuracy, accumulated errors (blind)

To measure the errors accumulated in this experiment, I set up the arena with a paper target denoting the starting point of the robot. In this target I marked the center of the robot (middle of the wheels) and wheel starting positions, so that the robot started in the same position each time.

I then set the parameters I defined in section 4.1.1 and placed the robot in the center of the target. The robot then made its maneuvers and eventually returning to the paper target, where I marked its new wheel positions. I repeated this 10 times for each set of parameters.

I then took the paper target and marked the center of each of the sets of wheel positions. Then I measured the distance from each of these center positions to the center position of the starting point. A sample of this can be seen in figure 4.1.

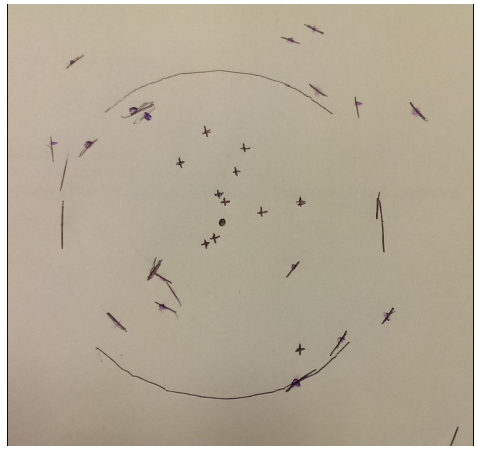

Figure 4.1: Sample of the results from one of the experiments. Shows the spread of final ending points (center of final wheel positions, marked by +) and the central starting point (marked by solid dot). The other lines are ending wheel positions

The results from these experiments are summarized in table 4.1

Maneuvers	Steps / Maneuver	Rotation / Maneuver	Total distance (cm)	Avg Error (cm)	Median Error (cm)
4	1000	120	103.9	1.4907	1.5
8	1000	120	207.9	1.667	1.67
12	1000	120	311.7	8.19	7.68

Table 4.1: Error measurements

4.3 Communication

In this experiment I was supposed to evaluate the feasibility of my communications protocol for use with my later two experiments. To do this I outlined a number of possible issues in section 4.1.2 that I wanted to examine, which I will go through stepwise here.

4.3.1 Accuracy

Accuracy in this sense mean how often the senders intended message matches the recipients interpreted message. I.e how often the measured pulse width falls within the window corresponding to the message intended by the sender. This depends on the accuracy of the measurements.

I measured the accuracy by transmitting the same message roughly 5000 times from one robot to another, using different pulse widths and outputted the pulse width measured by the recipient robot. I took this data and counted the number of occurrences of each of the measured pulse widths.

In the first test I set the targets high, pulsing 333ms HIGH, then 300ms LOW. The results can be seen in figures 4.2 and 4.3.

In the second test I set the targets in the lower range: pulsing 30ms HIGH and 40ms LOW. These results can be seen in figures 4.4 and 4.5.

There are some odd results in these tests. Especially in the latter two cases, where the target value was outpaced by its neighbors. However, all of the measurements fell within a window of $\pm$ 3ms from the target value. This is sufficiently accurate to hit the window I defined in the communication example in section 3.1.1 of $\pm$ 5ms.

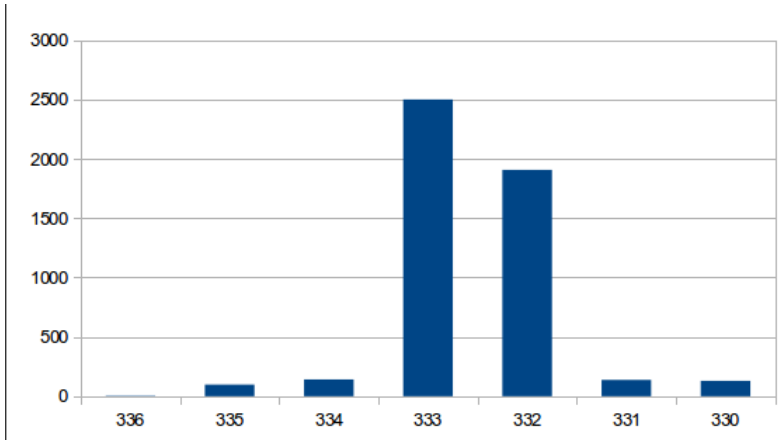

Figure 4.2: Test results from the first test. This is the HIGH portion of the signal. the target was 333ms. These are all the measurements, i.e. there were no occurrences of values outside of the chart.

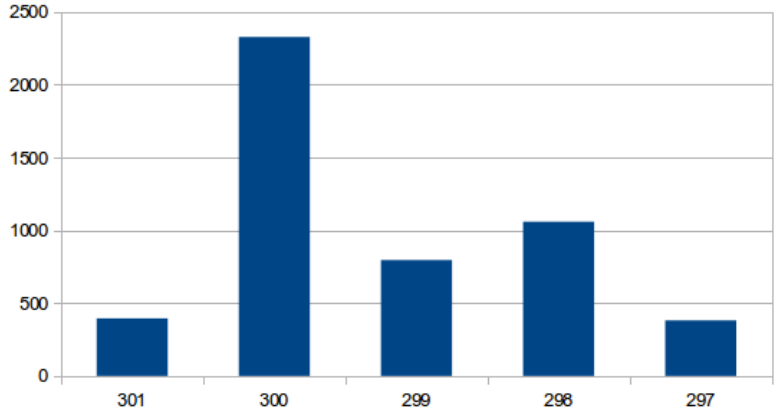

Figure 4.3: Test results from the first test. This is the LOW portion of the signal. the target was 300ms. Again, These are all the measurements, i.e. there were no occurrences of values outside of the chart

4.3.2 Number of possible messages

The number of possible messages is limited by the accuracy of the pulse width measurements. Based on the accuracy measured above, I can assume that I will

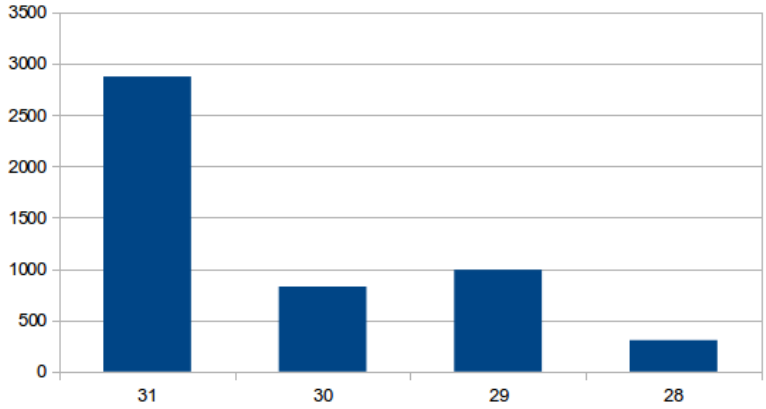

Figure 4.4: Test results from the second test. This is the HIGH portion of the signal. the target was 30ms. These are all the measurements, i.e. there were no occurrences of values outside of the chart.

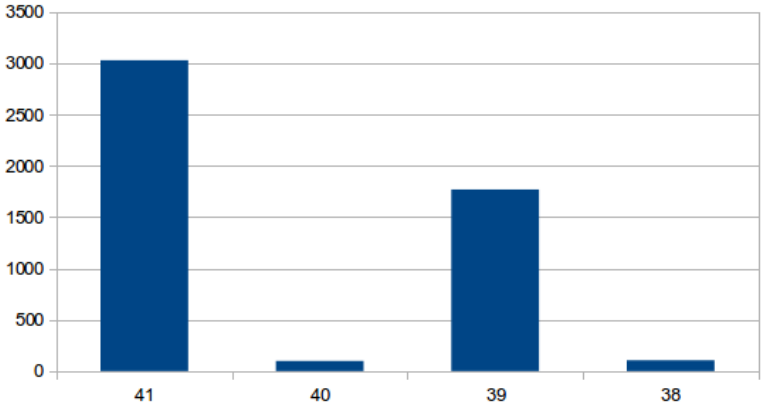

Figure 4.5: Test results from the second test. This is the LOW portion of the signal. the target was 40ms. Again, These are all the measurements, i.e.there were no occurrences of values outside of the chart

need a window of at least 7ms per pre-defined message (target $\pm$ 3ms).

In theory you can have a pulse width as long as you like, and thus also have as

many possible messages as you like, but, of course, a longer pulse-width means it will take longer to transmit the message. So it's preferable to keep the maximum pulse-width as short as possible. In my case, I considered using at most 8 different messages (one for each of the IR-LEDs. In that example I would need at most a 7ms*8 = 56ms pulse width, which is an entirely reasonable time frame in this context.

4.3.3 Errors due to noise

Unfortunately, due to a lack of robots available to me when I was testing this, I did not have an opportunity to test this properly.

4.3.4 Maximum error during communication

This is where my biggest problems showed itself during testing. Of course, when transmitting these directional messages, there is no guarantee that the robots are perfectly aligned, i.e. the sending robot is pointing directly into the recipient robot. I had considered this problem, but this system was never intended as a particularly accurate system. It was enough that the robots could point each other in the general direction of the target, so the robots could at least start searching in the right area.

To calculate the maximum error I measured both the spread of the IR-light from the diodes, as well as the field of view (i.e. the maximum angle from which it can receive signals) of the IR-detector. The spread from the IR-light I measured to be $\pm$ 12 degrees around the center (roughly 25 degrees total), which is within reasonable limits. However, when measuring the field of view of the IR-sensor, I found it to be about $\pm$ 60 degrees around the center (figure 4.6), which is too large spread to be useful for navigation.

This amount of spread makes the worst case scenario not very useful for navigation, as seen in figure 4.7

I tried several solutions to reduce this problem. One was to make some sort of blinders out of paper, to make a tunnel around each of the IR-receivers to narrow down the angle at which it could receive signals. However, for one this worked very poorly and irregularly, but it was also very finicky to do, and goes against one of the main principles of the ChIRP robot, namely ease of assembly and use. I discarded this idea after testing it briefly and noting little to no improvement.

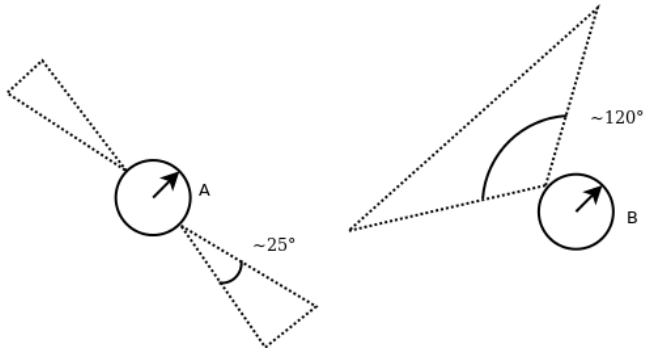

Figure 4.6: IR-LEDs spread (A) and IR-receiver field of view (B)

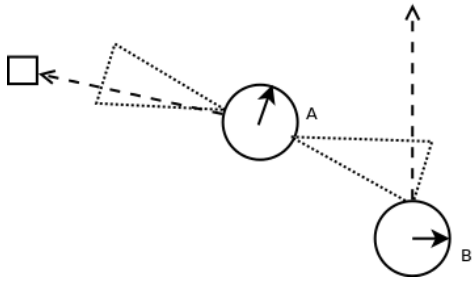

Figure 4.7: Worst case scenario. Receiving the signal at the maximum possible angle. Dashed lines represent the direction each robot calculates the object is in.

Another solution I tried was to use multiple sensors detect whether the signal was coming in from a wide angle. If the exact same incoming signal hit more than one neighboring IR-receiver, you could safely assume the signal was coming in from a wide angle and could thus be discarded. However, the design of the chassis, as well as the spread of the IR-LEDs makes this solution less useful. The chassis obscures some of the detectors from wide angles, meaning that if the light hits the neighboring detectors at a wide angle, the obscured detectors won't detect it. In addition, the spread of the IR-LEDS means that from further away, the light is going to hit more than one detector, even if it shines directly into one.

In the end I could not find a satisfactory solution to this problem without switching out the IR-sensors to ones with narrower fields of view. Because of this I could not complete the final two experiments.

Chapter 5

Evaluation and Conclusion

5.1 Evaluation and Discussion

Unfortunately, I never got to test my complete system properly. However, I did learn several things during this project.

For one, I was surprised at how accurate the bare path integration part worked. Inaccuracies of between than 1.5cm and 8cm after traveling blind up to 3 meters was better than I had expected, and more than accurate enough for my purposes. Of course, it is important to keep in mind the environment in which these tests were performed. In an entirely flat arena, devoid of anything the robot could get stuck on or collide with. It is clear that this most basic method would not work very well on rough terrain. At least not without additional sensors such as an accelerometer and compass to make corrections along the way. Of course, in my case it is a bit of a moot point, as the ChIRP robot is not capable of traversing rough terrain to begin with.

This method can still be useful in indoors environments, also in ones that are much larger than the arena I used.

As for the communication, my method of directional communication obviously did not work out, which is disappointing, because it is because of a problem I should have seen much earlier than I did. Still, I think the underlying protocol can be useful in other applications where you would need a quick method to

communicate pre-defined messages or small amounts of data between robots, with the condition that the data sent does not need to be completely accurate.

5.2 Conclusion

Although I never did find out what I set out to find out in this project, I still learned a few useful things.

First, path integration/dead reckoning is not as inaccurate as i first assumed, even running blind. I think it is feasible to use this in the future in other swarm applications.

Second, as mentioned above, while my communications protocol failed to work as intended in my project, I think the idea can still work in other applications where the direction of the incoming transmission is not important. As a way to quickly broadcast a message and a small amount of data to neighboring robots with very little overhead, I imagine it can be very useful in other swarm applications.

5.3 Future Work

There is of course a lot of work that is possible to do on this project.

First and most importantly, is to find a workable solution to communicate directions between robots. One solution could be to use different IR-receivers with narrower fields of view. Of course, this would mean replacing the ones on the robot, which is not really feasible on the ChIRP robots.

Another possible solution would be to align the robots' IR LEDs and receivers before transmitting, by turning on a LED and then rotating slowly in place until the robot finds the point at which the signal is the strongest. At which point the two robots should be aligned to a sufficient degree. Thes means a significant increase in overhead in the communications protocol, of course.

One of the biggest issues with this blind navigation, is how easily all the data is rendered useless if the robot collides with or snags onto something. The best way I think to alleviate this problem would be to attach a compass to the robot, to make corrections whenever the robot is thrown off course.

Bibliography

[1] Håvard Schei, Robert Versvik, Anders S Rye. *CHIRP: A small, modular, low-cost robot for swarm research and education.* Specialization project, NTNU, fall 2012

[2] Christian Skjetne, Pauline C Haddow, Anders S Rye, Håvard Schei, Jean-Marc Montanier . *The ChIRP Robot: a Versatile Swarm Robot Platform.* The 2nd International Conference on Robot Intelligence Technology and Applications 2013

[3] ChIRP website: http://chirp.idi.ntnu.no/

[4] Collett, Matthew, and Thomas S. Collett. *How do insects use path integration for their navigation?.* Biological cybernetics 83.3 (2000): 245-259.

[5] Von Frisch, Karl. *The dance language and orientation of bees.* (1967).

[6] Ronacher, Bernhard, et al. *Lateral optic flow does not influence distance estimation in the desert ant Cataglyphis fortis.* Journal of Experimental Biology 203.7 (2000): 1113-1121.

[7] Esch, Harald, and John Burns. *Distance estimation by foraging honeybees.* Journal of Experimental Biology 199.1 (1996): 155-162.

[8] Schmickl, Thomas, and Karl Crailsheim. *Trophallaxis within a robotic swarm: bio-inspired communication among robots in a swarm.* Autonomous Robots 25.1-2 (2008): 171-188.

[9] Nouyan, Shervin, and Marco Dorigo. *Chain based path formation in swarms of robots.* Ant Colony Optimization and Swarm Intelligence. Springer Berlin Heidelberg, 2006. 120-131.

[10] Jannik Berg, Camilla Haukenes Karud *Swarm intelligence in bio-inspired robotics.* Master's thesis, NTNU, 2011

[11] Gross, Roderich, and Marco Dorigo. *Towards group transport by swarms of robots*. International Journal of Bio-Inspired Computation 1.1 (2009): 1-13.

[12] Kube, C. Ronald, and Eric Bonabeau. *Cooperative transport by ants and robots*. Robotics and autonomous systems 30.1 (2000): 85-101.

[13] robtillaart, *A runningMedian Class for Arduino*. http://playground.arduino.cc/Main/RunningMedian