ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

RESEARCH GROUP FOR
Distributed Systems

Bachelor's Thesis

at the Institute for Pervasive Computing, Department of Computer Science, ETH Zurich

Distributed Algorithms for Kilobots: Constructing a Global Coordinate System from Local Information

by Tobias Kaiser

Autumn 2013

ETH student ID: 10-923-704
E-mail address: kaisert@student.ethz.ch

Supervisors: Dr. Christof Roduner
Prof. Dr. Friedemann Mattern

Date of submission: 28 February 2014

Declaration of Originality

I hereby declare that this written work I have submitted is original work which I alone have authored and which is written in my own words.
With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette:'

`http://www.ethz.ch/students/exams/plagiarism_s_en.pdf`

The citation conventions usual to the discipline in question here have been respected.
This written work may be tested electronically for plagiarism.

Zurich, 28 February 2014

Tobias Kaiser

Abstract

The relatively new field of swarm robotics is emerging and developing rapidly. The primary intention of swarm robotics is the coordination of a large collective of robots. The Kilobot is a swarm robot developed by the Self-organizing Systems Research Group at Harvard university. The Kilobots' strengths and weaknesses lay in its simplicity: low computational power, limited movement and communication abilities, but its cheapness and maintainability allow implementing and testing distributed algorithms on a big swarm at an affordable price.

This thesis shows an implementation of constructing a global coordinate system within a swarm of Kilobots. Global in the sense that it is known and accepted amongst the entire swarm. The only information comes from neighbour-to-neighbour communication and measurements of distance between neighbours.

As these swarm robots neither have an actual sense of their environment, nor do they know their exact position within the collective, it is crucial for many algorithms to first decide on some kind of orientation. Once the coordinate system is established, many new possibilities emerge, such as building shapes, foraging or displacement of the robots.

Acknowledgment

My special thanks go to professor Pekka Orponen from university Aalto, who provided the great opportunity, facilities and fundings, which made this thesis possible in the first place. Also to Antti Halme, with whom I enjoyed many vivid discussions, which helped me a great lot to accomplish my work.

I also want to thank professor Friedemann Mattern from ETH Zürich for letting me write my thesis in Helsinki and Christof Roduner for supervising my work.

Contents

1 Introduction

This thesis presents the work done on attempting to implement a solution to construct a global coordinate system within a collective of Kilobots. The algorithmic approach was designed by Michael Rubenstein [8]. The intended platform for the implementation of his work is the Kilobot [9]. During the construction, every member of the swarm runs with the same program and starts with the same amount of information. No main controlling unit was being used and no roles were assigned in advance. The Kilobot merely provides simple functionalities, such as computing, moving and exchanging of messages, and the range of communication is very limited. This is the reason why it is only possible to operate on local information.

Swarm robotics, as a discipline, is relatively young and defined by its design and study of robots organizing through extensive interaction and collaboration. Main inspirations are organisms such as social insects, like ants or bees, but also living cells growing in and forming a body. The sequence of executions of one single robot of the collective might seem nondeterministic and insignificant, but together a swarm achieves goals, which lay beyond the abilities of one single member operating on its own. Common tasks are assembling into shapes, exploring their surrounding, foraging or mining, where the latter two have not yet been implemented beyond simulation. Swarm robotics emphasizes decentralized, error-prone and highly scalable approaches.

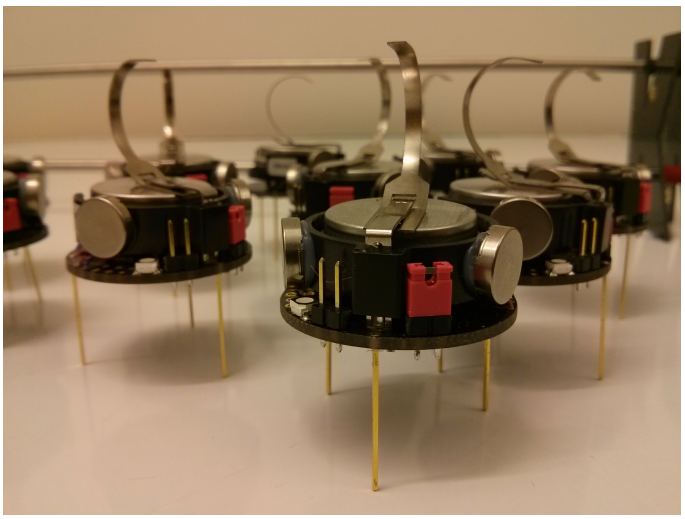


Figure 1.1: A swarm of Kilobots.

The Kilobot was designed with the purpose of providing a low cost swarm robot. It measures approximately 2.5 cm in diameter and is about 3 cm tall. As stated above, it comes with rudimentary moving, communication and sensing systems.

This thesis first presents a problem statement. After that, a small overview of similar work is provided, as well as some other swarm robotics platforms are introduced. It is followed by a detailed description of the algorithm used in this thesis. In the implementation part, first the specifications of the Kilobot is described in greater detail, followed by the challenges we encountered during the implementation and the results we gained. The last part of the chapter is a description of our implementation. Ultimately the reader can find a discussion of our work, and the thesis ends with a summary of its conclusions.

2 Problem Statement

The basic principle of swarm robotics is to make a collective of robots achieve a goal, which a single member of the swarm is not able to fulfil on its own. This approach heavily depends on collaboration. As a robot does not have any knowledge about its environment, location or number of neighbouring robots, there needs to be some constant structure, which provides at least a minimum of orientation. For example given the task of building a shape with a given swarm requires some sort of positioning system. One possibility to approach this problem is to create a global coordinator, which has total knowledge about the swarm and all of its members and therefore is able to instruct and command all of its followers. But since swarm robotics emphasizes decentralized solutions, having a leader over the whole swarm should be avoided. Another solution is to have a coordinate system, known and distributed amongst the whole swarm. With this coordinate system, every robot has knowledge about its exact location with respect to their fellow robots. Once this information is established, any shape building or localization of objects of any sort is feasible.

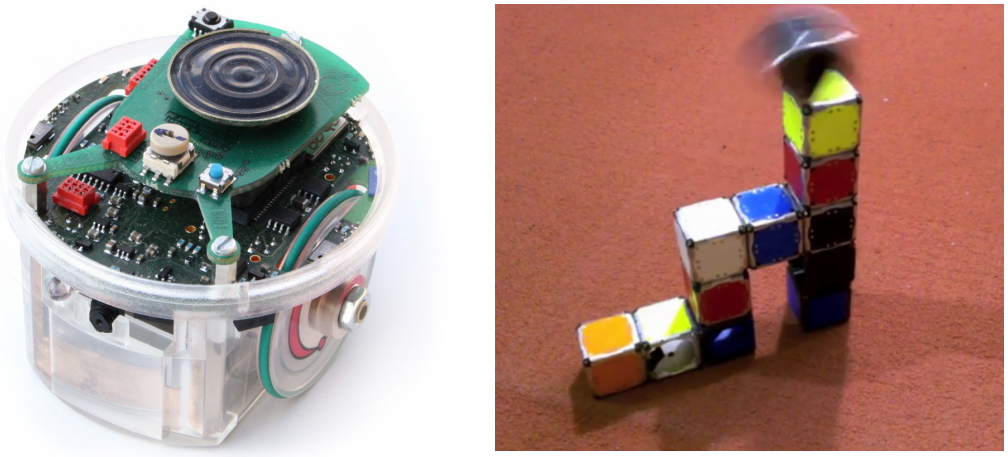
This thesis tackles the problem of constructing a consistent, global coordinate system within a swarm of robots. The intended platform is the Kilobot, which is defined by its simplistic and minimalistic design. As stated before, there is hardly any global knowledge amongst the members of the collective. As the communication radius of a single Kilobot is easily outranged by the area of the whole swarm, the construction can only rely on local information. While the intended task is easily feasible for a swarm of only three or four members, a more sophisticated approach is needed, as soon as no direct communication between two robots is possible any more. The algorithmic approach is described in the work of Michael Rubenstein [8]. As the Phd thesis discusses the theory, this Bachelor's thesis takes a practical approach and implements the construction of a global coordinate system. The algorithm emphasizes on scalability, meaning there is theoretically no limit to the size of the swarm.

3 Related Work

This chapter presents some common challenges met within swarm robotics in general, and the attempted approaches to solve these. In the first part different types of swarm robots that are currently used are presented and in the second part some general self-organizing and self-assembly problems are shown.

3.1 Swarm Robotics

Currently multiple swarm robotics platforms are being used. One of the most popular is probably the e-puck [1]. E-puck was developed at the École Polytechnique Fédérale in Lausanne (EPFL) with a purely educational purpose. It comes with various sensors, such as infrared proximity sensors, an accelerometer, microphones, a color camera, and actuators like stepper motors, a speaker and LEDs. Additionally several extensions can be connected to the e-puck, which even increases its functionalities.



(a) (b)

Figure 3.1: Picture of an E-puck (3.1a) and formation building of M-Blocks while mid-jump (3.1b).
Photo by Stéphane Magnenat, 2008. Frame taken from video ”M-Blocks Modular Robots” by John Romanishin.

The CSAIL group at the Massachusetts Institute of Technology recently developed the M-Block [7]. M-Block is a cube-shaped robot, which is able to connect to other M-Blocks through magnets on its sides and edges. It has no external moving parts and moving is possible through a flywheel enclosed within the cube. The flywheel is spun in the cube and abruptly stopped. Using the momentum, the M-Block is able to pivot and even jump from one location to another. With its magnets, a collective of M-Blocks is able to self-assemble any arbitrary shape. The current model is not actually programmable, but rather receives commands from an external computer. Future models will have the necessary specification such as a computational unit to run independently and decentralized as well.

The s-bot [6], also developed at EPFL, is one of the swarm robots, which supports actual physical interaction. It can interconnect with other s-bots with either a rigid gripper or a semi-flexible arm also ending in a gripper. While the rigid gripper is mostly used to form large chains of s-bots to overcome obstacles, the arm connector is used when a level of mobility is required between the robots. It is also possible to grab other objects using the grippers. Furthermore the s-bot is equipped with a camera, infrared proximity sensors, light sensors, humidity sensor, accelerometers, incremental encoders as well as torque sensors. It also features color detectors, a microphone and speakers. Messages can be exchanged via wireless LAN.

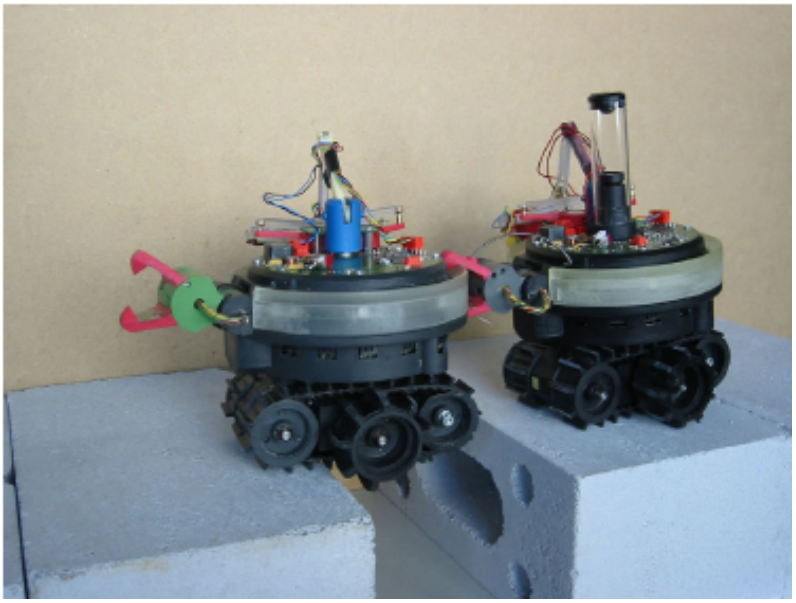


Figure 3.2: Two s-bots crossing a gap, while being connected with the rigid gripper. Source: [6].

Not yet fully published, but under vivid development is the robobee [1] (figure 3.3) at Harvard's School of Engineering and Applied Sciences. It is based on a bee as it appears in nature. It is a flying device weighting only 80 mg and has a wingspan of 3 cm. The wings reach a frequency of 120 Hz. Orientation is possible through vision sensors. The

[1] http://robobees.seas.harvard.edu/

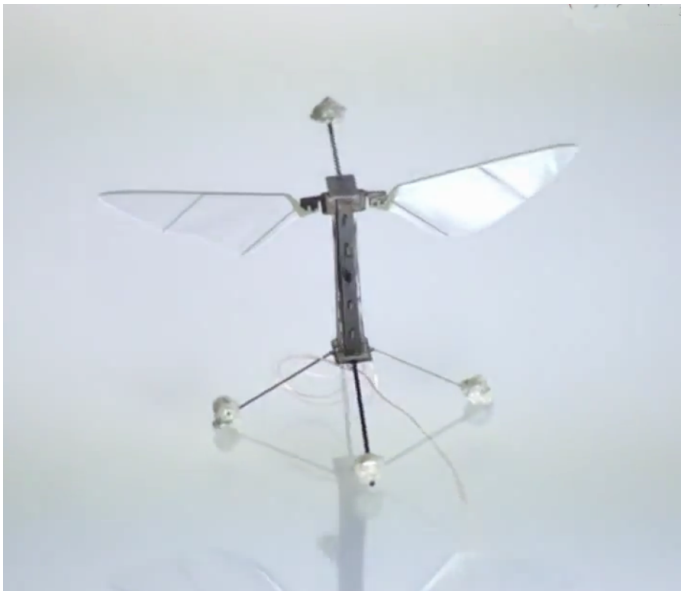


Figure 3.3: A robobee developed a research team at Harvard's School of Engineering and Applied Sciences. Frame taken from video "Flight of the RoboBee: Flying Robot Insect As Small As A Penny".

potential application areas of a robobee include pollinating trees, and thus effectively replacing natural bees if necessary, searching and recovering people after natural disasters, exploration in general, and surveillance.

3.2 Self-Organizing and Self-Assembly

There are various approaches to constructing consistent global coordinate systems within a collective of robots.
One of them is described in [14], where initially one dedicated seed robot is set at a predefined location. Whenever this seed robot is seen by a neighbour, a robot computes its coordinates by adding the vector between itself and the seed to the seed's coordinates. As soon as a robot succeeds in localizing itself, it becomes a seed as well and is able to help its neighbours to localize themselves. In the end, a coordinate system relative to the initial seed's position is spread over the whole swarm. One disadvantage is, that the seed's role needs to be assigned prior to the start of the execution. Thus the approach holds a single point of failure.

One approach for achieving self-assembly, fault-tolerance and self-repair is presented in [12]. Spears and Gordon use artificial physics to compute interactions between two agents, i.e. physical or virtual. Using artificial physics, allows distributed control over a collective of these agents. The proposed model uses Newton's law of universal gravitation. With this law either an attractive force is calculated, i.e. when two agents are too far from

each other, or a repulsive one, when the distance is too small. The aim is to model a hexagon and a square with the agents. Though the shapes can be perfectly assembled in a group of seven, respectively four agents, the global shape of the collective is somewhat arbitrary. To solve this problem, some global information was added to the model. Namely, every agent got a position, i.e. (x,y) coordinates. During the assembly, the agents sort themselves, while simultaneously applying the initial forces. So using distributed control, distributed computation was performed, which resulted in a perfect global hexagon or square respectively.

For shape formation one might not only consider self-assembly, but also self-disassembly. An example is described in [2]. First Gilpin et al. introduces a modular robotics system, called Miche. Like the M-Block, Miche is a cubic robot, which can build shapes using magnets on its sides to hold onto each other. Differences to the M-Block are, that it does not contain a flywheel within, it has its own computational power and for displacement it must rely on external forces, such as gravity. The process of forming a shape consists of four phases: neighbour discovery, localization, shape distribution and disassembly. During the neighbour discovery, the modules get manually assembled, while via message exchange each Miche robot registers its neighbours. During the next phase, each robot computes its location within this initial formation and sends the data to a Matlab program. Using this program a shape can be sculpted, which is then distributed to the robots during the shape distribution phase. In the final phase, every Miche not needed for the final sculpture disconnects from its neighbours, leaving the intended form behind.

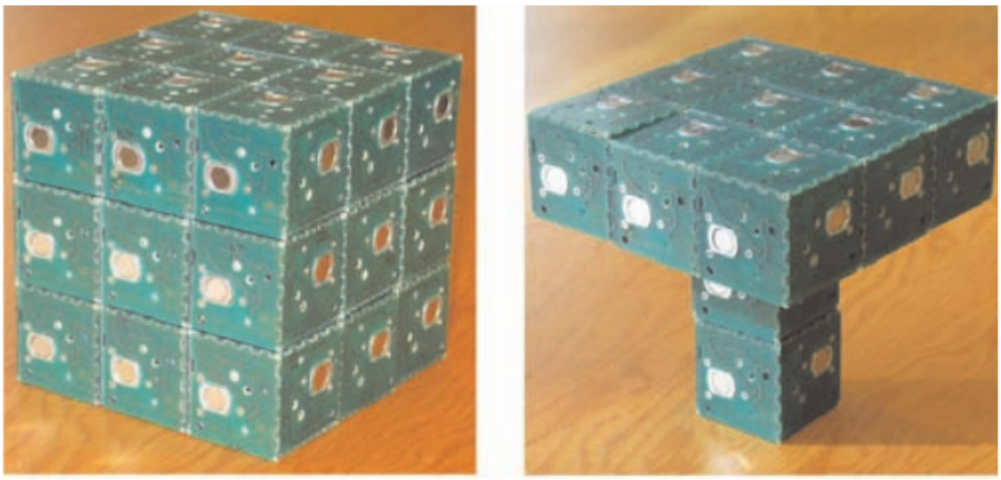


Figure 3.4: A collective of Miche robots in their initial setting (right) and in the determined shape after the disassembly (left). Picture taken from [2]

Shen et al. developed and introduced a digital hormone model [10] for solving the distributed control problem. It is inspired by various observations in nature, such as the forming of feathers in organisms. The principle is, that each member of the swarm dispenses hormones depending on its location and on the goal of the swarm. Based on a probabilistic function each robot computes its behaviour when encountering certain

hormones. The dispensation of the hormones is calculated with equations for hormone reaction, diffusion and dissipation. The model functions without assigned identifiers and works completely decentralized. The approach is highly eligible for foraging and gathering tasks, but complex shapes cannot be modelled precisely using the digital hormone model.

In [4], Hoff presents an approach on how to simulate foraging behaviour with a swarm of robots. Namely, the task is to find a source of "food" starting from a home location, the "hive". When found, the food needs to be delivered back to the hive. Hoff proposes three algorithms to solve this problem.
The first one is based on virtual pheromones. To simulate the pheromones, some robots become beacons, which spread out. When a robot encounters a beacon, it places its pheromones there. The pheromones are represented by a system of floating point numbers. The hormones will slowly decrease their values over time, meaning, less important pheromones decay gradually, which results in a trail leading from the hive to the food source. While the robots are foraging, they follow this path and simultaneously amplify the pheromones.
The second method, using a gradient, is similar. As before, beacons are employed, but this time they indicate how many beacons there are in between them and the origin, and how far away the food source is. With these indicators a simple gradient is generated, which a robot merely needs to follow to bring food from the source to its home.
The last algorithm is to find food by forming a line of robots, which then sweeps in a circular fashion around the hive. When the food source is encountered, the gradient algorithm is applied to transport the food from the source to the hive. After the description of his theory, Hoff continues with the implementation of each algorithm with a small swarm of e-pucks, and also the implementation of the gradient approach with a collective of Kilobots.

4 Algorithm Outline

In this chapter the algorithm implemented is being presented. The algorithm used in this thesis was proposed by Michael Rubenstein in his Phd thesis "Self-assembly and self-healing for robotic collectives" [8]. The way described to construct a global coordinate system is to build many local coordinate systems at first, which then are gradually merged into one global, unique coordinate system. The algorithm can be split into six different phases. In the following explanation, the same terminology as in [8] is used.

In the first phase, IDs are constructed, which serve the purpose of local identification. This is followed by the election of so called seeds. The requirements are that every non-seed robot sees at least two seed robots, as these seed robots will each construct a local coordinate system in phase 3. This happens by recruiting reference robots, which help to establish the x- and y-axes. Once these local coordinate systems are built, other Kilobots around the seed can localize themselves in the different coordinate systems. In phase 5 merging groups are created. Merging groups consist of three robots and are dedicated to two seeds each. These merging groups are used in phase 6 to gradually merge all the different local coordinate systems, until only one single, global coordinate system exists.

4.1 Phase 1: Construction and Distribution of locally unique identifiers

As no robot is initially assigned an individual identifier, the first phase is to construct a locally unique ID. The routine is as follows: first a random ID is generated and then broadcasted to the robot's neighbours. Every new identifier received gets stored. If an ID is encountered twice from two different robots, a message is sent, which causes those two robots to construct a new ID. Additionally these robots broadcast that they acquired a new ID, which leads to the neighbours dropping the stored ID and saving the new one. A robot constructs new IDs until it is certain that the assigned ID is locally unique. With locally unique meaning that no robot sees two robots, which share the same ID.

4.2 Phase 2: Seed Election

During the execution, a robot can have two different roles, either the one of a seed or that of a non-seed. In the beginning, no robot is assign to a role. A seed robot's task is to construct a local coordinate system. To ensure that enough of these local coordinate systems are distributed amongst the swarm, every non-seed robot must eventually see at least two seeds. Similarly, every seed must see at least one other seed.

To elect the seed robots, two protocols are executed, one for the so called top seed election and one for the bottom seed election. Starting with the top seed election, every robot sends its ID to its neighbours. Whenever a robot receives a value greater than its own ID, it does not become a seed. If none of the messages contain a greater ID, the robot becomes a top seed. The remaining non-seeds continue with the bottom seed election protocol. The idea is the same as before: Every non-seed broadcasts its ID and becomes a seed if no message is received that indicates that a neighbour is assigned a greater ID. Once a robot becomes a seed, it starts to announce its new role. In the end of the bottom seed election protocol, every robot checks whether it is able to see enough seeds. If this is not the case, it broadcasts a message, inducing a new bottom seed election with its neighbours.

This procedure is repeated, until every robot sees at least two, respectively every seed at least one other seed robot. The roles of the top and bottom seeds are identical, and they are not distinguished any more in the phases below.

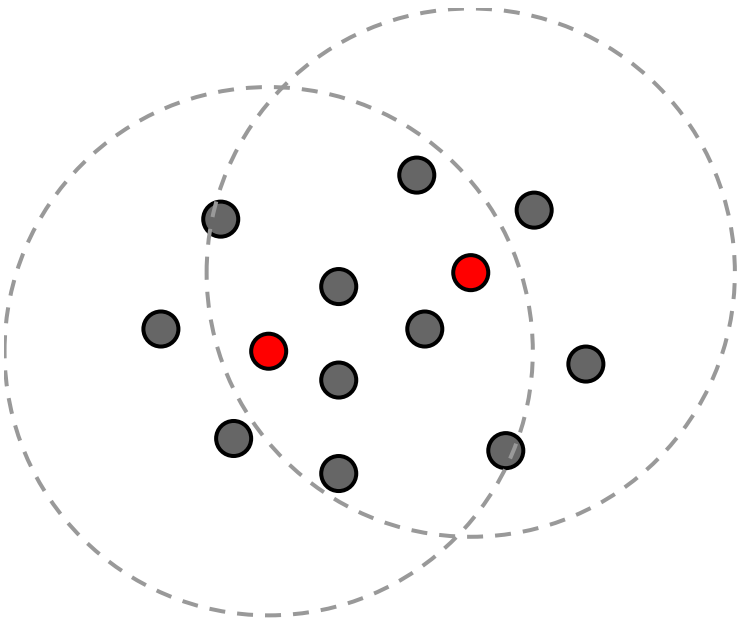


Figure 4.1: Possible excerpt of a collective. Red: seed robots with respective communication ranges.

4.3 Phase 3: Constructing the local Coordinate Systems

The next step for the seeds is to recruit reference robots. Each seed needs two references which must be able to communicate with each other. With these references, it is possible with trilateration to construct an x- and a y-axis, which then define a coordinate system. It does not matter, whether a reference robot is a seed itself or not. The seed (A) chooses its references (B, C) in such a way that the angle α in the triangle ABC is maximized, while B and C are still in each other's communication range. To do so, the distance between every potential reference pair must be inquired. Once the best pair is found, the seed can calculate the coordinates of each reference. The seed is at the origin of the system. Reference robot B gets located at $(\overline{AB},\ 0)$. With the y-axis being orthogonal to the x-axis, the coordinates of C is computed as $(\overline{AC}*\cos\alpha,\ \overline{AC}*\sin\alpha)$. A then transmits the corresponding location to its references.

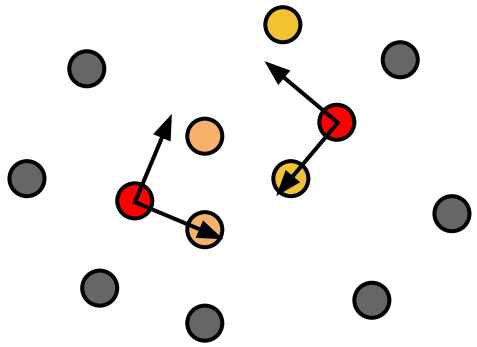


Figure 4.2: Two seeds with their local coordinate systems after recruiting reference robots (yellow and beige).

4.4 Phase 4: Localization of Robots

After a robot is localized in a coordinate system, it starts distributing its newly acquired coordinates. Whenever a robot sees two localized robots as well as the dedicated seed robot, it is able to compute its position in this local coordinate system. Once this is done, it also starts to announce its location, and therefore helps other robots to locate themselves.

4.5 Phase 5: Creating Merging Groups

To merge all the local coordinate systems, merging groups are constructed. A merging group consists of exactly three robots, which are within the communication range of each

other. The roles of the robots does not matter. These three robots must share two local coordinate systems, in which they are located. Supposing that robot X is located in the coordinate systems of the seeds K, I and H and robot Y in the ones of G, K and I. Then robot Z, assuming X, Y and Z can see each other, must be located in at least K's and I's coordinate system in order to form a merging group consisting of the members X, Y and Z. A robot can be part of an unlimited amount of merging groups, and two merging groups can also overlap. It might even happen, that two merging groups are identical, but assigned to different seeds. When the merging groups are built, they register to the two seeds they associate with.

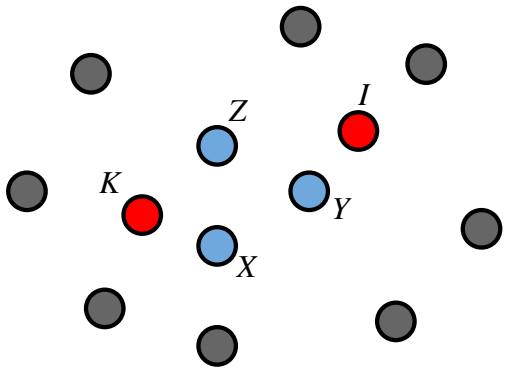


Figure 4.3: Two seed robots (K and I) with a common merging group (blue).

4.6 Phase 6: Merging the Coordinate Systems

Summarizing, up to now, every seed has constructed its own coordinate system. At least two other robots are located in that coordinate system and at least one merging group has registered to each seed. To describe its local coordinate system, a seed stores a vector $\vec{v}_{\text{seed}}$ and a 3x3 rotation matrix R_{seed}. $\vec{v}_{\text{seed}}$ is the shift of the origin of the coordinate system and R_{seed} is seen as the rotation of the initially created x- and y-axis. The location of a seed robot in its own local coordinate system can then be computed as $R_{\text{seed}} * (-\vec{v}_{\text{seed}})$. Initially $\vec{v}_{\text{seed}}$ is equal to the zero vector and R_{seed} is the identity matrix. This means, that in the beginning the point of origin of each coordinate system is located at the position of its seed. Assuming robot Z is located in seed A's coordinate system then the actual location of Z in A's coordinate system can be computed as $R_{\text{A}} * (\vec{x}_Z - \vec{v}_{\text{seed}})$, where R_{A} and $\vec{v}_{\text{A}}$ are R_{seed}, respectively $\vec{v}_{\text{seed}}$ of seed robot A and $\vec{x}_Z$ are the coordinates of robot Z in A's coordinate system, calculated in phase 4.

Now in order to gain one global coordinate system, the origins of all coordinate systems must be moved to the same location and handedness of the axes must be rotated such that they match. To achieve this, the coordinate systems get merged gradually. Phase 6 consists

of several rounds. In each round, the coordinate systems get shifted more and more to one common point of origin and the axis get rotated.

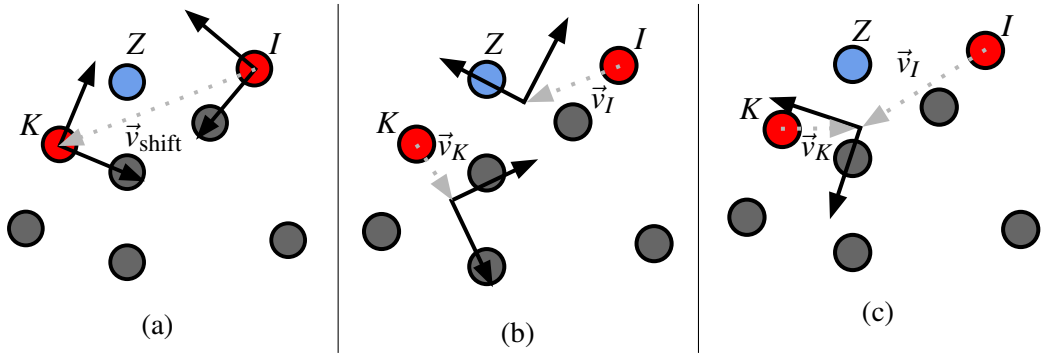


Figure 4.4: Mechanism of merging: in 4.4a I inquired an update from one member (Z) of a merging group common with K. 4.4b shows the coordinate systems after the first round. I's origin got closer to K and K had shifted its system to another seed. After several rounds in 4.4c the coordinate systems are merged and every origin is set to the same location.

For that purpose the merging groups were constructed. As the members are able to communicate with each other, they are able to exchange their coordinates in the local coordinate system of the seeds, I and K. With this knowledge, a three dimensional rotation matrix can be computed, which translates a vector from the coordinate system of I to the coordinate system of K. In order to do so, the TRIAD method [3] can be used. From this matrix the rotation R_{parallel} can be computed, which shows the rotation of I's coordinate system, such that it lies parallel to the coordinate system of K. Secondly, a vector $\vec{v}_{\text{shift}}$ can be computed, which denotes the shift of I's zero-point to the one of K. If the robot now updates R_{I} and $\vec{v}_{\text{I}}$ to R_{parallel} and $\vec{v}_{\text{shift}}$ nothing would be achieved, as the other seed's system is updated as well and thus, their origins would merely oscillate between several points. Instead $\vec{v}_{\text{I}}$ is updated to a vector, which describes the shift halfway to the origin of K's coordinate system, namely to $\frac{\vec{v}_{\text{shift}}}{2}$. In the same manner, R_{I} is multiplied by $R_{\text{half\_parallel}}$, which stands for the rotation of I's axes halfway to being parallel to K's axes. $R_{\text{half\_parallel}}$ can be found by interpolating the half rotation from the identity matrix to R_{parallel} using SLERP [11].

5 Implementation

Following, the reader will first encounter a description of the platform, the Kilobot, which we used to implement our system. After that, some challenges we found during the implementation will be discussed, followed by the description of the approach taken.

5.1 Platform

The targeted environment of the implementation is the Kilobot [9]. Kilobot was developed in 2010 at the university of Harvard with the intention of providing a low cost swarm robotics platform. As the hardware components of a Kilobot merely cost around 14 US$, a big collective of Kilobots is easily affordable. Also the programming, charging and controlling in general of a swarm of Kilobots can be easily done by a single person within a reasonable amount of time.

Besides a small computational unit, a Kilobot comes with some flash memory for the user program and bootloader, as well as a small amount of volatile memory and non-volatile EEPROM. Two vibration motors are attached on the sides, with which the robot is able to turn either left or right, or move forward. Communication is possible with a infrared light, which is reflected off the surface that the bot is standing on. The transmission of a message is in the form of a broadcast, which can be received within a radius of at least 7 cm, provided that there are no objects between sender and receiver. To prevent collision, the CSMA/CA protocol is used. A message consists of 2 bytes and 7 bits, which are written to a buffer, plus a checksum. So one data package consists of almost 4 bytes. If the infrared sender is turned on, the broadcast happens every 200 ms and consists of the bytes specified most recently in the buffer. When an incoming message is detected and the transmission is successful, it is saved to another buffer. A Kilobot can buffer up to three messages. In the case of an overflow, the oldest message gets dismissed. During the execution, a request can be sent to the buffer, whether new messages have arrived. If this is the case, the oldest message gets copied into an array which is intended to be accessed by the executing program. From the intensity of the signal the distance to the sender can be estimated with an error of up to $\pm$ 2 mm. A Kilobot is also able to measure ambient light and sense its battery voltage. No Kilobot is assigned to a fixed ID or other form of identification.

Summing up these abilities, we need to point out, that a Kilobot has barely any sense of its environment, meaning, it does not know how many other robots are part of the collective nor at what position the Kilobot itself and its neighbours are positioned. Although it can measure the distance to a neighbour, a Kilobot does not have the ability to locate its neighbours merely by exchanging one single message. Also even though the motors can be calibrated, due to inaccuracy, only simple paths of movements are feasible for a Kilobot. Until now, no collision avoidance protocol for the Kilobot is implemented. These were part of the reason, why this thesis does not focus on the movement of a robot, but more on an algorithm, which requires only message exchange.

The most important data of the Kilobot's specification [5]: The Kilobot is assembled with a microcontroller ATmega 328p [13] as CPU which operates on 8-bits at 8 MHz. As already hinted, it contains 32 KB of flash memory, 1 KB of EEPROM as non-volatile memory and 2 KB SRAM. The energy source is a 3.7 V rechargeable battery.

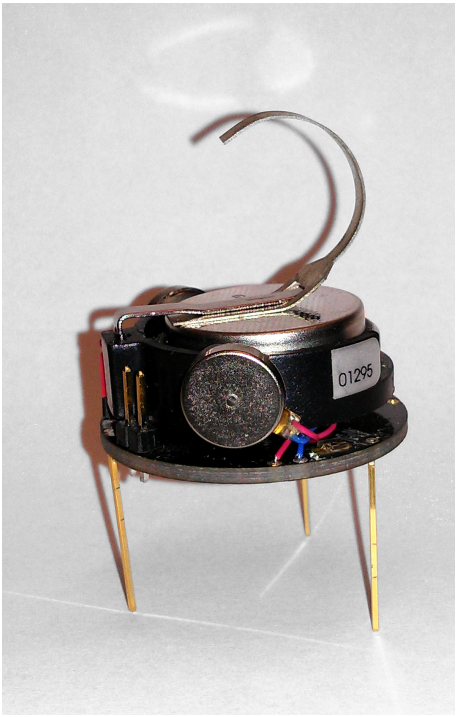


Figure 5.1: A Kilobot

A program is written in C. For compilation, avr-gcc is used which is part of the free AVR toolchain[1] to program Atmel microcontrollers. The output is a HEX file containing the machine code. This file is streamed to the Overheadcontroller (OHC), which then programs the whole swarm simultaneously via infrared. To debug a Kilobot, it has a serial port, which can be connected to the OHC to output short strings or integers via a computer terminal. Also by blinking a RGB LED, a Kilobot can indicate in what state it currently is.

[1] https://www.kilobotics.com/

A library, providing all the basic functionalities, was developed by Michael Rubenstein and is part of the toolchain.

Our swarm had up to 20 members, standing on a whiteboard surface, which is the Kilobot's intended ground to stand on.

5.2 Challenges

Due to the minimalistic nature of a Kilobot, we encountered the following challenges:

Low computational power: As an 8-bit processor with two kilobytes of memory, the Atmel 328p is already able to compute more than regular computers two decades ago, computationally intensive algorithms should still be avoided.

Message exchange: With messages containing only up to 23 bits, communication protocols have to be designed with care and every redundant exchange should be avoided. Because a new message can only be transmitted every 200 ms, protocols might consume a lot of time.

Noise and interference: Even though message exchange is generally very stable, noise and interference might disturb transmission.

Sensitivity to battery charge: Depending on the state of the battery of a Kilobot, it might behave differently. Typical effects are: unreliable response to the overhead programmer, message loss while sending or receiving, or weak LEDs.

No identification: As no Kilobot comes with a fixed identifier, it is not possible to reliably address a single specific Kilobot.

Decentralization: Because the philosophy of swarm robotics prioritises algorithms without any main controlling unit, the implementation has to make sure that the Kilobots respond to each other in meaningful ways. In other words, the robots have to be synchronised at least rudimentary, or, if it is not possible, at least can provide requested information independent from its state.

No prior knowledge about the environment: Since the size of the swarm and therefore the number of participants is unknown and no information about the topology is given, a general and scalable solution must be found to solve the given task.

Byzantine behaviour: During the process of implementation, unexpected and inconsistent behaviour became apparent with a few Kilobots. It seemed, that the infrared receiver did not fully function. Namely, some messages were never received and other messages were delivered with a delay. Even when the affected Kilobot was separated and no other signals could have corrupted the transmission, it showed

a similar misbehaviour. There was no apparent pattern to the receiving failure, it seemed, that depending on the content of the messages they simply were discarded. Our assumption is, that the infrared receiver needs a recalibration. Unfortunately we had not the required infrastructure to do so, so the robots considered corrupted were singled out.

5.3 Our Approach

This chapter describes our implementation. First the overall architecture will be explained, and followed by how we approached each phase as described in chapter 4. A presentation of the result can be found online[2].

5.3.1 General Architecture

A program on a Kilobot is run within an infinite loop in the main function. With every iteration, first interrupts get checked and then the actual program is executed. As these interrupts are important for example for receiving messages, there should not be a big delay between each iteration. Therefore we chose the overall architectural pattern to be a finite state machine. As the algorithm already consists of different phases, the implementation of a finite state machine was the most straightforward. Another advantage is the lightweight nature of the pattern, which actually makes it popular in embedded system design in general. We used 17 different main states, of which most are again divided into several substates.

Outgoing messages are always defined within different states. If a response is needed for a state transition, the requesting message gets sent, until an appropriate response is received, or a counter is reached. To receive messages, the function `receive_messsage()` nedds to be invoked. The message is processed accordingly to its type, or, if it is not meant for this recipient, dropped directly.

The pattern of a message is as follows: Every message's last seven bits indicate the type of the message. Depending on the type, the other two bytes are interpreted accordingly. Most of the time, the second byte indicates either the receiver's or sender's ID, and the first byte a type of data containing the required information.

[2] `http://n.ethz.ch/~kaisert/kilobot`

5.3.2 Implementing the Phases

Phase 0

The first state, INIT, is responsible for the initialization of several data structures and sets up the seed for the random number generator. The transition into the next state happens immediately and no computation for the actual algorithm happens here.

Phase 1

The construction of the coordinate system starts in the state CONSTRUCT_ID. Each Kilobot computes and assigns itself a random ID. In TEST_ID every Kilobot broadcasts its ID to its surrounding neighbours. Whenever a Kilobot receives an ID, it saves the ID in a linked list and additionally the distance to the neighbour bearing this ID. To ensure the local uniqueness of the IDs, every incoming ID is matched with the already received ones. If there is a mismatch in distance and ID, a message is broadcasted commanding the other Kilobots to drop this ID. The two Kilobots under this ID change back to CONSTRUCT_ID and thus receive a new ID and broadcast it again. This repeats, until locally no ID appears twice anymore. The transition to the next state happens after a counter is reached.

Phase 2

The next step, the seed election protocol, follows the description of Rubenstein quite strictly. It is invoked in the state ELECT_SEED_TOP. Per default, every Kilobot is assigned the same role. While distributing the ID, every robot which encounters a higher ID changes its state, and starts with the bottom seed election. When after several rounds the robot still remains in ELECT_SEED_TOP, it considers itself a seed. The same protocol applies for the bottom seed election, with the difference that those robots already elected as seeds do not participate anymore. It is applied in the next state, ELECT_SEED_BOTTOM.

```
1  static void state_elect_seeder_top()
2  {
3      //broadcast id for seeder election
4      message_out(EMPTY, id, M_ELECT_SEED_TOP);
5      BREAK;
6      if(receive_message() == 1) {
7          //if greater id is received -> do not become top seeder
8          if(message_rx[1] > id)
9              role = BOTTOM_SEEDER;
10     }
11     if(counter >= ELECT_SEEDER_COUNT_T) {
12         //after ELECT_SEEDER_COUNT_T rounds, if role is still ND,↩
               become top seeder
```

```
13          if(role == ND)
14              role = SEEDER;
15          state = ELECT_SEED_BOTTOM;
16          RESET_COUNTER;
17  #ifdef DEBUG_LED
18          blink_led(&led_turquoise);
19  #endif
20      }
21      ++counter;
22  }
```

Figure 5.2: Code excerpt from the top seed election, file: part1/BA.c

Whenever a Kilobot is chosen as a seed, it will shout it to its neighbours, which store the ID in a linked list. After the bottom seed election, in the state `CHECK_SEEDER`, every robot checks, whether it can see at least two, or in case of a seed, at least one seed. If this is not the case, it broadcasts a message, which leads the neighbours to make the transition from `CHECK_SEEDER` back to `ELECT_SEED_BOTTOM`. If there are no more requests for new elections, the Kilobots transit to `AWAIT_R_R`, which serves as rudimentary synchronisation state.

Phase 3

With the transition to `RECRUIT_REFERENCE`, phase 3 is initiated. In this phase every seed needs to recruit two references. For the recruitment, the seed needs to know, every distance between all its neighbours. For this crucial exchange a synchronous protocol was designed: Every non-seed simply waits for an inquiry from a neighbour. It does not initiate any interaction with any other Kilobot. For the seeds, the protocol is implemented as following: besides the initialization state, a seed will iterate through three different states. First a seed (A) will request the attention of a neighbour (B). Once this request gets acknowledged, the seed asks for the distance between B and another neighbour (C). As soon as the transmission is over, the seed makes a transition back to the first substate to inquire the distance between two other neighbours, until every distance between each pair of neighbours is known. the enclosing angle of the seed and B and C is computed on the fly and the best suitable pair of references is remembered. As stated above in the beginning, B remains in a busy waiting state until an inquiry message with matching ID is received. Then B changes to a, for other seeds, non-responsive state. It sends the acknowledgement to A, and listens, until it receives the neighbour's ID to which it has to check the distance. When this message is transmitted, it either responses with the distance, which already was saved in phase 1, or, if the neighbour can not be seen, sends zero as an indicator that neighbour C is out of B's communication range.

If a seed happens to break down and is not able to finish the protocol, robot B resets its state after a certain amount of other requests and start to funciton as initially. Similarly,

the seed will skip *B* after a certain amount of requests, indicating it is unresponsive and continues with the next neighbour.

At this point the program exceeded the 32 KB of the storage for user programs and bootloader and we were forced to split the implementation into two programs. This means, that after completion of the first part, the Kilobots write all important data, such as identification number, the list of all surrounding seeds and the list of all visible neighbours, into the non-volatile EEPROM, and are shut down afterwards. Then the second part of the algorithm is transferred to the Kilobots, all values are retrieved again and the execution of the program continues.

Up to now, every seed generates its local coordinate system. One of the references lays on the x-axis and the exact coordinates of the other reference is computed as described in section 4.3. The seed sends the correspondent coordinates to *B* and *C*, which now are located in this local coordinate system.

Phase 4

Before the localization of the other robots, state `AWAIT_LOCALIZATION` synchronizes the Kilobots again. Basically every Kilobot sniffs what messages are circulating. When there are no messages from the previous phase heard anymore, the state makes a transition to `LOCALIZE`.

To locate other robots in the different coordinate systems as well, every located Kilobot starts to shout its location in a known coordinate system. Due to the limitations in messaging it is a bit tricky to construct a reliable protocol for that purpose, which also operates within a reasonable amount of time. One possibility is to implement again a synchronized protocol in a similar manner of the protocol in phase 3, but as every neighbour might require to be updated multiple times, it is not a feasible approach. Also every neighbour needs the same information to be able to localize itself. Instead, the Kilobots communicate asynchronously. If a bot is not located in any coordinate system it remains quiet. When a robot becomes localized, it first broadcasts the seed's ID in whose coordinate system it had acquired a position. Whenever such a message is received, and the seed can be seen, an unlocated Kilobot stores the ID of the sender and the one of the seed. The seed's ID is followed by the corresponding X and Y values of the located Kilobot. When such a broadcast is transmitted, the receiver can match the ID of the entry in the list of announcers with the ID sent with the coordinate. Whenever two or more robots announced their location in the same coordinate system and the seed is within communication distance, another Kilobot can locate itself with simple trilateration in the seed's coordinate system. As soon as a Kilobot has a new location, it also starts shouting its location, such, that other robots are able to locate themselves as well.

To make sure all assigned locations are propagated, a Kilobot iterates over the list of locations several times. After reaching a counter, the robot makes a transition to

`AWAIT_FORMING_M_GROUP`, where the swarm synchronizes again, and then continues to `FORMING_M_GROUP`, initializing phase 5.

Phase 5

In phase 4, enough information was gathered to already form the merging groups. The implementation of phase 5 differs from the description in chapter 4. Since during the localization the position of every member has been inquired, there is no need for an actual message exchange. If two neighbours are found, which are located in the same coordintate system, a merging group is founded. The formation of the merging group is based on the assumption, that every member of the group comes to the same conclusion. Since the verification with the other members is not that crucial in this step, it is omitted. It is sufficient, when at least one member comes to the conclusion to form a merging group. In theory, a merging group should only be created, if all three members are aware of each other, but due to communication failures, it might happen that some members did not recognise all surrounding Kilobots. The decision to omit any transmissions in this state was made, because another message exchange, if synchronously or asynchronously, costs again a lot of time and the data gained only serves a verification purpose, but no new insights into the topology of the swarm can be made.

After the creation of the merging groups, every Kilobot registers its groups to the appropriate seeds. This is done by simply broadcasting a message.

Phase 6

In the final phase, merging all the local coordinate systems, small changes were applied to the protocol proposed by Michael Rubenstein. As the main bottleneck is the transmission of data, the implementation avoids that every Kilobot sends a hitherto computed 3x3 matrix and a vector to their seeds. Instead the program proposes that any seed (A) first randomly chooses a Kilobot (B) which registered as part of a merging group. The seed then requests the data from B. If B is part of more than one merging group for this specific seed, it also chooses randomly from these groups.

Now B computes the 3x3 rotation matrix. To approximate the halfway rotation, the matrix needs to be converted into a quaternion. As there is no benefit in converting the new quaternion back into a rotation matrix, but actually a loss of precision, not a 3x3 rotation matrix is being transferred to the seed, but rather a quaternion. Another benefit is, that the quaternion is only a four dimensional vector, meaning, when every value is stored in 16 bits, only eight messages are necessary to transfer the data instead of 18 in case of a matrix. Similarly the vector $\vec{v}_{\text{seed}}$ is calculated and transferred.

To keep the traffic low and reduce the transfer time, the protocol to transfer the quaternion and the vector, is implemented asynchronously. Meaning, the chosen member of the merging group gradually shouts the updated location of the seed, which then repeats the received entries of the quaternion and the vector for all its followers. When the update is complete, every Kilobot applies the new data to their location. After enough rounds, every coordinate system converges to each other, such that there is only one coordinate system remaining.

Not the complete phase 6 was implemented. Due to time constraints, only parts of it are functioning properly. The skeleton for the protocol exists in its basics and most functionalities to calculate the vectors and quaternions are implemented. Once the implementation is completed, one might program the Kilobot to produce some kind of gradient to visualize the coordinate system, simulated with their LEDs.

6 Results

This chapter presents the reults gained in this thesis. They are ordered according to each phase as described in chapter 4:

Phase 1: At the end of phase 1, every Kilobot is assigned to a locally unique identifier and its existence was being promoted amongst the proximate neighbours.

Phase 2: The seeds are elected reliably. Though it might occur that too many seeds will be chosen than necessary. This is due to unoptimized waiting times during the election and asynchronism.

Phase 3: Every seed is able to acquire the distances between its neighbours, and select the appropriate references. It is observed regularly, that the protocol used, does not terminate correctly. This is due to signal interference and noise, such that the non-seed robot does not receive an acknowledgement, stating the termination of the transaction. However the robot is able to recover by simply resetting its state after reaching a count.

Phase 4: Even though the traffic of message exchange is comparatively heavy in this phase, the Kilobots are able to locate themselves with the aid of their neighbours. There are certain flaws though, which make it impossible to gain consent about a global coordinate system. This will be discussed in chapter 7.

Phase 5: If phase 4 was successful, then so is phase 5, as mainly data is being processed, gained from phase 4. In the end the Kilobots are assigned to adequate merging groups and have registered them.

Phase 6: As stated above, phase 6 was not fully implemented. It is implemented, up to the point, where the random merging group is chosen. Also the code for computing the necessary quaternions is done. What is missing, is the implementation of the protocol necessary to exchange the quaternions and the vectors. The implementation has started, but tests, using dummy numbers, have not been successful yet. Also a rudimentary synchronisation among the swarm between the merging rounds is not implemented, and should be considered, when continuing the work.

7 Discussion

Due to time constraints the focus of the implementation did not lay on optimization. Hence there is a big potential for tweaking the delays and improving the waiting times for message passing and synchronisation. In the same direction goes the aspect of enhancing the robustness of the implementation. Another part is the memory usage, which can be reduced and optimized. One can also consider to rely more on asynchronous protocols, as they are much faster and have proven to be quite stable for this scenario.

Since communication relies on infrared, a signal gets mostly cut off, as soon as an obstacle, like another Kilobot, is placed between sender and receiver. With that limitation, reliable communication is only possible with a direct neighbour. This decreases the radius of visibility drastically for each robot. With that issue, the topology and the distribution of the local IDs within the swarm takes a much bigger role. Even when the conditions are fulfilled and every non-seed robot can see at least two seeds and ever seed at least one other seed, there are settings, in which the swarm is not able to reach consent about one global coordinate system. There are three cases, which make the construction of a global coordinate system not feasible when following the algorithm from chapter 4:

1. When a Kilobot can not locate itself within any local coordinate system. One example is shown in figure 7.1a. The two outer left Kilobots can only see two other robots, which have at least one (in this case two) assigned location in a coordinate system. Due to symmetry issues, a robot needs to see two located references plus the seed robot of the coordinate system. Thus, in 7.1a it is not possible for those two robots to acquire a positioning in any of the two coordinate systems.

2. No merging group can be constructed for a specific seed as in figure 7.1b. No matter how the other seed robots choose their coordinate systems, the blue seed will never be able to join a merging group with either of them. The problem is similar to the one discussed before: Only the seed and the two references are able to position them in the blue seed's system.

3. Even though merging groups could be constructed, they are placed such, that multiple "global" coordinate systems will emerge. Like in figure 7.1c, where the four robots on the left will establish a coordinate system and the four Kilobots on the right side as well.

The first case is not too dramatic. Theoretically this Kilobot can just be left out during the construction, and it can locate itself, as soon as a global coordinate system has been

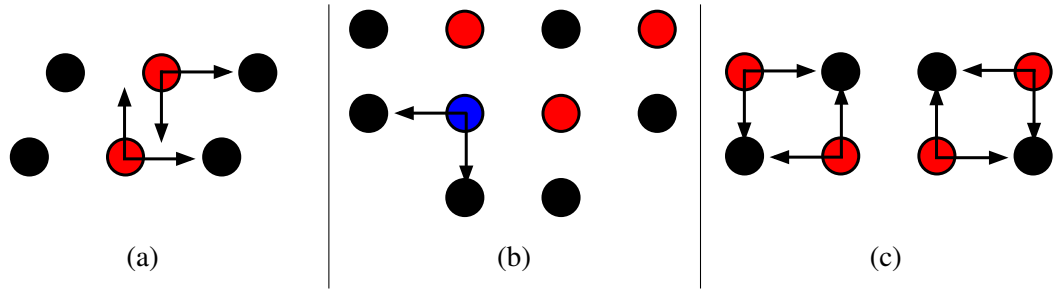


Figure 7.1: Three cases, in which a global coordinate system can not be established. The seeds are either red or blue. The chosen coordinate systems are indicated by the arrows, where the labelling of the x and y axes does not have any influence, and is omitted. In each case, the robots closest to the axes were chosen as references.

achieved. The same counts for case number two. One can try to continue without this seed such that all unlocated robots can locate themselves afterwards, but since one seed is lacking, it is possible that case 2 just transforms into case 3.

In the last case there are two possibilities how to resolve the problem. One is to just start over again and hope for a better distribution of the IDs and hence the seeds. Another solution is to simulate a new seed election, and force the seeds to be at the edges of each individual global coordinate system. Then the entire merging protocol can be repeated and the two or more semi-global coordinate systems eventually will merge as well. To actually detect this scenario, a Kilobot can test whether the measured distance to a robot matches the calculated distance. This can be done in a few simple steps in which the coordinates of the neighbours are requested and from this the euclidean norm is calculated and compared to the measured distance.

In order to keep the risk as low as possible that either one of these cases occur, the size of the swarm can be increased. With more robots, it is more likely that each local coordinate system is connected with one another, because there are simply more possibilities to do so. Another solution would be to increase the communication radius of the Kilobots. With that, more than simply the neighbouring robot can be seen and hence cases as in figure 7.1a occur less often.

As a platform, the Kilobot brings all the necessary requirements, but we have to bear in mind, that the Kilobot does not bring anything more. For example considering the messaging system, even though protocols are feasible and it is possible to rely on merely almost three bytes, already the addition of a fourth byte would ease the design significantly. The intention of the Kilobot was, to provide a first low-cost platform with the intention to give a start for swarm robotics to emerge as discipline and serving as educational purposes, in which it definitely succeeded.

8 Conclusion

In this thesis, an algorithm was presented, which describes how to reach consent about a global coordinate system within a swarm of identical robots. It relies only on local information and has almost no restrictions on the size of the swarm. We implemented the algorithm on a platform called Kilobot, a simple swarm robot. Even though, the implementation is not fully finished, it became clear that the proposed algorithm in Michael Rubenstein's Phd thesis is feasible for a Kilobot. None of the missing steps exceeds the Kilobot's abilities. The algorithm implemented proved to have some flaws, which can partially be avoided by either increasing the size of the swarm, or by selecting a more powerful, in the sense of range, communication system.

The program is written in C. During the implementation process many approaches wildly used in the field of embedded and, of course, distributed systems proved to be very useful.

The next step would be to finalize the program. Once a global coordinate system can be established reliably, approaches to constructing shapes or building paths can be taken. Even though Kilobot comes with many useful features for its costs, one might consider either using more powerful hardware such as the e-puck, or even develop their own robot tailored to its individual purposes.

Bibliography

[1] C. Cianci, S. Magnenat, J. Pugh, C. M. Alves, X. Raemy, J.-C. Zufferey, F. Mondada, D. Floreano, A. Klaptocz, P. J. Torres, P. J. Gonçalves, A. Martinoli, and M. Bonani. The e-puck, a Robot Designed for Education in Engineering. *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 1(1):59–65, 2009.

[2] K. Gilpin, K. Kotay, D. Rus, and I. Vasilescu. Miche: Modular Shape Formation by Self-Disassembly. *The International Journal of Robotics Research*, 27(3-4):345–372, Mar. 2008.

[3] H. D. BLACK. A passive system for determining the attitude of a satellite. *AIAA Journal*, 2(7):1350–1351, July 1964.

[4] N. H. III, R. Adviser-Nagpal, and R. Adviser-Wood. *Multi-robot foraging for swarms of simple robots*. Phd, Harvard University Cambridge,, 2011.

[5] Julien Tharing. Kilobot - User Manual. Technical Report March, K-Team, 2012.

[6] F. Mondada, A. Guignard, M. Bonani, D. Bar, M. Lauria, and D. Floreano. SWARM-BOT: from concept to implementation. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 2, pages 1626–1631. IEEE, 2003.

[7] J. W. Romanishin, K. Gilpin, and D. Rus. M-blocks: Momentum-driven, magnetic modular robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4288–4295. Massachusetts Institute of Technology, IEEE, Nov. 2013.

[8] M. Rubenstein. *Self-assembly and self-healing for robotic collectives*. Phd, University of Southern California, 2009.

[9] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *2012 IEEE International Conference on Robotics and Automation*, pages 3293–3298. IEEE, May 2012.

[10] W.-M. Shen, P. Will, A. Galstyan, and C.-M. Chuong. Hormone-Inspired Self-Organization and Distributed Control of Robotic Swarms. *Autonomous Robots*, 17(1):93–105, July 2004.

[11] K. Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH computer graphics*, 19(3):245–254, 1985.

[12] W. Spears and D. Gordon. Using artificial physics to control agents. In *Proceedings 1999 International Conference on Information Intelligence and Systems (Cat. No.PR00446)*, pages 281–288. IEEE Comput. Soc, 1999.

[13] I. Sram, S. P. W. M. Channels, and O.-c. A. Comparator. Atmel 8-bit Microcontroller with 4/8/16/32KBytes In- System Programmable Flash. Technical report, Atmel, 2013.

[14] K. Stoy and R. Nagpal. Self-repair through scale independent self-reconfiguration. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 2, pages 2062–2067. IEEE, 2004.