NTNU – Trondheim
Norwegian University of
Science and Technology

Self-Assembling to Improve Performance in Swarm Robotics

Joachim Halvorsen
Hege Beate Seilen

Master of Science in Computer Science
Submission date: June 2014
Supervisor: Keith Downing, IDI

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

Problem Description

This master thesis aims to show how a swarm of robots can cooperate, to improve their mobility in difficult environments. Self-assembling should increase their capability of handling environmental challenges, such as ice and strong wind. Through local interactions, the robots will decide autonomously if self-assembling is necessary, without human intervention or global control. Both the training and the performance evaluation of the robot controllers will be conducted in simulation.

- Assignment given: 15 January 2014
- Supervisor: Professor Keith Downing

Abstract

This thesis gives a brief introduction to the field of swarm robotics, and investigates the advantages of using self-assembling for swarm robots in difficult environments. The current research in swarm robotics has already demonstrated advantages of using self-assembling, including the ability of the swarm to drive up steep slopes and the ability to pull objects that are larger than the robots themselves. Robots can have multiple advantages by being small. With the ability to self-assemble the robot swarm can physically connect to become a larger entity, when this is advantageous. This project continues the research in swarm robotics, by investigating self-assembling in environments with complicating factors, such as ice, strong wind and obstacles.

We made a robot simulator to be able to perform experiments, and designed several scenarios with difficult environments. By using mechanisms inspired by social insects, the robots were able to perform complex tasks when working together as a swarm. Each robot was controlled by an arbitration architecture, where behavior modules vote for actions. The votes are weighted based on the general and specific importance of the behavior module, which is found by an evolutionary algorithm and an objective situation manager respectively.

The results from this thesis demonstrate that self-assembling can improve the mobility of a swarm of robots, in environments with complicating factors, such as ice, strong wind and obstacles. When gradually increasing the wind velocity in sequential experiments, we found that the use of self-assembling went from being unnecessary, to become a faster method to traverse the environment, and to finally become necessary in order to traverse the whole environment. The robots adapted to the different experiments by increasing the use of self-assembly as the wind velocity was increased. This thesis contributes to the field of swarm robotics, by demonstrating and analyzing the mobility advantage of the swarm robots by using self-assembling in these difficult environments.

Sammendrag

(This is a Norwegian translation of the abstract)

Denne avhandlingen gir en kort introduksjon til forskningsfeltet svermrobotikk, og undersøker fordelene med å bruke selvstendig sammenkobling for svermroboter i vanskelige miljøer. Forskning innen svermrobotikk har allerede demonstrert fordeler ved å bruke selvstendig sammenkobling, som muligheten for svermen til å kjøre opp bratte bakker og mulighetene til å trekke objekter som er større enn robotene. Roboter kan ha flere fordeler ved å være små. Med muligheten til selvstendig sammenkobling kan robotsvermen koble seg sammen og bli en større enhet, når dette er fordelaktig. Dette prosjektet fortsetter forskningen innen svermrobotikk, ved å undersøke selvstendig sammenkobling i miljøer med kompliserende faktorer, som områder med is, sterk vind og andre hindringer.

Vi har laget en robotsimulator for å kunne utføre eksperimenter, og har designet flere scenarier med vanskelige miljøer. Ved å bruke mekanismer som er inspirert av sosiale insekter, kan robotene utføre komplekse oppgaver ved å jobbe sammen som en sverm. Hver robot blir kontrollert av en meklingsarkitektur, hvor moduler for oppførsel stemmer på mulige handlinger som roboten kan utføre. Stemmene vektes basert på den generelle og den spesifikke innflytelsen til modulene, som er henholdsvis bestemt av en evolusjonær algoritme og en objektiv situasjonsbehandler.

Resultatene fra denne avhandlingen viser at bruken av selvstendig sammenkobling kan forbedre mobiliteten til svermroboter i miljøer med kompliserende faktorer, som områder med is, sterk vind og andre hindringer. Ved å gradvis øke vindhastigheten i sekvensielle eksperimenter, fant vi at bruken av selvstendig sammenkobling gikk fra å være unødvendig, til å gjøre robotene raskere på oppgaven, og til slutt være nødvendig for at robotene kunne traversere hele miljøet. Robotene tilpasset seg de forskjellige eksperimentene, ved å øke bruken av selvstendig sammenkobling ettersom vindhastigheten ble økt. Denne avhandlingen bidrar til forskningsfeltet innenfor svermrobotikk, ved å vise og analysere bevegelsesfordelene til svermrobotene når de bruker selvstendig sammenkobling i disse vanskelige miljøene.

Preface

This project is the authors' master thesis at the Department of Computer and Information Science, Norwegian University of Science and Technology.

We wish to thank our supervisor Professor Keith Downing, at the Department of Computer and Information Science, Norwegian University of Science and Technology, for his invaluable guidance throughout this project.

We would also like to thank Jean-Marc Montanier, Research Assistant at the Department of Computer and Information Science, Norwegian University of Science and Technology, for giving us valuable insight to the field of Swarm Robotics.

Joachim Halvorsen and Hege Beate Seilen

Trondheim, 4 June 2014

Contents

List of Figures

List of Tables

List of Abbreviations

2D 2-Dimensional

3D 3-Dimensional

ANN Artificial Neural Network

CPU Central Processing Unit

DAMN Distributed Architecture for Mobile Navigation

EA Evolutionary Algorithm

EANN Evolutionary Artificial Neural Network

EDSA Enhanced Directional Self-Assembling

GA Genetic Algorithm

GUI Graphical User Interface

S1 Scenario 1

S2 Scenario 2

S3 Scenario 3

S4 Scenario 4

Chapter 1

Introduction

In our master thesis, we have researched the state of the art in the field of swarm robotics. We have studied how a swarm of robots can cooperate, by physically connecting to each other, to improve their mobility in difficult environments.

This chapter is the introduction to the thesis. Section 1.1 contains the background and motivation for our master thesis. In Section 1.2 we present the goal and research questions. Section 1.3 describes the research method we have used. Section 1.4 explains the contributions we have made to the field of swarm robotics. Section 1.5 gives an overview of the thesis structure.

1.1 Background and Motivation

Swarm robotics is an emergent field of robotics, which is inspired by nature. The concepts and approaches in this field comes from studying animal species, which have evolved and adapted to survive, in the changing environment on earth, for millions of years. The animal species have developed from simple organisms to complex species. As in Darwin's evolutionary theory [Darwin, 1859], natural selection and mutations are concepts used in swarm robotics to evolve robot controllers. We find these concepts very interesting and believe it is possible to learn much more from nature.

For swarm robotics, the greatest inspiration from biology is the self-organization in animals. Animals use self-organization to gain multiple advantages in nature. Their individual behaviors are often based on simple rules, with only local interactions, and often no assigned leader. Still their collective behavior can be very complex. Some examples are, when birds fly in a v-formation to reduce drag and conserve energy, and when ants distribute their workload to build their nest and search nearby areas for food. The fire ant species use the approach of self-assembling, which is to form a physically connected structure, only by using local interactions. The fire ants use self-assembling to

create waterproof rafts of themselves to survive floods, and they exchange positions and use air bubbles to minimize the drowning of individual ants [Mlot et al., 2011]. Other biological approaches are also used in swarm robotics. These include nervous system modeling, which is often used to control robots, and processes inspired by evolution, which can be used to train these robot controllers.

We believe that robots will become an increasing part of our world in the future. In the last years, we have already begun to see robots in the everyday life, outside factories and research labs. This includes automatic lawn mowers, vacuum cleaners, and delivery robots used in the same rooms and corridors as people, like at St. Olavs Hospital in Trondheim. Robots can also be used as eyes and tools for humans in areas where humans cannot go, either because it is too dangerous or because the area is too small. The robots can be used for exploring the Moon and Mars, and for searching for people in collapsed or dangerous buildings, damaged by fires, earthquakes or tornadoes.

Traditional robots are already used in these areas, and small swarm robots can possibly perform similar tasks in the future. Having smaller robots can be advantageous in multiple situation. They are lighter, meaning they can more easily be carried into the area where they are needed, for instance by small quad-copters. Small robots can also get into smaller passages. With an ability to self-assemble, the small swarm robots can also get the advantages of being a larger entity, including the ability to pull large objects. Much research have been done in the field of swarm robotics, in the last decade, but most systems are still in the research lab. Before swarm robotics can move more into the real world, we believe more research is needed.

We believe that swarm robotics can be useful out in the real world, and want to contribute to the development within the field. During our research process, we found the projects Swarm-bots [Dorigo et al., 2006], Swarmanoid [Swarmanoid, 2010] and Symbrion [Symbrion, 2013], which have done experiments with self-assembling in swarm robotics. These projects show successful self-assembling, with swarms managing to climb slopes, cross holes and create structures. We found these studies very interesting, but their self-assembling systems were only explored in a few scenarios. We believe that self-assembling need to be considered in more environments before it can be used in the real world. To contribute, by continuing this research, we will explore the advantages of self-assembling for a swarm of robots in other difficult environments.

1.2 Goal and Research Questions

In this section, the goal and the research questions of this project are presented.

Goal statement:

> *The goal of the project is to demonstrate advantages of emergent self-assembling, for a swarm of robots in difficult environments.*

We will investigate the use of self-assembling in difficult environments by considering the two research questions defined below. We will create a robot simulator and perform experiments with the simulated robots in multiple scenarios.

Research Question 1:

> *How can self-assembling improve the mobility of a robot swarm in difficult environments?*

We will specifically look at environments with complicating factors, such as ice and wind. Self-assembling could possibly be advantageous in these environments, by increasing the grip on ice and to reduce the effect of a powerful wind. One property of these environments is that they could easily be altered in a simulator. By simply adjusting the strength of the wind, the performance and use of self-assembling could be very different in the same environment, which could be interesting to analyze.

Research Question 2:

> *How can the robots decide when to self-assemble through local interactions and sensing the environment?*

We will investigate how the robots are able to form a more global structure from local interactions. The robots are only able to sense the environment from their set of sensors, including light and distance sensors, meaning their view of the environment is very limited. Their communication methods are also very limited, having no digital communication, no global control and no explicit negotiation methods. It will be interesting to investigate if these simple robots are able to find reasons to self-assemble and then manage to complete the self-assembling process.

1.3 Research Method

To be able to address the research questions, we designed a robot simulator, where we could perform experiments with a swarm of robots, which are able to self-assemble into larger groups. By creating a robot simulator, we were able to do experiments on multiple environments, with multiple parameters and multiple robots, and still keep the experiments both fast and cheap. We made a virtual robot that was inspired by robots used within the field of swarm robotics. The robot has different sensors, wheels, and a gripper, to be able to sense the environment and act in response to it and other robots.

To evaluate the mobility of the swarm in difficult environments, we designed four different scenarios in our simulator, with challenging features including ice, strong wind and obstacles. We used artificial evolution to train the robot controller, which is based

on a known robot controller architecture. With the training, the robot could become better at navigating towards the goal and overcome these environmental challenges. The simulator was then used to evaluate the performance of different robot controllers, found during the artificial evolution.

Finally, we analyzed the resulting performance, features and situations we found using the best robot controllers, to be able to decide if the use of self-assembling actually improved the mobility of the swarm, in the different experiments.

1.4 Contributions

Our contribution to the field of swarm robotics is to continue to experiment with self-assembling in a swarm of robots, in other challenging environments. Our experiments demonstrate that self-assembling can improve mobility in environments with ice and strong wind. Depending on the strength of the wind in our experiments, the use of self-assembling goes from being unnecessary, to becoming a faster method, and finally to become a necessary method for the robots to traverse the difficult environment.

1.5 Thesis Structure

The thesis is divided into five chapters. Chapter 2 contains research from the field of swarm intelligence and swarm robotics, common solution techniques, a literature review covering related projects, and background discussion. Chapter 3 contains the methodologies that we have used and describes the robot, the scenarios, the robot controller, the choice of creating a simulator, how we modeled the physics in our simulator, and the evolutionary algorithm. Chapter 4 contains the results and analysis of the experiments we performed, and a discussion about them. Finally, Chapter 5 contains the conclusions of our project.

Chapter 2

Background

This chapter contains the background research for our thesis. Section 2.1 introduces concepts within swarm intelligence. Section 2.2 gives an overview of the field of swarm robotics and explains what evolutionary robotics can be used for. Section 2.3 covers the common solution techniques used in this field. Section 2.4 contains a structured literature review. Section 2.5 describes systems and projects that are related to our thesis. In Section 2.6 we discuss how the fields and projects from this chapter have influenced our own project.

2.1 Swarm Intelligence

Swarm Intelligence: From Natural to Artificial Systems [Bonabeau et al., 1999], is a book that contains an overview of the field of swarm intelligence, from the bio-inspired perspective to the first swarm robotics. This section will cover the most relevant concepts from this book.

The inspiration for swarm intelligence comes from social insects. Insects, such as ants, bees, wasps and termites can perform tasks that seem to have a complex, centralized control, like building large nests and find optimal paths for finding food. However, these tasks are the result of decentralized, distributed tasks, among insects with different responsibilities. In an ant colony for example, there are different kinds of workers. Some have the responsibility of building and expanding the nest, cleaning the nest, and feeding the brood, while others find food and defend the nest. The ants also adapt to changing environments and situations. If a group of workers is underrepresented or removed, others can change their tasks to compensate for the loss of workers. There is a high degree of plasticity in the division of tasks.

To obtain a decentralized control, and thereby self-organization, of a colony of social insects, interactions among the insects are essential. Interactions can be either direct or

indirect. Direct interactions can be physical contact, visual contact or chemical contact. Indirect interactions can be stigmergy, a concept where an insect modifies the environment and others are indirectly affected because they change their behavior based on the environment.

In the end of the book, the authors describe issues with applying swarm intelligence to solve problems. First, it is difficult to program the individual robots to make the swarm perform the desired task. Second, it is difficult to determine how complex each robot should be, in terms of communication, sensors, ability to learn, and knowledge of the environment. Last, there are two major problems for adaptive problem-solving systems: the lack of absolute reliability when unexpected events occur, and the lack of standard benchmarks to evaluate the performance. The authors also present solutions for these issues. For the first issue, evolution can be used to efficiently try out different solutions and come up with a good one. For the second issue, a solution is to start simple and extend the functionalities gradually. The lack of absolute reliability can be solved by thoroughly exploring the adaptive system's behavior. There is no simple solution to the lack of standard benchmarks. It requires a set of problems suited for evaluation of adaptive and swarm-intelligent systems.

2.2 Swarm Robotics

The book Bio-Inspired Artificial Intelligence [Floreano and Mattiussi, 2008] gives an overview of the field of Swarm robotics. Swarm robotics is self-organization of a swarm of autonomous robots. The robots are simple, which means that they follow simple rules, have simple sensors, mechanisms and electronics, and have decentralized control. The idea behind swarm robotics is to have simple robots working together to solve complex problems. This can be done by using emergent behaviors, similar to social insects, as described in Section 2.1. Usually a swarm of robots consists of three to a hundred robots, where the swarm can achieve more than the sum of the achievements of each robot. Different types of robots can work together in a swarm, for example flying robots, walking robots or robots with wheels, but we will focus on robots with wheels. In this section we will explain the phenomena self-assembling and self-aggregation, and in Section 2.2.1 we will describe what evolutionary robotics can be used for.

Self-assembling is a phenomenon where a swarm of robots gathers and connect to each other. Direct or indirect communication is often used within the swarm to know how and where to self-assemble. The robots can form a chain or different shapes when connecting to each other. Their gripper can connect to a connection ring or a specific connection area on another robot. The concept of self-assembling robots was introduced by Fukuda et al. [Fukuda et al., 1989] (as cited in [Bonabeau et al., 1999]). They developed a self-assembling cell structured robot, CEBOT, where each cell could connect, disconnect and communicate with each other. This was the beginning of trying to design self-assembling robotic systems.

Aggregation is another phenomenon, where a swarm gathers, but does not connect to each other. This can for example be used in a box pushing problem, where the swarm gathers around the box to push it collectively.

2.2.1 Evolutionary Robotics

Using artificial evolution to evolve software or hardware components for robots is called evolutionary robotics [Cliff et al., 1993]. In this section, we will focus on evolution of robotic software, and what it can be used for.

Evolutionary robotics can be used for exploring design spaces or scientific hypothesis, and for autonomous robots to develop their own controllers and body configurations [Floreano et al., 2007]. Evolutionary robotics can generate artificial brains and morphologies for robots. Autonomous robots can learn how to adapt to the environment by using evolved controllers. The robots can also learn to cooperate with each other to solve a task together. For self-assembling and morphologies, the use of evolutionary robotics is the most common approach. When using artificial evolution, it is sometimes possible to find solutions humans have not thought of. It is also possible to make task specific behaviors that can adapt to the environment. Floreano et al. [Floreano et al., 2007] believe that growing and further progression of evolutionary robotics eventually can lead to developing of new species of machines, capable of self-evolution.

Simulation is often used when evolving controllers, since it is very time expensive to transfer genotypes to physical robots for each generation. We will describe the advantages and drawbacks of using simulations in Section 2.3.5.

2.3 Common Solution Techniques

This section describes relevant common solution techniques within the field of Swarm robotics. These are: evolutionary algorithms, Brooks' subsumption architecture, an action selection architecture, artificial neural networks, and simulators.

2.3.1 Evolutionary Algorithms

Evolutionary algorithms (EA) are used to find good candidate solutions to optimization and search problems, including optimization of robot controllers. They are inspired by biological evolution, where a population of individuals is evolved through generations. The traits of each individual can be different, leading to a diverse population. Each individual can be selected to produce an offspring, and the offspring can then inherit traits from their parents, through recombination and mutation. Not every individual is selected for reproduction, but the selection is not completely random. The selection is inspired by Darwin's theories about natural selection [Darwin, 1859]. Individuals in

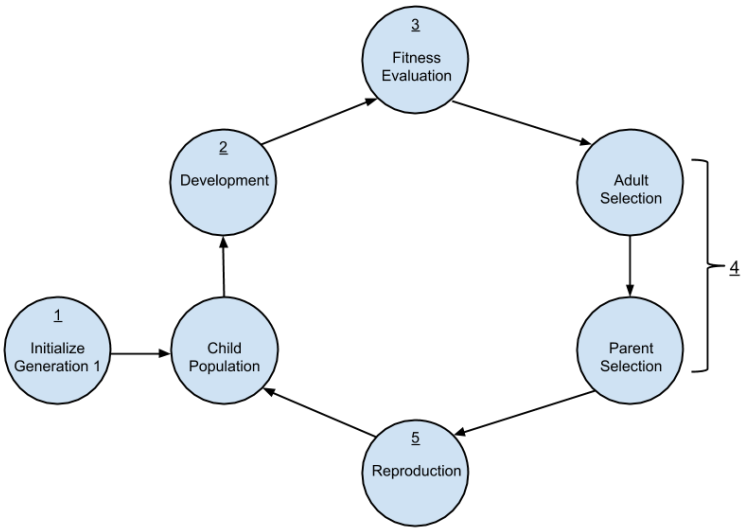


Figure 2.1: Primary generation loop in the evolutionary algorithm
Based on text in [Floreano and Mattiussi, 2008]

nature that have good traits, have a higher chance of finding food, avoiding predators and finding a partner. This makes it more probable that an individual with good traits is able to reproduce, compared to individuals without good traits. Individuals with higher fitness values in the artificial evolution are more likely to be selected to survive to adulthood and be selected to be a parent.

The process of the artificial evolution is described in Bio-Inspired Artificial Intelligence [Floreano and Mattiussi, 2008] and illustrated in Figure 2.1. First, the genotypes, representing the individuals' genomes, are initialized. Second, a developmental process is used to translate the genotypes into phenotypes. The phenotype represents the traits of the individual in the environment. Third, the fitness of each individual is evaluated in a fitness test, which tests the ability of the phenotype to handle a given problem. A numeric score, called the fitness value, is given to be able to compare different results. Fourth, different selection mechanisms decide which of the individuals that can become adults and then which of them are able to become parents. These methods combine exploitation and exploration. The fifth step is the reproduction process, where the genetic operator crossover is used to combine two parents to produce new individuals. Mutation can be used to make small random changes to the genotypes. One parent alone can also produce a new individual, by copying itself. When the best individuals are chosen for this, it is called elitism. Finally, the whole process starts from the second step again, and continues until it has iterated the given number of generations, or a good enough solution is found.

Most of the EAs used in the projects described later in this chapter, use a type of EA called genetic algorithm. In these algorithms the individuals operate on a binary rep-

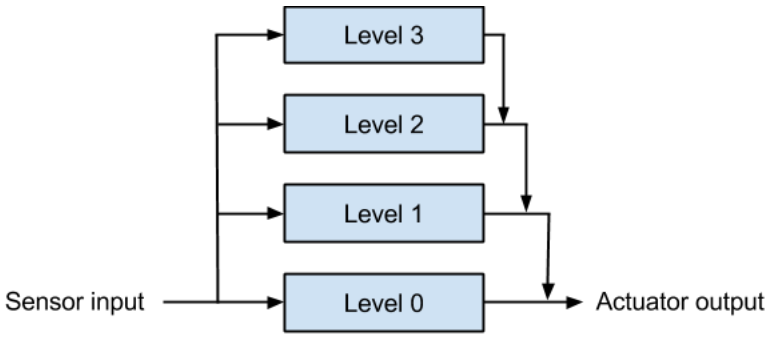


Figure 2.2: Brooks architecture, levels of control
The figure is based on text and figures by Brooks [Brooks, 1985].

resentation [Holland, 1992] (as cited in [Floreano and Mattiussi, 2008]). The binary encoding is often represented by a string or array of bits. Crossover and mutation can then be done by respectively combining two arrays, by taking a set of bits from each, and randomly flipping bits in the array. Other types of EAs include evolutionary programming, evolutionary strategies, multiobjective evolutionary algorithms and genetic programming. In genetic programming, for example, the genes can represent tree-based programs and circuits [Floreano and Mattiussi, 2008].

2.3.2 Brooks' Subsumption Architecture

Brooks' subsumption architecture [Brooks, 1985] is a reactive architecture for handling behaviors in a robot controller. A behavior can be anything the robot is designed to accomplish, including avoiding objects and constructing maps. The architecture has no centralized control mechanism or global memory, and the intelligence will arise from the robot's interaction with the environment.

Brooks describes the traditional decomposition of the robot controller as a horizontal sequence of modules, where the output from one module becomes the input for the next. Only the first module reads the sensor data, and only the last module controls the actuators. Brooks subsumption architecture has a vertical decomposition, where each behavior can both read from the sensor and send signals to the actuators, as illustrated in Figure 2.2. The different behaviors are represented by layers, which are ordered by their level of competence. Higher level layers have priority and subsume the roles of lower level layers. The coordination between the behaviors is done through inhibition and suppression. Inhibition happens when one layer prevents the output from another layer from reaching the actuators. Suppression happens when a layer overwrites the signal from a lower layer.

By using this architecture, it is possible to construct a controller layer by layer, without having to change the underlying layers. Each layer can have different goals, which means

that the robot can have different goals depending on the situation.

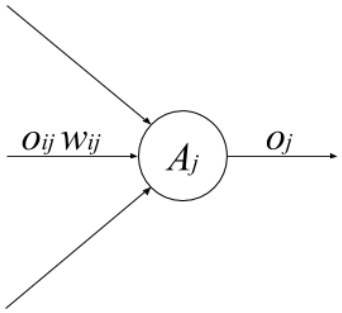
2.3.3 Arbitration via Action Selection

Rosenblatt and Payton proposed an alternative concept to the subsumption architecture [Rosenblatt and Payton, 1989] in 1989. The goal of the new architecture was to improve upon limitations they found by working with the subsumption architecture. One of their improvements was to increase the coordination among the components. In the subsumption architecture, a conflict among behaviors often result in situations where one behavior overrides the output from the other. The information from the lower level behavior is then completely lost. In the new architecture, coordination is achieved by a democratic voting mechanism. The behaviors cast weighted votes and the action that receives the most votes is selected.

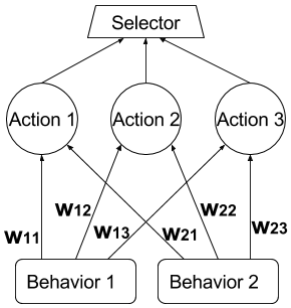
One example of an action node is illustrated in Figure 2.3(a). Each action node j receives votes from the behaviors i through its incoming edges. The output vote from a behavior node to an action node is denoted o_{ij} in the figure and can range from -1 to 1. Each of the edges from a behavior node to an action node can be weighted differently, with w_{ij}. This makes it possible to have behaviors that are more powerful than others. After all the votes in an action node are counted, it outputs the result o_j on the edge going out of the node. The overall action selection network is illustrated in Figure 2.3(b). The outputs from the actions all go to the module responsible for selecting the winning action, which selects the action with the highest output.

Rosenblatt and Payton also found that this architecture improved the modifiability of the system. New behavior modules could now be easily added without modifying the existing ones. They had found this aspect harder in the subsumption architecture, because of the need for a higher level behavior to decide if it should override a signal from a lower level behavior. Because it considers the output from the lower layers, the coupling between them is increased. With the increased coupling it becomes harder to make changes, including adding new behaviors. Another improvement compared to the subsumption architecture is that the need for the designer to assign priorities is reduced, by removing the hierarchy and only have weights that should be selected. The weights are easier to change, and can even be changed by another component in the system. The prioritized hierarchy is an aspect Pattie Maes has criticized the subsumption architecture for needing [Maes, 1991].

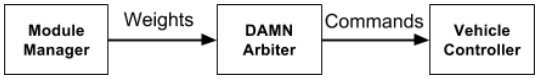
A module for changing the weights during runtime was proposed by Rosenblatt when creating DAMN. DAMN is a distributed architecture for mobile navigation [Rosenblatt, 1997], which builds upon the action selection architecture Rosenblatt and Payton proposed in 1989. The architecture is used in an example to control a vehicle with behaviors for avoiding obstacles, road following, avoid tipping over and more. The module manager, as seen in Figure 2.3(c), is one of the additions to the architecture. It is responsible for changing the weights on the edges, before the DAMN arbiter chooses an action that



(a) Weighted voting on actions. The figure is based on a figure by Rosenblatt and Payton [Rosenblatt and Payton, 1989].



(b) The action selection network. The figure is based on a figure by Rosenblatt and Payton [Rosenblatt, 1997; Rosenblatt and Payton, 1989].



(c) DAMN Structure. The figure is based on a figure by Rosenblatt [Rosenblatt, 1997].

Figure 2.3: The arbitration architecture

gets control over the vehicle controller. The DAMN arbiter is similar to the action selection network presented in Figure 2.3(b). The module manager did, however, only make changes to the weights in a predefined schedule in Rosenblatt's example system, and he admits there was no learning in their system at the time [Rosenblatt, 1997].

2.3.4 Artificial Neural Networks

Artificial neural networks (ANN) are computational models inspired by biological nervous systems, with good features such as robustness, flexibility and generalization [Floreano and Mattiussi, 2008]. An ANN consists of a network of artificial neurons with weighted connections. The neurons are divided into different layers: an input layer, an output layer and possible hidden layers. The input layer receives input from the environment. When used in a robot controller, it usually receives sensor input, where each sensor is represented by an input neuron. The output layer gives the result of the network. In a robot controller, the output neurons can represent actuators like motors and grippers, where each actuator is represented by a neuron. The hidden layers perform the calculations between the input layer and the output layer. The connections between neurons are weighted, which means that each neuron can have different influences on the other neurons.

Evolutionary artificial neural network (EANN) is an ANN where the weights of the edges between the neurons, and possibly other variables, are evolved. In evolutionary robotics,

reinforcement learning is often used, where the feedback comes from a simulator. A genetic algorithm can be used for the evolutionary process, where the weights and values are represented by the genotype.

It is possible to create very advanced structures in ANNs, with multiple layers and recurrent connections. However, most of the studies that are discussed later in this chapter, which use ANNs, use a single layer feedforward neural network in combination with different robot behaviors. The input neurons, in a single layer feedforward neural network, are connected directly to the output neurons, without directed cycles in the network.

2.3.5 Simulator

Using a simulator instead of real robots brings both advantages and drawbacks, which are discussed in this section. Additionally, two existing robot simulators are presented.

Advantages and drawbacks
Experiments with robot controllers in simulators are less time-consuming than on real robot hardware. Especially when dealing with learning and evolution, a high number of lengthy evaluations are required [Floreano and Mattiussi, 2008]. For real robot hardware, the environment and the initial positions of the robots should be restored, before each evaluation. This is a very time-consuming and repetitive process, especially for humans. Even though the process is faster in simulation it can still be very time consuming on a single PC. Evans et al. solved this by turning off unnecessary features of their simulator [Evans et al., 2010], like friction in their case, and distributing the evaluations among a cluster of 50 machines.

By using a simulator, we avoid the disadvantages of working with robot hardware. The hardware itself can be very expensive, and can easily be damaged during experiments. This can happen in several ways, including components overheating or poorly performing individuals crashing during the evolution [Floreano and Mattiussi, 2008]. Energy consumption is additionally a problem. Rapidly changing and recharging batteries is very time consuming. The other alternative is to use long energy wires to each robot, but with a high number of robots in a swarm, the cables can quickly become tangled. Simulation brings increased modifiability as different components, methods and environments can be altered and tested in the system.

The largest drawback is that a simulated system is not a perfect representation of the real world. A controller that has evolved or learned from simulated training might depend too much on reliable sensor readings or precise motor control. In real life, the sensors might get noisy readings, the motor can drift and the wheels can start spinning. In an experiment by Grefenstette et al [Grefenstette et al., 1990], their robot controller performed worse in real life than in the simulator. This was due to the noise on sensors in real life. They showed that this performance difference could be reduced by introducing noisy sensors in the simulation, which encouraged solutions that are more general.

Existing simulators
This section contains brief descriptions of two simulators often used for swarm robotics, ARGoS and Webots.

ARGoS is a multi-robot simulator developed for the Swarmanoid project [Pinciroli et al., 2012]. It can handle large swarms of robots of different types. ARGoS is, according to the authors of the article, the first multi-robot simulator to be both efficient and flexible. It has good performance and is highly customizable for different experiments. Both 2-dimensional (2D) and 3-dimensional (3D) physics engines can be used, also in parallel where the simulation space is partitioned into subspaces with different physics engines. The simulator supports three types of visualization: an interactive GUI based on Qt4 and OpenGL, a high-quality rendering engine based on POV-RAY, and a text-based visualization. The controllers need to be written in C++. ARGoS is a free, open source software that currently runs on Linux and Mac OS X.

Webots is a mobile robotics simulation software [Michel, 2004]. It was developed by the Swiss Federal Institute of Technology in Lausanne in 1996, and is now a commercial product available from their spin-off company, Cyberbotics Ltd. Webots can model and simulate any mobile robot, and includes a library for sensors and actuators, which can be used to modify or make a new robot. Controllers can be written in C, C++ , Java, Python or using MATLAB. Webots uses Open Dynamics Engine library for physics simulation. It uses a virtual time, which means that it can run faster simulations than what is possible for physical robots. A controller can easily be transferred to physical robots. Third party software can connect to Webots, to either supervise simulations or for global and local communication within a multi-agent system. The supervisor capability can be used for computationally expensive simulations like genetic evolution and neural networking.

2.4 Structured Literature Review

The purpose of this structured literature review is to get an overview of the field of swarm robotics. It is based on a template made by Anders Kofod-Petersen, at the Department of Computer and Information Science, Norwegian University of Science and Technology.

2.4.1 Identification of research

We used the search terms in Table 2.1 when using the online libraries: Google Scholar, IEEE Xplore, SpringerLink and ScienceDirect. The table is divided into groups and terms. Each group contains terms that are either synonyms, different forms of the same word, or terms that have similar or related semantic meaning, within the domain. A

logical expression of the search terms was created by applying AND between groups, and OR between terms within a group.

Domain specific sources were found talking to domain expert, Jean-Marc Montanier, who has a PhD in the field of swarm robotics, and now works as a postdoc at Norwegian University of Science and Technology. He recommended us to look into the work of Dario Floreano, and in the process we found relevant projects, including Swarm-bots and Symbrion. The researchers behind the Swarm-bots project have continued their research into two following projects Swarmanoid and E-Swarm. In addition, we found papers from the courses TDT04 Advanced Bio-inspired Methods and TDT11 Advanced Bio-inspired Technologies at Norwegian University of Science and Technology.

Table 2.1: Search term groups

	Group 1	Group 2	Group 3	Group 4	Group 5
Term 1	Swarm robotics	Assembling	Evolutionary algorithm	Simulation	Rough terrain
Term 2		Docking	EA		Slope
Term 3		Gripping	Artificial Neural Net		Hill
Term 4		Aggregation	ANN		Obstacle
Term 5			Evolution		
Term 6			GA		

2.4.2 Screening process

By performing a screening process we reduced the number of studies from 876 to 25. First, we did a basic screening by removing studies perceived irrelevant based on their title. Second, we did a quality screening in three steps, based on the criteria in Table 2.2. We used the remaining 25 studies as a basis to write the rest of this chapter.

Table 2.2: Inclusion and quality criteria for the screening process

Id	**Criteria**	**Screening step**
IC 1	The studys main concern is **P**	First
IC 2	The study is presenting empirical results	First
IC 3	The study focuses on **M**	Second
IC 4	The study describes an **S**	Second
QC 1	There is a clear statement of the aim of the research	Third
QC 2	The study is put into context of other research	Third

Inclusion (IC) and quality criteria (QC) for the screening process. We have defined the specific problem **P** as: how to get robots to traverse in environments where they cannot traverse alone, the methods **M** as: self-assembling and evolutionary algorithms, and the resulting system **S** as: a homogenous controller system for a swarm of robots.

2.5 Related Systems and Projects

This section describes the projects and studies we found in our literature review: the Swarm-bots project, the Swarmanoid project, the Symbrion project and four individual studies.

2.5.1 Swarm-bots

The Swarm-bots project is a swarm robotics project, coordinated by Dr. Marco Dorigo. It is inspired by the collective behavior in social insect colonies and other animal societies, especially the ability of self-organization and self-assembling [Dorigo et al., 2005]. The project's duration was from 2001 to 2005.

The main goals of the project were to make a new swarm robotic system called swarm-bot and to bring self-assembling to the forefront of multi-robot research [Dorigo et al., 2006]. The project used homogenous robots called s-bots. When the s-bots are assembled into a group, the multi-robot organism gets the name swarm-bot. The s-bots are by themselves relative simple robots, with limited sensors, motors and computational capabilities [Mondada et al., 2004]. The sensors include infrared proximity sensors, light and humidity sensors, and accelerometers. Each s-bot has one gripper for a firm and close grasp between two s-bots, and one gripper on a flexible arm, giving a wider grasp, where the robots can move more freely. Brooks' subsumption architecture is used in cases where it is possible to decompose the tasks into sub-tasks and corresponding behaviors, while evolutionary algorithms are used when training is needed. At the time of this project, little attention had been devoted to research into self-assembling of robots. The mechanism is highly complex, which was troublesome for existing systems [Dorigo et al., 2006]. By addressing and reducing this complexity, the ambition was to inspire more research and bring self-assembly to the forefront of multi-robot research. The project has led to publishing of several papers, most of which explores the possibilities of the swarm-bot and the control of the s-bots.

In experiments with aggregation and coordination, a behavior-based controller was used to move the robots into a chain formation [Nouyan and Dorigo, 2004, 2005]. The objective was to make a virtual chain between a food source and their common nest, where the distance between them was too large for a single robot to perceive them both. The controller had specific rules for transiting between the different behaviors. The behaviors are: search, explore, chain, and finish. The transportation of a large food item was considered in another experiment [Gross et al., 2005b]. This time the robots connected physically with their gripper, to get more pulling force and be able to transport a food item, which was bigger than the s-bots.

One study presents the first robot controller that is capable of functional self-assembling, which is the ability of the robots to assemble by themselves in response to the task or environment [O'Grady et al., 2005]. The robots' task is to navigate independently

towards a food source, represented by a light, which happens to be on the other side of a steep hill. If a robot finds a hill that is too difficult to pass over alone it will move back and change into an aggregation phase. In this phase it turns its LED lights blue, signaling that it needs help. After attracting nearby robots, they will self-assemble and align their direction, before moving towards the hill. In the experiment, a single robot could not complete the task, two robots succeeded 65 % of the time, and three robots succeeded 86.67 % of the time. The controller is made with a combination of behavior phases and an artificial neural network. The self-assembly process is also showed to work in more rough terrains [Gross et al., 2005a,b], however the advantages of self-assembly in these types of terrains are not considered in these studies.

Another study describes how the controllers, developed for hole avoidance, can be used to pass over a trough that can be bridged by a swarm-bot [Trianni and Dorigo, 2005]. The swarm-bot was already assembled before the start of these experiments, as the study is more about why the robots should self-assemble instead of describing how to do it. The decision of whether to move back from a gap or bridge over is a collective decision that emerges purely from the interactions between the s-bots and the environment. An ANN, whose parameters are set by an evolutionary algorithm, is controlling the robots. The fitness value is calculated by using the distance between the start and end point, divided by the maximum possible distance the swarm-bot could possibly cover. This gives a number between 0.0 and 1.0.

In an experiment, from another paper, different methods of communication between the robots are explored [Trianni and Dorigo, 2006]. The communication methods insects use can be grouped into indirect communication, direct interaction and direct communication. Ants use indirect (stigmergic) communication by leaving a trail of pheromones, to find good paths, but can also use direct interaction with their antennas. Direct communication is used by the honeybees with their waggle dance, to signal the direction and distance to interesting locations, such as sources for food or housing. This study looks at the use of different communication strategies in a hole avoidance scenario with s-bots connected in a square formation. The scenario was tested with no direct communication, handcrafted signaling and a completely evolved approach. The study concludes that the evolved approach gives the highest performance, by reducing the number of times the swarm fell into the hole.

Several of the robot controllers in the Swarm-bots project were made by evolving artificial neural networks. In one experiment, a feed-forward ANN was used to guide the robots in an assembling process. The s-bots' sensor inputs were preprocessed, before it was applied to the ANN [Tuci et al., 2005]. The ANN had three output neurons, one for each of the two wheels and one for the firm gripper. The connection weights were trained with evolution, where the fitness was evaluated using Swarmbot3D, a simulation tool created for this project.

Multiple studies discuss a significant high variance when evaluating fitness for robot controllers. There are several reasons for this. The robots are controlled by themselves,

with no global interactions or knowledge. How they act depends on what they see around them, locally, and this can change during each run. The robots are usually placed randomly in the environment, to make sure the controller works, no matter the starting positions. In addition, many robots are used, and most of them need to cooperate in order to get a satisfactory result. In order to obtain more reliable fitness evaluations, Dorigo et al. ran the evaluation eight times [Dorigo et al., 2004], and Trianni and Dorigo ran the evaluation five times [Trianni and Dorigo, 2005]. The average from these evaluations was used as the final fitness result. It is also interesting that they used only reproduction without re-combination. Both of them used an approach where the 20 % best individuals were selected for reproduction, and each generated five offsprings. Mutation was then used on the offsprings with a 3 % chance of flipping each bit. Dorigo et al. did not use elitism, while Trianni and Dorigo skipped mutation on one of each of the five offsprings.

The Swarm-bots project gave several contributions to the field of swarm robotics. It is one of the first works in which functional self-assembly in a homogeneous group of robots has been achieved [O'Grady et al., 2005]. It demonstrates that evolution is able to produce a self-organized system that relies on simple and general rules, scales well to the number of robots, and is robust to changes in the environment. It also brought both a hardware and a software robot system in the s-bots and the Swarmbot3D respectively.

The project has multiple implications for us. It has showed us that evolving controllers for a swarm of homogeneous robots is possible, which have influenced our decision to use artificial evolution to train our robot controllers. This does, however, increase to the computation effort, which grows even further if multiple evaluations of each individual is needed, to get an accurate fitness value. The project has experimented with the ability of a swarm of robots to navigate in rough terrain. This was a key inspiration for our thesis, especially regarding Research Question 1. This also meant that we did not have to research the same scenarios regarding self-assembling, as in this project, such as going up a steep hill and pulling large objects. The s-bots were also an inspiration when we designed our virtual robot, and decided which sensors to use. The main task of navigating towards a goal, represented by a light, will also be used in our system.

2.5.2 Swarmanoid

The Swarmanoid project followed the Swarm-bots project, with Dr. Marco Dorigo as coordinator. It was the first project to study how to design and control a swarm of heterogeneous robots, operating in a 3D environment [Swarmanoid, 2010]. The project's duration was from October 2006 to September 2010. In addition to using the s-bots from the Swarm-bots project during this project, three robot systems were constructed: a foot-bot, a hand-bot and an eye-bot. The foot-bot is similar to the s-bot, but is larger and has a different gripper. The new gripper is not capable of lifting another robot, like the s-bot's gripper is. The hand-bot can climb up shelves, and has a wire it can

connect to the ceiling, by launching a magnet. It uses the wire for safety reasons and to lower itself down to the ground, when holding an object. The hand-bot can also grab larger objects than the other robot types. To move on the floor, it needs help from at least two foot-bots. The eye-bot can fly and lift the other robots. It keeps an overview over the environment and communicates with the other robots, to guide them. By using these robots, the Swarmanoid project demonstrates how different types of robots can help each other. The ARGoS simulator was created during this project [Pinciroli et al., 2012].

During the Swarmanoid project, several articles were published. The next paragraphs describe the most relevant ones for this master thesis. Most of the experiments were performed with a swarm of either s-bots or foot-bots.

In the first article, a mechanism for autonomous self-assembling was introduced, called Swarmorph [O'Grady et al., 2009]. This is a mechanism for distributed morphology generation. The authors saw lack of morphology control as a limitation, since the resulting morphologies were not necessarily able to perform any tasks. The Swarmorph mechanism controls, through local morphology extension rules, the morphologies that are formed. By doing this, task specific morphologies can be made. The experiments were performed using a swarm of s-bots. All the experiments were first done in simulation, and then a few were performed using the real s-bots. The mechanism made the swarm of robots self-assemble into different morphologies, such as a line, a star, an arrow and a rectangle. The s-bots communicate through LED lights. A robot indicate with its LED lights that it has a free connection point. Non-connected robots wander around until they discover a free connection point. A robot that is navigating towards a connection point continuously calculates the approach vector. Swarmorph works in parallel, since a morphology can have several open connection points simultaneously. After performing the experiments, the mechanism was found both robust and precise, since they achieved a high success rate when building four different morphologies. They also found the mechanism to scale well, when using a large number of robots in simulation.

The second article introduces a method for self-assembling between two robots, without direct communication [Ampatzis et al., 2009]. The controllers are evolved artificial neural networks, which control all the actuators of the robots. Based on the robots' interactions, roles are allocated between them, which cause dynamic specialization of the robots. The fact that there are no direct communication means that the LED lights are not in use, unlike in the previous article. Two s-bots were used in the experiments. The two robots were placed next to each other before each experiment started, due to limitations regarding the camera range. Reinforced learning was used for the fitness evaluation, where the robots were rewarded based on a combination of an aggregation component, a collision component and a self-assembly component. This fitness evaluation was meant to lead to a collision-free self-assembling for the robots. The results of the experiments showed that it is possible to make two robots autonomously self-assemble, using controllers with evolved artificial neural networks. The scalability of this system was not tested in the experiments. A major contribution from this article is

to demonstrate that such controllers can be reliable and efficient for fine sensory-motor coordination, like connecting two robots. The authors state that it is important to understand which principles make evolution produce efficient rules for guiding robots, rather than identifying each of the rules.

In the last one of these articles, a control mechanism called enhanced directional self-assembling (EDSA) was introduced [Mathews et al., 2011]. This mechanism was used for fast morphology growth through high-speed communication in a swarm of foot-bots. A robot connects to another robot by first recruiting the best located neighbor, and second, guiding it to an optimal connection point on its chassis. Communication between the robots are based on a combination of radio and infrared lights. This makes them able to give location based messages to each other. To test the EDSA, the swarm was placed in a dynamic environment, so that the morphology was grown in response to its environment. An example of such a response is forming a chain of robots to cross a hole, too large for a robot to cross alone. In the experiments, morphologies were grown, and the results showed that the EDSA was precise, robust and fast. This also showed that the EDSA enabled morphology growth in motion, parallelism, adaptive recruitment and connection in dynamic environments. In the end of the article about the EDSA, the authors also presented ongoing work. First, how the EDSA can form larger task-specific robot morphologies, and second, how to form large morphologies by forming small segments in parallel and then assemble the small segments into the large target morphology.

From the articles above, we found useful points to consider for our project. Our robots have a connection area on the back, and a gripper in front, which would limit the morphologies to either chain formations or chain formations with branches. This makes the self-assembling easier for the robots, requiring less communication, and they should still be able to perform the task. The task-specific morphologies used in Swarmorph would be interesting to implement in our controllers, and can improve the performance, but would require a larger project scope. The different articles used different forms of communication between the robots. The first one used LED lights to guide robots towards free connection slots. The second one used no direct communication, and therefore had to place the robots close to each other to make them interact. In the last article, communication through radio and infrared lights was used. In our project, we considered the different types of communication, and ended up with using LED lights for communication between our robots. One of the reasons behind this decision was that we would use light sensors for converging towards the goal, and could use a very similar approach when the robots needed to communicate about converging towards each other.

2.5.3 Symbrion

Symbrion is a project about self-reconfiguring modular robots, and is a platform for exploring artificial evolution and evolve-ability [Symbrion, 2013]. It started in 2008 and finished in 2013. The robots used in Symbrion are different from the ones used in Swarm-

bot and Swarmanoid. Their mobility by themselves are more limited compared to a single s-bot or foot-bot. Their wheels are only used for movement in the self-reconfiguration process. However, they have the ability to self-assemble into much larger shapes, even in vertical formations, where each robot is only a small part of the system. This can lead to chained robots with snake-like movements, or robots with leg-like structures similar to insects, or possible structures not even imagined. When self-assembled these robots have increased mobility, and can be able to climb stairs, or walk over obstacles. The Symbrion robots do also have powerful on-board computational resources. These can be used to do on-board simulations, so that the robots can make adjustments in unknown environments.

Even though the robots are different from the ones we will use in our project, we can learn from their experiences, especially from the studies about evolutionary processes. One of the articles studies the fitness functions used in evolutionary robotics [Nelson et al., 2009]. In the article, it is argued that robot designers may not be able to provide appropriate control algorithms in the case of unforeseen situations or environments. Thus, the robot should be able to learn and adapt their controller without supervision from humans. In this process, the fitness function is responsible for comparing potential controllers. Further, the article evaluates the benefits of different fitness functions. A behavior based fitness function is hand-formulated to the task, and can measure multiple aspects of what the robot is doing. To create these functions, a high degree of a priori knowledge is needed. Another type, called aggregated fitness function, needs less a priori knowledge and only evaluates if the task is completed or not. Using less a priori knowledge is advantageous, as it can lead to solutions that are more general. However, by using an aggregated fitness function, a working solution can be hard to find, as the evolutionary process can not get feedback about being close to a solution, which can be problematic in the selection process. The article also discusses functional incrementing fitness functions. This is the process of explicitly training sub-behaviors, one at the time, before training the more complex behaviors. This addresses the problem that an initial population might have large problems with completing a difficult task. In these methods, the designer is responsible for both the evaluation metrics and for structuring the search path of the controller's search space. This can lead to a restriction on the possible cooperation methods between the behaviors, and the evolution might not be able to reach the entire solution space.

One of the experiments of the Symbrion project involved improving the evolutionary engine by including a distance and diversity measure during the evolutionary process [Winkler et al., 2011]. It was shown that there is a correlation between the relative diversity of the population and the relative increase in fitness between generations. A low diversity probably means that the whole population is converged to individuals in a local optimum. Exiting the local optima is very advantageous, as it can lead to better solutions in a shorter time. This can be accomplished by increasing the mutation rate or by initializing new individuals.

The studies from Symbrion imply an advantage of using a behavior based fitness function,

compared to an aggregated or incremental fitness function. This discussion impacted our project, and we chose to create a behavior based fitness function. Winkler et al. demonstrated that a diversity measure could be used to exit local optima and speed up the evolutionary process. This discussion influenced our project, because speeding up the evolutionary process is important when evolving robot controllers. This is normally a very tedious process, which several of these related projects confirm, and we need an acceptable runtime to do numerous experiments.

2.5.4 Other systems

This section describes four individual articles found in our literature review.

A study from 2007 focuses on self-organized aggregation in swarm robotic systems [Soysal et al., 2007]. The approach of evolving an artificial neural network controller is investigated and compared to handcrafting a probabilistic controller. The evolved controller uses a single-layer feedforward neural network. Inputs from microphones and infrared sensors are fed directly to three output neurons: the left wheel, the right wheel and a speaker. In the evolutionary algorithm, elitism is used by passing the 10 % best of the individuals directly to the next generation. The rest of the individuals are recombined using crossover with 80 % probability, and mutation with 1 % probability of flipping each bit. The probabilistic controller is a finite state machine with behaviors as states. Switching between the behaviors is managed by specific rules, such as when another robot is close, or by a probability rule. The results showed that the evolved controller performed better than the probabilistic, even when trying multiple strategies for the probabilistic controller. The article ends with four rules of thumb for evolving robot controllers. The most important one for us is to keep the number of simulation steps as low as possible, but still give the robots enough time to accomplish their goal. This is important to be able to increase the number of evaluations, and keep an acceptable runtime of the system. The number of simulation steps is the number of times the simulator is updated during a single simulation. This can be reduced by either reducing the number of updates per second or by reducing the total duration of each simulation. The number of evaluations is the number of times a specific controller will be tested in a simulation, before combining the results to a fitness value. When combining these results, the article states that using the minimum score from the evaluations is better than using the average, the median, or the maximum score. The research showed that, in this case, an artificial neural network controller had higher performance than a probabilistic controller. When evolving our controller, we will test if the rules of thumb can improve the results from our evolutionary processes.

A study from 2009 describes an amoeboid modular robot called Slimebot [Shimizu and Ishiguro, 2009]. The Slimebots use velcro strap to connect, and directly contacting electrodes to communicate with each other. In the control mechanism there is an oscillator in combination with different control algorithms. The purpose of this study was to show real-time adaptive reconfiguration of a swarm of Slimebots, by demonstrating that an

assembled swarm can reconfigure when there are obstacles ahead. An example of this is when an assembled swarm of robots approaches a narrow pathway, and they need to reconfigure the shape of the swarm to get through. The results showed that this was possible both in simulation and with physical robots, and that the Slimebot enabled both adaptive amoeboid locomotion and reconfiguration. The results also showed high adaptivity, high scalability and high fault tolerance. This study shows an example of how a swarm of robots can adapt as a response to the environment.

An evolutionary robotics approach is presented in a study from 2010 [Ohkura et al., 2010]. This approach is to design controllers by evolving artificial neural networks. The performance depends on the topological structure of the neural network. To find the best structure, four recurrent neural networks were tested in simulations, on a cooperative package pushing problem. The results of these experiments showed that the artificial neural network where the hidden layer was a small-world graph, had the best performance, flexibility and scalability. In the experiments, ten robots with sensors, and four packages were used. Different packages required different number of robots to be pushed. The input layer in the neural network represents the sensors, and the output layer represents the motors. The fitness function in the evolution is based on points that the swarm gets for pushing a package to the goal line, for touching a package for x number of ms, and for the distance between a package and the goal line when the timesteps run out. In the experiments, the robots start next to each other behind a specific line, if they successfully complete the task, five additional tasks are given. For the additional tasks, they start on a random position in the field, to avoid overfitting. The fitness points are summed up for the tasks, with more weight on the first task. If a swarm does not complete the first task, the collected points are given as the fitness points. Giving points for the distance to the goal, was an inspiration when we made our fitness function. We also used a similar environment setup by combining the random placing of the robots, with placing them in a fixed area, in our case a defined starting area, when starting a simulation.

A novel heterogeneous robot approach enabling aerial robots to aid and control ground-based self-assembling robots is studied in a paper related to the E-Swarm project [Mathews et al., 2012]. The E-Swarm project continued the research of the Swarmanoid project. The aerial robots, the eye-bots, use their enhanced view of the environment to detect the need of self-assembling the foot-bots on the ground. Two images, taken by the eye-bots, are used to create a height map representation of the environment. After the height map is ready, a simulation on-board the eye-bots is used to determine what morphology the foot-bots should use. Finally, each of the foot-bots are guided into the target morphology formation, and the navigation task is started. In their experiment, the foot-bots were guided over a steep slope towards a food resource on the other side. The eye-bot was stationary attached to the ceiling, and a pre-calculated height map was used, which means that this part was not shown. This study is relevant for both of our research questions. It shows a way in which self-assembling improves the mobility of a robot swarm, and a way to get the robots to decide when to self-assemble. This means that

we can direct our research towards finding other possible improvements for mobility of a swarm, and ways to decide when to self-assemble.

2.6 Background Discussion

In this section, we discuss how the studies described in this chapter have influenced our project. We will explore how these studies solved the problem of traversing in difficult environments, how they made their robots self-assemble, what the limitations of current research are, and how the studies in general have impacted our project.

Traversing in difficult environments
Physically connecting robots is a solution which can lead to improved mobility in difficult environments. In the projects described, self-assembling improved the swarm's ability to drive up steep slopes, pass over gaps and remain upright in uneven environments. During our research, we found two main kinds of robots that can connect physically to each other. The first kind is the small, wheeled robot with grippers, used in the Swarm-bots and Swarmanoid projects. The second is the kind of modular robots used in the Symbrion project. These robots work as parts and joints in larger structures of robots, and might not rely on wheels to move around. Most of the solutions involve no global interactions or control. Each robot is controlled by analyzing the input from their sensors, like infrared and proximity sensors, and applying the results to the actuators, such as wheels and gripper.

There are several methods used, to make robots self-assemble. One method is to let the creators design a complete, handcrafted controller. The communication methods, the behaviors, and the handling between sensor input and actuator output, are defined and implemented by the creators. Another method is to use evolutionary robotics. Here, the communication methods, behaviors, and handling of input and output are only partially defined by the creators, and are then evolved to adapt to a given problem or environment. Most of the systems discussed in this chapter, included aspects that were handcrafted by the designer, such as the communication and the behaviors of the robots, and other aspects that were evolved through an evolutionary process, including the handling between sensor input and actuator output.

Limitations of current research and our contributions
We think that interaction and collaboration between researchers and studies could be utilized more in this field. The research we found was often inspired by other research, in their methods and hardware, but we did not find much work on improving upon the work of others in similar settings with the same robot hardware. In the field of machine learning, one can find hundreds of standard and free datasets. By using these, it is easy to compare multiple classifier algorithms. The field is increasingly brought further, as better algorithms are found and discussed. We think that more collaboration and standard benchmarks in swarm robotics would enhance the field, and we hope to see

more of this in the future.

The current research in self-assembling swarm robotics has managed to show that self-assembling can improve performance of a swarm in multiple specific situations. Before swarm robotics goes from the research labs and into use in the outside world, experience with more general adaptivity is needed. We think that the current research has shown a good start, and the researchers should continue to search for more situations and environments, where self-assembling would give an advantage to a robot swarm.

It is natural to consider improving on these areas in our project, as it can lead to contributions to the field. The lack of standard benchmarks is too large in scope for this project and should be considered by researchers with years of experience in the field. We are, however, able to contribute to the current research by looking at more situations and environments, where self-assembling in a swarm of robots can be advantageous. We have not found any research about how self-assembling can improve the mobility of swarm robots in environments with both ice and strong wind. We will contribute to to the field of swarm robotics by researching self-assembling in these kinds of situations.

Impacts on our project
As implied over, the research we found was a significant inspiration when deciding to do this project, especially the experiments describing the ability of a swarm of robots to navigate in rough terrains. The robots we are going to use are inspired by the s-bots from the Swarm-bots project and the foot-bots from the Swarmanoid project. The research demonstrated that evolved controllers could get better performance than handcrafted controllers. This strengthened our desire of using an EA in our project. These systems involved complex subsystems such as ANNs, EAs and simulation of robots. The studies we found gave us recommendations for improvements to one or more subsystems. This includes using an average over multiple simulations, to get a more precise fitness value in the EA. More information on impacts from each individual study or group of studies can be found in the previous sections where the studies are discussed.

The research on the different robot controller architectures were also interesting when deciding if we should use one of them or make a new architecture. This impacted our project, because we chose to create a controller based on Rosenblatt and Payton's architecture [Rosenblatt and Payton, 1989], with arbitration via action selection. We also decided to use a manager similar to the module manager, and multiple behavior modules proposed by Rosenblatt [Rosenblatt, 1997]. This decision is described more thorough in the Methodology chapter.

Chapter 3

Methodology

In this chapter, we describe the tools and methods we have used to investigate our research questions. The overview of the system is presented in Section 3.1. Section 3.2 contains the description of our robots. The scenarios used in our experiments are demonstrated in Section 3.3, and the robot controller is described in Section 3.4. The choice of creating a simulator is explained in Section 3.5, and the modeling of the physics in the simulator is described in Section 3.6. Section 3.7 contains the description of the evolutionary algorithm used in this project.

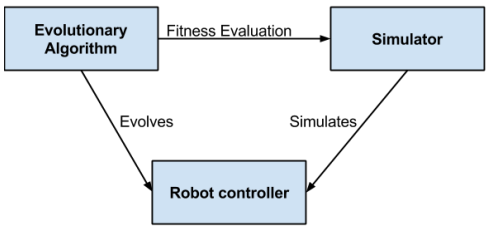
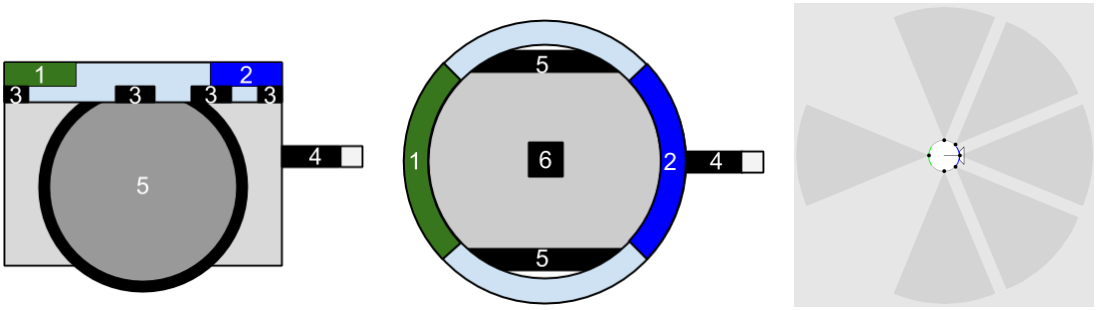


Figure 3.1: Primary system components and their interactions

3.1 System Overview

Our system consists of three primary components: the robot controller, the simulator, and the evolutionary algorithm. The interactions between these components are illustrated in Figure 3.1. We use the evolutionary algorithm to evolve the robot controller, by adapting weights and thresholds in the controller. The simulator is then responsible for performing the fitness evaluation on the controller. The components are designed to have low coupling between them. One can for instance use the evolutionary algorithm to solve entirely different problems, by only making small changes to a few key classes (fitness evaluator and development). The low coupling should make it easier to modify, reuse or replace each component.



(a) Side view (b) Top view (c) Top view, in the simulator

Figure 3.2: Multiple views of the robot, all facing right

3.2 The Robot

The robot swarm consists of a set of equal robots. We wanted to design the robots in a realistic way, and chose to keep their properties close to the e-puck robot [Cyberbotics Ltd., 2007], an already existing physical robot. We have also been inspired by the s-bot and foot-bot, described in the Background chapter. Our robot is illustrated in Figure 3.2. We use similar sizes as the e-puck robot, which is 5 cm in height, and has a diameter of 7 cm on the top ring. The wheels have a radius of 2.5 cm. The total weight of the robot is 200 g, including wheels and sensors. We have given small improvements to the robot's sensor and motor abilities, to make the robot able to reach a top speed of 0.25 m/s. It was important for us to keep the robot realistic, even though we did all our experiments in simulation, to get more meaningful results, which are more applicable outside the simulated world.

The devices and features highlighted in Figure 3.2 are: two partial rings of LED-lights (1 and 2), multiple light sensors (3), a gripper (4), two wheels (5) and an accelerometer (6). The LED-lights are primarily used to signal the direction of the robot to other robots. If a robot wants to self-assemble to another it uses its gripper. The gripper contains a small magnet, which can be connected to the back of other robots. The green LED-light makes it possible for the robots to converge toward this connection area, without requiring communication that is more advanced. The light sensors are, similarly to the e-puck robots, used for both light and distance detection. The distance is calculated by sending out infrared light and measuring how long it takes before it is reflected back. The range of the distance sensor is highlighted as the cone shape coming out of each sensor (the black dots on the robot) in Figure 3.2(c)

The robot also has hole detection sensors, below the light sensors. These work in a similar way as the distance detection in the light sensors, but the sensors are tilted downwards, which makes their range shorter. The hole detection sensors can detect a hole in the ground within 10 cm of the sensor.

3.3 The Scenarios

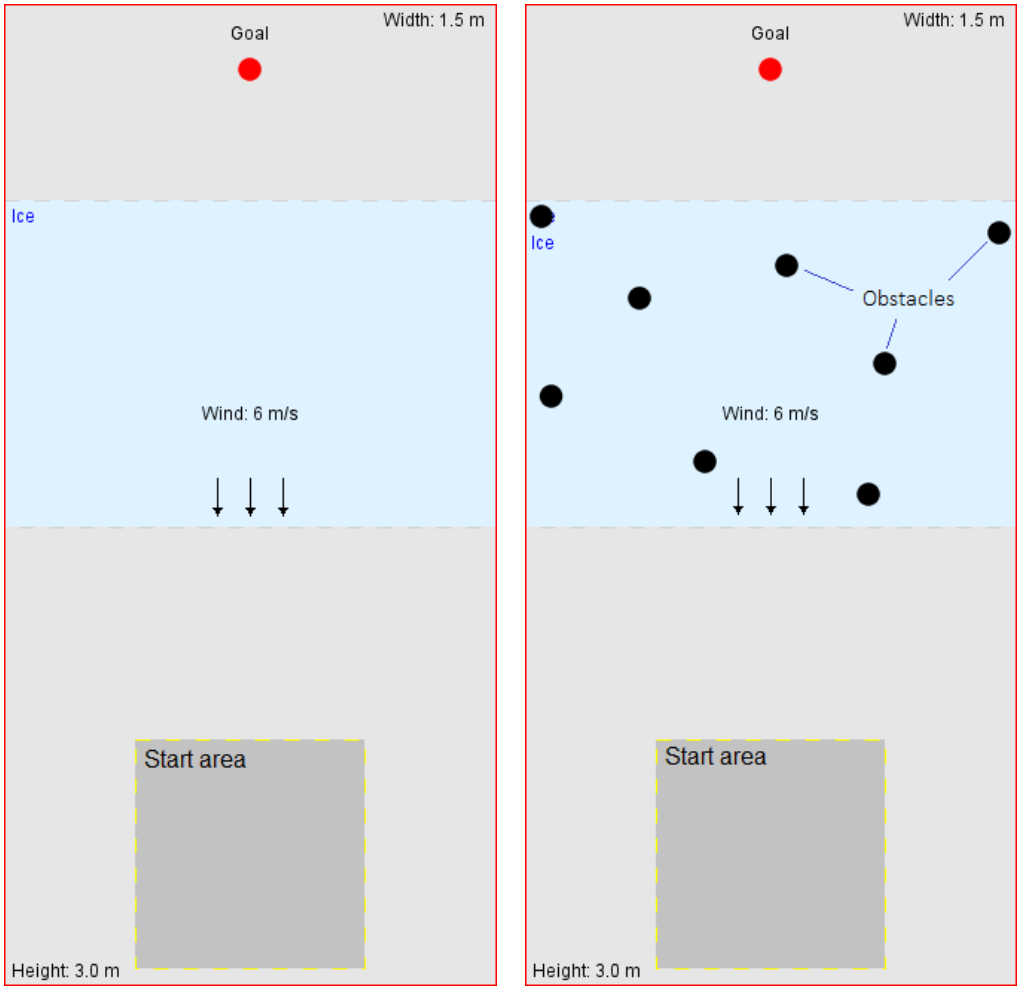
We have performed experiments with the robots on four scenarios (environments). Each scenario has a starting area, where the robots are placed in the start of every simulation. Both the position inside the starting area and the rotation of the robots are randomized. These random elements are used to avoid overfitting and make the robots more general, by not only training on a specific starting condition. The main task for the robots is then to move to the goal object. This object could be anything, but is often a food source represented by a light. The robots are able to sense the light and converge towards the goal. This approach, with a food source as the goal, is also used in experiments in the Swarm-bots project [O'Grady et al., 2005].

Environmental features such as strong wind and areas of ice, make the task harder for the robots, and are present in all our scenarios. The scenarios are made in order to investigate if the swarm is able to improve their performance by cooperating and overcome these challenging environmental features. Three of the scenarios have additional elements, like obstacles, blocking the way towards the goal, or holes, which the robots should avoid falling into. The wind in each scenario extends across the whole environment and has a fixed direction. The velocity of the wind can easily be changed before starting each experiment.

We have kept the size of each scenario and the number of robots in the swarm consistent to make each scenario comparable to the others. The scenarios are 1.5 meters in width and 3 meters in height (when seen from above). There are walls around the environments, to keep the robots from going outside the boundaries. We performed the experiments with a swarm of four robots. This is enough robots to make them able to explore the environment on their own, and to self-assemble into bigger structures. Having a small number of robots is important to keep the runtime of the system low.

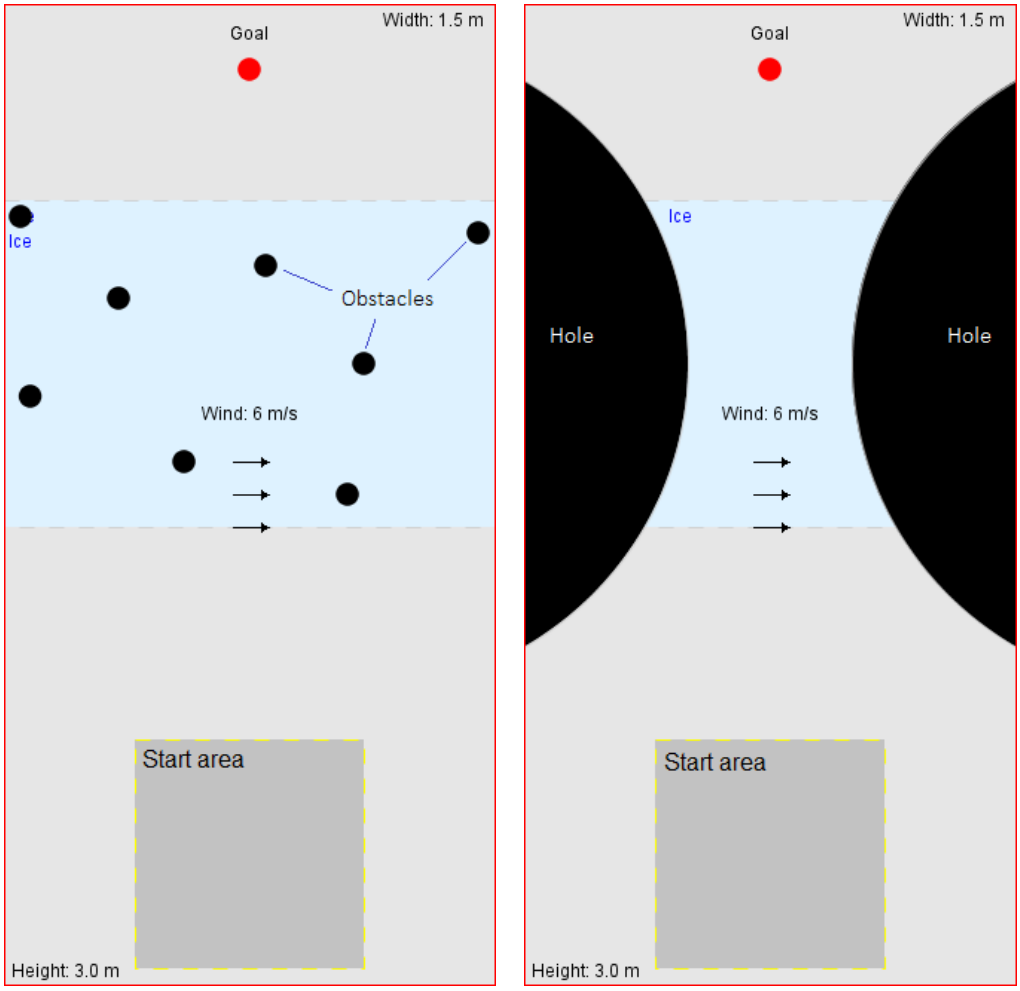
Scenario 1 serves as the basic experiment on a plain environment, with only a single ice area and wind going south. It is illustrated in Figure 3.3(a). This scenario is interesting because the robots need to travel in the opposite direction of the wind to get to the goal. The ice can cause the robots to spin or slide. The combination of ice and strong wind can cause the robots to be pushed back towards the starting area.

Scenario 2 has a similar layout as scenario 1, but in addition it has obstacles spread around the environment. This is illustrated in Figure 3.3(b). The obstacles, resembling rocks, can both have a positive and a negative effect for the performance of the robots. They are additional objects the robots need to avoid, and they force the robots to steer in between them, making the task harder. Moving between the objects should be even harder if the swarm has assembled into a bigger structure. The positive aspect for the robots is that the obstacles can block a part of the wind. When the robots are behind an obstacle they should be less affected by the wind, making the movement easier. The combination of both positive and negative aspects makes this scenario interesting to investigate.



(a) Scenario 1: Opposing wind (b) Scenario 2: Opposing wind with obstacles

Figure 3.3: Scenarios with opposing wind



(a) Scenario 3: Side wind with obstacles (b) Scenario 4: Side wind and holes

Figure 3.4: Scenarios with wind from the side

Scenario 3 is similar to scenario 2, with the same starting area, ice area, goal and obstacles. The difference is that in this scenario, the wind comes from the side, going in the east direction. Scenario 3 is illustrated in Figure 3.4(a). The wind should not directly prevent the robots from moving towards the goal, but it can make it difficult to move left against the wind. This means that it can be challenging for the robots to steer in between the obstacles. As in scenario 2, the robots can get an advantage of moving behind obstacles, because the obstacles can shield the robots from the wind. To prevent the robots from choosing an easy solution, by driving along the right wall, we placed an obstacle near the end of the ice area on the right side. This should cause more trouble for the robots when moving towards the goal, when the wind is strong. It is the only obstacle that has been strategically placed.

Scenario 4 is illustrated in Figure 3.4(b). As in the previous scenarios, the same starting area, ice area and goal area are used. Scenario 4 has no obstacles, and the wind direction is from west to east. There is one hole to the left of the ice area and one to the right of the ice area. In strong wind the robots could easily be pushed into the right hole. On the other hand, if a robot tries to move too fast against the wind, it could fall into the left hole. If a robot falls into a hole, it is not able to get out and can thereby not reach the goal. An interesting part of this scenario, is that self-assembling can make the group more capable of handling the strong wind, however a larger group might be a problem when avoiding the holes, especially when the wind is pushing from the side.

3.4 The Robot Controller

The controller, used to decide how the robot should act in the world, is based on multiple reactive behavior modules. Each behavior module is responsible for a single behavioral goal for the robot, such as avoid collisions or converge towards the goal. The implementation of our robot controller is described in this section.

3.4.1 Behavior Coordination and Action Selection

To coordinate the behaviors, we used a reactive architecture, based on Rosenblatt and Payton's architecture for arbitration via action selection [Rosenblatt, 1997; Rosenblatt and Payton, 1989]. Having worked with the subsumption architecture in a previous project, we agreed with the limitations identified by Rosenblatt and Payton, and their proposed improvements, which is described in the Background chapter.

A reason to use this architecture is to increase the cooperation between the behaviors. This can lead to smarter decisions. An example situation is: the collision avoidance behavior tries to avoid an obstacle by turning either left or right, and the goal converging behavior senses a weak light from the goal on the right side, and will rotate towards it. In this situation, collision avoidance is more critical than goal converging, because the

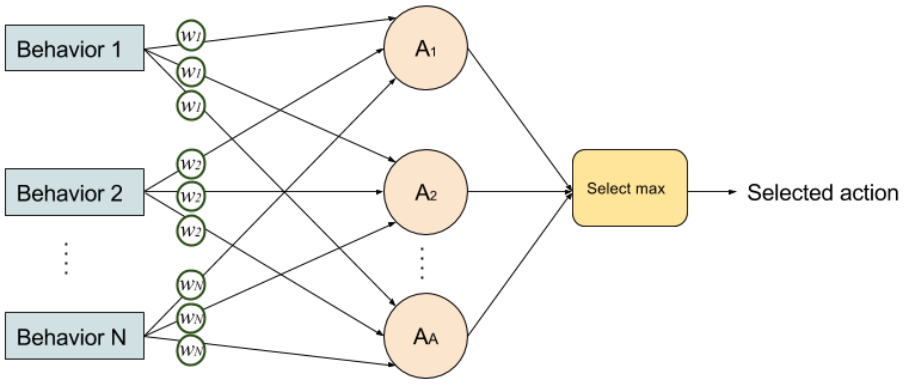


Figure 3.5: The action selection network
Partially based on a text and figure in Behavior-Based Robotics [Arkin, 1998] and the DAMN architecture [Rosenblatt, 1997]. All behaviors cast weighted votes for actions. The action receiving the most votes is selected, and the action is applied on the actuators.

light from the goal is very weak at this distance. By only considering the most critical behavior, the result would be to turn right 50 % of the time and left the other 50 %. This is only a good result for the goal converging behavior 50 % of the time. By voting for actions and considering both behaviors, the result would always be to turn right in this exact situation. The robot would then both turn away from the obstacle, and turn towards the goal, making a good result for both behaviors.

The architecture for arbitration via action selection requires no states or phases for the robots. Using phases, was an approach used in the Swarm-bots project, when self-assembling was used to get the robots up a steep hill [O'Grady et al., 2005]. Having the whole swarm agree upon a state would be hard. This requires both more communication, and negotiation, meaning the robots would need to be more advanced, and more time would need to be used to negotiate.

With our architecture it is possible to divide the behaviors in such a way that each behavior is not very complex by itself. With these simple behaviors, we managed to get good results without using an ANN to control the behaviors. Introducing an ANN would possibly also make the scope of the project too large, considering the time restrictions and the other complex components.

There are both predefined behaviors and actions in our system. These actions include move forward, and turn hard to either the left or the right side. Each behavior uses data from the sensors, to decide which actions it wants to select. To choose an action as a group, a plurality voting procedure is used. The process is visualized in Figure 3.5. Each behavior can cast weighted votes for multiple actions. The action receiving the highest number of votes is selected, and its response should be applied to the actuators. The action selection network selects a new action every time the controller is updated,

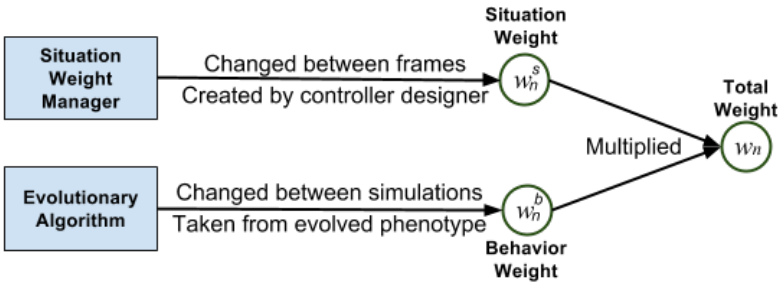


Figure 3.6: The weight update process
The total weight depends on the evolved behavior weight and the situation weight. The resulting total weight is used in the action selection network in Figure 3.5.

which is 30 times per second in our simulator.

The votes for each action are summarized, using Formula 3.1:

$$vote_a = \sum_{n=1}^{N} vote_{an} w_n \tag{3.1}$$

- $vote_a$ is the total sum of the weighted votes given to action a
- $vote_{an}$ is the vote given to action a, by behavior n
- w_n is the weight for behavior n

A vote can be either positive or negative. Negative votes are used to reduce the chance of selecting an action. If a robot has a wall on the right side, the collision avoidance behavior will give a negative vote to the turn right action. All the votes are from -1 to 1. We do not use any restrictions on the total sum of votes coming from a single behavior, as this could lead to situations where it is better to vote for a single action, even though a second action is also very good for the behavior. We have other mechanisms in place, such as the behavior weight, to make sure that one behavior is not too influential without being helpful.

Weight for behavior n is calculated using Formula 3.2:

$$w_n = w_n^b w_n^s \tag{3.2}$$

- w_n is the weight for behavior n
- w_n^b is the behavior weight, given by the phenotype, for behavior n
- w_n^s is the situation weight, given by the situation-weight manager, for behavior n

The votes are weighted, based on two types of weights. Both is in range from 0 to 1, and are multiplied to get the resulting total weight. The weight update process is presented in Figure 3.6. The first type of weight is the behavior weight w^b. It reflects the general importance of the behavior module. This weight is similar to the weight used by

Rosenblatt and Payton [Rosenblatt and Payton, 1989]. It can be changed before starting a simulation, but remains constant until the simulation is over. Each behavior weight was hand designed in Rosenblatt and Payton's system. We are using more behaviors than Rosenblatt and Payton, and want to have more learning in the system. We use our evolutionary algorithm to find good behavior weights.

The second type of weight we used is the situation weight w^s. It reflects the importance of the behavior in the current situation of the robot. Changing the weights during runtime by a module manager was proposed by Rosenblatt [Rosenblatt, 1997] as part of the DAMN structure. Our approach is slightly different from the one in DAMN, as their module manager alters the behavior weights. We keep the behavior weights constant during the simulation, but use an additional weight, the situation weight, to alter the importance of the behavior based on the situation of the robot. These situation weights are adjusted during the simulation by the situation-weight manager. This objective manager is similar to the module manager proposed by Rosenblatt. We changed the name, to make it more explanatory to its responsibility, in our opinion. The situation weight can, for instance, be used to give a high weight to the collision avoidance behavior, if the robot detects an object in close proximity. In our controller, each situation weight can have one of three predefined values. A w^s of 0.0 means the behavior should have no influence, a w^s of 0.5 means that the behavior should have normal influence and a w^s of 1.0 means that the behavior is very important in this situation, and should have high influence. Using a larger set of predefined values is possible here. However, in our case, we wanted a large part of the exploration to be left to the artificial evolution, instead of handcrafting too many specific situation weights.

We decided to use these two weight types, because each of them has a role we want in the system. With the situation weights, we are able to contribute our knowledge to make the robots perform better on a general basis. The behavior weights, on the other hand, can be adapted by evolution to make the robots perform better on more specific cases and scenarios. With a large set of behaviors, it is possible that emergent properties can arise in the system. These properties could be both surprising and interesting. The solutions found by artificial evolution could also be better than the solutions we would handcraft, and are easier to adapt to different environments, as we can run the evolution on new environments to adapt the behavior weights. A running example of voting in the robot controller, with these two weight types, the behaviors, and the actions is demonstrated in Appendix A.

3.4.2 Actions

An action is an operation that the controller can perform on the robot. The behaviors vote for an action to be applied to the actuator, in each frame. The actions have a predefined and fixed response. For instance, the turn hard right action rotates the left wheel forward and the right wheel backward, to make a right turn.

Our approach is slightly different from what Rosenblatt used in the DAMN architecture [Rosenblatt, 1997]. Their controller was used on a vehicle with four wheels. Two of them could rotate, for turning, and two where connected to the motor, to give the vehicle speed. They used both a turning arbiter and a speed arbiter to choose both a turn action and a speed action. Only one arbiter and action selection process is needed in our system, since our robot only has two wheels. A hard turn is achieved by rotating the wheels in opposite directions, and a turn while moving forward is achieved by rotating both wheels forward with different velocities. Because the turning and the speed are so interconnected in our system, and that the robot's max speed is only 0.25 m/s, we decided to only use one arbiter and to have actions, which decide both speed and turning.

Table 3.1: Overview of the actions

Action	**Short description**
Turn hard left	Rotate left on the spot
Forward left	Move forward, while turning left
Forward	Move forward
Forward right	Move forward, while turning right
Turn hard right	Rotate right on the spot
Backward	Move backward
Grip	Try to connect to another robot by gripping
Release grip	Release the physical connection from another robot

An overview of the actions we used in our controller is presented in Table 3.1. The actions are related to moving the robot, or to physically connect the robot to another robot. The grip action tries to connect the gripper (in front of the robot) to another robot. At the same time, it rotates the wheels forward at a slow speed, to make sure the other robot is within the range of the gripper. By doing this, the robots are able to assemble into bigger groups. The gripping process takes 2 seconds, to make it more realistic. During this time, the robot is not able to do other actions. The final action we use is the release grip action. It makes the gripper let go of the other robot. This can be used to get out of situations where the whole group of assembled robots is stuck, or in other situations where assembling is no longer needed.

3.4.3 Common Thresholds

Multiple behaviors use common thresholds, to determine if the light and distance sensor senses a close or a far away object. We used our evolutionary algorithm to find the thresholds that would get the best results. The threshold variables are presented in Table 3.2.

Table 3.2: Distance and light thresholds, used in the controller

Variable name	Description	Example value
Γ_{DF}	Distance threshold, for far distances	30 cm
Γ_{DC}	Distance threshold, for close distances	10 cm
Γ_{LF}	Light threshold, for sensing a light source far away	0.0002 W
Γ_{LC}	Light threshold, for sensing a close light source	0.005 W

3.4.4 Behaviors

This section describes each of the behaviors used in our controller, together with how they vote for actions and how their situation weight is calculated. The purpose of the behaviors is to move the robot, assemble with other robots, increase the safety of the robot, or recover from situations where the robot has not been able to move. Each behavior is presented in Table 3.3, and then described more thorough in the following sections.

Table 3.3: Overview of the behaviors

Behavior	Short description
Goal converging	Converge the robot towards the goal
Wander	Move around in a random direction
Maintain turn	Avoid unnecessary oscillations when turning
Self-assemble	Converge towards, and physically connect to other robots
Follow leader	When assembled, follow the robot in front
Collision avoidance	Avoid obstacles in the world, like walls or rocks
Hole avoidance	Avoid falling into holes
Stagnation recovery	Get away from situations where the robot is stuck
Assemble recovery	Release the grip and move away from the group

Goal Converging
The goal converging behavior tries to converge the robot towards red light. It will vote 1.0, the full positive vote, in the direction with the strongest sensed light, which has a close resemblance to red light. Each sensor is here related to an action. The front sensor will, for instance, vote on the drive forward action, and the sensor on the right side of the robot will vote on the turn hard right action. To achieve this we use Formula 3.3, where x identifies the sensor. $Light_r(x)$ gets its value from the light sensor, but only if the measured color is approximately red. An example situation is illustrated in Figure 3.7. Here, the forward right action gets a full vote, and the forward and hard right actions gets a partial vote. We have a special case for the sensors that go diagonally, which have two related actions. They will normally vote for a forward turn, but if the distance that their sensor measures is very low ($distance < \Gamma_{DC}$), they will vote for a

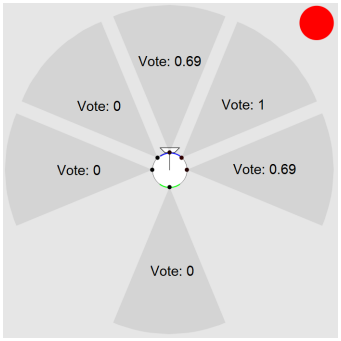


Figure 3.7: The goal converging behavior
Example of votes when the robot is close to the red goal, before weight is considered.

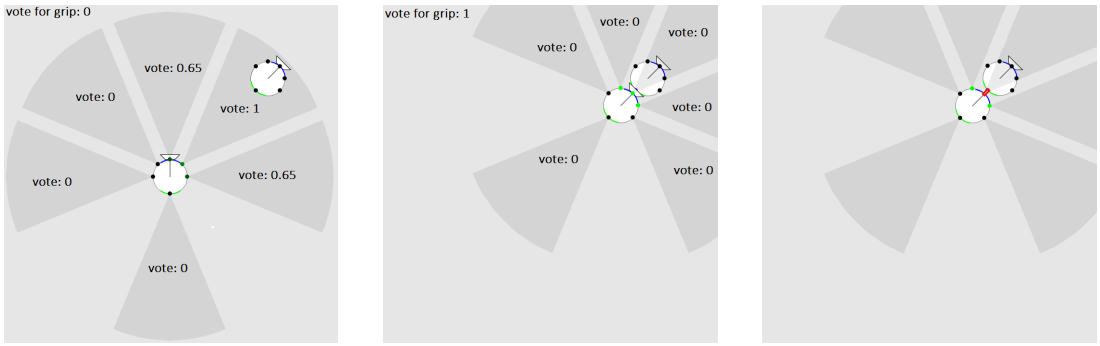
hard turn, to avoid crashing into the nearby object. The situation weight of the goal converging behavior becomes larger than 0 if red light is detected (sensed light $> \Gamma_{LF}$). By looking at the strength of the sensed red light, it decides if the light source is close (sensed light $> \Gamma_{LC}$). The situation weight becomes 1 if it is close, and 0.5 otherwise. Formula 3.3 is used to find the goal converging vote:

$$vote(x) = \frac{light_r(x)}{max(light_r(X))} \tag{3.3}$$

- $vote(x)$ is the vote in direction x, from this behavior
- $light_r(x)$ is the sensed light in direction x , if the color is approximately red
- $max(light_r(X))$ is the strongest and approximately red light sensed in any direction

Wander
The wander behavior is the basic movement behavior, making the robot move around, searching the environment. This is especially helpful in situations where the other behaviors are not voting, as it prevents the robot from standing still. This behavior will vote either a full positive vote or a vote of 0.2 for multiple movement actions, to make the robot move around randomly. Each action has its own probability of getting a full positive vote. These are 1.0 for forward, 0.6 for soft turns, 0.4 for hard turns and 0.4 for driving backwards. The situation weight is always 0.5 for this behavior as it should always have normal influence.

Maintain Turn
The maintain turn behavior helps the robot to turn in a more continuous way, by reducing the stuttering that happens if the robot changes the turning direction frequently. The action selection network will find a new action 30 times per second, so reducing this stuttering is important. At the same time, this behavior should not cause the robot to take longer turns than necessary. To accomplish this, full negative votes are given in the opposite direction of the ongoing turn. If the last action was turn hard right, then both turn hard left and forward left will get a negative vote from this behavior. The situation



(a) Step 1: Converge towards the other robot

(b) Step 2: Connect to the other robot's connection area (in the back), by using the gripper, when it is close enough

(c) Result of self-assembling, the red line over the gripper illustrates the physical connection

Figure 3.8: The self-assemble behavior
In a situation where another robot is seen, example of votes before weight is considered

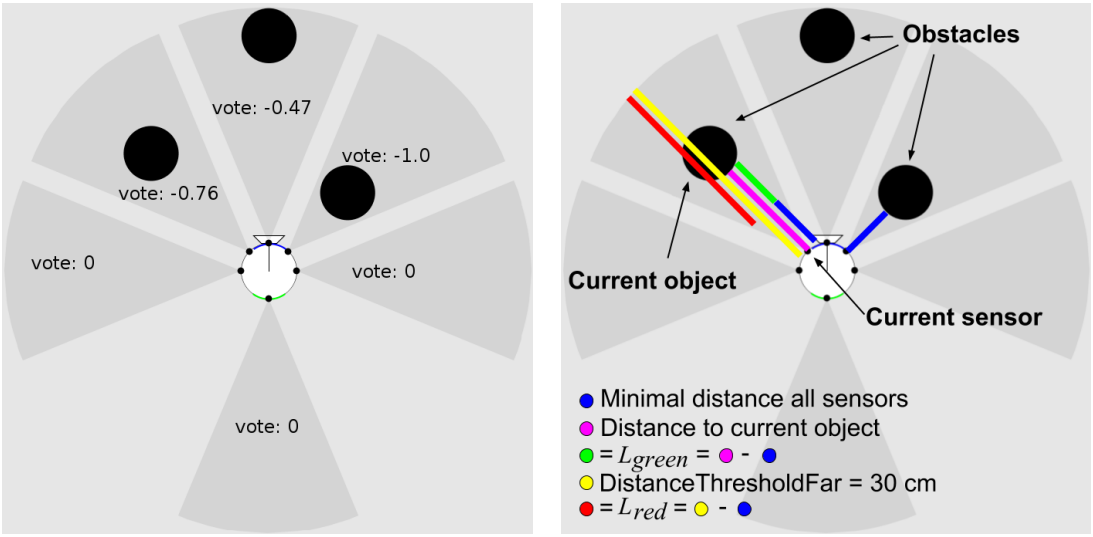
weight is always 0.5 for this behavior as it should always have normal influence. The idea of this behavior was found in the article about the DAMN architecture [Rosenblatt, 1997].

Self-Assemble
The self-assemble behavior is divided into two steps. The first step is to converge towards other robots, as illustrated in Figure 3.8(a). It will give a full positive vote in the direction with the strongest green light, which is the color of the backlights of the robots. The converging part is the same as in the goal converging behavior, where each sensor votes for an action, with a vote that is relative to how strong the sensed light is compared to the strongest detected light of the right color. The second step happens when the robot is close to the connection area of another robot. It will then give a full positive vote for the grip action, to connect to the other robot, as illustrated in Figure 3.8(b). The result of two self-assembling robots is illustrated in Figure 3.8(c). In this case, the behavior will ignore inputs from the other sensors. Formula 3.3, from the goal converging behavior, is rewritten to concern green light instead of red light, resulting in Formula 3.4 for converging (step 1):

$$vote(x) = \frac{light_g(x)}{max(light_g(X))} \tag{3.4}$$

- $vote(x)$ is the vote in direction x, from this behavior
- $light_g(x)$ is the sensed light in direction x , if the color is approximately green
- $max(light_g(X))$ is the strongest and approximately green light in any direction

There is a special case during the converging phase of this behavior. If the back sensor



(a) Example of votes before weight is considered

(b) Illustration of the distances used in formula 3.5: The vote depends on the lengths represented with a green and a red line

Figure 3.9: The collision avoidance behavior

senses the front light (blue light) of another robot behind it, it will try to wait for it by voting negative on both the forward action and the forward while turning actions.

The situation weight can become larger than 0 if the robot has not already connected to another robot. The situation weight is then decided by the strongest light of the right color, detected by the sensors. Green light is considered the right color for all of the sensors except for the back sensor, which according to the special case mentioned above also considers blue light. The situation weight will be 0 if no light of the right color is detected (sensed light $< \Gamma_{LF}$), 1 if the strongest detected light is close (sensed light $\geq \Gamma_{LC}$), and 0.5 otherwise.

Collision Avoidance

The collision avoidance behavior will give a full negative vote in the direction that is closest to another object. The role of this behavior is only to prevent the robot from going in that direction, and not to say which other direction to move in. Other behaviors should make the robot move. This behavior should not avoid light sources like the goal, as this would be counter productive. If there are close objects in multiple directions, the other directions will get a partial negative vote, depending on the distance. An example situation with multiple objects is illustrated in Figure 3.9. Formula 3.5 is used to find the vote from this behavior, in the direction related to sensor x:

$$vote(x) = \frac{L_{green}}{L_{red}} - 1 = \frac{distance(x) - min(distance(X))}{\Gamma_{DF} - min(distance(X))} - 1 \tag{3.5}$$

- $vote(x)$ is the vote in direction x, from this behavior: $vote(x) \in [-1, 0]$
- $L_{green} = distance(x) - min(distance(X))$, illustrated in Figure 3.9(b)
- $L_{red} = \Gamma_{DF} - min(distance(X))$, illustrated in Figure 3.9(b)
- $distance(x)$ is the detected distance, with the distance sensor, in direction x. (purple line in Figure 3.9(b))
- $min(distance(X))$ is the shortest detected distance in any direction (blue line in Figure 3.9(b))
- Γ_{DF} is the far distance threshold (yellow line in Figure 3.9(b))

The situation weight of this behavior is 0.5, if the robot can sense another object in medium proximity ($distance < \Gamma_{DF}$), which is not a light source. This weight is further increased to 1 if the detected object is in very close proximity ($distance < \Gamma_{DC}$).

Follow Leader
The follow leader behavior is only active when the robot has connected to the back of another robot. It will vote to follow the light of the robot in front of it, which is usually green. The one exception is when the robot in front sends out an orange light. This signal means that the robot in front is trying to drive backwards. The robot might be driving backwards because it has come to a dead end or it senses a hole in the ground. Therefore, it is important that the follow leader behavior replicates that action. A full positive vote is given for driving backwards, and negative for driving forward, including forward turns. This behavior has normal influence, giving a situation weight of 0.5, when the robot has connected to another robot, and 0 weight in other situations. Formula 3.4 is also used to find the vote from this behavior, in the direction related to sensor x, when orange light is not detected. This behavior will only look at the three front sensors.

Hole Avoidance
The hole avoidance behavior should vote to avoid falling into holes. It gives a full negative vote in every direction a hole is detected. If either the front left sensor or the front right sensor detects a hole, the forward action will be given a full negative vote as well, as illustrated in Figure 3.10. If the front sensor detects a hole, all of the three forward directions will be given full negative votes. The robot has a hole detection sensor which can detect a hole in two distances, where the closest distance is 5 cm, and the longest is 10 cm. This behavior will not vote differently for the two distances, which means that it will vote a full negative vote for a hole detection in any of the two distances. The situation weight, on the other hand, will be 1 if a hole is detected within the closest distance, 0.5 if a hole is detected within the longest distance, and 0 otherwise.

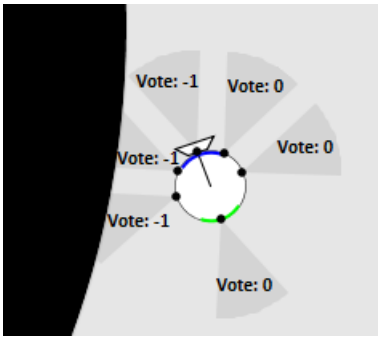


Figure 3.10: The hole avoidance behavior
In a situation where the robot is close to a hole. The sensor shadow represents the hole detection sensors (10 cm range). The values of the votes are before weight is considered.

Stagnation Recovery
The stagnation recovery behavior will start acting if the robot has not moved in 3 seconds. It will then try to move the robot away, with a full positive vote in any direction without objects, and a partial vote in directions with objects, depending on the distance to them. It continues this voting procedure until the robot is moving, and then for 3 additional seconds. This behavior gets the situation weight of 0.5 if the robot has not moved in 3 seconds. This is further increased to 1.0 if it has not been able to move in the last 6 seconds. Formula 3.6 is used to find the vote from this behavior, in the direction related to sensor x:

$$vote(x) = min\left(1.0, \frac{distance(x)}{\Gamma_{DF}}\right) \tag{3.6}$$

- $vote(x)$ is the vote in direction x, from this behavior
- $distance(x)$ is the detected distance, with the distance sensors, in direction x
- Γ_{DF} is the far distance threshold

Assemble Recovery
The assemble recovery behavior will start acting if the robot is connected to other robots and has not been able to move in 3 seconds. It is then assumed that the whole group is stuck, and this behavior will try to disconnect the robot from the group, by voting for the release grip action. It will keep doing this until the robot releases the grip or the robots manage to move again. Similar to the stagnation recovery behavior, the situation weight is 0.5, if the robot has not moved in 3 seconds. The weight is further increased to 1.0 if it has not been able to move in the last 6 seconds. The situation weight of this behavior is always 0.0 if the robot is not connected to another robot.

3.5 The Choice of Creating a Simulator

The use of simulation is a common approach when developing robot controllers. The simulator is responsible for performing the fitness evaluation for the evolutionary algorithm. In this section, we present the requirements we have for the simulator, and evaluate these requirements for two popular simulators used in the field of swarm robotics, before explaining the choice of creating our own simulator.

The following requirements are what we need of a simulator:

R1: The robots
The robots need wheels to move around, lights and light sensors to see the environment and other robots, and a gripper to be able to physically assemble with the other robots.

R2: Environment and physics
To experiment with our scenarios, the robots will need to move around in a 2D environment. Using 3D is also feasible, but the performance loss can be too significant, resulting in a preference for 2D environments. The environment and physics engine should be able to handle both normal ground properties and ice, obstacles, wind and holes. The physics engine should handle friction, collisions and physical connections between robots.

R3: Easy collaboration with our other components
The evolutionary algorithm needs to be able to start a simulation with parameters such as the phenotype. The result should then be returned to the evolutionary algorithm automatically. This requires easy collaboration between the simulator and the other components.

R4: Well documented
To be able to set up the simulator, make the necessary changes in the system, and create the communication channels to our other components, the simulator should have a thorough and easy accessible documentation.

R5: License to use
The simulator should be either free to use or licensed for us to use.

Discussion of the options
Based on the requirements for the simulator, we considered using either ARGoS, Webots or writing our own 2D simulator. ARGoS and Webots are two popular simulators within swarm robotics. They are described in Section 2.3.

The first option was to use ARGoS. ARGoS is open-source and free, which fit very well to requirement R5. We knew that it would support our type of robots, as it supports the foot-bots from the Swarmanoid project and the s-bots from the Swarm-bots project. However, we found this simulator to be hard to use, as it did not fit well with requirement R4, which also made it hard to test requirement R2 and R3.

Webots is better on requirement R4. It has a substantial documentation and supports multiple programming languages and all main operating systems. We have seen examples of gripping robots with this simulator, and have experience using it in multiple earlier swarm robotics projects. However, their license program limits a few features to the professional edition, including the fast simulation mode. This mode is not available in the education edition, which is the edition we can access. We need fast simulations for the evolutionary algorithm, meaning we can not use Webots without the fast simulation mode.

The third option was to write our own 2D simulator. An advantage is that we can make sure that it fits well with all our requirements. The trade-off is that it can be very time consuming to make. While trying to create a prototype, we found it hard to implement the collision detection and the resulting reaction. We found an open-source plug-in JBox2D [Murphy, 2014], for collision handling and general physics, which should make this process less time consuming. By making our own simulator, it is also easier to distribute parts of it to multiple machines or cores/threads, to make it more efficient. In a preliminary experiment, we managed to solve a problem around five times faster, using eight threads compared to only using one thread (4 cores and 8 threaded CPU). This should help to make the evolution faster.

Based on our requirements, the problems we had with the ARGoS simulator, and the limitations for the use of all the Webots features, it was clear that none of the considered existing simulators were a good match for our project. Searching for more possible simulators was possible, but the two we considered as the best candidates did not match, and many simulators are not using aspects we need, such as friction, so we decided to create our own 2D simulator. We could then make an optimal simulator for our use, with only the most essential features.

3.6 Modeling Physics

We have used a physics engine called JBox2D [Murphy, 2014], to model the basic physics of our simulator. It models the world and can handle collision detection, friction and movement forces. JBox2D uses formulas for classical mechanics, including Formula 3.7-3.9, which we have verified with a formula booklet [Utdanningsdirektoratet, 2005].

The velocity formula for constant acceleration a (constant in a single step), where v is the new velocity, v_0 is the previous velocity, and t is the duration of a step:

$$v = v_0 + at \tag{3.7}$$

The movement formula for constant velocity v (constant in a single step), where s is the new position, s_0 is the previous position, and t is the duration of a step:

$$s = s_0 + vt \tag{3.8}$$

Newton's 2. law of motion, for the relation between acceleration a and the forces F:

$$\sum F = ma \tag{3.9}$$

The physics engine did not directly support the use of wind, ice, light, and a top down view, with friction on the ground (it is built for a side view). We use this section to describe how these features are added to the physics engine.

3.6.1 Movement from Motors and Wheels

The movements of the robot are simulated by modeling a motor connected to the wheels. By doing this, we find the rotational speed of both the motor and the wheels. This is needed to calculate acceleration and max speed for the robot. The following formulas, Formula 3.10, 3.11 and 3.12, describe how the maximum motor force is calculated [V-Neun and Neun, 2005].

In Formula 3.10, the torque from the motor, τ_{motor}, is calculated:

$$\tau_{motor} = \tau_{max}\left(1 - \frac{\omega_{motor}}{\omega_{free}}\right) = mg\mu\left(1 - \frac{\omega_{motor}}{\omega_{free}}\right) \tag{3.10}$$

- $\tau_{max} = mg\mu$ is the maximum torque of the motor [Nice, 2001].
- m is the mass of the robot: 0.2 kg
- g is the gravity constant: 9.81 m/s^2
- μ is the friction coefficient, depending on the surface and if the robot is sliding
- ω_{free} is the rotational free speed (no load speed) of the motor: 4829 RPM (calculated backwards to get a max speed of 0.25 m/s)
- ω_{motor} is the rotational speed of the motor, based on the previous step.

The gearbox torque, the torque on the wheels, is calculated in Formula 3.11:

$$\tau_{gearbox} = \eta\varrho\tau_{motor} \tag{3.11}$$

- η is the efficiency of the motor: 100 %.
- ϱ is the gear ratio of the motor: $\frac{1}{50}$
- τ_{motor} is the motor torque, defined in Formula 3.10

The maximum forward force on the wheels, F_{wheels}, is calculated in Formula 3.12:

$$F_{wheels} = F_a - R_{roll} = \frac{\tau_{gearbox}}{r} - R_{roll} \tag{3.12}$$

- F_a is the acceleration force

- R_{roll} is the friction resistance when rolling, defined in Formula 3.13
- r is the wheel radius: 2.5 cm

The motor gets an input related to how much the motor should be used. Applying the value one gives maximum rotation speed forward, and minus one for maximum reverse rotation speed. Different values for the two wheels leads to the robot turning. For example, a left input of minus one and a right input of one lead to a counter-clockwise turn on the spot. Input values between minus one and one can also be applied, and be used for a soft turn. The controller is responsible for setting these inputs.

3.6.2 Friction

The simulator operates with different kinds of friction: static friction, kinetic friction and rolling friction. It also operates with both normal ground and ice. We used friction coefficients from a formula collection [Utdanningsdirektoratet, 2005], with minor adjustments and additions. The friction coefficients are listed in Table 3.4:

Table 3.4: Friction coefficients

Friction/Ground	Normal	Ice
Static	0.5	0.04
Kinetic	0.4	0.03
Rolling	0.001	0.001

Formula 3.13 is the formula for friction:

$$R = mg\mu \tag{3.13}$$

- m is the mass of the robot: 0.2 kg
- g is the gravity constant: 9.81 m/s^2
- μ is the friction coefficient

3.6.3 Wind Force

We use formulas from fluid dynamics to calculate dynamic pressure, and use this pressure to find the force inflicted on an object standing in the wind. Formula 3.14 is used to find the dynamic pressure in a fluid [The Engineering Toolbox, 2014b]:

$$p_d = \frac{1}{2}\rho v^2 \tag{3.14}$$

- p_d is dynamic pressure in pascals $[N/m^2]$
- ρ is the density of the fluid $[kg/m^3]$
- v is the velocity of the fluid $[m/s]$

To calculate the drag force on an object from the dynamic pressure, both the projected surface area of the object and the drag coefficient should be considered. The robot has a cylinder shape, so parts of the wind should be pushed to the side without giving much impact force. This is accounted for in the drag coefficient c_d of the object. For small cylinders, like the robot, the drag coefficient $c_d = 0.8$. Formula 3.15 is found by multiplying these factors. It is used to find the drag force of the fluid [The Engineering Toolbox, 2014a]:

$$F_d = p_d c_d A \tag{3.15}$$

- F_d is the drag force $[N]$
- c_d is the drag coefficient for the object in the fluid
- A is the projected surface area of the object (the part that the fluid can hit) $[m^2]$

Formula 3.16 is found from combining Formula 3.14 and 3.15:

$$F_d = \frac{1}{2}\rho v^2 c_d A \tag{3.16}$$

Inserting variables related to wind and the robot leads to Formula 3.17:

$$F_w^{max} = \frac{1}{2}\rho_{air} v_w^2 c_d A_{robot} \tag{3.17}$$

- F_w^{max} is the drag force from the wind, on a robot fully exposed to the wind $[N]$
- ρ_{air} is the density of air in 0 °C : 1.29 kg/m^3 [The Engineering Toolbox, 2012]
- v_w is the velocity of the wind [m/s]
- c_d is the drag coefficient of the robot: 0.8
- A_{robot} is the projected surface area of the robot: 28 cm^2

The robots are not necessarily fully exposed to the wind. If one robot is behind another, the robot in front should absorb and block a part of the wind. Calculating the percent of wind, γ, in an area, could be very performance heavy, and it needs to be updated every frame of every simulation. To use this with an evolutionary algorithm and keep an acceptable runtime, we had to make a simplified model.

The model we made divides the wind area into small cells of 1 cm^2, each representing the percent of wind γ_c, in the given subarea (cell). Each cell gets its value based on its

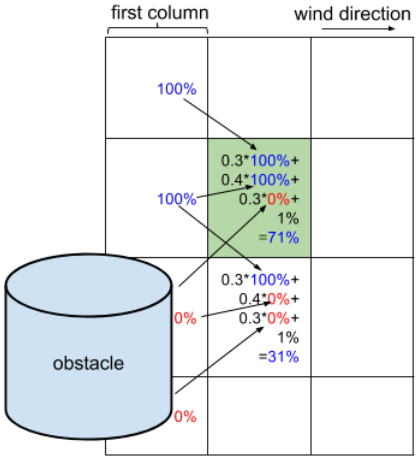


Figure 3.11: The model for calculating approximate blocking of the wind Update rules example on a small area. The percentage in each cell is γ_c , the percent of wind in the area. Cells are updated based on neighbor cells.

neighbors. The update rule starts by setting all cells to 100 % in the row or column where the wind comes from. In the example in Figure 3.11 the wind direction is east, so the first column from the left is set to 100 %. All the cells covered by an obstacle are then set to 0 %, as the obstacles block the wind in those cells. All the other cells are then updated sequentially, starting from the row or column where the wind comes from. They are updated based on three neighbor cells, depending on the direction of the wind. Because the wind is going towards east in the example, the cells should be updated based on the neighbor cells from the west side (including north west and south west), using Formula 3.18. The green cell in Figure 3.11 is an example of an ongoing update. It takes 40 % of its value from the left (west) cell and 30 % from each cell diagonally to the left. Also a small constant ψ is added, which we found to be good at 1 % for the current cell size. This represents the wind coming over the object. The model assumes that the wind is in a direction of either north, east, south or west. The results from performing these calculations on a larger area with multiple objects are demonstrated in Appendix B.

The update rule depends on the wind direction. The coordinates of the neighbor cells need to be changed according to this direction. Formula 3.18 works when the wind is going from west to east:

$$\gamma_{c(x,y)} = 0.3\gamma_{c(x-1,y-1)} + 0.4\gamma_{c(x-1,y)} + 0.3\gamma_{c(x-1,y+1)} + \psi \tag{3.18}$$

- γ_c is the percent of wind in cell c (an area of the world)
- x and y is coordinates in the cell array
- ψ is the constant related to wind coming over the object: 1 %

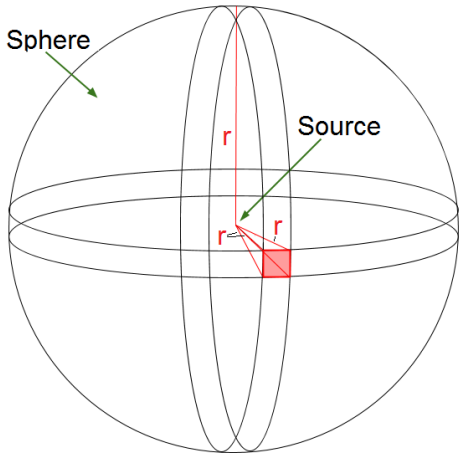


Figure 3.12: Inverse square law illustrated
Partially based on text and figures from HyperPhysics [HyperPhysics, 2012a]. The surface of the red square is fixed at 1 m^2. As the radius r increases, the surface of the red square becomes relatively smaller, compared to the surface of the whole sphere.

Formula 3.19 is used to reduce the drag force depending on the exposure:

$$F_w = \gamma_c F_w^{max} \tag{3.19}$$

- F_w is the drag force from the wind, depending on the wind exposure $[N]$
- γ_c is the percent of wind in cell c (an area of the world)
- F_w^{max} is the drag force from the wind $[N]$, with full wind exposure (Formula 3.17)

3.6.4 Light

The use of light was not supported directly in the physics engine. We found that the best approach for us was to calculate light directly between each sensor and light source. This is a compromise between the accuracy and the performance of the simulator, and was needed to get a usable runtime. This does however mean that we do not get any reflections in our simulator, all the light gets absorbed when it hits a surface. Before using the formulas below, to calculate the light a sensor sees from each light source, we check that the light source is visible from the sensor.

Sensed light can be measured by the light intensity per square meter, called illuminance [HyperPhysics, 2012a]. We use the inverse-square law to reduce the measured intensity, as the distance r from the light source increases. The inverse-square law comes from looking at the radiation in all directions as a sphere, and is illustrated in Figure 3.12. When the distance r increases, the energy (going in all directions) is spread over a larger surface. The energy that hits each square meter of the surface is then decreased.

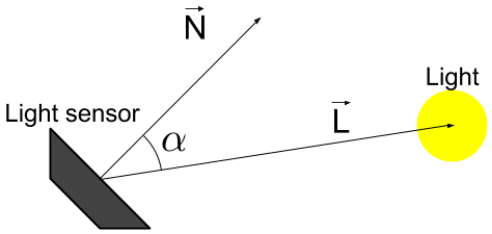


Figure 3.13: The light angle of impact, on the sensor
The angle α between the light sensor's normal $\vec{N}$ and the vector to the light source $\vec{L}$.

Formula 3.20 defines the illuminance on the light sensors:

$$E = \frac{P_0}{A_{Sphere}} = \frac{P_0}{4\pi r^2} \tag{3.20}$$

- E is the illuminance, the light intensity per square meter $[W/m^2]$
- P_0 is the power from the light source [W]
- A_{Sphere} is the surface area of the sphere, with radius r, from Figure 3.12 $[m^2]$

The light has a maximal effect on the sensor only when the sensor points directly towards the light source, meaning the angle α, between the sensor and the light source, is $0°$. This angle is illustrated in Figure 3.13. As α increases, the effect of the light decreases. If α becomes larger than $90°$, the sensor is not able to detect any light from the light source. This reduction of the effect is accounted for using Lambert's cosine law [Lambert, 1760]. It states that the intensity of the light that lands on a surface is proportional to the cosine of the angle α between the illuminating source and the normal of the surface.

Formula 3.21 Lambert's cosine law [Lambert, 1760]:

$$\text{light direction factor} = cos(\alpha) \tag{3.21}$$

- α is the angle between the sensor surface normal and the direction of the light source

The radiant flux Φ_E is the total power of the light that lands on the surface of the sensor [HyperPhysics, 2012b]. It is used further as the value the robot can read from the sensor, and is measured in watts. It is found by multiplying the illuminance with the sensor surface area and the light direction factor, resulting in Formula 3.22:

$$\Phi_E = EA_{sensor}\ cos(\alpha) \tag{3.22}$$

- Φ_E is the radiant flux, the power of the light landing on the sensor surface $[W]$
- E is the illuminance, defined in Formula 3.20

- A_{Sensor} is the surface area of the sensor: 1.0 cm^2
- $cos(\alpha)$ is the light direction factor (Lambert's cosine law)

3.7 The Evolutionary Algorithm

We have used an evolutionary algorithm to evolve the controller of the robots. In the following sections we will describe why we used an evolutionary algorithm, the genetic encoding and translation, how we calculated the fitness value for an individual, the selection mechanisms, and the reproduction process. The main evolutionary process together with a figure showing the generation cycles, are presented in Section 2.3.1.

3.7.1 Reasons to use Evolutionary Algorithms

We wanted to use an evolutionary algorithm to train the robots, to increase the learning capabilities of our system. From our background research, we found that many of the related projects had success with using an evolutionary algorithm in swarm robotics. This included training robot controllers, evolving artificial neural networks for the robot behaviors or controllers, and evolving communication methods.

Using an evolutionary algorithm gives advantages for us. Evolutionary algorithms are good optimization tools on large search spaces that do not require a needle in haystack solutions, such as our system. Multiple solutions can give good results in our system, but the search space is very large. We are evolving 9 behavior weights and 4 thresholds, each of them having 5 bits (n = 5), resulting in $2^{(9+4)5} = 3.7 * 10^{19}$ unique combinations. The algorithm should still be able to find good solutions, without using too much time.

Having an automatic search procedure also makes it easier to adapt the controller to new scenarios. These scenarios might require new controller solutions, which can be found by rerunning the evolutionary algorithm with the new scenarios. This can be done without requiring further human changes of the design or parameters of the system. The evolved controllers might also end up using solutions humans think of as strange and would possibly not consider at all. The evolutionary algorithm, however, does not give any explanations to the solution, which can make them harder to analyze. Even though this is a drawback, we have not found any alternatives that we could use, which also would give more explanations to its solutions.

3.7.2 Genetic Encoding and Translation

The genetic encoding is illustrated in Figure 3.14. The genotype consists of binary values and is m genes long, where each gene consists of n bits. To translate the binary genotype

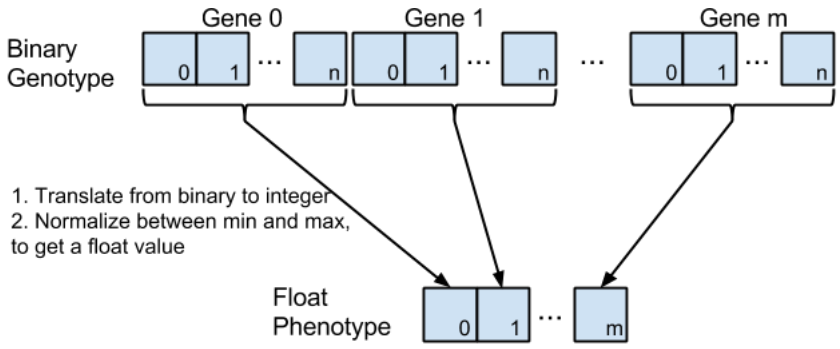


Figure 3.14: The genetic encoding in the evolutionary algorithm

into the float phenotype, each gene is translated into an integer and then normalized between a minimum and a maximum value, using Formula 3.23:

$$float(b) = min + \frac{Integer(b)}{maxInteger(n)}(max - min) \tag{3.23}$$

- $Integer(b)$ is the integer value of the binary number b, e.g. $Integer(1101) = 13$
- $maxInteger(n)$ is the biggest integer with n bits, e.g. $maxInteger(5) = 31$
- min and max is the minimal and maximal number in the float range

The different genes represent the behavior weights and the thresholds used by the controller. For a behavior weight, the minimum and maximum values are 0.0 and 1.0 respectively. For all the thresholds, the minimum value is 0.0002, while the maximum value for the distance threshold is 0.3, and the maximum value for the light threshold is 0.5. We use five bits to represent each gene ($n = 5$), which means that each float in the phenotype can be evolved to $2^5 = 32$ different values. We also tried with a higher number of bits, but it did not give an observable impact on the results.

3.7.3 Fitness

The fitness evaluation returns a number representing the performance of the individual on a specific problem. In our case, the robots get a score between 0.0 and 1.0 on their performance on getting to the goal, where 1.0 is the best possible value. The simulator performs the fitness evaluation. When the simulator is called to evaluate a phenotype, the float numbers from the phenotype are mapped into the controller. The controller is then evaluated in a predefined scenario, where the robots can use up to 90 seconds to reach the goal area.

To get a good fitness value, the robots need to reach the goal area fast. The fitness function is visualized in Figure 3.15. We use the method of giving partial credit before the robot reaches the goal, based on the distance between the robot and the goal, in the

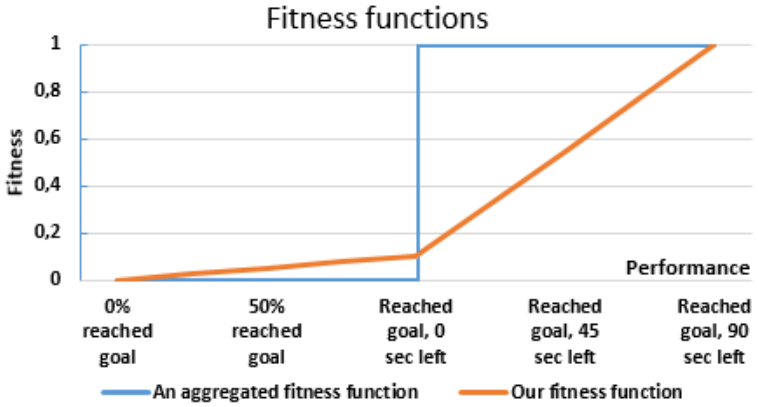


Figure 3.15: Visualizing the fitness function of the evolutionary algorithm
The fitness function in our system (brown), uses partial credit for time and distance in the fitness score. The aggregated fitness function (blue) gives either 1 or 0 and nothing in between.

end of the simulation. If the robot reaches the goal, the elapsed time will also be used as a part of the fitness value. The use of an aggregated fitness function was discussed in Section 2.5.3 and is also presented in Figure 3.15. The aggregated fitness functions gives 1.0 if the task is finished and 0.0 otherwise. Our fitness function gives at least two benefits compared to the aggregated fitness function. The first benefit is that it helps the evolution when comparing solutions that were not able to reach the goal. A solution that nearly reached the goal is better than one that did not move at all. By slightly modifying a solution that nearly reached the goal, the new solution might be able to reach it. The second benefit is that it makes room for improvements also after reaching the goal. By reducing the time needed to reach the goal, the robots become better at the task.

The method of subtracting the optimal time from the total time used on the task, is used in several of the related projects. We found it hard to calculate the optimal time in our experiments, given the different unpredictable effects of the environmental features, such as ice and wind. We decided to only use the total time, and to not subtract the optimal time. This makes it impossible to get a perfect fitness score of 1.0, since the robots will always use more than zero seconds to get to the goal area, but the results from different scenarios are more comparable. This decision should not affect our selection process negatively, because only the individual with the highest fitness value or a random individual is selected, and the relative fitness value among the individuals does not matter.

Each of the four robots is rated independently in the simulator, and the average score from the robots is used for the overall fitness value. As mentioned, the fitness is based on both the distance to the goal and the elapsed time.

The score based on distance is calculated using Formula 3.24:

$$S_D^r = 1 - \frac{d(t)}{d(0)} \tag{3.24}$$

- S_D^r is the score of robot r depending on the distance from the goal
- $d(t)$ is the distance from the goal area at time t, d(0) = the initial distance

The score based on the time used to reach the goal, is calculated using Formula 3.25:

$$S_T^r = \begin{cases} 1 - \frac{g(r)}{t_{max}}, & \text{if goal is reached} \\ 0, & \text{otherwise} \end{cases} \tag{3.25}$$

- S_T^r is the score of robot r depending on the time used to reach the goal area
- $g(r)$ is the time robot r has used to reach the goal area
- t_{max} is the maximal simulation time: 90 s

The fitness scores are then combined and averaged using Formula 3.26. We found that using 0.9 for the time weight and 0.1 for distance weight had two positive effects. First, it made the time part of the fitness value more important, but at the same time it gave the robots room for improvements in the beginning, even before they are able to reach the goal. Second, when using a $t_{max} = 90$ seconds, a fitness reduction of 0.01 means that the average time increased with one second. This makes it easy to see the differences in time between two fitness values.

Formula 3.26 combines the distance and time scores for each robot, to an overall fitness value:

$$Fitness = \frac{1}{|R|} \sum_{r=1}^{R} (w_t S_T^r + w_d S_D^r) \tag{3.26}$$

- R is the set of all robots
- w_t is the weight on the time part of the score: 0.9
- w_d is the weight on the distance part of the score: 0.1

Using the average of multiple samples to reduce variance

As we discussed in the background chapter, the variance when simulating a controller could be significant. It is a common solution technique to run a simulation multiple times and either use the average, the minimal or the median result. Soysal et al. found it best to keep the simulation steps as small as possible to be able to increase the number of evaluations of each controller, and use the minimum when combining the results [Soysal et al., 2007] .

However, after experimenting with using average, minimal and median to combine the results, we found that using average was more reliable than both minimal and median.

The minimal could have a large drop in one sample of a good controller, for instance if three robots assembled together and a forth robot wandered alone. Median was more reliable, but the results could jump a little here as well. Average gave the most consistent results, so we decided to use average going forward. We found that running the simulation three to five times before averaging the results, gave a good balance between lowering the variance and keeping an acceptable runtime. The average of three simulations was only used when finding statistics from hundreds of runs of the evolutionary algorithm, as this was very time consuming, otherwise the average of five simulations was used.

Increasing the performance
It is a slow process to perform the fitness evaluation for each individual. Each fitness evaluation is fortunately independent of the others, making them suitable to do in parallel. By starting new threads for each simulation, the system could handle up to eight simulations at the same time, each of them with four robots. This made the system around five times faster, on a CPU with four cores and eight threads.

3.7.4 Selection

The selection process first decides which individuals should survive, and thereafter which of them should become parents. This can be seen as the artificial method of natural selection in the population. Natural selection is a term Charles Darwin used to describe the process of real evolution, where traits become less common if they have a negative effect on the chance of an individual's reproduction, and more common if they have a positive effect [Darwin, 1859].

In the evolutionary algorithm we have tested different selection mechanisms, based on the ones described by Floreano and Mattiussi [Floreano and Mattiussi, 2008]. These include rank selection, tournament selection, full generational replacement and generational mixing. The selection process is divided into two steps. The first step is the adult selection process, were the individuals that survive to adulthood is selected. After the adults are selected, we have a parent selection process. Here, a number of adults are selected to become parents. A single adult can be selected multiple times as a parent, which can result in multiple offsprings from the same individual.

We use full generational replacement as the adult selection mechanism. In this mechanism, all the individuals in the child population will become adults and replace the old adult population [Floreano and Mattiussi, 2008]. We have chosen to use this mechanism because we want to keep all individuals, to maintain high diversity in the population. We also want to make sure that the individuals that are competing against each other in one generation are also tested in the same generation. This is to make sure an individual with a high fitness value does not get influence over several generations, without being evaluated each time, in case it was very lucky in the first evaluation.

We use tournament selection as the parent selection mechanism. In this mechanism,

the individuals from the adult population are divided into tournament groups [Floreano and Mattiussi, 2008]. We use tournament groups of size 5. Each group will have a winner chosen by the highest fitness value. However, there is a probability of e, to choose a random winner. New tournaments are performed, with different groups each time, until the number of winners are equal to the population size. This means that an individual can win more than one time. Since there are new random groups each time, the individuals should get to compete against different individuals in multiple different groups. All of the winners become parents. We chose to use tournament selection after testing and comparing with the parent selection mechanisms, such as fitness proportionate, rank selection, and sigma scaling [Floreano and Mattiussi, 2008]. We found that by using tournament selection, the fitness values were higher earlier in the generations, i.e., the plots for maximal fitness and average fitness were steeper. The parameters for group size and the probability variable e have been chosen to get diversity in the population. With groups of size five, and a population size of 20, there are enough groups to make sure that not only a few individuals win each tournament. At the same time, the groups are large enough to make sure that the individuals with low fitness values do not win too often. The value for the probability variable e is chosen to be 0.3, because we want a few random winners to keep a diverse population.

3.7.5 Reproduction and Elitism

The reproduction phase of the evolutionary algorithm uses the processes crossover and mutation, to create new genotypes, and elitism to copy the best genotype. They are all used to make the individuals of the new child population.

Crossover is a process with similarities to two parents making two children. Two parents make two new genotypes in the algorithm, by copying a part of the genotype from the first parent, and the rest from the second. The second genotype is made in the same way, but using the opposite parents for the first and second part of the genotype. The point where the parent is changed is called a crossover point, and is in our case selected randomly. It is also possible to use multiple crossover points, but we saw no advantage of using more than one.

Using crossover can also have a negative effect on the individuals, because all behavior weights are relative to one another. For instance, behavior a could be the most important behavior in parent one, with a behavior weight of 0.5 and an average behavior weight of 0.2, and still not be among the most important behaviors in an offspring, after crossover. For parent one, 0.5 is a high weight, because the average weight for this individual is much lower. In parent two, on the other hand, the average behavior weight could be 0.7. If we consider an offspring, where behavior a is taken from parent one and the rest is taken from parent two, the weight of behavior a will suddenly be below the new average behavior weight, even though it was the highest behavior weight for its parent. This means that behavior a will have a low influence on the offspring, even though it had a very high influence on the parent. We still found that using crossover gave good results,

but we ended up with a lower crossover rate than we have used in previous projects, because of this negative effect. We ended up with a crossover rate of 0.6, which means that 60 % of the parent population takes part in the crossover process. The last 40 % of the parent population is copied directly to the genotype pool.

The mutation mechanism in the EA is inspired by natural mutation. For evolutionary algorithms, it is often distinguished between a mutation rate for each individual and one more specific for each bit or gene. The first one is the chance that a new genotype will be selected for the mutation process, and the second one is the chance that a single bit or gene should be mutated. We keep the first rate at 100 %, meaning all individuals can be mutated, and then we have a low chance of 1.5 % for each bit to be mutated. If a bit is selected for mutation, then it is flipped, giving it the opposite value. With 65 bits ($9 * 5 + 4 * 5$) a mutation rate of 1.5 % means that one bit will be changed on average ($1/65 = 1.5\%$). Having one or fewer bit mutations on average is preferred, to not create too much change in a single generation. Too many changes could lead to very random individuals, making the evolution process harder. Mutations can happen on genotypes created from both crossover and copying of the remaining parents, but not on the elites, as they should remain unchanged. Using an adaptive mutation rate can speed up the evolutionary process. This is demonstrated by Winkler et al., who used a distance measure to increase the mutation rate in populations with low diversity [Winkler et al., 2011]. We did not implement an adaptive mutation rate, because our evolutionary process consistently found good solutions using few generations and a small population size, so these measures were not necessary to perform our experiments.

We also use elitism, which instead of creating a new genotype by crossover and mutation, directly copies the genotype of the best adults. Good solutions can easily change to bad ones by mutation or crossover in our system. For instance if the weight for the goal converging behavior is turned very low by mutation, or if two parents with high fitness values produce a child with a poor combination of behavior weights, as discussed. By using elitism, we ensure that the best solutions are not lost in the following generation. We used two elites, to make sure the genotypes from the best individuals are kept unchanged, and at the same time keep the diversity by letting most of the population evolve using crossover and mutation.

Chapter 4

Results and Discussion

In this chapter, the results of our experiments are presented and analyzed. Section 4.1, contains the resulting fitness plots from running the evolutionary algorithm on the scenarios, with a fixed wind velocity of 6 m/s. Section 4.2 contains the result of the swarm's performance, when the wind velocity is gradually increased from 0 m/s to 8 m/s between evolutions. Section 4.3 contains the weights and thresholds of the best controllers we found in three different experiments. Section 4.4 contains a thorough analysis and discussion of the presented results, and a discussion of common situations found when running the best controllers in different experiments.

4.1 Results from the Evolutionary Algorithm

The results from running the evolutionary algorithm on all scenarios are presented in this section. These runs use our standard setup, with wind velocity at 6 m/s. The fitness plots for each scenario are placed in Figure 4.1 The fitness of the average and the best (maximal) individual in each generation are plotted, together with the standard deviation of the fitness values. Each of these plots comes from averaging the results from running the evolutionary algorithm 10 times on the given scenario, to give more accurate data. The runs on each scenario use the same parameters to keep it consistent. Appendix C, contains similar fitness plots, without averaging over 10 evolutions. The parameters used for the evolutionary algorithm and the simulator are presented in Table 4.1.

The results from scenario 1, where the wind comes from the north, are presented in Figure 4.1(a). We observe a steep climb in the first generations, both for average and maximal fitness. After about 10 generations, the line for maximal fitness stops growing, around a value of 0.78. A fitness value of 0.78 means that the average robot in the simulations used 22 seconds to get to the goal area. This is a short time, considering the

Table 4.1: Parameters for the evolutionary algorithm and the simulator

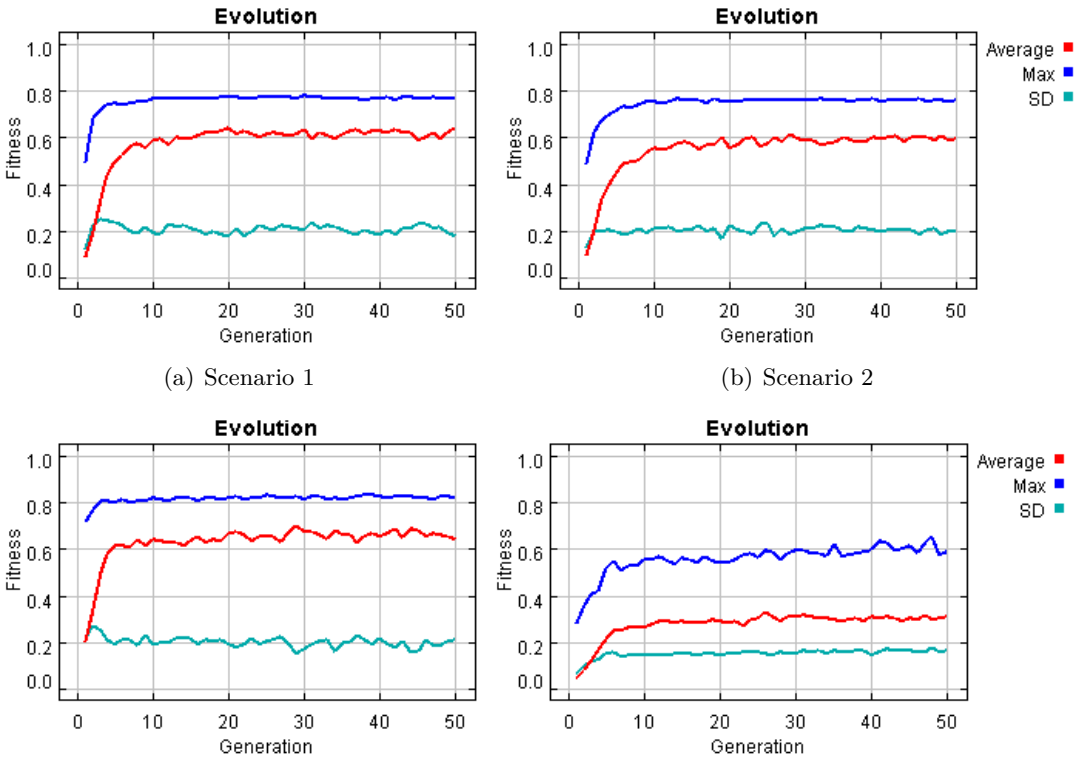
Population size	20
Generations	50
Adult selection	Full generational replacement
Parent selection	Tournament selection
Tournament group size	5
Tournament randomness factor e	0.3
Mutation rate (per bit)	1.5 %
Crossover rate	60 %
Number of bits per gene	5
Number of elites	2
Number of samples per plot point	10 evolutions
Runs per fitness evaluation	5 simulations
Number of robots	4
Wind velocity	6 m/s

wind and ice they need to overcome in this experiment. The line for standard deviation is around 0.2 throughout the evolution.

The resulting plot from scenario 2, in Figure 4.1(b), is similar to the one from scenario 1. This is observed in both the maximal and the average fitness lines, which are just marginally lower than in scenario 1. The difference between scenario 1 and 2 is the obstacles added in scenario 2. It is interesting that these had little impact on the result in these plots. The obstacles bring both advantages, by reducing the wind, and disadvantages, by forcing the robots to navigate between them, which is especially hard in bigger groups. Having both a negative and a positive effect might be the reason why the results are very similar. Another interesting observation is that the lines for maximal fitness and average fitness in scenario 2, are very steep in the first 5 generations, and then they flatten out after 10 generations. The line for standard deviation does not show any significant increase or decrease.

Figure 4.1(c) presents the results from scenario 3. We observe that both the maximal fitness and the average fitness are slightly higher than in scenario 1 and 2. The difference from scenario 2 is that the wind comes from the west and not from the north. This makes it easier to move towards the goal, which is north. The obstacles will still reduce the movement, but can also block a part of the wind. It is interesting to see that the line for maximal fitness reaches 0.8 after only 3 generations, and that the line for average fitness reaches 0.6 after 5 generations. This could mean that it is easy to find a controller that works well in this scenario.

In scenario 4, the wind comes from the west, and there is a large hole on each side, which means that the robots only have a small passage they can go through. Figure 4.1(d) presents the resulting plot from this scenario. We observe that both the maximal and



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

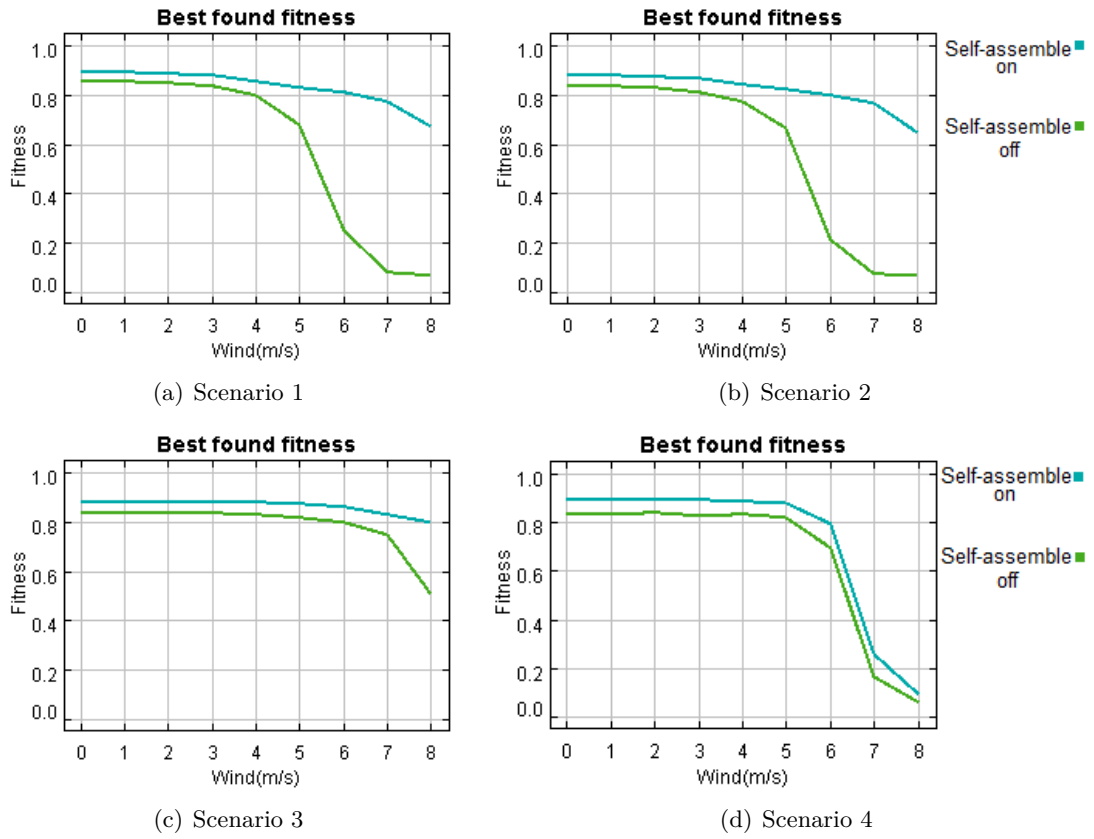
(d) Scenario 4

Figure 4.1: Fitness plots from the evolutionary algorithm
Each point in the plot are averaged from 10 runs of the evolutionary algorithm. The lines show the average, the maximal and standard deviation of the fitness values.

average fitness are lower than in the other scenarios. This indicates that this scenario is harder than the other scenarios. The lower fitness values mean that the robots on average use longer time to reach the goal, if the goal is reached at all. In this case, the best individuals got a fitness value of around 0.65, which means that the average robot reached the goal in 35 seconds. If a robot falls into a hole it will contribute zero points to the individual's fitness score, resulting in a large reduction in the fitness score for every robot that falls.

4.2 Results with Various Wind Velocities

We have used the three types of plots found in Figure 4.2-4.4, to explore the use of self-assembling in wind velocities between 0 and 8 m/s and on the different scenarios. We describe each type of plot in general, before we consider the information they give in the different scenarios.



(a) Scenario 1 (b) Scenario 2

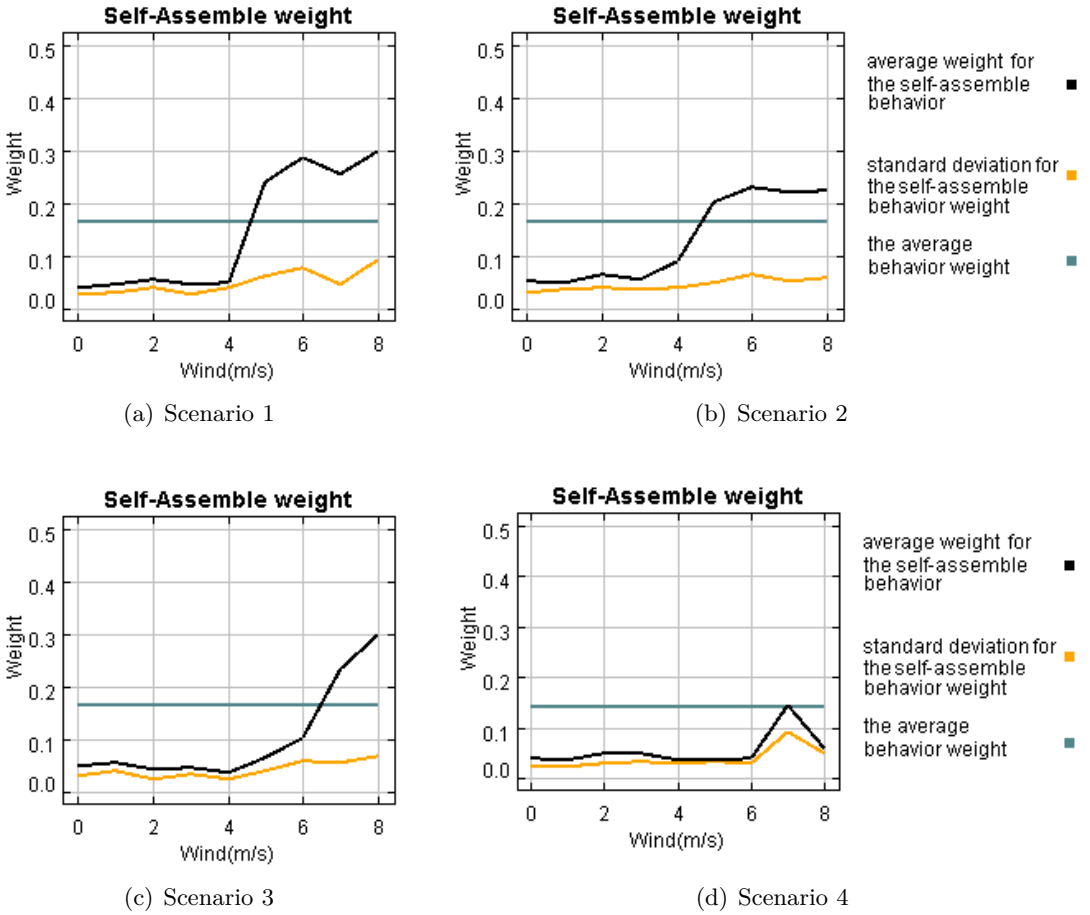
(c) Scenario 3 (d) Scenario 4

Figure 4.2: Plots of the best found fitness in different wind velocities
Both with self-assembling turned on and off. Using the average from 8 samples of evolutions for each point in the plot.

Description of the figures

In Figure 4.2, the best found fitness is plotted, depending on the wind velocity. The turquoise line on the plot resembles the normal situation in our system. The behavior weights are trained with evolution and the robot can choose to use the self-assembling behavior. The green line resembles a situation where the self-assemble behavior is turned off by setting the behavior weight to be 0.0.

The difference between the turquoise and the green line can indicate the importance of the assemble behavior. If the two lines have similar fitness values, then the use of the self-assembling behavior gave no significant improvements to the situation, indicating that it was not very important in that scenario in the given wind velocity. If the turquoise line is higher than the green line, then the use of the self-assembling behavior gave advantages in the situation. This can indicate that it was an important behavior in that scenario, with the given wind velocity. It is important to note that the turquoise line should never be below the green line. If the use of self-assembling is disadvantageous in



(a) Scenario 1

(b) Scenario 2

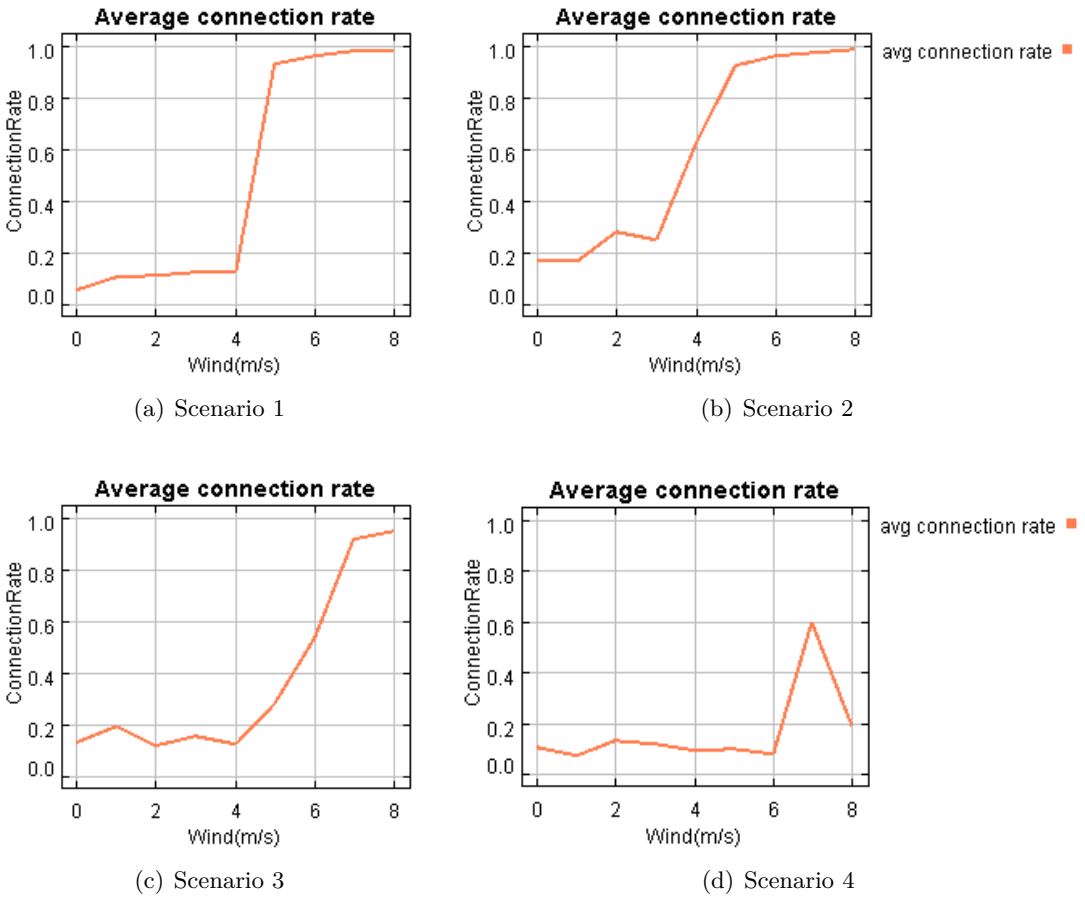
(c) Scenario 3

(d) Scenario 4

Figure 4.3: Plots of the average self-assemble behavior weight in different wind velocities Using the average from 8 samples of evolutions for each point in the plot. All weights are normalized, so their sum is 1.0. Standard deviation on the self-assemble behavior weight is listed as SD.

the scenario, the evolution will adapt and turn it off (give it zero weight). The result should then be that the turquoise and the green line have equal fitness values, as the use of self-assembling is the only difference between them.

The weights for the self-assemble behavior are plotted in Figure 4.3. It uses data from the same runs used in Figure 4.2, with self-assembling turned on. It brings additional information about the actual weight on the self-assemble behavior. The black line illustrates the average weight for the self-assemble behavior. All the weights in this plot are normalized, so that the sum of the weights in all the competing behaviors for an individual will be 1.0. The weights are normalized, because only the relative weight is important when comparing among other individuals. The average behavior weight in



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

Figure 4.4: Plots of the connection rate in different wind velocities
The connection rate is the percentage of robots that connected to another robot (self-assembled). Using the average from 8 samples of evolutions for each point in the plot.

different individuals can be vastly different. For scenario 1 to 3 there are normally six competing behaviors, resulting in an average behavior weight of $1/6 = 0.17$ after normalization. This is illustrated with the gray line in these plots. Here, the follow leader, hole avoidance and assemble recovery behaviors are not competing with the self-assembling behavior, as there are no holes in the scenario and the other behaviors cannot be used at the same time as self-assembling. Note that in scenario 4, the hole avoidance behavior is active, resulting in seven active behaviors and an average weight of $1/7 = 0.14$ after normalization. The average weight for self-assembling is the average taken from all individuals in the last generation, and the yellow line illustrates the standard deviation from the same data.

The average connection rates are plotted in Figure 4.4. It illustrates the percentage of

robots that have connected to another robot during the simulation. It takes data from all individuals in the last generation, using the same runs with self-assembling on as in Figure 4.2 and in Figure 4.3. These plots give additional information about the actual use of self-assembling. Connecting with the gripper is the method the robots have for self-assembling in our simulator. When a robot connects to another, both robots will be considered as having been connected.

Parameters
The runs on each scenario use the same parameters as the ones described in Section 4.1, with two exceptions. The number of simulations per fitness evaluation is reduced to 3 (from 5), and the number of generations is reduced to 25 (from 50), to get an acceptable runtime. Each data point in these plots comes from the average value when running the evolutionary algorithm eight times. These artificial evolutions were evaluated on each of the nine wind velocity integers between zero and eight, both with assembly on and off.

Using more samples of evolutions per data point, smaller steps for wind velocity, more generations, and a higher number of evaluations per fitness test was something we wanted, but could not do because of the runtime. We also turned off the evolving of thresholds, and set the thresholds to the example values given in Section 3.4.3. Doing these adjustments could result in a small drop in the maximal found fitness value, but this compromise was needed to reduce the runtime. Usually the maximal fitness value stopped growing before the 15th generation, which means that the reduction in the number of generations should not have a large effect on the results. Considering the need to do 144 (8 samples * 2 situations (on and off) * 9 wind integers) runs of the evolutionary algorithm for each scenario, the runtime was still numerous hours per scenario.

Scenario 1
Running the system with scenario 1 resulted in the (a) plots in Figure 4.2-4.4. In the plot for best found fitness (Figure 4.2(a)), we observe that for wind velocities under 4 m/s, the green line (self-assemble off) is nearly on the same level as the turquoise line (self-assemble on). The fitness values are above 0.85 in both cases, meaning the robots used less than 15 seconds on average to reach the goal. As expected, more wind caused the robots to use more time to reach the goal. This is observed in both lines, which decrease as the wind increases. The rate of the decrease is, however, much higher when self-assembling is turned off. When the wind goes from 5 to 6 m/s we observe a significant decrease in the fitness in the green line (self-assemble off) and only a small decrease in the fitness value of the turquoise line (self-assemble on).

We observe that the average weight of the self-assemble behavior (black line), in Figure 4.3(a), is considerably lower than the average behavior weight (gray line). We note that the standard deviation is around 0.03, which is very low, meaning that most individuals have a weight that is close to the average weight for the self-assemble behavior. We also note that the self-assemble behavior weight is still not 0.0. Similar to what we observed from the first plot, there is a significant change in this plot for wind velocities in the range

4 to 5 m/s. The average weight on the self-assemble behavior goes from 0.07 to 0.23 in this wind velocity range. From 5 m/s wind, the weight on self-assembling is higher than the average behavior weight. We observe this sudden change in Figure 4.4(a) as well. The average connection rate, the percentage of robots involved in self-assembling goes from being well below 0.2 when the wind velocity is 4 m/s or less, to becoming above 0.9 when the wind reaches 5 m/s in velocity.

Scenario 2
The results from scenario 2 are illustrated by the (b) plots in Figure 4.2-4.4. Figure 4.2(b) is very similar to the plot from scenario 1, Figure 4.2(a). In the plot for the self-assemble weight, Figure 4.3(b), we observe that the black line (average self-assemble weight) started at 0.06 and increases to 0.1 when the wind velocity is 4 m/s. This is earlier than for scenario 1, which has a nearly straight line from 0 to 4 m/s. The self-assemble weight in scenario 2 increases even more for the wind velocities 5 and 6 m/s, before it flattens out from 6 to 8 m/s, at a weight of 0.23. The increase was lower than for scenario 1, which went from weight 0.07 to 0.3, between the wind velocities 4 and 8 m/s. In Figure 4.4(b), we observe that the average connection rate has higher values from the beginning for scenario 2 compared to scenario 1. There is a small increase from 0.18 to 0.25 for the wind velocities from 0 to 3 m/s. For the wind velocities 4 and 5 m/s, there are significant increases in the average connection rate. From a value of 0.93 for wind velocity 5 m/s, the average connection rate slowly increases towards a value of 1.0 for 8 m/s.

Scenario 3
Having very similar plots for scenario 1 and 2, the differences in the results from scenario 3 are clearer. The plots from scenario 3 are presented in the (c) plots in Figure 4.2-4.4. In the best found fitness plot (Figure 4.2(c)), we observe a less steep decline in fitness, as the wind velocity increases. The turquoise line (self-assemble on) goes from a fitness value of around 0.9 to 0.8, which is a much smaller decline compared to scenario 1 and 2, where this line went from around 0.9 to around 0.7 in both scenarios. In the green line (self-assemble off), the change is even larger. This line begins with a fitness value around 0.85, when the wind velocity is 0 m/s, in all three scenarios. In the two first scenarios the green line ends with a fitness value around 0.1, when the wind velocity is 8 m/s. In scenario 3 however, this line only drops to a value of about 0.5. Another interesting observation from this figure, is that it is first at wind velocity 8 m/s, that the difference between the lines is larger than 0.2. This happens much earlier in the first two scenarios, at 5 m/s wind in both cases.

The plot of the weights for the self-assemble behavior (Figure 4.3(c)) in scenario 3 is more similar to the first two. The black line, representing the weight on the self-assembling behavior starts off very low, and so does the standard deviation. Similar to the first two scenarios, the black line starts increasing when the wind velocity becomes larger than 4 m/s. The line grows to a weight of 0.3 when the wind velocity is 8 m/s, which is the same result as in scenario 1. The increase is however much slower than in both of the first two scenarios. The point where the self-assemble behavior weight becomes larger

than the average behavior weight is between 6 and 7 m/s, compared to between 4 and 5 m/s in the first two scenarios. In the connection rate plot, in Figure 4.4(c), we observe similar characteristics. It looks like the ones from the first two scenarios, as it has a very small connection rate from the beginning, and by the end, it has grown to near 100 %. The increase is, however, much slower than in the first two scenarios.

Scenario 4
The results from scenario 4 are illustrated by the (d) plots in Figure 4.2-4.4. For scenario 4, the plot in Figure 4.2(d), is very different from the other scenarios. It is similar to scenario 3 from wind velocities of 0 to 5 m/s, but after that, both the turquoise (self-assemble on) and the green (self-assemble off) lines decrease significantly, as the wind velocity is gradually increased to 8 m/s. The turquoise line goes from a value of 0.9 to 0.1, and the green line goes from a value of 0.83 to 0.07. This is the only scenario where the turquoise (self-assemble on) line decreases almost as much as the green (self-assemble off) line. It is interesting to see that even though they both decrease so drastically, the turquoise line is still above the green line, which means that the self-assemble behavior gave at least a small advantage.

The plots for self-assemble behavior weight, in Figure 4.3(d), and average connection rate, in Figure 4.4(d), are also very different from the other scenarios. In the plot for the self-assemble weight, the black (average for assemble weight) line is nearly straight from 0 to 6 m/s, at a value around 0.05. The value of the average behavior weights are lower in this scenario, because we also use the behavior for hole avoidance, which is not used in the other scenarios. At wind velocity 7 m/s, the black line (average weight for the self-assemble behavior) increases towards the gray line (the average behavior weight) at 0.14, and then it decreases for 8 m/s, towards 0.06. In this scenario, the self-assemble weight is thereby never above the average behavior weight. The plot for average connection rate illustrates that the connection part of the self-assemble behavior is not used very often, compared to the other scenarios. The connection rate is approximately a straight line from 0 to 6 m/s, at a value around 0.1, then it increases towards 0.6 for a wind velocity of 7 m/s, and drops towards 0.2 for 8 m/s. This pattern corresponds to the plot for self-assemble weight. In the plots for the other scenarios, the lines start increasing after 3 to 4 m/s and continue to increase steeply until the last wind velocity at 8 m/s. Scenario 4 is the only scenario where the connection rate has a significant decrease.

4.3 Resulting Behavior Weights and Thresholds

In this section, we present the best individuals we found in three different situations and compare their behavior weights and thresholds. We chose scenario 1 with 0 m/s wind and 6 m/s wind, to see how the wind affects the weights and thresholds, and scenario 2 with wind at 6 m/s, to compare the effect of the different scenarios. The parameters used here are the same as used in the evolutions described in Section 4.1.

Table 4.2: Evolved controller variables

Symbol	S1(0 m/s)	S1(6 m/s)	S2(6 m/s)	Description
w_1^b	$^{(a)}$**0.26**	$^{(a)}$**0.16**	$^{(a)}$**0.25**	The goal converging b. weight
w_2^b	0.08	0.15	0.07	The wander b. weight
w_3^b	0.27	0.01	0.11	The maintain turn b. weight
w_4^b	$^{(b)}$**0.02**	0.23	0.16	The self-assemble b. weight
w_5^b	0.16	0.11	0.01	The follow leader b. weight
w_6^b	$^{(c)}$**0.10**	$^{(c)}$**0.17**	$^{(c)}$**0.14**	The collision avoidance b. weight
w_7^b	(No holes)	(No holes)	(No holes)	The hole avoidance b. weight
w_8^b	0.0	0.04	0.14	The stagnation recovery b. weight
w_9^b	0.10	0.14	0.12	The assemble recovery b. weight
Γ_{DC}	0.17	0.09	0.18	Distance threshold, close distances
Γ_{DF}	$^{(d)}$**0.27**	$^{(d)}$**0.30**	$^{(d)}$**0.30**	Distance threshold, far distances
Γ_{LC}	0.06	0.05	0.02	Light threshold, close
Γ_{LF}	$^{(e)}$**0.002**	$^{(e)}$**0.002**	$^{(e)}$**0.002**	Light threshold, far away

Evolved controller variables for the robots running on Scenario 1 (S1) and Scenario 2 (S2), for wind velocity 0 m/s and 6 m/s

The result is presented in Table 4.2. The weights are here normalized to make them comparable, as only the relative weight between the behaviors affects the influence of each behavior. When comparing the weight, we look at trends in the data and not the exact weights. This is because multiple solutions can give good results, and there are additional variance sources because of the dependencies between the weights. For instance if the self-assemble behavior weight is 0 or nearly 0, then the follow leader and the assemble recovery behavior weight will not matter much for the individual. The weights of these behaviors will then not have a significant effect on the fitness value, making them difficult to train. We observed trends in the weights in Table 4.2, and have verified that these trends happen over multiple runs, and not only from the ones listed in this table.

The first trend is that the goal converging behavior weights, marked with (a) in the table, are higher than the average behavior weight ($1/8 = 0.125$). This behavior is, as the name implies, responsible for making the robots finish their main task, moving to the goal, so it is not surprising that it is evolved to have an above average weight in all three situations. The second trend is that the self-assemble behavior weights, marked with (b) in the table, are much higher in the situations where the wind velocity is 6 m/s compared to 0 m/s. This is similar to what we observed in the results presented in Section 4.2.

A third trend is that the collision avoidance behavior weights, marked with (c) in the table, are high. They are either near average or above the average behavior weight in all three situations. There is no fitness punishment or possibility of destroying components

when the robots collide in our simulator, but there are still reasons to have a high collision avoidance. A collision can take time to navigate away from and the friction will cause the robots to move slower if they are bumping into a wall. In addition, driving into a wall means that the robot is not driving in the right direction.

We observe that the distance threshold for far distances are always near the maximal value of 0.3 (the distance sensor is able to detect obstacles 0.3 meters away). These are marked with (d) in the table. For the collision avoidance behavior, a high threshold means it will start avoiding an obstacle as soon as it is detected. Note that the situation weight of the collision avoidance behavior becomes 0.5 if an obstacle is detected closer than this threshold. The situation weight becomes 1.0 if the distance is shorter than the distance threshold for close distances. This threshold had much lower values than the values for the far distance threshold, but the variety was also larger.

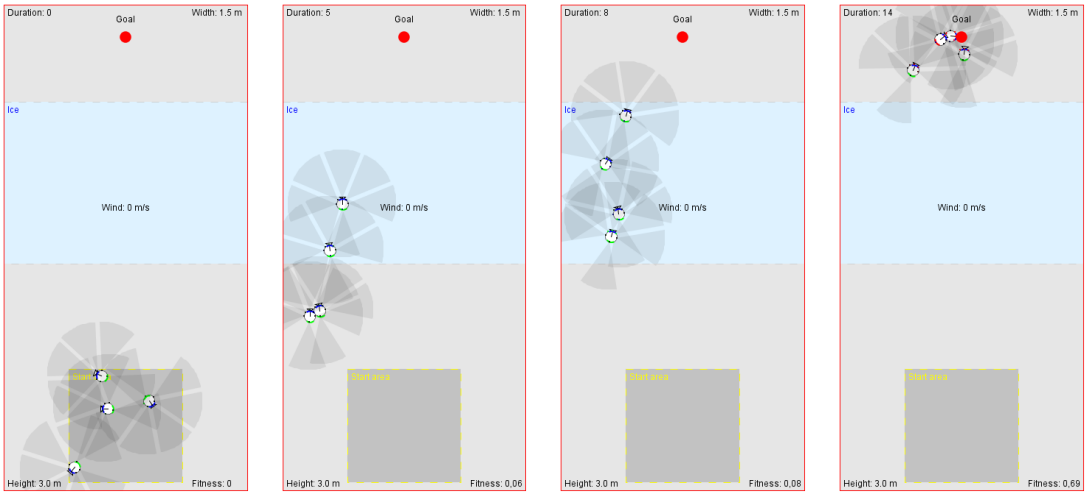
The light threshold for recognizing that a light sensor has sensed a light far away, is also evolved to an extreme value, the minimal value in its range (0.002 W), marked with (e) in the table. Compared to a higher threshold, a lower threshold leads to the robots being able to start converging at longer distances. With a threshold this low, the goal converging behavior will try to converge towards the goal as soon as it senses light that is close to red light and above 0.002 W. This is reasonable because it is advantageous to converge early, and there are no disadvantages to do so in the simulator. However, a real sensor would possibly have more noise in the data, resulting in a higher threshold to make sure it is not just noise.

4.4 Discussion

We use examples from running simulations and the results from the previous plots, to analyze the use of self-assembling in these experiments. We begin by discussing the use of self-assembling when the wind is low (0 to 4 m/s), and then consider the differences when the wind velocity is increased, before analyzing additional situations, including situations where self-assembling gave few advantages.

4.4.1 Self-assembling in Low Wind Velocities

Self-assembling gave small improvements when the wind velocity was low. This is indicated by the small difference in the fitness values plotted in Figure 4.2 for wind in the range 0 to 4 m/s, in all scenarios. With these low wind velocities, the robots could easily navigate the environments, and are able to reach the goal fast, even when they travel alone. A common run of the simulation is illustrated in the screen shots in Figure 4.5. The opposing wind scenario is used, but the wind velocity is set to 0 m/s. Without the wind we observe that it is easy for the robots to move from the start area to the goal.



(a) $t = 0$ (b) $t = 5$ (c) $t = 8$ (d) $t = 14$

Figure 4.5: Screenshots: connection not used in an experiment without wind
Taken on scenario 1, with opposing wind (0 m/s) and the ability to self-assemble turned on. Screenshots taken on time t, the duration from the start of the simulation.

In this run they all reached the goal in less than 15 seconds, without connecting to each other.

From Figure 4.3, we observed a clear trend when the wind is in the range of 0 to 4 m/s. The average weight on the self-assemble behavior, the black line, is around 0.05 to 0.1 for all scenarios. This is lower than the average behavior weight after normalization, which is 0.17 in scenario 1-3 and 0.14 in scenario 4. The low weights for the self-assembling behavior indicate that this behavior is not much used in this situation.

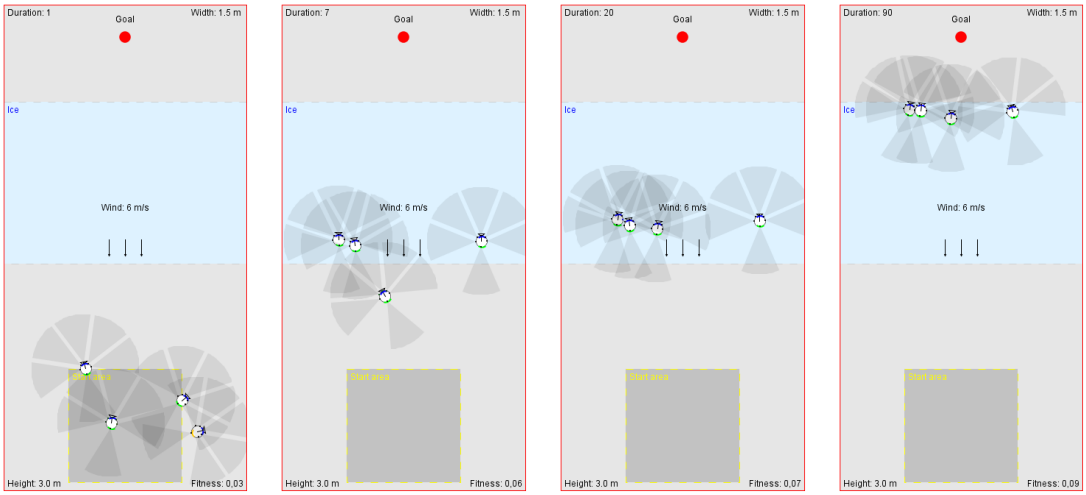
This is confirmed by the low connection rates in Figure 4.4, when observing the same wind range. We observed that the standard deviation line, in Figure 4.3, has a low value of around 0.04 in all scenarios, for low wind velocities. This means that most individuals have a self-assemble weight that is close to the average self-assemble weight for the entire population. This can indicate a high selection pressure on this behavior weight, and that having a self-assembling weight close to this value is important to get the best results. The low weight on the self-assembling behavior in the given wind range is observed in each scenario, and each of them were found using multiple samples. The consistency indicate that this result is not random. Given that the weight for the self-assembling behavior is both important and consistent, and at the same time is well below the average behavior weight, we believe that it must be a reason to why evolution has nearly turned it off in this wind range.

We have found three disadvantages for using the self-assembling behavior, which can be the reason for the low behavior weight when the wind is low. The first is that

it is harder for the robots to navigate in a self-assembled structure. With their limited communication methods, they are not able to vote among them to decide which direction to move in. They are only trying to follow the movement from the robot in front of them. In scenario 4, we also observed that the entire group of assembled robots could fall in the holes if one of them fell in, and in scenario 1 and 2 it was harder for a larger structure to navigate between the obstacles. We will look closer at these situations later. The second disadvantage is that it takes time to self-assemble. We observed that the best robots are on average able to reach the goal in 10-15 seconds in scenario 1, when there is no wind. The 2 seconds used on the connection process becomes a significant amount of time in this situation, especially if it happens with multiple robots, connecting in sequence. The third disadvantage is that trying to self-assemble can lead the robot in other directions, than what other behaviors want. Votes from the self-assembling behavior might result in not performing the action that had the most votes before. For example, if there is a goal on the left side and another robot on the right side of the robot, depending on the distances and the behavior weights, the robot might choose to self-assemble with the other robot, instead of going towards the goal.

With these disadvantages in mind one could easily believe that the best individuals would turn the self-assembling behavior off. However, as we observed in Figure 4.3, this is not the case. The average behavior weight for self-assembling is lower than the average behavior weight, but not completely off. In Figure 4.2 it is also clear that using self-assembling gave small advantages also in the wind range of 0 to 4 m/s, as the turquoise line is always above the green line in each scenario. We have found two reasons for why this low self-assembling weight is still advantageous. First, it is so low that it will not often interfere with other important tasks. Second, by going towards other robots there are better chances of going in a good direction, since the other robot is heading that way. The other robot is probably not driving straight into a wall, and might even have sensed the goal and is on the way towards it. This may also explain why the robots are grouped together on the left side in Figure 4.5, which occurred in many of the runs we observed.

Note that going towards other robots is listed as both an advantage and a disadvantage. If the self-assembling weight is very high, its votes might be so large that the votes from other behaviors does not matter much. It can then happen that it would do the opposite of other behaviors. This could be a disadvantage, like in the example of not going towards the goal, if it sees a robot in another direction. However, when the weight on self-assembling is very low, the votes from the other behaviors matter more. Now it is probably only able to influence the decision between actions that the other behaviors also want. This results in a possible advantage, because these decisions can be slightly better, as there are more cooperation.



(a) $t = 1$ (b) $t = 7$ (c) $t = 20$ (d) $t = 90$

Figure 4.6: Screenshots: robots using a long time to overcome the wind and ice alone Taken on scenario 1, with opposing wind (6 m/s) and the ability to self-assemble turned off. Screenshots taken on time t, the duration from the start of the simulation.

4.4.2 Self-Assembling in Higher Wind Velocities

As expected, the environments become more difficult for the robots to traverse when the wind has a higher velocity. This is indicated by all plots in Figure 4.2. In all scenarios the fitness value is higher when the wind velocity is 0 m/s compared to 8 m/s, both when the ability to use self-assembling is turned on, and off. In scenario 1 and 2 the plots are very similar. When the wind goes from 4 m/s to 8 m/s and the self-assembling behavior is turned off, the fitness result in both plots (Figure 4.2(a) and 4.2(b)) falls from a value of around 0.8 to a value below 0.1. Note that this fitness value is below the value given to individuals for reaching the goal (with an average robot), which is 0.1. This means that the average robot is not even able to reach the goal in the given 90 seconds.

A common run of the simulation with a strong wind and the ability to self-assemble turned off, is illustrated in the screenshots in Figure 4.6. Scenario 1, the opposing wind scenario, is also used here, but this time the wind velocity is set to 6 m/s. The strong wind makes the environment harder to traverse for the robots. The main challenge brought by the wind is that it counteracts the forward force from the robot motors, making their acceleration and velocity lower. This can cause problems with the robots moving forward, especially when the robots are driving on ice, as their forward force is already much lower because of the reduced grip. From the observations when running the simulator and the screenshots depicted here, we can confirm that it is the traversing over the ice area that takes the longest time for the robots. Three of the robots reached the ice area after 7 seconds in Figure 4.6, but none of them had passed the ice area when

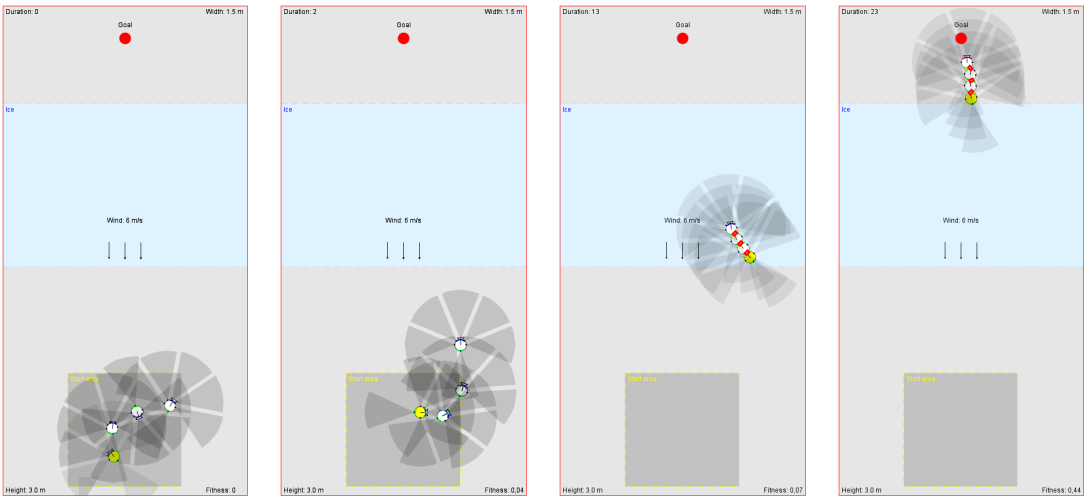
the simulation stopped after 90 seconds. This is a significant difference from the run in Figure 4.5, where the robots only spent about 10 seconds on crossing the ice area.

When the robots start to have problems overcoming the wind and ice alone, we observe a significant increase in the average weight for the self-assemble behavior. This increase is clear in scenario 1 to 3 in Figure 4.3. For instance in scenario 1, the weight for the self-assemble behavior is increased from 0.06 to 0.25, when the wind goes from 4 m/s to 5 m/s. The weight goes from being way below the average behavior at 4 m/s, which is 0.17, to become above average at 5 m/s. When the assemble weight increases, the actual connection rate, observed in 4.4(a), increases as well, as expected. The connection rate changes from being below 0.2 to becoming above 0.9 in the same wind range. This displays that the robots actually complete the connection process over 90 % of the time when the wind is strong. The results from both scenario 1 and 2 are similar.

In scenario 3, we also observed that the self-assembling behavior weight starts below average, at low wind velocities. Similar to scenario 1 and 2 the self-assembling behavior weight gets higher when the wind gets stronger. In this scenario however, the first time this weight is larger than the average behavior weight, is when the wind velocity is between 6-7 m/s. This is later than in scenario 1 and 2, where it already happened with wind between 4-5 m/s. In scenario 3, the wind comes from the side, which resulted in lesser significance of the impact from the wind. From Figure 4.2(c) we observed that the first significant impact from the wind on the fitness values happened at a higher wind velocity in this scenario, compared to the other scenarios.

The increased use of self-assembling in the first three scenarios gave significant improvements for the robots. When the wind goes from 4 m/s to 8 m/s and the self-assembling behavior is turned on, then the fitness result in both plots (Figure 4.2(a) and 4.2(b)) falls from a value of around 0.85 to a value around 0.65-0.7. The decrease for the turquoise (self-assemble on) lines are smaller than for the green (self-assemble off) lines in these scenarios, which fell from a value of around 0.8 to a value below 0.1. The fitness score of 0.65 means that the average robot used 35 seconds to reach the goal. This is better than when self-assembling was turned off, where not all robots reached the goal. In scenario 3, it is also clear that using self-assembling is an advantage when the wind is 8 m/s. The turquoise line decreases very slow in this scenario, while the green line has a significant fall from around 0.75 to around 0.5 when the wind velocity is between 7-8 m/s.

When self-assembling is turned on, in the same situation as in Figure 4.6, the robots perform considerably better. A common situation that appeared in these new runs is illustrated in the screenshots in Figure 4.7. The robots start to follow each other from the beginning, and are able to self-assemble into a chain of robots, before reaching the ice area. By moving in a chain formation in the opposite direction of the wind, the robots are able to traverse the ice area in less than 15 seconds. This is a very good result compared to the situation we presented in Figure 4.6, where they could not use self-assembling. In that situation, they used more than 80 seconds in the ice area, and were not past the ice at the end of the 90 given simulation seconds.



(a) $t = 0$ (b) $t = 2$ (c) $t = 13$ (d) $t = 23$

Figure 4.7: Screenshots: robots self-assemble to overcoming the wind and ice together Taken on scenario 1, with opposing wind (6 m/s) and the ability to self-assemble turned on. Screenshots taken on time t, the duration from the start of the simulation. On $t = 13$ the robots have connected to each-other with their gripper.

We have found three main reasons to why self-assembling is helpful when the wind is strong. The first reason is that a robot moving against the wind can block much of the wind from robots right behind it. This results in increased thrust from the robot behind, which increases the forward push on the robot in front. The second reason is that pushing other robots is easier if some of them are on ice and others are not. The robots that are not on ice have great grip, while the robots they are pushing or pulling have low friction against the ice beneath them. The third reason is already discussed when we analyzed why the robots still use self-assembling, but with a low weight, when the wind velocity is low. The reason is that following other robots can be advantageous, because there might be a good reason to why a robot is heading in a specific direction.

We observed in the fitness plots in Figure 4.1 that good controllers with fitness values around 0.8 can be evolved in scenario 1 to 3, this was also confirmed in Figure 4.2. In these scenarios, a fitness value close to the maximal found during the whole evolution was found in less than 10 generations. This implies that finding a good solution, in these scenarios, is not very hard for the evolutionary algorithm. Scenario 4 however, ended up with a much lower maximal and average fitness value in Figure 4.1(d). The evolutionary algorithm did not find a better solution than the ones giving a fitness value slightly above 0.6. In the plot in Figure 4.2(d), the result for wind velocity 6 m/s, was slightly higher. The reason for this might be high variance in the results. We believe that the high variance comes from the fact that this scenario is hard for all controllers, because the robots can easily fall into the holes.

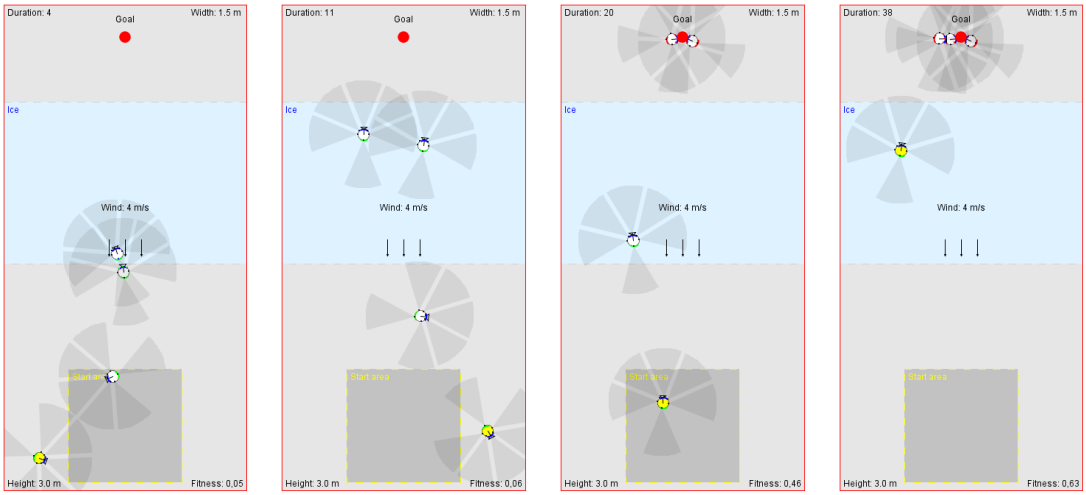
The plot for best found fitness for scenario 4 (in Figure 4.2(d)), gave very different results compared to the plots from the other scenarios. For the lower wind velocities, the best found fitness is above 0.8, like in the other scenarios. The green line, taken from runs where assembling is turned off, suddenly has a drastic decline, when the wind is in the range 5-7 m/s. Having a sudden drop in this green line is also similar to the other scenarios. The major difference however, is that the turquoise line, for runs using self-assembling, has an equally large drop as the green line. In the other scenarios, using self-assembling gave a large advantage, but with the lines being that close in this scenario, we cannot say that a very significant advantage was found.

For scenario 4, the plots that show the use of self-assembling in Figure 4.3(d) and average connection rate in Figure 4.4(d), are also very different compared to the other scenarios, which all have more similar plots. The average weight for the self-assembling behavior never becomes larger than the average behavior weight, and the average connection rate is lower compared to the other scenarios. The self-assembling behavior weight does have a spike, going up to the average line at 7 m/s, but decreases again at 8 m/s. As we will discuss when looking at specific situations from this scenario, robots moving in a self-assembled swarm can easily fall into a hole, dragging the whole group down. This could be the reason for why the self-assembling behavior weight is low in this scenario, even when the wind is 8 m/s. It is, however, important to note that the fitness score is very low at 7 m/s and 8 m/s wind in Figure 4.2(d). This could mean that none of the solutions did significantly better than the rest, resulting in a high variance in the evolved weights. This variance could also be a part of the reason for the spike at 7 m/s, in the plots for the self-assemble weight in Figure 4.3(d) and the connection rate in Figure 4.4(d).

The sudden increase in use of self-assembling

The use of self-assembling had a sudden and large increase between the wind velocities 4-6 m/s, in scenario 1, 2 and 3. To understand why the increase was so sudden, it is important to look at the difficulties the robots would get, if they had not used self-assembling. We will consider situations from scenario 1, to find the reason for this sudden increase in the use of self-assembling. We have already discussed the problems that the robots had when going over the ice area alone, with a wind velocity of 6 m/s. These problems were also observed in the screenshots in Figure 4.6. When running the simulation on the same scenario, with wind velocity decreased to 4 m/s, we observed a clear difference, which is presented in the screenshots in Figure 4.8. In this wind velocity, the robots can still easily traverse the ice area. This is observed from $t = 4$, where the two leading robots are in the beginning of the ice area, and they are nearly at the end of it when $t = 11$.

We have already implied, from analyzing the plot in Figure 4.2(a), that traversing this scenario becomes much harder without self-assembling, when the wind velocity is increased from 4 m/s to 6 m/s. These screenshots support this implication. When using self-assembling to a large degree, we observed, in the screenshots in Figure 4.7 and from analyzing the plots in Figure 4.2(a), that traversing the environment becomes easier,



(a) $t = 4$ (b) $t = 11$ (c) $t = 20$ (d) $t = 38$

Figure 4.8: Screenshots: the robots easily overcomes the ice area with 4 m/s wind Taken on scenario 1, with opposing wind (4 m/s) and the ability to self-assemble turned off. Screenshots taken on time t, the duration from the start of the simulation.

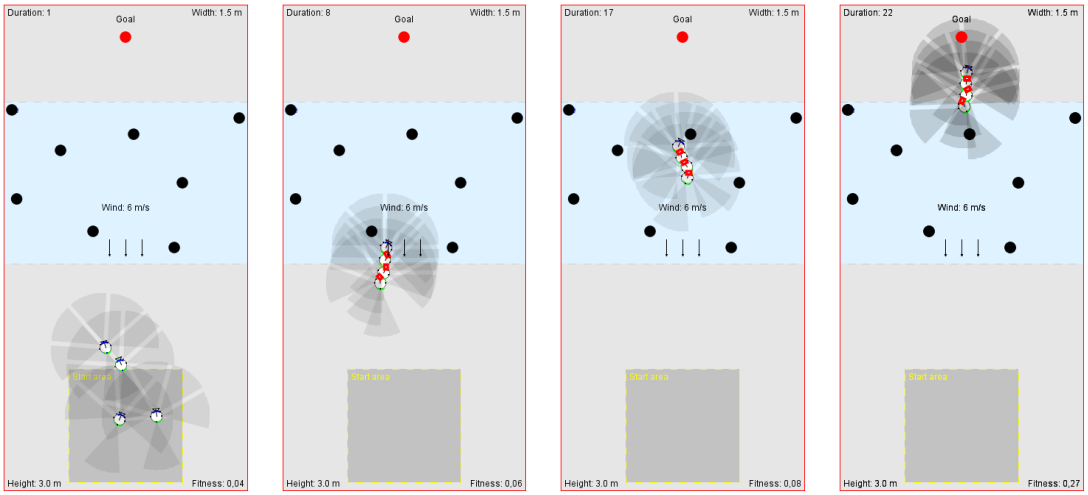
even with the wind velocity at 6 m/s.

Traversing the environment with wind at 4 m/s is easy for the robots. The disadvantages of heavy use of self-assembling overcomes the benefits. When the wind is increased to 6 m/s, the environment becomes much harder to traverse. The advantages of using self-assembling to a high degree is now much larger, while we believe the disadvantages are the same as before. The advantages now out weights the disadvantages. This is the reason for the large and sudden increased use of self-assembling, between wind velocities of 4 m/s and 6 m/s, in scenario 1.

4.4.3 Analyzing Additional Situations

We ran simulations of the best controllers found during the evolutionary runs in Section 4.1, and found interesting patterns in how the robots behaved in the different scenarios. The wind speed was set to 6 m/s in all scenarios. In this section, we describe situations from different scenarios and explain how they occurred. We also explain why self-assembling gave few advantages in scenario 4.

A situation that often occurs in scenario 2, is that the robots connect to each other in a chain formation and use the obstacles as shields from the wind, as illustrated in Figure 4.9. By moving in a chain formation, they can also shield the wind for each other, since they are moving against the wind direction, which means that they can move faster than one robot alone. In this situation, the behavior weights for self-assembling,

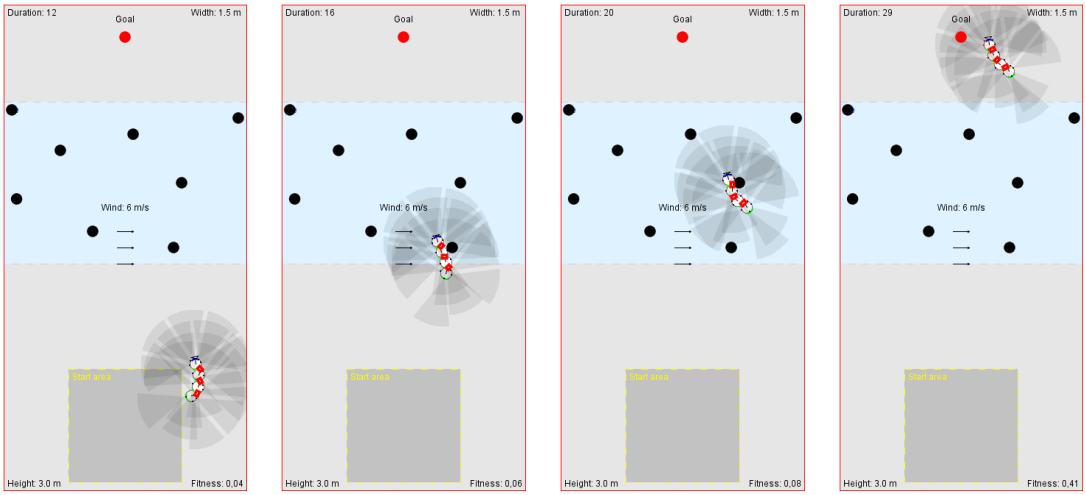


(a) $t = 1$ (b) $t = 8$ (c) $t = 17$ (d) $t = 22$

Figure 4.9: Screenshots: the robots reach the goal in a chain formation (S2) Taken on scenario 2, with opposing wind (6 m/s). Screenshots taken on time t, the duration from the start of the simulation.

follow leader, collision avoidance and stagnation recovery were high, while the assemble recovery behavior weight was low. The combination of these behavior weights made the swarm able to move easily between the obstacles towards the goal, without being stuck in any obstacles. The high weight on the collision avoidance behavior helped the swarm not to collide with the obstacles, resulting in a lower chance of them becoming stuck. The distance thresholds were especially important for obstacle scenarios. The threshold for the far distance range was evolved to 19 cm, while it was evolved to 3.9 cm for the close distance range. This made the robots able to get close enough to the obstacles to be shielded from the wind, but not become stuck to the obstacles. Self-assembling, in combination with goal converging and follow leader helped the robots to move fast towards the goal. The goal converging behavior made the first robot in the chain stay focused on the goal, while the follow leader behavior made the rest of the robots follow the movements of the robots in front of them. Having a low behavior weight for the assemble recovery behavior means that the robots will seldom disconnect from each other, when they have first connected. The controller has thereby learned through evolution that self-assembling in this scenario, with this wind velocity, is beneficial. This is also confirmed by the high connection rate for wind velocity 6 m/s in the plot in Figure 4.4(b), and by the high average behavior weight for self-assembling in the plot in Figure 4.3(b).

In scenario 3, we observed the same situation as in scenario 2, that when the self-assemble behavior weight was high, usually all the robots connected to each other in a chain formation and moved between the obstacles towards the goal. This situation

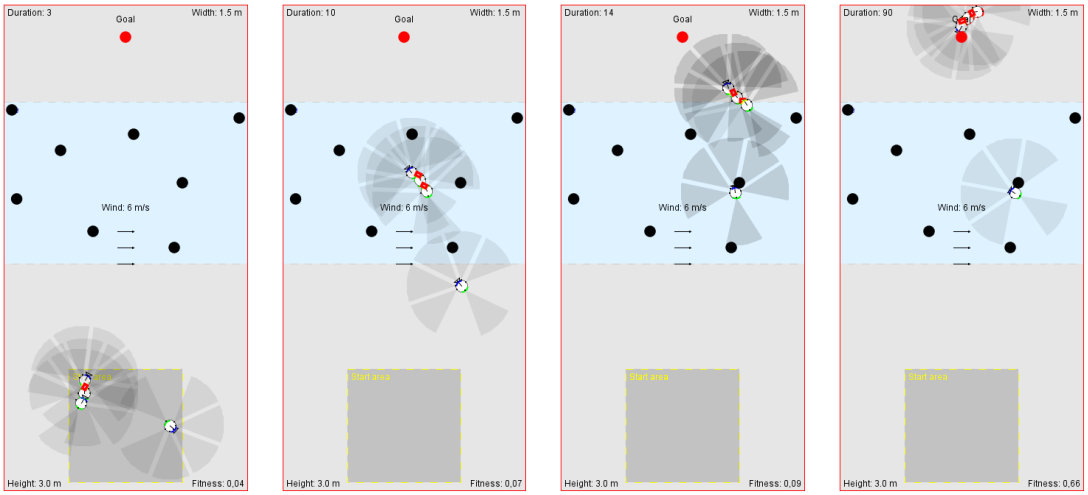


(a) $t = 12$ (b) $t = 16$ (c) $t = 20$ (d) $t = 29$

Figure 4.10: Screenshots: the robots reach the goal in a chain formation (S3) Taken on scenario 3, with wind from the side (6 m/s). Screenshots taken on time t, the duration from the start of the simulation.

is captured in Figure 4.10, and is also documented by the plots in Figure 4.3(c) and Figure 4.4(c), for wind speed from 6 to 8 m/s. The obstacles in this scenario can be a disadvantage in low wind speeds, because it is easy for long chains of robots to become tangled around them, and to be slowed down from friction against the obstacles. In cases with high wind speeds, on the other hand, the obstacles can give advantages. If a chain of robots move on the right side of the obstacle, at least one robot will be sheltered from the wind, while they pass by. The sheltered robot(s) will not be pushed as much in the wind direction as the others, and can thereby keep the others from sliding as much as they would if the obstacle had not been there. This effect will be beneficial until the last robot in the chain has passed the obstacle. A chain of robots moving on the left side of an obstacle, as in the figure, can experience a similar effect. The difference is that the obstacle will no longer shelter the robots, but can physically keep them from being pushed by the wind. Even if only a few robots are near an obstacle, it will affect the rest of the chain as well, because the robots will not be pushed as far as they would have been without the obstacle. When a robot is in contact with an obstacle, friction between them can slow down the robot. However, in the situations we observed, the swarm was always able to drag the colliding robots away from the obstacles.

We sometimes observed that one or two robots could be left behind, and not always reach the goal. This happened more often when the self-assemble behavior weight was lower than the average behavior weight. An example of this situation is captured in Figure 4.11, where a robot becomes stuck behind an obstacle, and does not manage to reach the goal before the maximal time is used. In this situation we also observed that

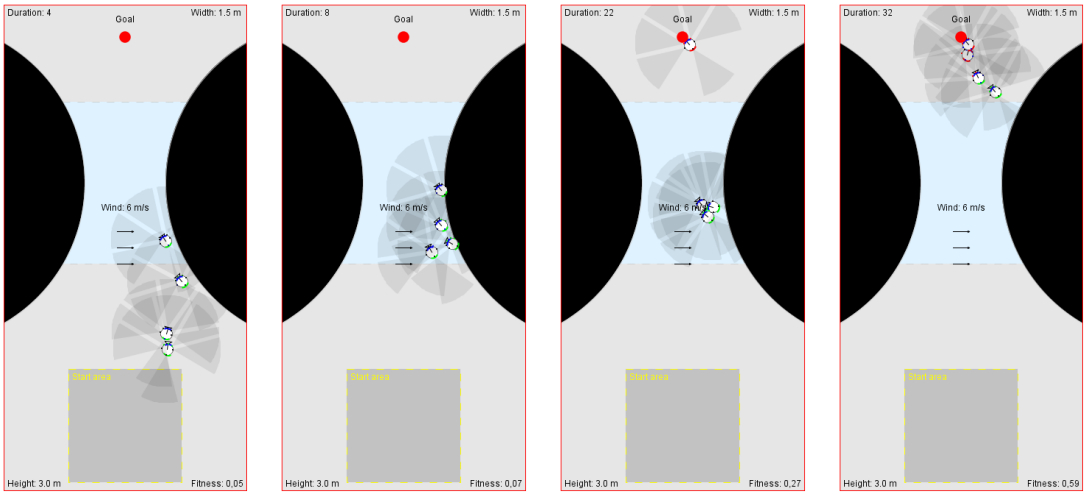


(a) $t = 3$ (b) $t = 10$ (c) $t = 14$ (d) $t = 90$

Figure 4.11: Screenshots: situation where one robot is left behind
The rest of the robots reach the goal by connecting to each other. Taken on scenario 3, with wind from the side (6 m/s). Screenshots taken on time t, the duration from the start of the simulation.

the stagnation recovery weight was lower than the average behavior weight, which made it difficult for the robot that was alone, to get away from the obstacle. In this case, the robot was turning and moving forward and back again, because of the wind. The robot was not standing completely still, which meant that the stagnation recovery behavior did not activate the recovery mode. We observed that the goal converging behavior weight was high, and that the front sensor of the robot could sense the red light from the goal. This means that the forward action got a strongly weighted vote from the goal converging behavior, in addition to the standard vote from the wander behavior. The collision avoidance behavior voted to get away from the obstacle, but did not vote negative on the forward action, because the front sensor did not sense the obstacle. This can possibly have caused most of the turning movements of the robot. The collision avoidance behavior weight was lower than the goal converging weight, in this case. The result was that the winning action shifted between being the action of the direction of the goal and the left turn action. In addition, the robot was unable to get away from the obstacle because of the friction between the robot and the obstacle and the strong wind that went in the opposite direction of the robots' forward direction. Since this situation does not occur too often, the controller has not learned to avoid it. In most cases, the robot will manage to get away from the obstacle, or get help from the other robots. This situation shows the importance of self-assembling in this scenario.

In scenario 4, we observed that the self-assemble behavior influenced the movement of the robots, even though its behavior weight was very low in all of the controllers we looked

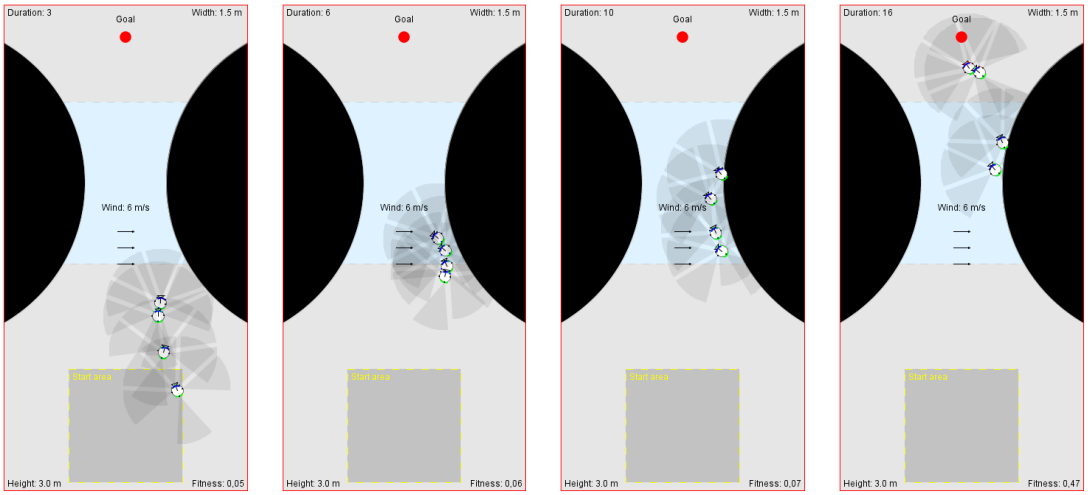


(a) $t = 4$ (b) $t = 8$ (c) $t = 22$ (d) $t = 32$

Figure 4.12: Screenshots: situation where triangular formation is used to reach the goal Taken on scenario 4, with wind from the side (6 m/s) and holes. Screenshots taken on time t, the duration from the start of the simulation.

at, including the ones in Figure 4.3(d). A low weight on the self-assemble behavior is usually not enough for activating the grip action. This is confirmed by the plot for average connection rate in Figure 4.4(d). Even small weights can, however, have an impact on the movement actions, and make the robots follow each other. This is similar to the situations where low weights on the self-assembling behavior, was used in low wind velocities to make the robots follow each other without connecting, as described in Section 4.4.1. Scenario 4 is a special case since the self-assemble behavior weight is evolved to be low in experiments with high wind velocities as well.

We also observed that the robots sometimes would form a triangular formation, without physically connecting to each other. This is captured in Figure 4.12. In this situation, the robots got the benefits of wind blocking, and also the ability to move freely without limitations of being connected to others. When they did not need the help from the others anymore, they broke out of formation and continued towards the goal. This situation can be explained by the combination of two advantages. First, the advantage of following each other. Second, by the advantage of moving faster when another robot is blocking the wind. The wind blocking can explain how they got into the cluster formation, since the robots blocking the wind would move slower than the robots in shelter of them, in the opposite direction of the wind. In very few cases, we observed the same triangular formation, in an assembled group of robots. This kind of situation seldom occurs since the robots have small connection areas, leading to a small chance that two robots manage to connect to the same area. Because the gripper is fixed on the front of the robot, the triangular formation can lead to the connecting robots not facing



(a) $t = 3$ (b) $t = 6$ (c) $t = 10$ (d) $t = 16$

Figure 4.13: Screenshots: situation where the robots move along the right hole edge The robots are able to reach the goal. Taken on scenario 4, with wind from the side (6 m/s) and holes. Screenshots taken on time t, the duration from the start of the simulation.

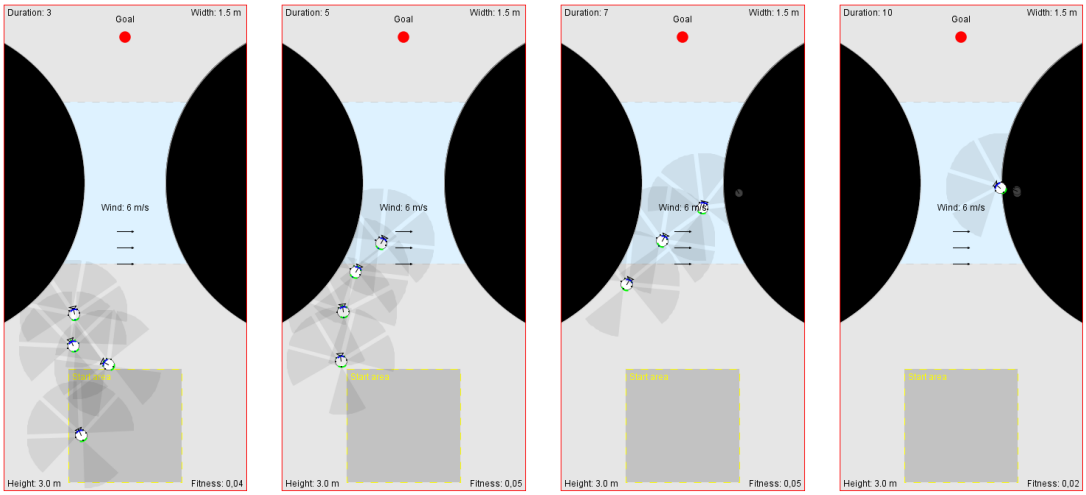
in the exact same direction, as each other or the robot in front. This slows the robots down. The connection area is currently not very large, to avoid this problem.

Self-assembling gave few advantages in scenario 4

When doing experiments on scenario 4, we observed that self-assembling gave few advantages. The weight of the self-assemble behavior in this scenario was usually very low, as discussed earlier. This indicates that a high weight on this behavior is not advantageous in this scenario. The following paragraphs contain a discussion of a situation where connecting to each other would not be an advantage for the robots, and another situation where neither connection nor following each other was an advantage.

In the first situation, we observed that the robots often reached the goal if they followed each other on the right side of the environment, and then moved slowly next to each other with their backs to the edge of the right hole. Their direction was angled between the direction of the goal and the direction opposite of the wind direction, to be able to move towards the goal. An example of this situation is captured in Figure 4.13. In this situation, it was an advantage to be able to follow each other without connecting to each other.

A second situation that often occurred, was that the robots followed each other towards the left hole. They then turned away from this hole, making them move in the direction of the wind straight towards the right hole, without knowing it. When they finally sensed the right hole, it was too late to avoid it, before falling in one by one. This situation



(a) $t = 3$ (b) $t = 5$ (c) $t = 7$ (d) $t = 10$

Figure 4.14: Screenshots: situation where the robots move along the edge of the left hole The robots then turn right to avoid the hole, and are later not able to get away from the right hole. Taken on scenario 4, with wind from the side (6 m/s) and holes. Screenshots taken on time t, the duration from the start of the simulation.

is captured in Figure 4.14. This is a situation which is hard to avoid, because there is no way for the controller to learn not to move towards the left in the beginning of the simulation. The hole avoidance behavior works hard to avoid the second hole, but does not succeed because of the short range of the hole sensors, and the large momentum the robots have achieved when moving in the same direction as the wind, while on ice. If the robots had been connected to each other, it would probably have been even harder to try to avoid the hole, since the movement of the robots would depend on other robots. The controller could have evolved an even lower weight for the assemble behavior to try to avoid that all of the robots follow if one robot turns towards the left side of the environment in the beginning. The self-assemble behavior thereby has few advantages in this situation. However, the assemble weight was not set to 0.0 in any of the cases we looked at, because it is an advantage to follow each other in other situations.

Self-assembling would not help in these two situations, since the robots need to carefully adjust their direction along the way, to be able to resist the wind and not falling into the holes. If they move in the direction of the wind, they would slide on the ice, and fall into the right hole. Moving in a self-assembled chain formation can make it difficult to obtain a good direction relative to the wind. If a robot, connected to other robots, fall into a hole, it is hard for the others to avoid falling in as well, even though they let go of the falling robot. Since the wind comes from the side, robots in a chain formation do not help blocking the wind for each other, while going straight towards the goal. Long chains are thereby not advantageous in this scenario. However, we have seen cases where only

two robots connect to each other and have reached the goal. We believe, as mentioned earlier, that the controllers have a very low weight on the self-assemble behavior to avoid the disadvantages of the long connection chains, but still have enough weight to be able to follow each other in situations where this is beneficial, and sometimes connect in small chains.

If the robots were able to align their directions after connecting, then other formations, like a triangle, could be used without the problem of slowing them down. This could possibly improve the result in scenario 4, because the robots could then block more wind from each other. This possibility is discussed further in the section about further work (Section 5.3).

Chapter 5

Conclusion

During our master thesis, we have given a brief introduction to the fields of swarm robotics and evolutionary robotics, and focused on the area of self-assembling. In this chapter, we start by giving an overview of the project. In Section 5.1 we discuss the goal and research questions. In Section 5.2 we discuss our contributions to the field of swarm robotics. In Section 5.3 we present suggestions to further work.

In Chapter 2, we gave an overview of related projects. During the research phase we found that we could contribute to the field by performing experiments with self-assembling, in new and difficult environments. We also studied the robot and simulator technologies, and the solution techniques that the researchers used. These related projects had a large influence on our choices, when designing our system.

Chapter 3 contains detailed descriptions of our choices of technology, solution techniques, simulator design, robot design, and robot controller architecture. We built a system for experimenting with self-assembling in different scenarios. The system consists of a simulator and an evolutionary algorithm, and is designed to evolve robot controllers that are capable of traversing difficult environments. The scenarios we designed, contained different combinations of ice areas, wind, holes, and obstacles, to make it difficult for the robots to navigate to the goal area. We used virtual robots, influenced by robots used in the related projects. The robots have sensors for measuring distance, and the intensity and color of light. They also have distance sensors that are tilted downwards for detecting nearby holes. The robot controller was built using a reactive architecture based on arbitration via action selection, where behavior modules vote on actions. An evolutionary algorithm evolved the behavior weights and sensor thresholds used by the controller.

In Chapter 4, we described and discussed the results from our experiments, and explained how and why different situations occurred. In three of the scenarios, we found that self-assembling could be an advantage, when traversing ice areas in high wind velocities. In the fourth scenario, on the other hand, self-assembling gave few advantages, as the sce-

nario became too hard to traverse in strong wind, even when using self-assembling.

5.1 Goal Evaluation

The goal of this project was to demonstrate advantages of emergent self-assembling, for a swarm of robots in difficult environments. By creating a simulator for robots with evolvable controllers and self-assembling abilities, we managed to demonstrate that self-assembling can improve the mobility of the swarm in environments with complicating factors, such as ice and strong wind. These experiments served as the basis for Chapter 4, and gave us insight to answer the research questions defined in Section 1.2. This section contains a thorough discussion of answers for these research questions.

Research question 1: *How can self-assembling improve the mobility of a robot swarm in difficult environments?*

Other researchers have already found situations where self-assembling can improve the mobility of a robot swarm. O'Grady et al. found, during the Swarm-bots project, that self-assembling can get a group of robots over steep slopes, which could not be traversed by a single robot [O'Grady et al., 2005]. Trianni and Dorigo found, that a swarm can self-assemble into a bridge structure, and pass over a trough [Trianni and Dorigo, 2005]

In our experiments, we found that using self-assembling was advantageous in multiple scenarios with ice and wind. When the wind velocity was low, the advantage was small and only resulted in slightly faster movement of the swarm. However, when the wind velocity was high, the advantage was more significant. In these experiments, the robots were able to traverse the whole environment by using self-assembling, but without this feature the robots got nowhere.

The first way that self-assembling improves the mobility, is that the group is more resistant against a strong wind, when being assembled. If the robots are assembled in a chain formation and are going straight towards an opposing wind, the first robot will absorb much of the force from the wind, meaning that the robots behind are more shielded from the wind. Being less affected by the wind, the robots behind are able to get a larger thrust, and can give a stronger push forward, meaning that the first robot gets pushed farther. Being assembled is thus beneficial both for the first robot and for the ones behind, in this situation.

Being able to push and pull other robots when assembled, gave improved mobility when the robots were on ice. When the robots are in a long chain formation in environments with ice, it often happens that parts of the assembled group are on an ice area while others are not. If the robots in the back of the group are on the normal ground and the rest are on ice, then the robots in the back can easily push the robots on the ice forward. The robots on the normal ground have good grip on their wheels, while the others have

low friction against the ice. A similar advantage happens if the robots in the front are on normal ground and pulls the other robots out of the ice area. These two techniques made it easier for the robots to traverse the ice area.

Being assembled was also advantageous when strong wind came from the side, and the robots traversed an ice area with small obstacles. A robot going alone could easily be pushed to the side by the strong wind. If however, a larger assembled group was pushed to the side by the wind, they could more easily be stopped by colliding with the small obstacles, resulting in the group being more able to move straight forward. The robot colliding with the obstacle would be slowed down from friction between them, but the other robots were able to pull the whole group forward, in the situations we observed.

In order to self-assemble, the robots needed to group at the same location. This resulted in a need to converge towards other robots, which made the robots faster in traversing the environments in our experiments. By going after other robots, we observed a decrease in the time used searching around, resulting in shorter time used to find the correct direction to go in.

We have now demonstrated reasons to why self-assembling can improve the mobility of a robot swarm in difficult environments. We believe there are more difficult environments where self-assembling can improve the mobility for a robot swarm, and more reasons to how this can happen. This is discussed more thorough in the section about further work.

Research question 2: *How can the robots decide when to self-assemble through local interactions and sensing the environment?*

We found that it was possible for the robots to decide when to self-assemble, with very limited communication abilities and only local control. In Chapter 3, we described our behavior based controller architecture, which uses variables evolved by artificial evolution. Using this approach, the artificial evolution defines how much influence the self-assembling behavior should have. We found that the general influence of the self-assembling behavior was often high when the wind was strong, because self-assembling could then improve the mobility of the swarm. The final decision of whether to try to self-assemble or not, was continuously evaluated in each robot controller, where each behavior module interpreted the sensor input, and then voted for which action to use. The robot could then choose the action leading to self-assembling, either if this action corresponds to the action other behaviors voted on, or if the self-assembling behavior has so much influence that its single vote is enough to choose this action.

The need to converge towards other robots gave an interesting effect in our system. In experiments without strong wind, the self-assembling behavior was generally given low influence. With the low influence, the robots seldom managed to self-assemble physically with other robots. The converging part, however, still managed to affect the movement. This could happen in cases where the sum of other votes were close among the actions

with the most votes from the other behaviors, and the self-assembling behavior could manage to influence the final decision. The robots then got the advantages of converging, without the disadvantages of physically connecting. This, and other outside the box solutions, found by exploiting the system in ways the designers do not even think of, makes it very interesting, in our opinion, to work with evolutionary robotics.

We believe more research should be done to answer this research question further. This includes more research into how the robots can decide when to self-assemble, in more dynamic environments. We will examine this when discussing further work.

5.2 Contributions

The current research in swarm robots with self-assembling abilities, presented in Section 2.5, have managed to use self-assembling to improve performance of the swarm, in several situations. These include the ability to drive up steep slopes, pass over gaps (holes), remain upright in uneven environments and pull large objects. These projects were an inspiration for our research, and we contributed to the field by exploring additional use of self-assembling. Using swarm robots with self-assembling abilities, can give advantages over other robots, including being very small, getting through small passages, and still be able to assemble into a bigger structure and pull large objects. By finding more advantages of using self-assembling, we believe the use of swarm robots can become a more viable alternative to traditional robots in the future.

We have found that self-assembling can lead to improved mobility of the swarm in simulated environments, with ice and strong wind. By using self-assembling in experiments with strong wind, the swarm could usually traverse the whole environment. The swarm was, however, not able to traverse the environment, when self-assembling was turned off. These experiments demonstrate that self-assembling improved the mobility of the swarm.

In other experiments, we found that it was possible to traverse the environment, even without self-assembling. Depending on the strength of the wind, a swarm using self-assembling could be many times faster in traversing the environment, compared to swarms without this ability. These experiments also demonstrate that self-assembling can improve the mobility.

In experiments with weak or no wind, it was not necessary for the swarm to use self-assembling, as the swarm was able to traverse the environment fast also when robots were going alone. In these cases, we observed that self-assembling was rarely used, as it would take too much time to accomplish.

The main lessons from this project:

> *Using self-assembling is not always an advantage for a swarm of robots in a specific situation, but it is in general very advantageous to have this ability. Self-assembling can greatly improve the mobility of the swarm in environments with complicating factors, such as ice and strong wind, and it can easily be turned off when it is not needed.*

5.3 Further Work

In this section we present further work, to give more answers for research question 2, improve the performance in the most difficult scenario, and find additional situations where self-assembling improves the mobility of a swarm of robots.

Increased Adaptability
We believe research question 2 should be investigated further, especially to consider how the robots can decide when to self-assemble in dynamic environments. Our experiments are performed on difficult, but static environments, meaning the robot controllers are trained to navigate in a specific environment with a specific and constant wind strength. In real life situations, the environments are more dynamic. For instance, the wind could suddenly increase. The robots should adapt to this or other changes.

The robots cannot currently adjust their behavior weights while running, to adapt to a changing environment. One possible solution could be to train the controllers to adapt to multiple environments and their possible changes. Another possible solution could be that the robots are able to evolve new controllers while running. The robots could then run a new artificial evolution, when the environment has changed. A wind sensor on the robots could be used to let the robots become aware of the change, and then evolve a new controller.

Training controllers on new and unknown environments would still be hard. To be able to find a good controller, the whole environment must be known for the simulator. The sensors on the robots can only sense small distances. They cannot be used for sensing the entire environment from a distance. One solution to a similar problem was proposed during the E-Swarm project [Mathews et al., 2012], where an aerial robot was used to get an overview of the environment, and this information was used to guide the swarm on the ground.

Increasing the adaptability in more general situations, with new and unknown environments, would be required before these types of robots can be used in dynamic and unknown real world scenarios.

Additional Formations

In our experiments, self-assembling gave advantages to mobility in all scenarios except for one. In this scenario the forward chain formation, which we typically observed in other experiments, gave no or few improvements. Moving in a chain formation often resulted in the swarm falling into a hole. They did not manage to shield each other from the wind, and if one robot fell into a hole, the rest could easily be dragged down with it. We observed that when moving in a triangle formation, the robots where able to traverse the scenario better, but were often slowed down, because their alignment in this formation was not perfect, meaning their heading directions were not the same. Their small connection areas and their fixed located gripper, would prevent them from heading in any other direction than the direction of their gripper, and would make it hard for two robots to connect to the same robot.

If the robots were able to align their directions after connecting, then other formations, including a triangle formation, could be used without the problem of slowing them down. However, this requires big changes. The robots would need a gripper that can rotate around their body, to be able to align with the other robots. They would need a bigger gripping area, like a gripping ring, so that other robots are able to connect to them in any direction. The robots would also need more advanced communication, to be able to decide which direction the group should be facing. Coordinating their movements would also require more communication, as it would no longer be enough to follow the robot in front of them, when multiple robots can be in front of the group. With the ability to form other formations, the robots could form a denser group, and shield each other better from the wind.

Additional Situations

Both in former research and in this thesis, situations where self-assembling can improve the mobility of a swarm of robots are identified. We believe that more of these situations should be identified. Self-assembling can possibly improve the mobility of small robots in deserts or other large areas with sand. A larger group can possibly be better at avoiding the problem of becoming stuck in the sand. The large group can also be more robust in situations where sandstorms can occur. For robots that move over large distances, it is also possible that self-assembling can reduce fuel usage, giving larger reach before refueling. Every situation where self-assembling can improve the mobility brings advantages to these kinds of robots, and progressively makes them a better alternative to use.

Appendix A

Example of Voting

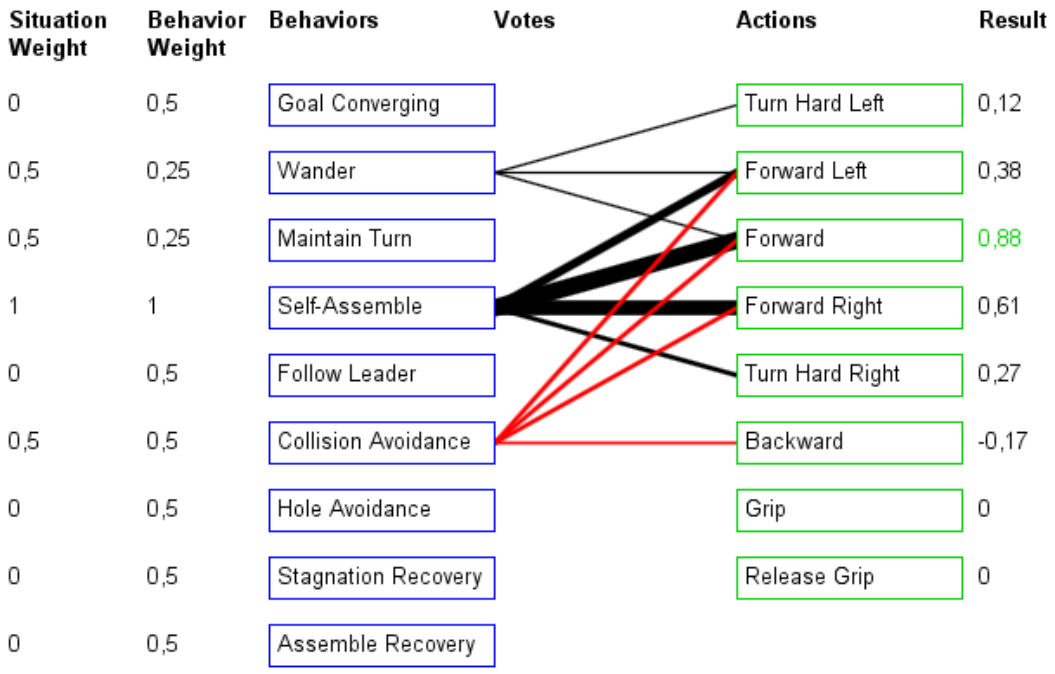


Figure A.1: A running example of weighting, behaviors, voting and actions
The lines from behaviors (blue boxes) to actions (green boxes) represent the weighted votes on the actions. Thicker lines represent larger votes and red lines represent negative votes. The forward action was selected in this example. It received votes from both the wander behavior and the highly weighted (in this case) self-assemble behavior. It received negative votes from collision avoidance, but the positive votes were larger.

Appendix B

Resulting Wind Blocking

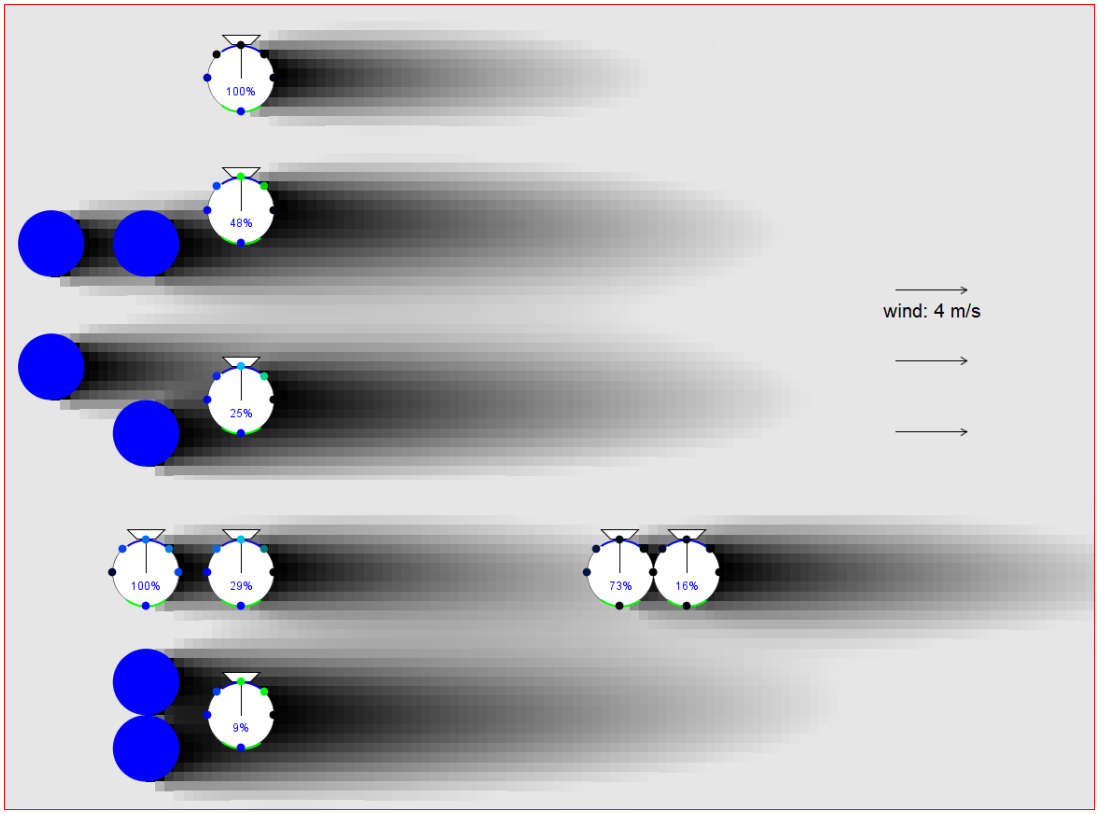
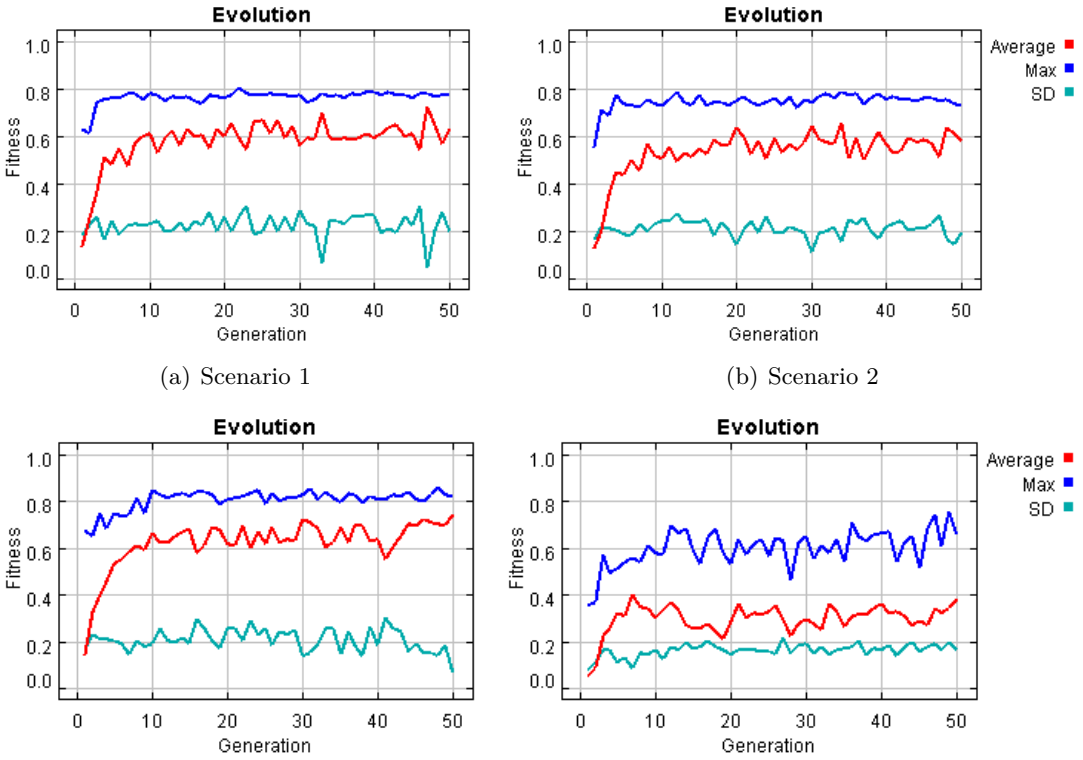


Figure B.1: Resulting wind blocking in the simulator
The environment is divided into cells of 1 cm^2, where dark cells represent areas blocked from the wind. Black cells represent areas with 0 % wind. Here we have eight robots and six blue obstacles. The wind coming from west is blocked on the east side of the objects. The number on the robots represent the wind percent where they stand.

Appendix C

Additional Fitness Plots



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

Figure C.1: Fitness plots for a single evolution
The result from a single evolutionary run (one sample), for each scenario, with the average from 5 simulations for each fitness evaluation. The lines show the average, the maximal and the standard deviation of the fitness values.

Appendix D

System Instructions

The system requires at least Java version 7. All the components we have implemented, in addition to Java 7 and the libraries we have used, should be able to run on Windows, Mac OS X and Linux. They are currently tested on Windows 7. A CPU with multiple cores and threads are recommended to reduce the runtime of the evolutionary algorithm.

To run the system, open the project in an integrated development environment for Java, such as Eclipse. The classes for running the system are located in the folder: Swarm\common\run. Use the class RunSingleEvolution to run the evolutionary algorithm. The best phenotypes are saved automatically and placed in the folder: Swarm\data\results\phenotypes. The class Replay is used to run visual simulations using the saved phenotypes. New scenarios can be added by inserting a new ".ini" file in the folder: Swarm\data\environments. Use a similar format as used in the other ".ini" files in this folder.

Bibliography

Christos Ampatzis, Elio Tuci, Vito Trianni, Anders Lyhne Christensen, and Marco Dorigo. Evolving self-assembly in autonomous homogeneous robots: experiments with two physical robots. *Artificial life*, 15(4):465–84, January 2009.

Ronald C. Arkin. *Behavior-based Robotics*. Bradford book. MIT Press, 1998.

Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999.

Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of robotics and automation*, 2(1):14–23, 1985.

Dave Cliff, Phil Husbands, and Inman Harvey. Explorations in Evolutionary Robotics. *Adaptive Behavior*, 2(1):73–110, 1993.

Cyberbotics Ltd. e-puck, 2007. URL `http://www.cyberbotics.com/e-puck/e-puck.pdf`. Visited on 2014-01-15.

Charles Darwin. *On the Origin of Species by Means of Natural Selection or the Preservation of Favored Races in the Struggle for Life*. Murray, London, 1859.

Marco Dorigo, Vito Trianni, Roderich Gross, and Thomas Halva Labella. Evolving Self-Organizing Behaviors for a Swarm-Bot. *Autonomous Robots*, 17(2/3):223–245, 2004.

Marco Dorigo, Elio Tuci, Roderich Gross, Vito Trianni, Thomas Halva Labella, Shervin Nouyan, Christos Ampatzis, Jean-Louis Deneubourg, Gianluca Baldassarre, Stefano Nolfi, Francesco Mondada, Dario Floreano, and Luca Maria Gambardella. The SWARM-BOTS Project. *Swarm Robotics*, 3342:31–44, 2005.

Marco Dorigo, Elio Tuci, Vito Trianni, Roderich Gross, Shervin Nouyan, Christos Ampatzis, Thomas Halva Labella, Rehan O'Grady, Michael Bonani, and Francesco Mondada. Swarm-bot: Design and implementation of colonies of self-assembling robots. *In Computational Intelligence: Principles and Practice*, pages 103–136, 2006.

William C. Evans, Grégory Mermoud, and Alcherio Martinoli. Comparing and Modeling Distributed Control Strategies for Miniature Self-Assembling Robots. *IEEE International Conference on Robotics and Automation*, pages 1438–1445, 2010.

Dario Floreano and Claudio Mattiussi. *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press, 2008.

Dario Floreano, Phil Husbands, and Stefano Nolfi. Evolutionary Robotics chapter 61. In *Handbook of Robotics*, pages 1423–1451. 2007.

Toshio Fukuda, Seiya Nakagawa, Yoshio Kawauchi, and Martin Buss. Structure decision method for self organising robots based on cell structures-CEBOT. *1989 International Conference on Robotics and Automation*, pages 695–700, 1989.

John J. Grefenstette, Connie Loggia Ramsey, and Alan C. Schultz. Learning sequential decision rules using simulation models and competition. *Machine Learning*, pages 355–381, 1990.

Roderich Gross, Michael Bonani, Francesco Mondada, and Marco Dorigo. Autonomous Self-assembly in a Swarm-bot. *Proc. of the 3rd Int. Symp. on Autonomous Minirobots for Research and Edutainment*, pages 314–322, 2005a.

Roderich Gross, Michael Bonani, Francesco Mondada, and Marco Dorigo. Autonomous Self-assembly in Swarm-bots. *IEEE Transactions on Robotics*, 22(6):1115–1130, 2005b.

John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Bradford Books. MIT Press, 1992.

HyperPhysics. Inverse Square Law, 2012a. URL `http://hyperphysics.phy-astr.gsu.edu/hbase/forces/isq.html`. Visited on 2014-01-20.

HyperPhysics. Radiant Flux, 2012b. URL `http://hyperphysics.phy-astr.gsu.edu/hbase/vision/radiant.html`. Visited on 2014-01-20.

Johann Heinrich Lambert. *Photometria*. W. Engelmann, 1760.

Pattie Maes. The Agent Network Architecture (ANA). *SIGART Bulletin*, 2(4):115–120, July 1991.

Nithin Mathews, Anders Lyhne Christensen, Rehan O'Grady, Philippe Retornaz, Michael Bonani, Francesco Mondada, and Marco Dorigo. Enhanced directional self-assembly based on active recruitment and guidance. *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4762–4769, September 2011.

Nithin Mathews, Alessandro Stranieri, Alexander Scheidler, and Marco Dorigo. Supervised Morphogenesis: Morphology Control of Ground-based Self-assembling Robots by Aerial Robots. *Proceedings of the 11th International Con- ference on Autonomous Agents and Multiagent Systems*, 1:97–104, 2012.

Olivier Michel. Webots TM : Professional Mobile Robot Simulation. *International Journal of Advanced Robotic Systems*, 1(1):39–42, 2004.

Nathan J. Mlot, Craig A. Tovey, and David L. Hu. Fire ants self-assemble into waterproof rafts to survive floods. 2011.

Francesco Mondada, Giovanni C. Pettinaro, Andre Guignard, Ivo W. Kwee, Dario Floreano, Jean-Louis Deneubourg, Stefano Nolfi, Luca Maria Gambardella, and Marco Dorigo. Swarm-Bot: A New Distributed Robotic Concept. *Autonomous Robots*, 17 (2/3):193–221, September 2004.

Daniel Murphy. JBox2D, a java physics engine, 2014. URL `http://www.jbox2d.org/`. Visited on 2014-01-06.

Andrew L. Nelson, Gregory J. Barlow, and Lefteris Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57(4): 345–370, April 2009.

Karim Nice. How Four-Wheel Drive Works, April 2001. URL `http://auto.howstuffworks.com/four-wheel-drive.htm`. Visited on 2014-01-21.

Shervin Nouyan and Marco Dorigo. Chain Formation in a Swarm of Robots. March 2004.

Shervin Nouyan and Marco Dorigo. Chain Based Path Formation in Swarms of Robots. *Ant Colony Optimization and Swarm Intelligence*, 4150:120–131, 2005.

Rehan O'Grady, Roderich Gross, Francesco Mondada, Michael Bonani, and Marco Dorigo. Self-assembly on Demand in a Group of Physical Autonomous Mobile Robots Navigating Rough Terrain. *Advances in Artificial Life*, 3630:272–281, 2005.

Rehan O'Grady, Anders Lyhne Christensen, and Marco Dorigo. SWARMORPH: Multirobot Morphogenesis Using Directional Self-Assembly. *IEEE Transactions on Robotics*, 25(3):738–743, 2009.

Kazuhiro Ohkura, Toshiyuki Yasuda, and Yukihiko Kotani. A Swarm Robotics Approach to Cooperative Package-Pushing Problems with Evolving Recurrent Neural Networks. *SICE Annual Conference 2010*, pages 706–711, 2010.

Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, November 2012.

Julio Kenneth Rosenblatt. DAMN: a distributed architecture for mobile navigation. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2/3):339–360, 1997.

Julio Kenneth Rosenblatt and David W. Payton. A fine-grained alternative to the subsumption architecture for mobile robot control. *International Joint Conference on Neural Networks*, 2:317–323, 1989.

Masahiro Shimizu and Akio Ishiguro. An amoeboid modular robot that exhibits real-time adaptive reconfiguration. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1496–1501, October 2009.

Onur Soysal, Erkin Bahceci, and Erol Sahin. Aggregation in Swarm Robotic Systems: Evolution and probabilistic control. *Turkish Journal of Electrical Engineering and Computer Sciences*, 15(2):199–225, 2007.

Swarmanoid. The Swarmanoid Project, 2010. URL `http://www.swarmanoid.org/`. Visited on 2013-08-22.

Symbrion. The Symbrion Project, 2013. URL `http://www.symbrion.eu/`. Visited on 2013-09-20.

The Engineering Toolbox. The Engineering Toolbox: Air Density, 2012. URL `http://www.engineeringtoolbox.com/air-density-specific-weight-d_600.html`. Visited on 2014-01-06.

The Engineering Toolbox. The Engineering Toolbox: Drag Coefficient, 2014a. URL `http://www.engineeringtoolbox.com/drag-coefficient-d_627.html`. Visited on 2014-02-06.

The Engineering Toolbox. The Engineering Toolbox: Dynamic Pressure, 2014b. URL `http://www.engineeringtoolbox.com/dynamic-pressure-d_1037.html`. Visited on 2014-02-06.

Vito Trianni and Marco Dorigo. Emergent Collective Decisions in a Swarm of Robots. *2005 IEEE Swarm Intelligence Symposium*, 2005:249–256, June 2005.

Vito Trianni and Marco Dorigo. Self-organisation and communication in groups of simulated and physical robots. *Biological cybernetics*, 95(3):213–31, September 2006.

Elio Tuci, Roderich Gross, Vito Trianni, Francesco Mondada, Michael Bonani, and Marco Dorigo. Cooperation through self-assembling in multi-robot systems. *ACM Transactions on Autonomous and Adaptive Systems*, 1(2):115–150, November 2005.

Utdanningsdirektoratet. *Tabeller og formler i fysikk*. Gyldendal, 2005.

John V-Neun and John Neun. Advanced Drivetrain Calculations (Presentation slides), 2005. URL `http://thinktank.wpi.edu/cgi-bin/index.cgi?n=Article:ArticleView&rid=44`. Visited on 2014-01-23.

Lutz Winkler, Heinz Worn, and Adrian Friebel. A distance and diversity measure for improving the evolutionary process of modular robot organisms. *2011 IEEE International Conference on Robotics and Biomimetics*, pages 2102–2107, December 2011.