UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

EXPERIMENTS IN EVOLUTIONARY COLLECTIVE ROBOTICS

André González Amor de Bastos

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2011

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

EXPERIMENTS IN EVOLUTIONARY COLLECTIVE ROBOTICS

André González Amor de Bastos

DISSERTAÇÃO

Projecto orientado pelo Prof. Doutor Paulo Jorge Vaz Cunha Dias Urbano
e co-orientado pelo Prof. Doutor Anders Lyhne Christensen

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

2011

Agradecimentos

Ao longo do meu percurso académico, inúmeras foram as pessoas que me ajudaram, inspiraram ou simplesmente me disseram a coisa certa, na altura certa. De entre essas pessoas, quero destacar a minha mãe, o meu pai, a minha madrinha e os meus avós, por todo o tempo e esforço que investiram em mim. Quero fazer uma referência especial à minha irmã, Ana Rita, por me fazer lembrar que não vale a pena crescer e a Kátia Paramés, Joana Paramés e Rosa Machado, que embora não façam parte da minha família, as considero como tal.

Esta tese é também dedicada a todos os amigos que fiz na faculdade, por terem estado ao meu lado nos bons momentos, mas sobretudo nos menos bons. Gostaria de agradecer em particular a João Costa, João Lopes, Louis Philippe, Bruno Seixas, Frederico Miranda, Geraldo Nascimento, Diogo Serrano, Christian Marques, Marco Lourenço, Raúl Simplício, Davide Nunes, Bruno Correia, Joaquim Tiago Reis, Carlos Alvares, Andreia Machado, Filipe Gil, João Ferreira, Alexandre Gabriel, Vera Conceição e a todos aqueles que, directa ou indirectamente, estiveram relacionados com o LabMag.

Gostaria de agradecer ao professor Paulo Urbano, ao professor Anders Christensen e ao professor Sancho Oliveira por me terem ajudado, de forma incansável e exemplar, a ultrapassar todos os obstáculos que enfrentei durante a escrita desta tese. Foi realmente um privilégio ser orientado por professores tão excepcionais.

Finalmente, gostaria de agradecer a todos os professores da Faculdade de Ciências da Universidade de Lisboa, com quem tive o prazer de ter aulas ao longo destes anos, pelos conhecimentos e valores que me transmitiram. Espero conseguir fazer uso de tudo o que me ensinaram de forma a dignificar tanto a faculdade, como a profissão de engenheiro informático.

A mis abuelos

Resumo

A robótica evolucionária é uma técnica que visa criar controladores e morfologias para robôs autónomos, usando técnicas de computação evolucionária, tais como os algoritmos genéticos. De forma semelhante ao principio darwiniano de reprodução do mais apto, o algoritmo genético selecciona os indivíduos mais aptos de cada geração para criar a próxima e assim sucessivamente, até um controlador adequado para a tarefa escolhida seja alcançado.

O principal objectivo deste trabalho é o estudo da emergência de comportamentos colectivos em grupos de robôs autónomos, usando técnicas de evolução artificial para evoluir controladores adequados. A emergência de protocolos explícitos de comunicação nas experiências é também estudado, com o objectivo de entender a sua influência nos comportamentos que os controladores desenvolveram.

A evolução artificial de controladores pode ser uma tarefa demorada, e a natureza aleatória das primeiras gerações destes pode danificar os robôs (podem surgir comportamentos que levem os robôs a embater contra paredes ou a embater uns contra os outros, etc). Para contornar estes problemas, tanto a evolução dos controladores como os seus respectivos testes foram feitos num simulador. Estes são redes neuronais recorrentes com entradas temporais, cujos pesos das ligações sinápticas, tendência e a taxa de decaimento estão codificados em cromossomas. Os cromossomas são criados usando um algoritmo genético e avaliados por uma função de avaliação desenhada especificamente para a tarefa que os controladores terão que efectuar.

O simulador JBotEvolver (http://jbotevolver.sourceforge.net) foi o sumulador utilizado para realizar as experiências. Este simulador, escrito na linguagem Java, é um projecto de código aberto que permite simular o comportamento de grupos de robôs num dado ambiente. Para efectuar a evolução artificial do controlador de um grupo de robôs, o simulador usa algoritmos genéticos, em conjunto com uma função de avaliação, previamente escolhida consoante a tarefa em questão. Um interface gráfico permite ao utilizador ver uma representação do ambiente, dos robôs e do seu comportamento. Todos os parâmetros da simulação podem ser alterados ao editar um ficheiro de configuração, que é único para cada experiência.

Quando a simulação está a decorrer, um conjunto de ficheiros de dados e de configuração são criados, com os valores de *fitness* de cada geração, a melhor geração até ao momento

e gerações anteriores. Desta forma é possível avaliar os resultados de uma determinada geração, fazer estudos sobre como é que a evolução decorreu, etc.

O simulador pode também fazer evolução artificial de forma distribuída. O lado servidor distribui pedaços de informação para os clientes processarem e enviarem de volta. Esta característica do simulador é extremamente útil, especialmente quando são feitas simulações que exijam uma grande capacidade de processamento.

Os robôs simulados no conjunto de experiências descrito nesta tese são de forma circular, com dez centímetros de diâmetro, duas rodas que se movem de forma independente, com velocidades a variar entre [-5cm/s, 5cm/s] e um conjunto de sensores que variam consoante o tipo de experiência. Um actuador de cor, que permite aos robôs variar a sua cor em todo o espectro RGB é também incluido nas experiências em que se estuda a emergência de protocolos de comunicação explícitos.

Nesta tese, três temas distintos são abordados pelas experiências: agregação auto organizada, escolha colectiva e gestão de energia. No estudo da agregação auto organizada, um controlador foi criado através de evolução artificial, para fazer que um grupo de robôs que inicialmente se encontram distribuidos aleatóriamente pelo ambiente, se agreguem num único grupo e se mantenham unidos. Para evoluir este controlador foi utilizada uma população de cinco robôs com sensores que lhes permitiam detectar outros robôs próxims, no entanto para efectuar um estudo de escalabilidade, o controlador foi testado com dez, quinze e cem robôs.

Para o estudo de comportamentos de escolha colectiva, foi criado um ambiente com diversas marcas luminosas. O objectivo era que os robôs encontrassem e escolhessem colectivamente uma dessas marcas formando um único grupo dentro delas. O controlador criado para esta experiências foi evoluido usando um grupo de cinco robôs com sensores proximidade entre robôs e sensores de luz para detectar as marcas luminosas e um ambiente com duas marcas luminosas, a distarem dois metros uma da outra. Estes sensores desenhados para detectar marcas luminosas possuem um alcance de dez centímetros e os robôs iniciam a simulação situados entre as marcas, pelo que os robôs têm que procurar activamente pelas marcas. Para estudar a escalabilidade do controlador produzido, este foi testado com grupos de dez e quinze robôs e num ambiente com cinco marcas luminosas. A influência de protocolos explícitos de comunicação foi estudada também, ao equipar os robôs com um actuador de cor e sensores que detectam mudanças de cor nos robôs nas proximidades.

Finalmente, para o estudo de comportamentos de gestão de energia, um ambiente com duas fontes de energia foi criado e um grupo de robôs com uma energia limitada foi usado. O objectivo do controlador produzido foi o de manter o maior grupo de robôs possível intacto, ou seja sem nenhum chegar a um nível zero de energia, o máximo de tempo possível. As fontes de energia usadas nesta experiência apenas podem carregar dois robôs simultaneamente, sendo que se um terceiro robô se juntar, todos os robôs

deixam de receber energia. Os robôs utilizados para esta experiência possuem sensores de proximidade entre robôs e sensores que detectam as fontes de energia, bem como o número de robôs que estas se encontram a carregar num dado instante. O controlador foi evoluido usando um grupo de 5 robôs com autonomia de cerca de 80 segundos, durante 120 segundos. Para estudar a escalabilidade do controlador, foram usados grupos de sete e dez robôs e o tempo da simulação foi aumentado para 150 e 200 segundos. O estudo de protocolos de comunicação explícitos foi também estudado ao adicionar um actuador de cor e sensores que detectam mudanças de cor nos robôs nas proximidades, tal como foi feito na experiência de comportamentos de escolha colectiva.

Palavras-chave: Robótica evoluionária, redes neuronais, algoritmos genéticos, escolha colectiva, agregação auto organizada, gestão de energia.

Abstract

Evolutionary robotics is a technique that aims to create controllers and sometimes morphologies for autonomous robots by using evolutionary computation techniques, such as genetic algorithms. Inspired by the Darwinian principle of survival of the fittest through reproductive success, the genetic algorithms select the fittest individuals of each generation in order to create the next one and so forth, until a suitable controller for the designated task is found or for a certain number of generations.

The main goal of this work is to study the emergence of collective behaviors in a group of autonomous robots by using artificial evolution techniques to evolve suitable controllers. The emergence of explicit communication protocols in the experiments is also studied in order to understand its influence on the behaviors the controllers evolved.

Since artificial evolution can be a time consuming task, and because of the random nature of the controllers produced in early generations can damage real robots, a simulator is often used to evolve and test the controllers. The controllers used in this study are Continuous Time Recurrent Neural Networks whose weights of the synaptic connections, bias and decay rates are encoded into chromosomes. The chromosomes are produced by using a genetic algorithm and evaluated by an evaluation function designed specifically for the task that simulated robots have to perform.

The controllers produced through artificial evolution are tested in terms of performance and scalability. The components of the simulator, such as evaluation functions, environments, experiments, physical objects and so forth are described. Some guidelines of how to create such components, as well as some code examples, are available in the report to allow future users to modify and improve the simulator.

Keywords: Collective evolutionary robotics, artificial evolution, self-organized aggregation, collective choice, resource management, artificial neural networks

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Since the concept of robot was created by the Czech writer Karel Čapek in 1920, people have dreamt of building an intelligent machine able to perform tasks with more efficiency than humans. Nowadays, robots are a reality in our society (even though not in the way that Karel Čapek imagined in his play) and robotics is a serious field of research. Two of the branches of robotics research are evolutionary robotics and collective robotics.

One of the main inspiration sources for evolutionary and collective robotics is nature and the way that life evolved in our planet, from unicellular organisms to complex beings, generation by generation. From the vast diversity of living organisms, insects are among the oldest, more numerous and simple life forms present on our planet. However, despite they are simple in terms of cognitive capacities they show some complex collective behaviors that allowed them to prosper in a variety of ecosystems.

Several studies have been made regarding complex insect societies and its results are very useful to evolutionary and collective robotics research: insects are among the oldest animals on Earth, and therefore, the modern insects we see today, are the result of a long process of successful evolution. They have complex societies that strive for the survival of the colony instead of the individual. Much of their behavior is therefore for the benefit of the society and decisions are made in a collective distributed way. Also, like a colony of insects is able to establish a nest in a given location and exploit the natural resources available, a swarm of robots should be able to exploit an energy source in the most efficient way.

In this study, I aim to simulate the evolutionary process to obtain a similar collective decision making behavior and also to obtain some resource management behaviors that will allow a group of robots to survive in a given environment. The emergence of explicit communication protocols in the experiments is also studied in order to understand its influence in the behaviors the controllers developed. The robots used in the experiments are controlled by a Continuous Time Recurrent Neural Network (CTRNN) [4] whose weights of the synaptic connections, bias and decay rate are encoded in chromosomes. The chromosomes are put of an evolutionary algorithm and each chromosome is evaluated by an evaluation function designed specifically for the task the controllers have to perform. Like in the Darwinian principle of reproduction of the fittest, the fittest chromosomes are selected to create the next generation and so forth, until a suitable controller for the designated task is achieved or for a certain number of generations.

The artificial evolution process can be a very time consuming task so, in order to make it faster and because of the random nature of the early controllers that can actually damage physical robots (evolution can follow a path in early iterations that may lead robots to bump against walls for example), it is simulated by software.

The research in evolutionary robotics can be applied in many fields of study: in robotics, to develop efficient controllers for real-world robots that interact in complex environments, in biology, to study biological neural networks and even to study the theory of evolution itself.

Advances in those fields may lead to real-world applications of evolutionary robotics: more efficient and intelligent robots may be developed to explore environments that are too dangerous or unknown for humans, a better understanding of the human brain may be achieved and it may even give us some insights into evolution in general.

1.2 Objectives

The aim of this report is to evolve and study emergent collective behaviors in simulated groups of robots and to study the evolution of communication and its advantages and disadvantages. Since there are an infinite number of real world problems that a controller could be evolved to solve, I focused my research in three of the most basic behaviors a robot must have to perform more complex tasks: self-organized aggregation, collective choice and energy management. Collective choice is arguably one of the most common behaviors present in every group of living beings in nature and plays an important role on the survival of the colony (ants, cockroaches and some species of fishes are a good example).

In order to survive, a group of living beings has also to develop strategies to manage the resources available in the environment, and that is also an interesting behavior I wanted to study and tried to replicate through artificial evolution. Also, the scalability of the controllers produced is studied to analyze the possibility of evolving controllers in a simple environment, with a small number of robots that can perform well in more complex environments, with larger groups of robots.

1.3 Contributions

There is some remarkable research in this field regarding aggregation, collective choice and language emergence like we can observe in the related work section. In this thesis, the collective choice and the energy management experiments compare populations capable of explicit communication through colors and populations without that ability. The fitness trajectories obtained by both populations are also compared in terms of how fast a good solution is evolved.

Aggregation, collective choice and energy management is studied and I tried to evolve those kinds of behaviors in the simulator, using Continuous Time Recurrent Neural Networks (CTRNN) [4] and an evolutionary algorithm which will be explained in chapter 3. The experiments involved a population of robots controlled by the same neural network and a set of environments. The environments were designed to study how would aggregation, collective choice and energy management behaviors evolve in scenarios with different characteristics. To study simple aggregation, a simple environment with no landmarks was used. Collective choice was studied by adding landmarks, which the robots could detect at a limited distance, and therefore force the population to develop strategies to find the landmarks and collectively choose one. Communication was then added to the previous problem to study the effect that this ability has in the strategies developed. To study if more complex strategies would emerge, the landmarks were replaced by energy sources and a limited amount of energy was given to the robots. The energy sources can charge a limited number of robots simultaneously, so an energy management strategy must emerge to maintain the entire population alive. Like the previous problem, energy management was also studied with communication capable robots. So far I could not find in literature descriptions of experiments where a population of robots had to learn how to manage an energy source using evolutionary robotics as described in this report. If these experiments are not a novelty, they will be among a very few.

1.4 Structure of the document

Chapter 2 is a short historic overview of evolutionary robotics (ER) and its current state of the art, referring some of the most relevant work that has been done so far. Chapter 3 describes the simulator used, as well as its main components and the evolutionary algorithm used. It also explains how to create new components, such as new sensors, actuators, environments and so forth. Chapter 4 describes the experiment about self-organized aggregation, chapter 5 describes the collective choice experiments with and without populations capable of communicating through a color actuator and chapter 6 describes the energy management experiments, again with and without populations capable of communicating through a color actuator. There will be a total of five experiments, presented from the most simple to the most complex scenario. Since the neural networks begin with random values, a remarkably good or bad controller can be achieved and it will not provide good data about the effectiveness of the evaluation function used. To mitigate this fact, every experiment was run 10 times (evolutionary runs) with random initial seed numbers.

Each experiment will be presented as follows:

* Purpose of the experiment in the context of my work.
* Experimental setup used
* Description of the fitness function used
* Presentation of the results
* Discussion

The controllers produced are also tested in slightly different conditions than the ones used in their evolution by varying the number of robots used and/or the number of landmarks/energy poles present in the environment, so that characteristics like scalability can be studied.

Finally, in chapter 7 some conclusions are drawn regarding the experiments performed, a comparison between the goals set in the preliminary report and the goals achieved in the final report and some future work is suggested. Appendix A to G offers code examples of the various components of the simulator and also some guidelines of how to create them. An explanation and an example of the configuration files are also present.

Chapter 2

Related work

2.1 Historic overview

The foundations of evolutionary robotics can be tracked back to late 50's, when the evolutionary computation concept was developed [2], [14], [13]. Nevertheless, evolutionary computation field remained relatively unknown to the broader scientific community for about thirty years [2].

The situation changed during the 80's, when computational power available started to be sufficient to conduct some experiments and to solve real-world optimization problems. In 1985, a number of international conferences on techniques, such as genetic algorithms and its theoretical aspects and evolutionary programming, were held, but it was not until the 90's the researchers in the different fields of evolutionary computation started to cooperate [2].

It was on one of those conferences in the 90's, the national council in Rome, the foundations of evolutionary robotics were laid. In 92 and 93, Floreano and Mondada at the EPFL in Laussane, Switzerland, and a research group at COGS, at the university of Sussex, reported experiments on artificial evolution on autonomous robots [36]. The results were promising, and that triggered a wave of activity in labs around the world.

Lately, due to the difficulty to scale-up the complexity of the robots tasks, research started to focus more on the theoretical questions of the field, rather than the engineering challenges.

2.2 Evolutionary collective robotics overview

The primary goal of evolutionary robotics (ER) is to develop methods for automatically synthesizing intelligent autonomous robot systems [35]. Although most of research in ER is about controllers, EA is also applied in the creation of robot morphologies [26], but this topic will not be discussed further in this report.

Evolutionary robotics is not depending on research about nature but it uses some of its results: there are some remarkable works about collective choice regarding cockroachesÕ [22] and ants [10]. Garnier et. al. [16] studied aggregation behaviors in cockroach larvae. In this study, different aggregation behaviors were observed, such as wall following and two different resting behaviors. The rest time in a single shelter has been proved to influence in which shelter cockroaches (Blatella) and ants (Oecophylla) will aggregate [10]. Jean-Mark Ame et al. [1] took this research a step further and developed a mathematical model to reproduce these collective responses, based on the relation between the resting period in a given site with the number of individuals. Studies about swarms in nature influenced collective robotics: there has been some attempts to mimic aggregation behaviors, as described in [15], where a group of Alice [6] micro-robots were used to perform the task. An even more realistic scenario was proposed by [24], where flocking behaviors were evolved in ecosystems consisting of plants, herbivores and predators.

Even though nature is a main source of inspiration, collective robotics also take some more articial approaches to perform aggregation and cooperation behaviors. [29] studied seeded aggregation and flocking using sound signals (chorusing). A similar experiment was proposed in *Evolving mobile robots able to display collective behaviors* [3], where robot controllers were evolved to make the robots aggregate and move together toward a light target. In the experiments described in this paper, the controllers evolved several strategies to perform the given task. Using a probabilistic model, N. Correll and A. Martinoli [7] showed how Alice micro-robots could aggregate and a more simple approach for aggregation and flocking was taken by C. Moeslinger, T. Schmickl, and K. Crailsheim [33], by describing a simple flocking algorithm to be used with very simple swarm robots which works without the need for communication, memory or global information. *Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors* [38] showed how a group of 3 robots with minimal sensor capabilities could aggregate and move as a group using simulator evolved controllers. Remarkably, the team members dynamically adopted roles to make the group movement easier. V. Trianni et al. [44] describes how it is possible to use artificial evolution to find simple solutions to the aggregation problem, mainly by exploiting some invariants present in the environment and the complex interactions among the robots and between the robots and the environment.

An important aspect of a society is how information is transmitted between its members [28]. The individuals that form a society can be very complex (like humans), or very simple (like bees or ants) in terms of cognitive or sensorial capabilities, but they will still have the need to communicate in order for the society to survive. The complexity of the organisms will influence the way communication is performed: humans are arguably the most complex known organism and we developed a very complex form of communication, and on the other hand, ants rely on very simple strategies, like the deployment of pheromones to mark a path for food [21]. Research in evolutionary robotics has been trying to make communication emerge among populations of simple robots in terms of cognitive and sensorial capabilities so they can accomplish tasks in a more efficient way. The emergence of communication might seem very unlikely, because the robots need to produce signals that are useful and react to useful signals. However, like Maynard Smith said ”*It's not good making a signal unless it is understood and a signal will not be understood the first time it is made*” [27], which means that the adaptive variations that led to the production of useful signals or to the exploitation of the same signals are adaptively neutral, unless they are developed at the same time. Nevertheless, experimental results showed that communication protocols can emerge in some cases.

One of the earliest attempts to make communication emerge using artificial organisms can be found in the G. Werner and M. Dyer paper [45]. The aim of this work was to evolve simple communication protocols for mate finding: female individuals had the ability to see males and emit sounds and male individuals were blind but able to perceive the sounds emitted by females. The results showed a progression of generations that showed increasingly effective mate-finding strategies. It also showed a number of groups with different signaling protocols. Angelo Cangelosi and Domenico Parisi performed some experiments where neural networks living in an environment evolved a simple language to help other individuals to decide whether a mushroom is edible or poisonous [37]. A similar experiment was performed in an environment with a food source and a poison source with robots capable of emitting a blue light [32]. The robots had to find a way to warn the rest of the population when they find a food source or a poison source. The way the population was evaluated proved to be important in the final behavior. When selection was made at an individual level, the robots started to emit deceiving signals so they could exploit more efficiently the food source. The communication protocol the robots developed was not uniform in all the experimental runs: in some cases, the food source was signaled by the robots by emitting the blue light and in other cases the robots signaled the poison with the blue light. The question of how a population of initially non-communicating robots might develop communication skills that are functional with respect to the task that the robots have to perform without being rewarded for communication is addressed in *Evolution of implicit and explicit communication in mobile robots* [9]. In this article, teams of 2 e-puck robots [34] with neural networks evolved in a simulator had to find 2 landmarks and switch between them as often as possible. The robots were able to communicate through a bluetooth connection but they were not rewarded for that. Results showed that populations able to explicitly communicate developed a communication protocol to help them to coordinate and cooperate. The robots also developed 2 different strategies to perform the task, which are thoroughly explained in the article.

More recently, some open challenges regarding the evolution of language in embodied agents were pointed out and some methods to address those challenges in a correct way were also established [30] [43]. The strategic aspects of communication are also important factors to take in consideration in order to better understand and guide the evolution of communication protocols [18]. A solution to the problem of honest communication among agents (which was faced in [32] and [31] for example), is presented: *costly signalling*. The main idea behind this concept is that if a signal is costly, the simple fact that the signal is present reveals something of value, even when in the presence of a deceiving signal. Some examples of *costly signalling* can be found in our society: Ownership of a Rolls Royce is a credible signal of wealth because only a wealthy person can afford one.

Societies in nature also have to develop strategies to manage the resources available. There has been lots of research about resource management, which are out of the scope of this report, but arguably the most famous is the *Tragedy of the commons* [19]. In this article, it is shown that the efficient management of resources by a growing population is impossible. The example given in the article shows how a profitable situation for an individual can be harmful for the society. There has been some research regarding this question like in *Autonomous robots sharing a charging station with no communication: a case study* [41] adapted to ER, where a group of self-sufficient mobile robots share a charging station using simple mechanisms or in *Market-basedcoordinationofrecharg- ing robots* [25] where market-based coordination techniques are used to effectively allocate recharging tasks to maximize the productivity and efficiency of the team. However, artificial evolution is not used in both cases. Surprisingly, it seems that the emergence of behaviors that would allow a group of robots to be self-sufficient has never been studied thoroughly.

All the experiments described above show that artificial evolution can be successfully applied to synthesize effective controllers. More than that, artificial evolution can develop controllers that would be nearly impossible to achieve if explicitly programmed ([3].

Chapter 3

The simulator

3.1 Simulation environment

To perform the experiments, we are using the JBotEvolver (available at http:// jbote-volver.sourceforge.net).

This simulator, written in the Java language, is an open source project that allows us to simulate the behavior of a population of e-puck like robots in a given environment. The number and type of actuators and sensors available on the robots are customizable, which means that we can simulate any given type of sensor and actuator.

JBotEvolver can conduct artificial evolution of a controller of a population of robots, in a given environment. To do that, the simulator relies on neural networks and evolutionary algorithms, along with an evaluation function previously chosen. A graphical user interface allows the user to see a representation of the environment, the robots and their behavior. Settings can be chosen by editing a configuration file, which will be unique for each experiment we conduct.

When the simulation is running, several data files and configuration files are created, regarding fitness values of each evolution, the best generation so far and the generations previously generated. This way it is possible to evaluate the evolutions results of a given setup.

The simulator can also perform evolution in a distributed manner. The server side of the simulator will distribute chunks of information for the clients to process and send back the results. This is a very useful feature to perform evolution with very complex experimental setups that require a lot of processing power.

3.2 Components of the simulator

3.2.1 The robots

The simulator can handle multiple types of robots, which are defined by the user. A robot is, in its core, a set of sensors, actuators and pre-programmed behaviors. However, it is possible to create more complex robots by extending the *Robot* class. The robots are customizable, which means that the number and type of sensors and actuators and the size of the robots can be defined by altering the configuration file. Sensors and actuators can be also programmed so the number of setups and types of robots the simulator can handle is vast. An example of a configuration file can be found on Appendix B.

To perform the experiments described in this document, a differential drive, cylindrical robot model with 10 cm of diameter was used. The actuators are the two wheels that allow the robot to move and an RGB actuator used as a communication means, when communication is used. The wheels can be controlled independently allowing the robots to steer and rotate and the RGB actuator can produce colors in the entire RGB color spectrum, so the controllers can evolve a language to communicate. An example of an actuator and some guidelines of how to create new actuators can be found on Appendix E.

Each robot is also equipped with a variety of sensors that will detect other robots nearby, the landmarks used, the energy poles and the robot's internal energy. Each sensor detects a specific type of object: another robot, the landmark object and the energy pole. Except the internal energy sensor, all sensors are disposed in the robot's body, as shown in figure 3.1, and have an opening angle of 135°, so the sensors can cover enough space and give the controller accurate information of the location of the physical objects (both distance and relative position with respect to the robot). The range varies according to the experiment needs. If there are no sources within the sensor's range and opening angle, its readings will be 0, otherwise the readings will be based on the distance to the closest source (c) according to the following equation:

$$s = \frac{range - d_c}{range} \tag{3.1}$$

or

$$s = \left(\frac{range - d_c}{range}\right) \cdot \tau, \tag{3.2}$$

in the case of the energy pole sensor, where *range* is the sensor's detection range, d_c is the distance between the closest source *c* and the sensor and τ is the energy level, normalized between 0 and 1, of the energy pole. To get results that are closer to the ones expected if the controllers were to be used in the real world, the sensors also have a noise factor. An example of a sensor and some guidelines of how to create new sensors can be found on Appendix D.

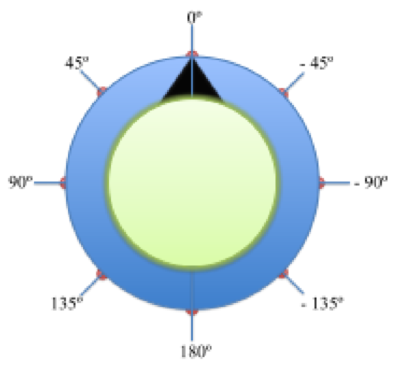


Figure 3.1: Scheme of the robot used in the simulator

3.2.2 The controllers

Controller classes can be created to control the robots, which means that the robots can use multiple techniques to navigate in the environments, from simple state machines to neural networks.

In the experiments described in this document, the robots are controlled by a Continuous Time Recurrent Neural Network [4] which consists of three layers of neurons: an input layer with N neurons, depending on the number and type of sensors used, a hidden layer with 5 neurons and an output layer with 2 or 3 neurons depending whether the robots are using the color actuator or not. The input neurons Ii are connected to the sensors, so they will react to other nearby robots, landmarks, energy poles or internal energy level. The neurons in the hidden layer are fully connected and are governed by the following equation:

$$\tau_i \frac{dH_i}{dt} = -H_i + \sum_{j=1}^{N} \omega_{ji} I_i + \sum_{k=1}^{5} \omega_{ki} Z\left(H_k + \beta_k\right), \tag{3.3}$$

where τ_i is the decay constant, H_i is the neuron's state,ω_{ji} the strength of the synaptic connection from neuron j to neuron i, β the bias terms, and

$$Z(x) = \left(1 + e^{-x}\right)^{-1} \tag{3.4}$$

is the sigmoid function. τ_i, β and ω_{ji} are genetically controlled network parameters. Cell potentials are set to 0 when the network is initialized or reset and circuits are integrated using the forward Euler method with an integration step-size of 0.1.

The output layer is fully connected to the neurons in the hidden layer. The activation of the output neurons is given by the following equation:

$$O_i = \sum_{j=1}^{N} \omega_{ji} Z\left(y_j + \beta_j\right) \tag{3.5}$$

Each output neuron is connected to an actuator and it will be responsible for its behavior. Outputs O_1 and O_2 can control left and right wheels respectively, and their output is linearly mapped to speeds between [-50 cm/s, 50 cm/s]. Output O_3 controls the color of the robot when the color actuator is used.

All robots have instances of the same controller so the population is homogeneous. Figure 3.2 shows a diagram of a neural network used when the robots used have proximity, landmark and color sensors, wheels and color actuator.

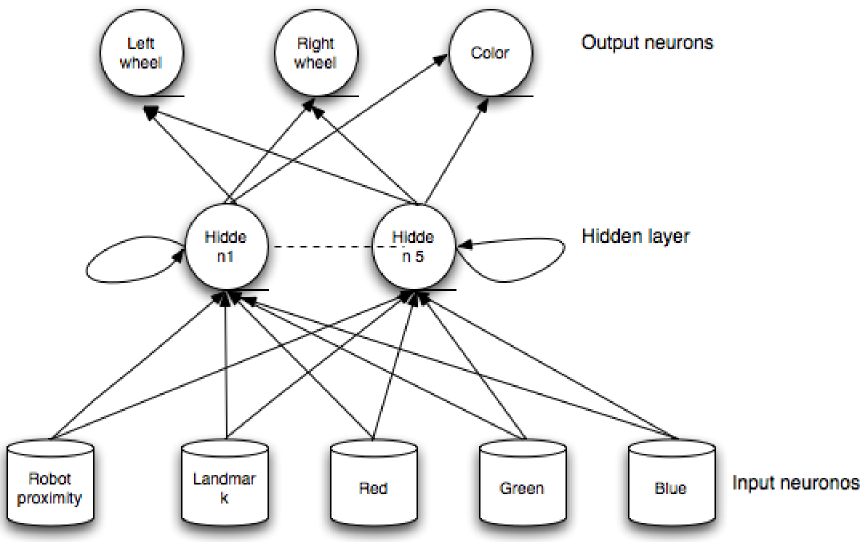


Figure 3.2: Diagram of a neural network used in the experiments

3.2.3 The evolutionary algorithm

Evolutionary algorithms (EAs) are computer programs that attempt to solve problems by using the Darwinian concept of evolution [8]. In EAs, individuals are represented by a vector that encodes a single possible solution to the problem. The EA starts with a random initial population of μ individuals. Each individual is evaluated by a fitness function and then assigned a fitness value. Individuals with higher fitness values are considered to be better solutions and thus used to produce (and sometimes to be part of) the next generation and so forth, until a goal fitness is achieved or a limit number of generations is reached. The next generation is produced by using mutation, recombination and/or crossover operators. The simulator refers the evolutionary algorithm as the population. Population classes are in the *evolutionaryrobotics.populations* package and extend the *Population* abstract class.

Evolutionary algorithms have different implementations [23]. There are three main implementation instances of EAs: Genetic algorithms, developed by Holland [20] and thoroughly reviewed by Goldberg [17], evolution strategies (ESs) developed by Rechenberg [39] and Shwefel [40] and evolutionary programming (EP) developed by L. J. Fogel et al. [12] and refined by D. B. Fogel [11].

In the set of experiments that will be described in this report, a genetic algorithm was used to perform evolution ([μ, λ] algorithm). The algorithm uses generations that consist of 100 chromosomes each (λ). The chromosomes encode the weights of the synaptic connections, bias and decay constant of the neural network. The topology of the neural network is not subject to evolution, but hand designed and defined globally and externally. The fitness of each chromosome is sampled (which means the chromosome is tested a number of times and the final chromosome fitness will be the average of the fitness values achieved in each sample) and the 5 best are chosen (μ) to create the next generation, discarding the remaining ones. Each one of the 5 chosen parents will create 19 children and finally, both parents and children are used to create the next generation. The genotype of the children is obtained by adding a random Gaussian offset to each gene, with a 15% of probability (mutation).

Parameters of the evolutionary algorithm used, such as chromosome number, number of time steps, number of generations and mutation rate can be set in the population section of the configuration file. Different evolutionary algorithms can also be created and used.

3.2.4 Evaluation functions

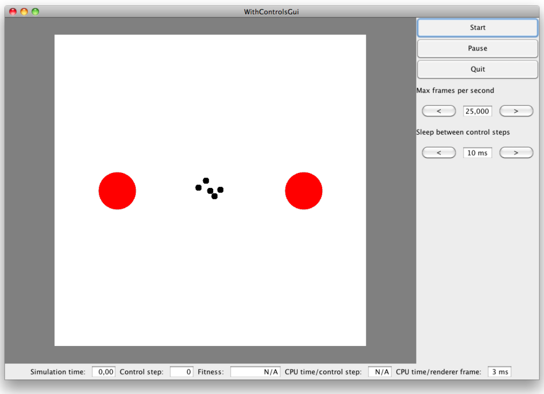
To evaluate the performance of the chromosomes of each generation, each experiment has an evaluation function. The evaluation function scores the chromosomes with a fitness value. The fitness of each chromosome is sampled a number of times during a number of control steps defined in the configuration file, and selection is based on the average fitness obtained. The fitness value is always calculated at group-level with an evaluation function.

The evaluation functions used will be described in detail in each experiment description. See appendix A for an example of an evaluation function and some guidelines of how to create new ones.

3.2.5 The environments

The environment is the virtual place where the simulation takes place. It is chosen by the user in the configuration file, as well as its parameters. An environment has a set of physical objects and rules to govern the interactions between them and the robots, such as collisions. The number, size, placement and so forth of the physical objects is also defined in the environment.

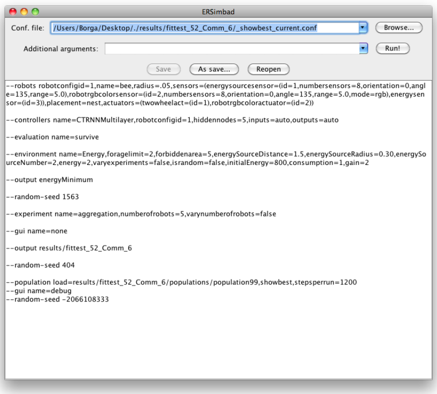
The environments may or may not be static: it is possible, for example, to make the number, size and/or position of a given physical object vary in each sample by adding a parameter in the environment section of the configuration file. Figure 3.3(a) shows an example of an environment and figure 3.3(b) shows an example of a configuration file.



Max frames per second: 25,000
Sleep between control steps: 10 ms

Simulation time: 0,00 Control step: 0 Fitness: N/A CPU time/control step: N/A CPU time/renderer frame: 3 ms

(a) Energy Management environment



Conf. file: /Users/Borga/Desktop/./results/fittest_52_Comm_6/_showbest_current.conf

Additional arguments:

```
--robots robotconfigid=1,name=bee,radius=.05,sensors=(energysourcesensor=(id=1,numbersensors=8,orientation=0,angl
e=135,range=5.0),robotrgbcolorsensor=(id=2,numbersensors=8,orientation=0,angle=135,range=5.0,mode=rgb),energysen
sor=(id=3)),placement=nest,actuators=(twowheelact=(id=1),robotrgbcoloractuator=(id=2))

--controllers name=CTRNNMultilayer,robotconfigid=1,hiddennodes=5,inputs=auto,outputs=auto

--evaluation name=survive

--environment name=Energy,foragelimit=2,forbiddenarea=5,energySourceDistance=1.5,energySourceRadius=0.30,energySo
urceNumber=2,energy=2,varyexperiments=false,israndom=false,initialEnergy=800,consumption=1,gain=2

--output energyMinimum

--random-seed 1563

--experiment name=aggregation,numberofrobots=5,varynumberofrobots=false

--gui name=none

--output results/fittest_52_Comm_6

--random-seed 404

--population load=results/fittest_52_Comm_6/populations/population99,showbest,stepsperrun=1200
--gui name=debug
--random-seed -2066108333
```

(b) Configuration file example

Figure 3.3: Configuration file and simulator environment

3.2.6 Physical objects

Physical objects are all the objects present in the environment where the evolution will occur. Everything that the robots can interact with (besides the robots themselves) is a physical object. New physical objects can be created, with their rules of interaction, in the simulator An example of a physical object and an explanation of how to create one can be found on Appendix C.

3.2.7 The experiments

The experiment classes define the rules of the experiment, like the way the robots will be placed in the environment, the types of robots the experiment will have (enemy and friendly robots for example will have different setups and those setups will be defined here) or even if the number of robots can vary in each sample and how is that variation made. The environment is also created in the experiment class, as well as the chromosome and the controller. If the controller is a neural network, its weights are also defined here. An example of an experiment class and some guidelines for creating new ones can be found on Appendix G.

3.3 Contributions to the simulator

Some functionalities have been developed to the JBotEvolver simulator but not all have been used to perform the experiments described in this report. Mainly this is due to poor or inconclusive results that the experiments developed achieved. However, I believe they may be useful in the future. Among the new functionalities for the robots we can highlight the *Near robot* sensor which detects nearby robots, an artificial pheromone to mimic the ants communication, as well as the respective actuator (the mechanism that places the pheromone in the environment) and sensor.

Two new environments were created, namely the *Landmark* and the *Manage* environments. In the *Landmark* environment, landmarks, which the robots can detect, are placed. The type of placement can be regular, which means the landmarks will be placed along a circumference and at a constant distance from each other, or random, where the landmarks will be randomly placed in the arena without overlapping. The size of the landmarks can also be customized as well as the distance that separates each landmark. The *Manage* environment works in a similar manner, but the objects that it contains are charging poles. This environment has all the customization options available in the *Landmark* environment and it also allows to define the number of robots each landmark can recharge at a time, the initial energy of the robots, their energy consumption per time unit and their energy gain when recharging per time unit. Both environments can also change during simulation: it is possible to change the number and placement of the landmarks or charging poles available on each simulation run or sample. The way the number and placement of the objects varies is constant: for example, if the simulation has 4 samples, the objects will be placed in 4 fixed (randomly assigned) different places, and the number of object will increment by adding the original number of landmarks 4 times. That will allow the controllers to evolve in a set of different environments, thus making the controllers more general.

A new experiment was also created, called *Aggregation*. In this experiment, the number of robots can vary in the same way the number of objects present in the environment.

Several evaluation functions were created to evaluate the controllers, namely the *CenterOfMass* and *CenterOfMassAndClusters* (which counts how many clusters of robots are presents at any time) to evaluate self-organized aggregation, *Recruiting* to evaluate collective choice and *Survival* to evaluate energy management in the *Energy management* experiments.

This report also explains in detail how to create sensors, actuators, experiments, environments, evaluation functions and configuration files to the simulator so it can be used as a user manual for the simulator.

Chapter 4

Self-organized aggregation

4.1 Introduction

Aggregation is one of the most pervasive behaviors in nature [5]: fishes aggregate so they can look like a larger animal to potential predators, some insects, like the cockroaches aggregate in valuable locations to survive and establish their societies [46] and so forth. Life would not be possible without aggregation phenomena: atoms aggregate and form new elements due to the weak nuclear force, thus creating the building blocks of life. Aggregation behaviors can be enhanced by environmental clues, such as light, humidity or food abundance to name a few. However, aggregation can also be self-organized: cockroach aggregation behaviors result from an emergent cooperative decision [42]. The study of aggregation and cooperative decision is important to swarm robotics: usually, swarm robotics uses simple robots that have to cooperate and that cooperation usually requires the robots to be close to one another. Also, aggregation is an important behavior for swarms of robots to self-assemble or perform pattern formations.

This experiment aims to evolve a self-organized aggregation behavior. The robots must form a single cluster and that cluster should be stable (i.e. should not have robots leaving the cluster once they are in). The place the robots aggregate is not important for this experiment, so the cluster can be formed anywhere in the environment.

4.1.1 Experimental setup

For this experiment, I used a population of 5 differential drive robots, with 8 robot proximity sensors. The opening angle of the sensors is 135° and the sensor range is 5.0 meters. 10 evolutionary runs were performed with this setup with different initial random seeds for 100 generations each. The fitness of each genome was sampled 4 times in trials of 2 minutes of virtual time (1200 control steps) each. In each sample, the robots are placed at a random position, so evolution cannot use the starting point of the robots to enhance aggregation, also the bias values in the robot proximity sensors vary.

The environment used to perform the experiments is an infinite space, with no landmarks or walls, so the only reference the robots will have will be other robots nearby.

4.1.2 Evaluation function

To evaluate the controllers produced by the evolution, I used an adaptation of the approach described by Trianni et al.[44]: the evaluation function checks the coordinates of all robots and calculates the center of mass, which means that coordinate will always be closer to where the majority of the robots are. The fitness value represents the average distance of the group from its center of mass multiplied by a factor, added to the sum of the neighbors of each robot, multiplied by a factor:

$$f_s = \sum_{t=0}^{\omega} \frac{\frac{1}{n}\left(\frac{10}{11} \cdot \sum_{i=1}^{n}\left(1 - \frac{d_i(t)}{\delta}\right) + \frac{1}{11} \cdot \sum_{i=1}^{n}\left(\frac{\rho(i)}{n*(n-1)}\right)\right)}{\omega} \tag{4.1}$$

where f_s represents the fitness of a sample, n is the number of robots used, t is the number of the current control step, $d_i(t)$ is the distance of the i^{th} robot to the center of mass, δ is the sensor range of the robots, $\rho(i)$ is the number of neighbors of the i^{th} robot and ω is the number of steps per run. The distance $d_i(t)$ varies between 0 and the maximum range of the sensor used. The fitness is calculated at group level and varies between [0, 1]. The final fitness value of a chromosome will be the average of all fitness values obtained in each sample.

4.1.3 Results

In all 10 performed evolutionary runs of the experiment, the robots managed to aggregate in a very efficient way: as we can observe in fig 4.1, which shows the fitness trajectories of all evolutionary runs, the values obtained are close to 0.9 which represents a quick aggregation. The fitness trajectories also show that an optimal controller was reached early on the evolution process.

Even though there is elitism in the evolutionary algorithm used, the fitness can still drop from generation to generation, as observed in the fitness trajectory. This is due to the fact that a random seed is used in each sample, which will define sensor noise and initial position. The random seed can be such that will make the robots perform worst (i.e. a slightly higher sensor noise). For that reason, each experiment is also performed a number of times, or evolutionary runs, so the results obtained are accurate and independent from the random nature of the initial position of the robots, sensor noise and the first generation.

Ten samples of the last generation of each evolutionary run were run to observe the strategies the robots developed to aggregate: the robots simply converge to the population's center of mass. When a single cluster of robots is formed, the cluster will wander through the arena. The cluster is cohesive but it will never stop moving, and that is the

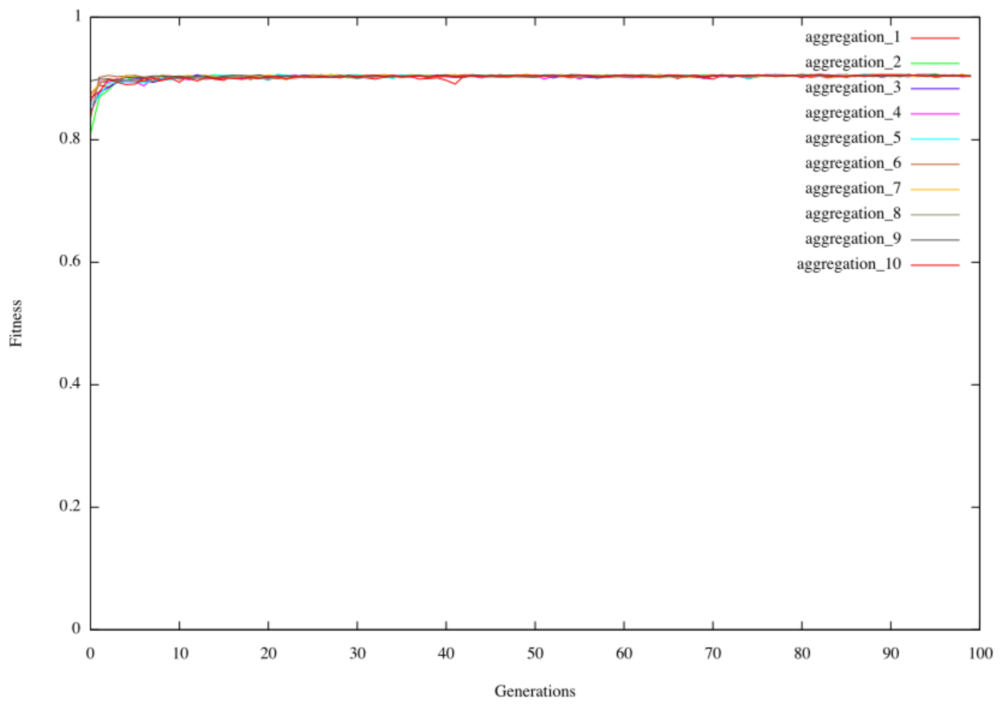


Figure 4.1: fitness trajectories of all 10 evolutionary runs

reason for the collective movement observed. The cause of this behavior is the fact that every robot will try to get as close as the population's center of mass as possible. Since the center of mass is not an environmental cue (and therefore the robots are unable to detect it), the robots will first form groups with the closest robots that will merge with other groups and so forth, until a single cluster is formed. The robots do not have the notion of group size, so in order to ensure that a single cluster is formed, the movement observed in clustered robots is of great help. The behavior described is also important for the scalability of the controllers evolved: the robots will be always searching for other robots while maintaining the cluster so there will be no limitation in terms of number of robots that can aggregate

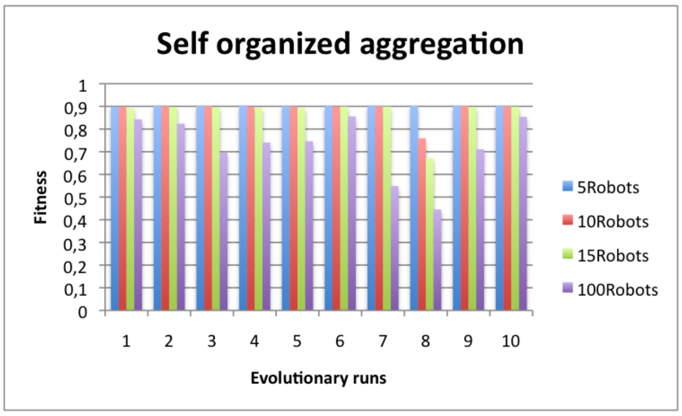
Scalability

To study scalability, and confirm that the solution achieved to perform self-organized aggregation is highly scalable, the best generation of each evolutionary run was tested with 5, 10, 15 and 100, 10 samples each, and the results is the average of the fitness values obtained.

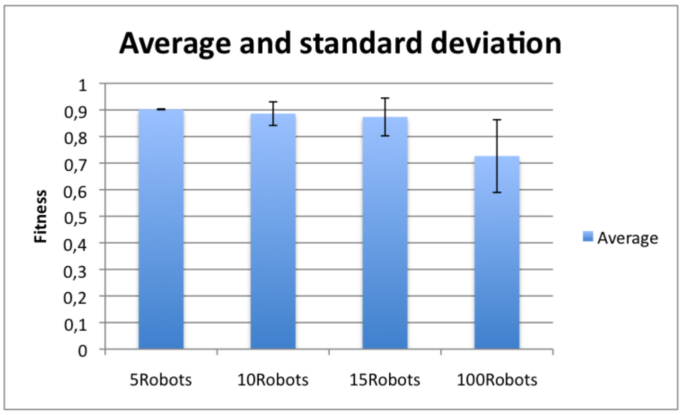
Figure 4.2(a) compares the fitness values achieved by all 10 evolutionary runs when tested with 5, 10, 15 and 100 robots and figure 4.2(b) shows the average fitness value and standard deviation.

With populations with 5, 10 and 15 robots, every controller achieved fitness values close to 0.9, which means the solution achieved by evolution using 5 robots can aggregate different sized groups of robots.

To study how the controller would handle extremely large groups of robots, a test with a population of 100 robots was performed 10 times on the last generation of each evolutionary run. The results show a drop on the fitness values, which can due to the cluster size: the bigger the population, the bigger the cluster diameter, so the robots at the outer parts of the cluster, despite they are in the cluster, they are far from the center of mass thus reducing the final fitness value. Another reason is the time the robots take to form the cluster when the population is large: the cluster formation starts with small clusters that merge until a single cluster is formed. The larger the population, the more time it will take to form a single cluster, thus reducing the final fitness value also.



(a) Scalability comparison



(b) Average and standard deviation

Figure 4.2: Scalability

4.1.4 Discussion

In this chapter, self-organized aggregation behaviors were evolved, evaluated and tested. The results show that with the evaluation function and the robot setup used, controllers can be successfully evolved to perform self-organized aggregation. The controllers produced proved to be efficient and scalable.

However, some issues can be pointed out. The evaluation function does not take in consideration the diameter of the final cluster, thus the final fitness values of large populations will not be as accurate as with small populations. During my work, I tried to mitigate this problem, without success, by developing a formula where the center of mass would be a circle instead of a point in the arena. That way, every robot within the center of mass circle diameter would have a distance value of 0 and thus maximizing the fitness value. As future work, developing a more precise way to score large populations performing self-organized aggregation will be, in my opinion, very useful. The fact the robots have a very large sensor range can also bias the results: the robots don't need to find their neighbors as they are all within sensor range. The time the experiments take to be completed was a limiting factor that made me make the decision of not study self-organized aggregation any further. However, a relation between fitness achieved and sensor range could be a good way to continue the study of self-organized aggregation.

My next step was to study the evolution of collective choice behaviors. How will evolution develop controllers capable to choose a landmark and aggregate on it? The next section describes how collective choice behaviors were evolved and the strategies that emerged.

Chapter 5

Collective choice

5.1 Collective choice without communication

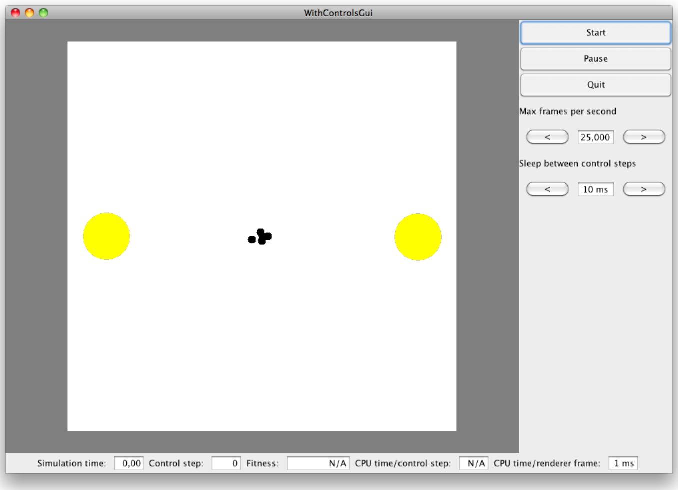
The next experiment aims to evolve collective choice behaviors. Collective choice behaviors are present in several societies, such as insect societies, and those behaviors are the main source of decision. In fact, the strength of this kind of societies relies on the fact that the group interest is always above the individual interest. Members of this kind of societies often have very limited cognitive and communication capabilities, so decisions have to be made collectively. To do so, societies have to develop some strategies to explore the environment and make decisions accordingly like if the society was, in fact, a single organism.

In this set of experiments, I evolved some collective choice behaviors, based on the way a group of cockroaches, which are limited in terms of cognitive and communication capabilities, can collectively choose a shelter [10]. The robots must cluster in one of the landmarks placed in the arena. The sensor range is very limited, so the robots must adopt a strategy to first find a landmark and then cluster inside.

5.1.1 Experimental setup

For this experiment, I used a population of 5 differential drive robots, with 8 robot proximity sensors, and 8 landmark sensors. The opening angle of the sensors is 135° and the sensor range is 5.0 meters for the proximity sensors and 0.1 meters for the landmark sensors, which means that the robots will not detect the landmarks when the simulation starts, but they will have to actively search for them. 2 landmarks are used, with 0.30 meters of diameter and at 2.0 meters from each other. Fig 5.1(a) shows how the landmarks were placed to perform the evolution. The number and position of the landmarks never changes during the simulation, however, to study scalability, a 5-landmark environment is also used. In this case, the landmarks are placed in a circle like shown in fig. 5.1(b) 10 evolutionary runs were performed with this setup with different initial random seeds for 50 generations each. The fitness of each genome was sampled 4 times in trials of 2 minutes of virtual time (1200 control steps) each.

In each sample, the robots are placed at the center of the environment so a robot will never begin close, or even inside a landmark, which would over simplify the experiment since a landmark could already be found by a robot before the beginning of the experiment. The bias of the sensors and the way the individual position of the robots in the initial cluster varies in each sample.

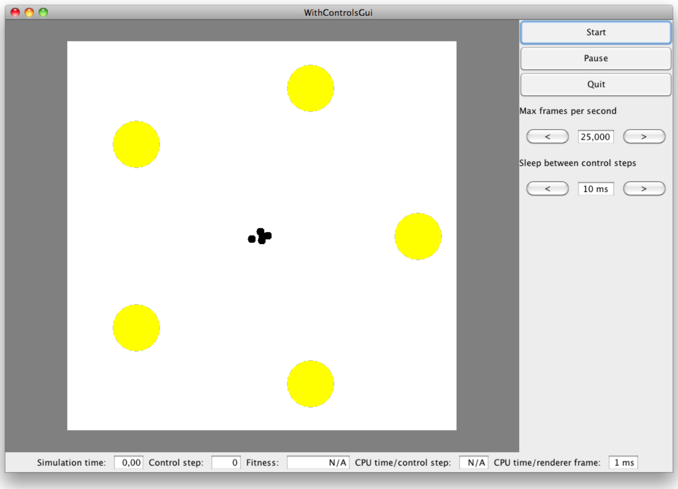


Max frames per second 25,000

Sleep between control steps 10 ms

Simulation time: 0,00 Control step: 0 Fitness: N/A CPU time/control step: N/A CPU time/renderer frame: 1 ms

(a) Simulation environment used to perform evolution



Max frames per second 25,000

Sleep between control steps 10 ms

Simulation time: 0,00 Control step: 0 Fitness: N/A CPU time/control step: N/A CPU time/renderer frame: 1 ms

(b) The 5 landmark environment

Figure 5.1: Environments

5.1.2 Evaluation function

To evaluate the controllers produced, I developed an evaluation function that first calculates the nearest landmark from the population's center of mass and then averages the distance between each robot and the landmark calculated before. The final fitness value of a chromosome will be average of the fitness values obtained in each sample.

$$f_s(t) = \sum_{t=0}^{t} \frac{\frac{1}{n}\sum_{i=1}^{n}\left(1-\frac{d_i(t)}{\delta}\right)}{\omega} \tag{5.1}$$

where f_s represents the fitness of a sample, n is the number of robots used, t is the number of steps of each generation, $d_i(t)$ is the distance of the i^{th} robot to the nearest landmark of the population's center of mass, δ is the sensor range of the proximity sensor and ω is the number of steps per run. The distance $d_i(t)$ varies between 0 and the maximum range of the sensor used. The fitness is calculated at group level and varies between [0, 1].

5.1.3 Results

In all 10 performed evolutionary runs of the experiment, the robots managed to find a landmark and aggregate in it in a very efficient way: as we can observe in fig 5.2, which shows the fitness trajectories of all evolutionary runs, the values obtained are close to 0.9 which represents a quick aggregation. The fitness trajectories also show that an optimal controller was reached early on the evolution process.

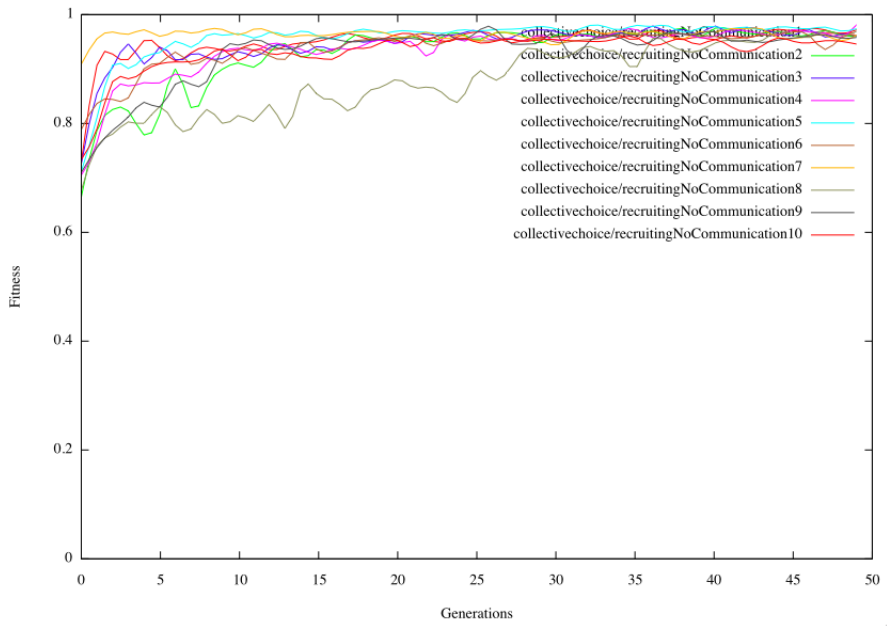
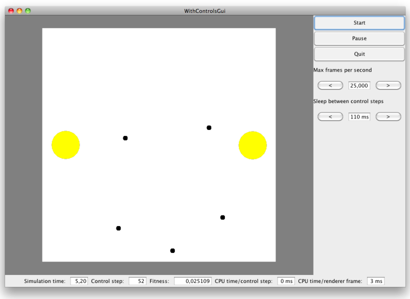


Figure 5.2: fitness trajectories of all 10 evolutionary runs

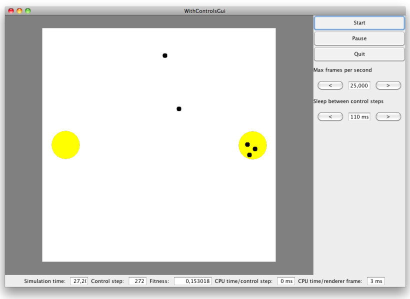
The strategies adopted by the populations to find and aggregate inside a landmark are very simple but effective. I was able to distinguish 2 different strategies, which I will call *Following* and *Scouting*.

In the *Following* strategy, all the robots will split the initial group and start to move in circles, with a variable radius, until a landmark is found by one of them. When a robot finds a landmark, it will stop. This will create a chain reaction: the robot right behind the one who found the landmark will reach the landmark and stop and so forth until all robots are inside the landmark. Figures 5.3(a), 5.3(b) and 5.3(c) show the process of finding a landmark and clustering in it.



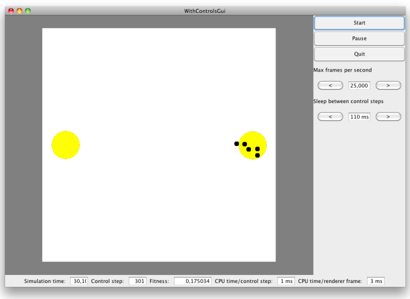
Max frames per second: 25,000
Sleep between control steps: 110 ms
Simulation time: 5,20 Control step: 52 Fitness: 0,025109 CPU time/control step: 0 ms CPU time/renderer frame: 3 ms

(a) The robots move in a circle



Max frames per second: 25,000
Sleep between control steps: 110 ms
Simulation time: 27,2(Control step: 272 Fitness: 0,153018 CPU time/control step: 0 ms CPU time/renderer frame: 3 ms

(b) The landmark is found

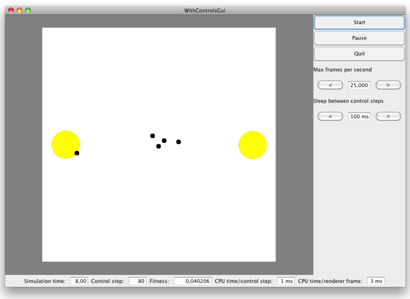


Max frames per second: 25,000
Sleep between control steps: 110 ms
Simulation time: 30,1(Control step: 301 Fitness: 0,175034 CPU time/control step: 1 ms CPU time/renderer frame: 3 ms

(c) The robots cluster

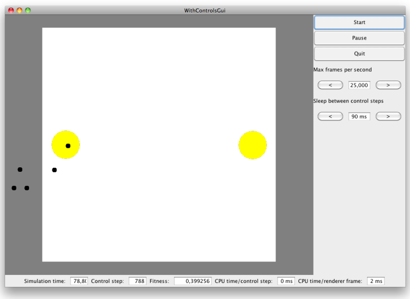
Figure 5.3: *Following* strategy

In the *Scouting* strategy, only a small number of robots will leave the initial cluster and start to move in circles. The remaining robots will move also, clustered, according to the scout robots positions: the group will never be too far away from the scouts and the scouts will, from time to time, join the cluster. If a robot finds a landmark, it will stay inside, thus pulling the other robots inside too. Figures 5.4(a), 5.4(b) and 5.4(c) show the process of finding a landmark and clustering in it.



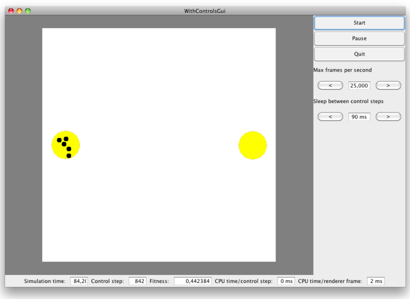
Max frames per second 25,000
Sleep between control steps 100 ms
Simulation time: 8,00 Control step: 80 Fitness: 0,040206 CPU time/control step: 1 ms CPU time/renderer frame: 3 ms

(a) The scout finds the landmark



Max frames per second 25,000
Sleep between control steps 90 ms
Simulation time: 78,8(Control step: 788 Fitness: 0,399256 CPU time/control step: 0 ms CPU time/renderer frame: 2 ms

(b) The remaining robots follow



Max frames per second 25,000
Sleep between control steps 90 ms
Simulation time: 84,2(Control step: 842 Fitness: 0,442384 CPU time/control step: 0 ms CPU time/renderer frame: 2 ms

(c) The robots cluster

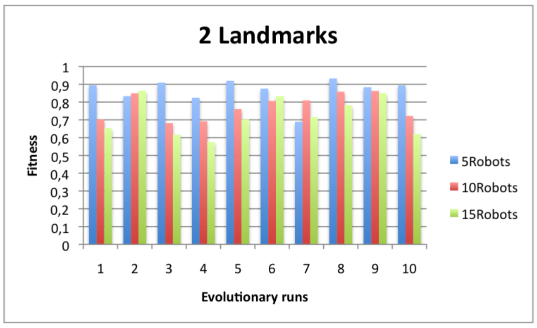
Figure 5.4: *Scouting* strategy

Scalability

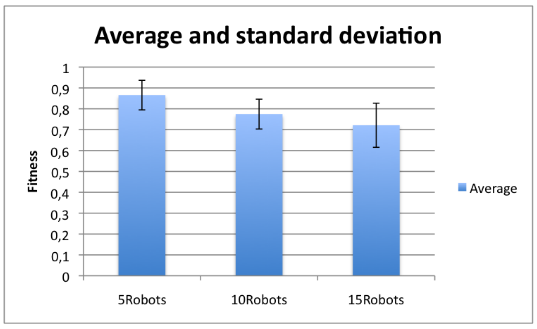
To study scalability, the controllers achieved by each evolutionary run were tested the 2 and 5 landmark environments, each one tested with 5, 10 and 15 robots, in a total of 6 tests. Each test was performed 10 times, with different initial seed numbers and the results are the average fitness values obtained.

The first scalability test shows the behavior of the controllers in an environment with 2 landmarks. Figure 5.5(a) and figure 5.5(b) shows a significant drop of the achieved fitness value of the controllers when the number of robots is increased. This drop is caused when the robots find the 2 landmarks and form 2 different clusters or when the population takes long to find a landmark and cluster. However, when just 5 robots are used, they managed to find and cluster inside a landmark in all of the evolutionary runs. By observing the robot's strategies, I concluded that the controllers who developed a *scouting* strategy scale better when the number of robots is increased. evolutionary runs 2, 6 and 8 are the ones who developed a more clear *scouting* behavior and therefore the ones with better overall results.

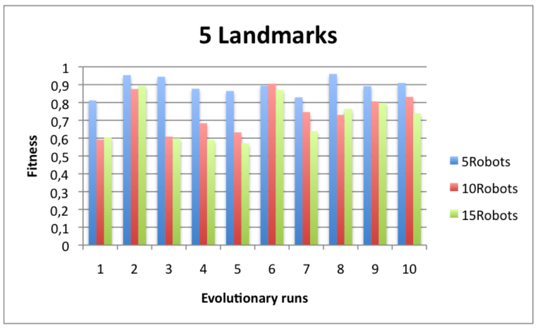
The scalability test with 5 landmarks obtained similar results as we can observe in figures 5.5(c) and 5.5(d). Except the controller on evolutionary runs 1 and 5, all controllers managed to find and cluster in one of the 5 landmarks of the environment. In terms of number of robots, scalability is slightly worse than in the 2-landmark environment, however, evolutionary runs 2, 6, 8 and 10 managed to find and cluster in a landmark efficiently. Once again, those controllers show a *scouting* strategy, proving once again its superiority in terms of scalability.



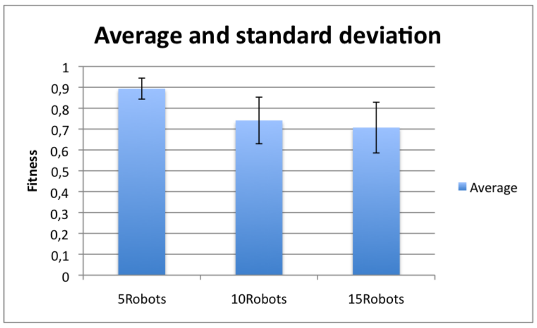
(a) 2 landmarks with 5, 10 and 15 robots



(b) Average and standard deviation, 2 landmarks



(c) 5 landmarks with 5, 10 and 15 robots



(d) Average and standard deviation, 5 landmarks

Figure 5.5: Average and standard deviation comparison

5.1.4 Discussion

In this set of experiments, some collective choice behaviors were observed. The robots adopted 2 main strategies, *Following* and *Scouting*, to successfully find a landmark and aggregate in it. The *Scouting* strategy seems to be superior. This can be explained by the fact that only a limited number of robots will leave the initial cluster, where they begin the simulation, to search for a landmark. On the other hand, the *Following* strategy relies on brute force to find the landmarks, which means spreading the population as much as possible, hoping some robot will find a landmark. However, when the populations are bigger, several robots can find different landmarks, thus forming a cluster per landmark found.

The results presented in this experiment showed that between 30% and 40% of the evolutionary runs developed a *Scouting* strategy, but more evolutionary runs would be necessary to verify this success rate. However, this study shows that it is possible to evolve a controller general enough to solve other problems than the ones faced during evolution. As future work, an evaluation function capable to raise the percentage of controllers showing a *Scouting* strategy would be interesting.

Insects capable of making a collective choice differ in a very important aspect from the robots used to perform these experiments: they can communicate. So the questions raised when analyzed these results are: will communication between robots emerge? And if so, will it improve the results?

5.2 Collective choice with communication

The purpose of this experiment is to study the emergence of communication and its influence of in the collective choice behaviors. The robots now have a channel to communicate, with a color actuator that can take any color in the RGB spectrum, which other robots will detect.

In this experiment, evolution will create strategies to choose a landmark and also a language the robots will use to communicate. Since communication is not the main goal, the controllers will not be scored based on the way communication is done, but rather if they can accomplish the task or not, which is to collectively choose a landmark and cluster in it. The configuration of the robots used in this experiment is similar to the setup used in section 5.1: the only difference is the usage of a color sensor and actuator as a communication means.

The results of this experiment will be compared to the results obtained in the experiment showed on section 5.1 and some conclusions about the usefulness of communication in this particular problem will be drawn.

5.2.1 Experimental setup

For this experiment, I used a population of 5 differential drive robots, with 8 color sensors, 8 landmark sensors and 8 robot proximity sensors. The opening angle of the sensors is 135° and the sensor range is 5.0 meters for the color sensors and robot proximity sensors and 0.1 meters for the landmark sensors. An additional actuator is also used to make the robot change colors. The color actuator can take any color in the RGB spectrum and the color will be used to communicate. 10 evolutionary runs were performed with this setup with different initial random seeds for 50 generations each. The fitness of each genome was sampled 4 times in trials of 2 minutes of virtual time (1200 control steps) each. 2 landmarks were used, with 0,30 meters of diameter and at 2.0 meters from each other. The number and position of the landmarks never changes during the simulation, however, to study the scalability of the controllers produced, a 5-landmark environment is also used. In this case, the landmarks are placed in a circle like shown in fig. 5.8(b). The bias of the sensors and the way the individual position of the robots in the initial cluster varies in each step.

The evaluation function used is the same used in section's 5.1 experiment, so the communication itself will not contribute to the overall fitness.

5.2.2 Results

Fig 5.6 shows the fitness trajectory of all 10 evolutionary runs. As we can observe, like in the experiment described in section 5.1, controllers able to perform the task emerged in early generations and by the last generations, the fitness values reached values near 1.

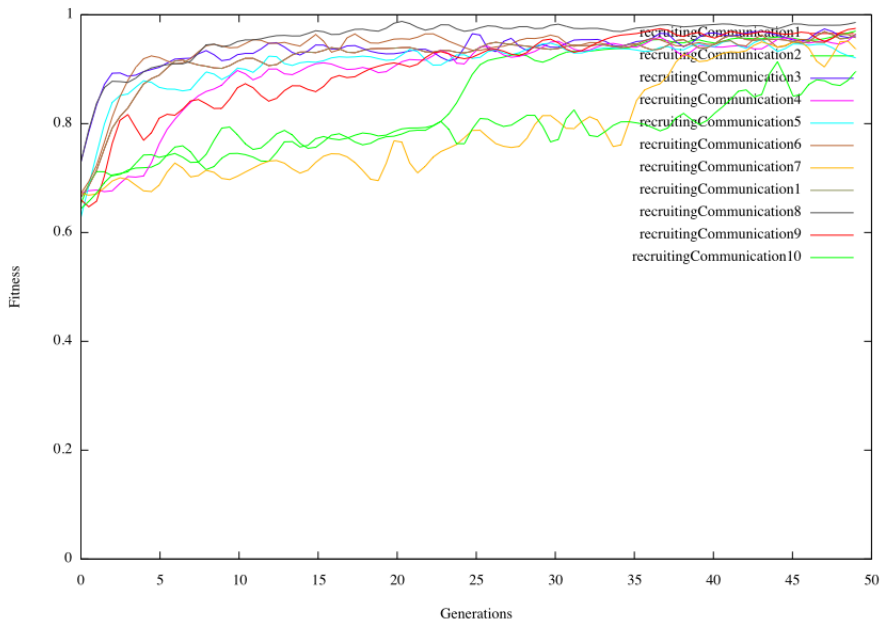
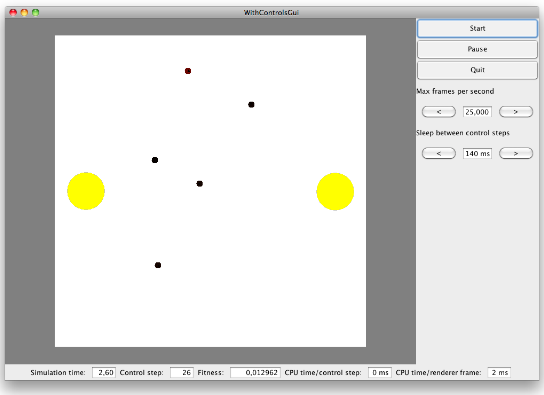


Figure 5.6: fitness trajectory of all 10 evolutionary runs

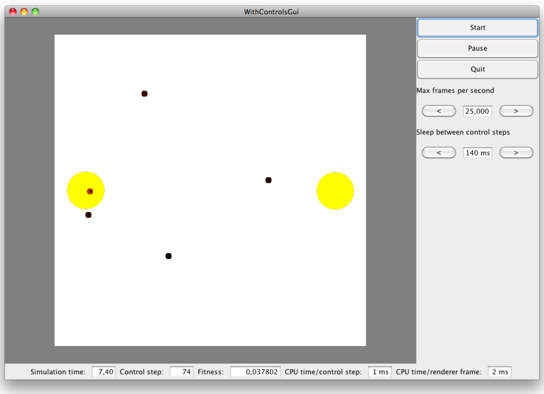
The main difference between the results in this experiment and the experiment described in section 5.1 is in the strategies developed. The *Scouting* and *Following* behaviors not always emerged, instead the robots had more erratic behaviors. The reason for this phenomenon is in the fact that some generations evolved a form of communication, as we can observe in figure group 5.7, to notify the other robots when they find a landmark, so strategies based on predictable movements, like *Following* and *Scouting* are less probable to emerge.



Max frames per second 25,000
Sleep between control steps 140 ms

Simulation time: 2,60 Control step: 26 Fitness: 0,012962 CPU time/control step: 0 ms CPU time/renderer frame: 2 ms

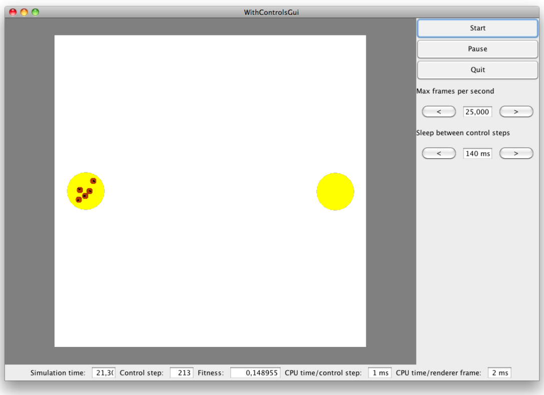
(a) The robots search for a landmark



Max frames per second 25,000
Sleep between control steps 140 ms

Simulation time: 7,40 Control step: 74 Fitness: 0,037802 CPU time/control step: 1 ms CPU time/renderer frame: 2 ms

(b) The landmark is found



Max frames per second 25,000
Sleep between control steps 140 ms

Simulation time: 21,3(Control step: 213 Fitness: 0,148955 CPU time/control step: 1 ms CPU time/renderer frame: 2 ms

(c) The robots cluster

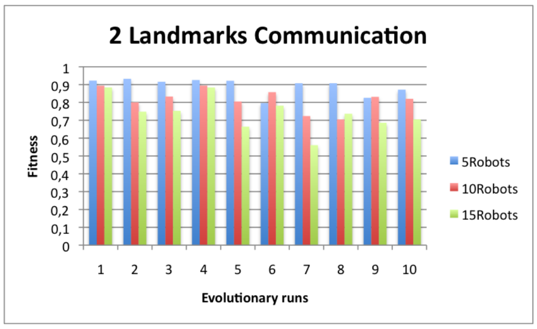
Figure 5.7: Robots communicating to find a landmark

evolutionary runs 5, 6, 7 and 10 did not develop a communication protocol. Since the color sensor works like the proximity sensor when it detects a color in another robot, some evolutionary runs adopted strategies similar to the ones adopted by previous experiment, where the robots couldn't communicate by changing colors.

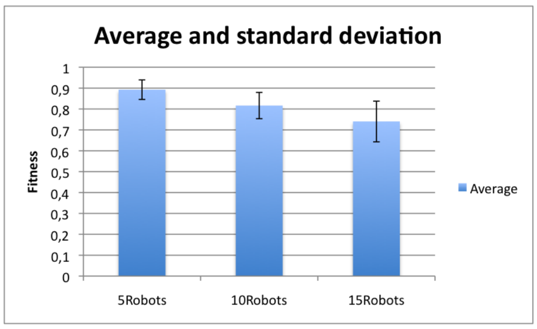
The meaning of some color changes observed on the robots is sometimes hard to determine. The robots can create a communication protocol with the entire RGB spectrum, thus having nearly an infinite number of colors to use.

Scalability

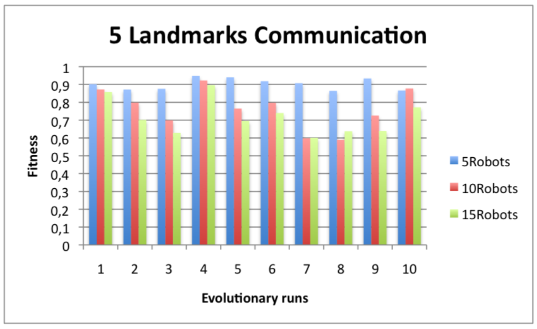
Like in the experiment described in section 5.1, the controllers achieved by each evolutionary run were tested the 2 and 5 landmark environments, each one tested with 5, 10 and 15 robots, in a total of 6 tests. Each test was performed 10 times, with different initial seed numbers and the results are the average fitness values obtained. The results obtained are shown in figure 5.8(a, b, c and d).



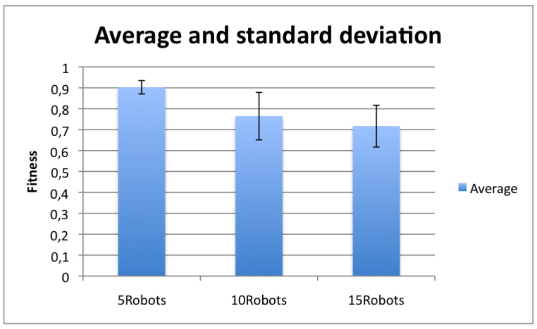
(a) 2 landmarks with 5, 10 and 15 robots



(b) Average and standard deviation, 2 landmarks communication



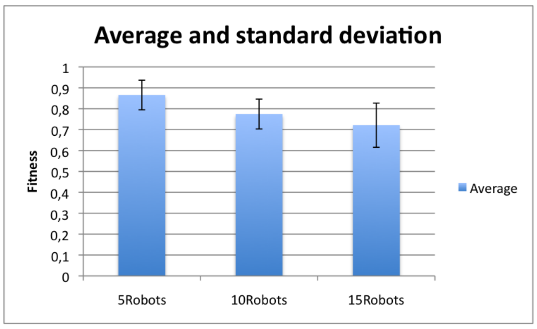
(c) 5 landmarks with 5, 10 and 15 robots



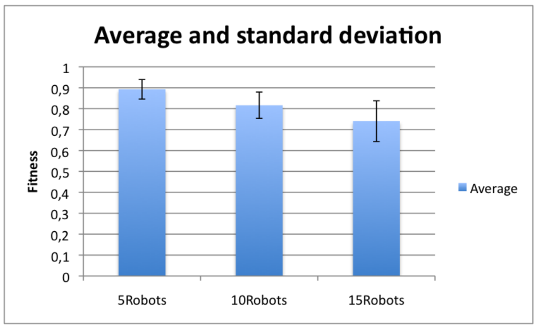
(d) Average and standard deviation, 5 landmarks communication

Figure 5.8: Scalability study

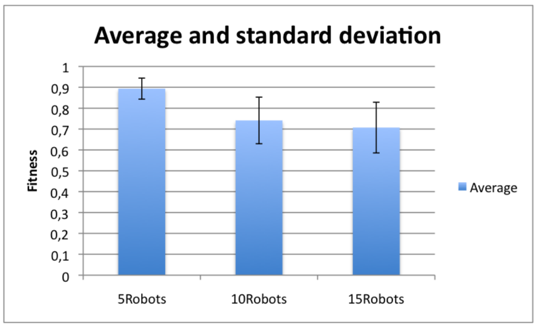
As we can observe in the result comparison between populations with and without the color actuator (figures 5.9 (a, b, c, and d)), the overall results with 2 landmarks and 5 robots are higher than the ones achieved with robots unable to communicate through colors. Results achieved with 10 and 15 robots are also higher in general. This means that communication helped the controllers to be more efficient to perform the task and made them also more scalable in terms of population size.



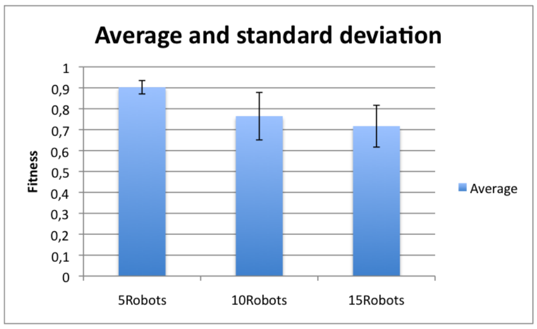
(a) Average and standard deviation, 2 landmarks no communication



(b) Average and standard deviation, 2 landmarks communication



(c) Average and standard deviation, 5 landmarks no communication



(d) Average and standard deviation, 5 landmarks communication

Figure 5.9: Result comparison between populations with and without the color actuator

Unlike the previous experiment, in section 5.1,the *Following* and *Scouting* behaviors are not clearly observable in most of the evolutionary runs. However, when communication protocols did emerge, some populations showed a very clear *Scouting* behavior. evolutionary runs 1 and 4 are a nice example of this phenomenon, and can scale remarkably well: the robots move as a flock, while one or two robots orbit around the main group until a landmark is found. When the landmark is found, the scout robots will stop inside and the remaining robots will move as a flock until they are inside the landmark. Communication can be observed clearly as well: the scout robots change color when they leave the cluster and start to search for a landmark. When the scout robots find a landmark, they change to a different color again thus calling the rest of the group. Since the robots do not scatter around the environment, the probability of two robots to find two different landmarks and therefore form two different groups is low. However, in evolutionary run 4, when such scenario occurs, the main group will choose to join the nearest scout, forcing the remaining one to leave the landmark and join the group.

When tested with 5 landmarks, the overall results are better than the ones obtained in the previous experiment (section 5.1). evolutionary runs 1 and 4 are the ones with better overall results, and are also the ones who developed communication protocols and a *Scouting* behavior.

5.2.3 Discussion

This set of experiments aimed to study if communication protocols would emerge and if so, to study if the robots were more efficient to find a landmark and form a single cluster on it than robots without communication capabilities. The results showed that populations where robots are able to communicate will not always develop a communication protocol. Communication itself does not make a population perform better in the task: the movement coordination the population develops is also of great importance. This experiment showed some populations capable of communicate and with poor performance, like on evolutionary run 3, and populations that did not develop communication protocols having good performances, like evolutionary run 10. The most successful populations are the ones that efficiently use both communication and movement coordination. Evolutionary run 4 developed a communication protocol and also a *Scouting* behavior, and it can perform well in both 2 and 5 landmark environments, with 5, 10 and 15 robots.

The simple fact of the robots being able to detect each other is a form of communication and that might be a reason for communication protocols to not have emerged in some evolutionary runs of this experiment. However, communication and movement coordination proved to be a powerful combination for controllers to perform collective choice behaviors that are general enough to work well under different setups than the ones used during evolution. Nevertheless, only 1 out of 10 evolutionary runs developed such combination of communication and movement coordination (evolutionary run 1 developed a similar strategy but not as efficient and as scalable). Overall, populations able to communicate through colors performed slightly better than the populations without that capability.

Chapter 6

Energy management

6.1 Energy management without communication

Every society is dependent of the resources available. In order to endure, societies have to find a balance between the ratio the resources are produced and the ratio they are consumed. This is often a non-trivial task and it can raise complex morality questions. Garret Harding extended and popularized William Forster Lloyd's concept known as *The tragedy of the commons* in an article[19], and proved that, as it is impossible to win a Tic Tac Toe match against an opponent that completely understands the game, the efficient management of resources by a growing population of selfish agents is also impossible. The example given in the article shows how a profitable situation for an individual can be harmful for the society.

In a hypothetical society of robots, energy would have to be acquired from an energy source and that energy source could be finite. In order for the society to survive, an efficient management of this resource would be of major importance. The question we address in this section is the possibility for a resource management strategy to emerge from artificially evolved controllers. To do so, I created an environment containing energy sources, which the robots will use to recharge themselves by approaching them. Those energy sources are, however, limited: only a limited number of robots can be recharged simultaneously by each energy source or else no robot will receive energy.

6.1.1 Experimental setup

For this experiment, I used a population of 5 differential drive robots, with 8 robot proximity sensors, 8 energy source sensors and an internal energy sensor for the robots to know their current energy level. The opening angle of the proximity and energy source sensors is $135°$ and the sensor range is 5.0 meters for both. 2 energy sources are used, with 0.30 meters of diameter and at 1.5 meters from each other. Each energy source can recharge up to 2 robots simultaneously. However, if more robots are inside the energy source, the recharging process will stop and the robots will start to consume energy as if they were outside the energy source.

10 evolutionary runs were performed with this setup with different initial random seeds for 100 generations each. The fitness of each genome was sampled 4 times in trials of 2 minutes of virtual time (1200 control steps) each. The bias of the sensors and the individual placement of the robots in the initial cluster vary in each sample.

The robots will begin with 800 energy units, consume 1 unit of energy per control step and gain 2 units of energy per control step when they are recharging. If a robot reaches an energy level of 0, it will stop and it will not be possible to recharge it. On the other hand, if a robot reaches the maximum energy value, it will simply stop charging, even if the robot is still inside the energy source.

In each sample, the robots are placed at the center of the environment so a robot will never begin close, or even inside a landmark, thus biasing the experiment.

6.1.2 Evaluation function

The main goal of this experiment is to study the strategies developed to make the entire population survive as long as possible, so solutions like maximizing the average energy level of the robots or simply maximize the number of robots with energy values above 0 will simply not work: the most simple solution will be to sacrifice a robot or make the robots live just the 1200 control steps, thus making the solution not scalable in time. To evaluate the controllers produced, I developed an evaluation function, which simply takes in consideration the energy value of the robot with the lower value and the number of robots alive at the end of the sample:

$$f_s = \delta * 0,9 + \left(\frac{\gamma}{n}\right) * 0.1 \tag{6.1}$$

where f_s represents the fitness of a sample, δ is the energy value of the robot with the lowest energy value at the last control step, γ represents the number of robots with an energy value above 0 and n is the number of robots. The final fitness value of a chromosome will be the average value the weakest robot had at the end of each sample.

6.1.3 Results

Figure 6.1 shows the fitness trajectory of each evolutionary run. The fitness values represent the energy level of the weakest robot in the population, so we can conclude that in all evolutionary runs the controllers developed a strategy to manage the usage of the energy landmarks. The fitness trajectory also shows that in all evolutionary runs, the populations found an optimal solution very early, since after generation 20 the fitness values of all evolutionary runs are near 1, which is the maximum value possible.

There is no significant difference among the strategies adopted by the controllers of each evolutionary run: the robots first split in two groups (a group of 2 and a group of 3) and head for an energy landmark. The group with 3 robots starts to move in circles in the border of the energy landmark (see fig. 6.2), so the three robots are never simultaneously inside.

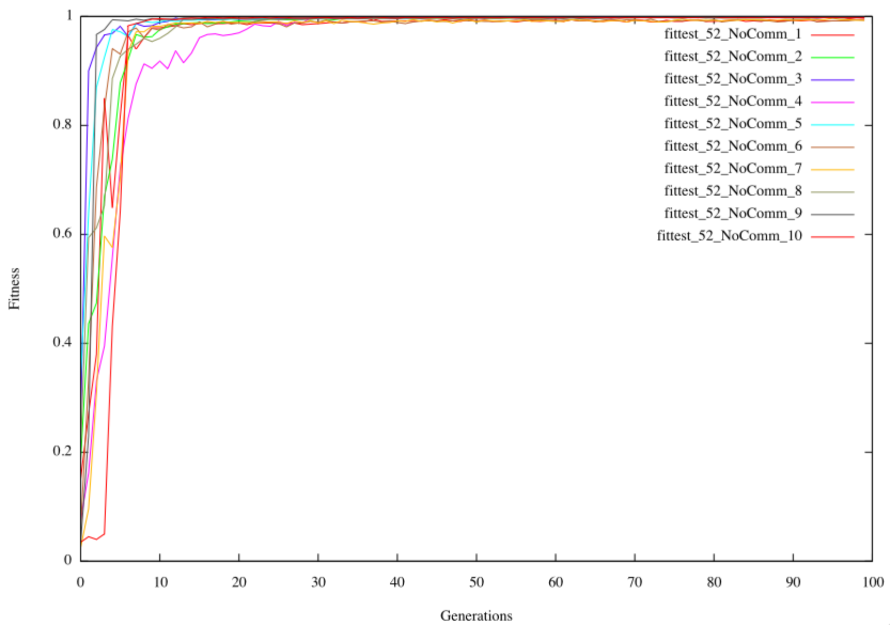
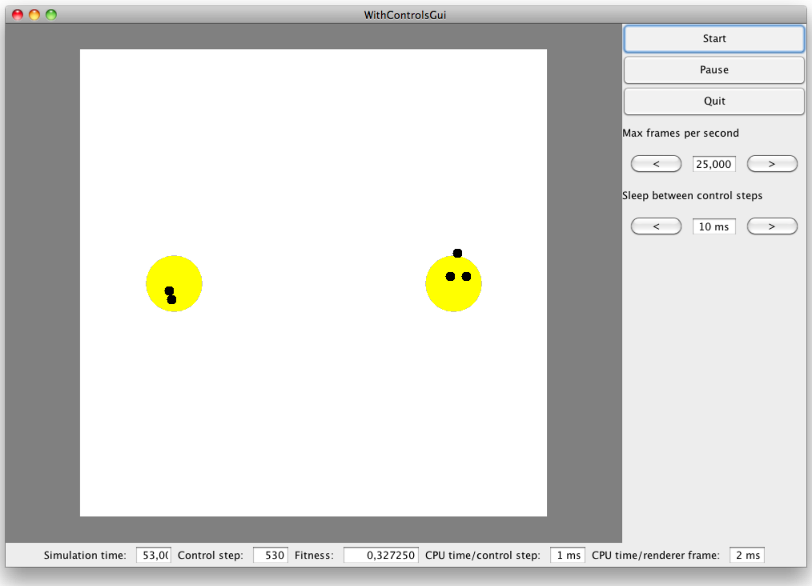


Figure 6.1: fitness trajectory of all 10 evolutionary runs



Max frames per second 25,000

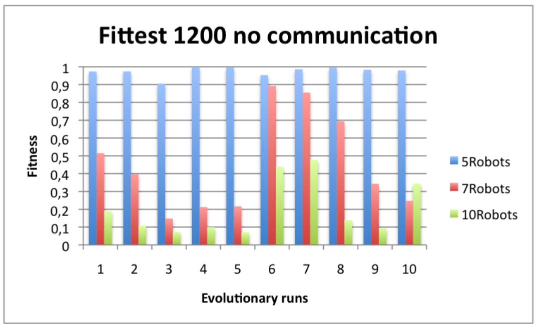
Sleep between control steps 10 ms

Simulation time: 53,0(Control step: 530 Fitness: 0,327250 CPU time/control step: 1 ms CPU time/renderer frame: 2 ms

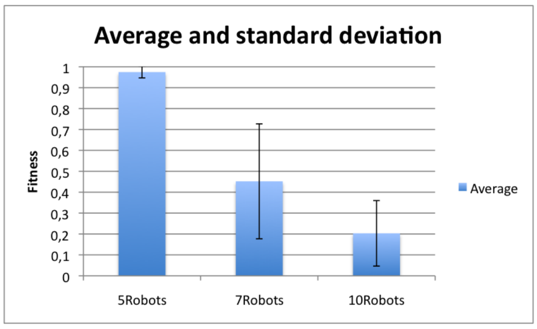
Figure 6.2: Energy management

Scalability

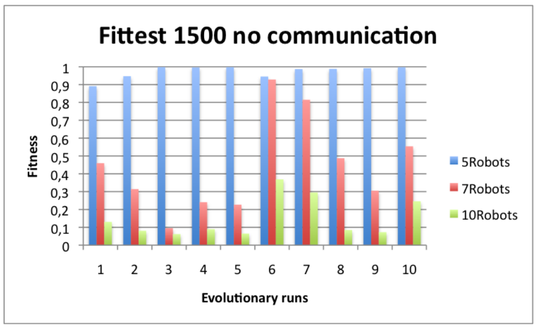
To study scalability, 10 samples of the last generation of each evolutionary run were analyzed. For each evolutionary run, populations of 5, 7 and 10 robots were used, during 1200, 1500 and 2000 control steps. The results are shown in figure group 6.3. .



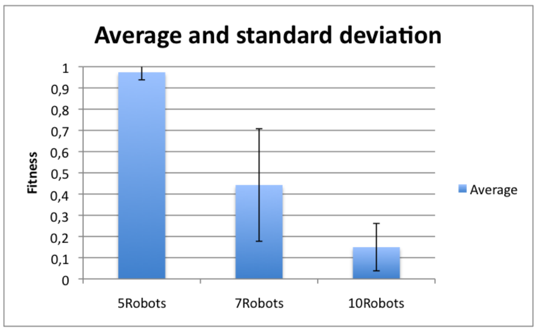
(a) 1200 steps



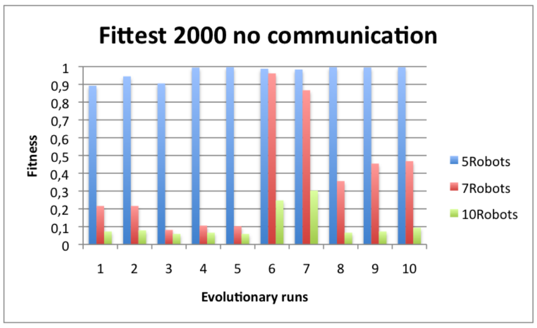
(b) Average and standard deviation



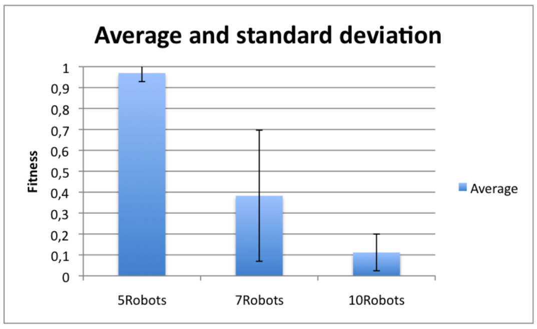
(c) 1500 steps



(d) Average and standard deviation



(e) 2000 steps



(f) Average and standard deviation

Figure 6.3: Scalability study

As we can observe, populations with 5 robots can scale well in terms of time, which means the controllers evolved strategies to make the entire population to survive as long as possible and not just the 1200 control steps they faced during evolution. However, in terms of population size, the controllers were not as successful. The fitness drop was expected, because there are more robots to recharge. The percentage of populations that did not lose any member until the end of the simulation was 100% for 1200 steps simulations, 90% for 1500 steps simulations and 70% for 2000 steps simulations. When the number of robots is increased to 10 the percentage dropped to 60% for 1200 steps simulations, 40% for 1500 steps simulations and 20% for 2000 steps simulations.

Both time and population size seemed to have influence in the survival rate of the robots. Nevertheless, evolutionary runs 6 and 7 developed controllers able to manage the energy landmarks so that a 100% survival rate of the robots could be achieved even in the hardest scenario, 2000 control steps simulations and 10 robots.

The strategy that emerged to manage bigger populations with the same amount of energy is just to make the robots rotate faster in the border of the energy landmarks. This simple behavior is the only possible behavior the robots could develop, since they have no way to communicate to nearby robots their energy level. When the energy level reaches a critical value, the robots will simply force their entrance in the energy landmark, thus forcing someone out.

6.1.4 Discussion

In this experiment, robots with a limited quantity of energy were deployed in an environment where, in order to survive, they would have to learn how to manage the 2 energy sources available in the most efficient way. Results showed that such is possible: the strategies that emerged not only allowed the populations to survive during the 1200 steps used during evolution, but also allowed to extend their lifespan beyond that. In some evolutionary runs the controllers were even able to maintain the lowest level of energy of the weakest robot above 50%. However, when the population grows, the solutions achieved are not able to manage the energy landmarks with the same degree of efficiency. The fitness drop observed when the number of robots is increased was expected: since there are more robots to recharge and there is a limited number of robots that can be recharged at a time, the robots will have to spend more time waiting for their turn to recharge and the charging times will be shorter, therefore the energy levels will have to be lower. Despite this fact, evolutionary runs 6 and 7 showed that an efficient management of the energy landmarks with a larger group of robots and for a longer period of time is possible. Evolutionary runs 6 and 7 showed that, in a group of 10 robots, the robot with the lowest energy level had between 30% and 40% of its total energy, and that level is even higher when the simulation time is 1200 steps. This means that the solutions achieved have even more scalability potential.

This experiment raises a question: how can we produce more controllers able to scale as well as the ones produced in evolutionary runs 6 and 7? In the next experiment, the robots will be able to explicitly communicate by a color actuator. The results will show whether this communication ability will have influence in the performance of the controllers produced.

6.2 Energy management with communication

The aim of this last experiment is to study the emergence of communication and its influence in the energy management problem. To do so, the robots were equipped with a color actuator that can take the entire RGB color spectrum and color sensors to detect color changes in nearby robots.

Like in section 5.2, evolution will develop strategies for the robots to manage the energy landmarks in an efficient way, so the population survives as long as possible. Since the robots now have a communication channel (the color actuator), some communication protocols may emerge to help to achieve the goal. However, like in section 5.2, communication protocols will not be evaluated, but rather how the population can manage the energy landmarks in an efficient way. For this experiment, the setup of the robots will be similar to what was used in section 6.1, with the exception that in this case the robots will use a color sensor and a color actuator as a communication mean.

The results of this experiment will be compared to the results obtained in 6.1 and some conclusions about the emergence of communication protocols and its influence in the task will be drawn.

6.2.1 Experimental setup

For this experiment, a population of 5 robots is used. Each robot is equipped with 8 robot proximity sensors, 8 color sensors, 8 energy landmark sensors and an internal energy sensor. The opening angle for the sensors is 135 and all the external sensors have a range is 5.0 meters. A color actuator is also used in each robot to make them change colors. The actuator can take any color in the RGB spectrum and it will be used as a communication form.

2 energy sources are used, with 0,30 meters of diameter and at 1.5 meters from each other. Each energy source can recharge up to 2 robots simultaneously. 10 evolutionary runs were performed with this setup with different initial random seeds for 100 generations each. The fitness of each genome was sampled 4 times in trials of 2 minutes of virtual time (1200 control steps) each. The bias of the neural network and the individual position of each robot in the initial cluster varies in each sample.

The robots will begin with 800 energy units, consume 1 unit of energy per control step and gain 2 units of energy per control step when they are recharging. If a robot reaches an energy level of 0, it will stop and it will not be possible to recharge it. On the other hand, if a robot reaches the maximum energy value, it will simply stop charging, even if the robot is still inside the energy source. In each sample, the robots are placed at the center of the environment so a robot will never begin close, or even inside a landmark, thus biasing the experiment.

The evaluation function used is the same used in section 6.1, so communication will not contribute to the overall fitness value directly.

6.2.2 Results

Figure 6.4 shows the fitness trajectory of all 10 evolutionary runs. Once again, the fitness values represent the energy value of the weakest robot in the population, so we can conclude that by generation 70 approximately, the entire population survived the entire 1200 control steps in every evolutionary run. However, we can observe an important difference between this experiment and the previous experiment, where the robots did not have the color actuator: populations took significantly more generations to learn how to manage the energy landmarks efficiently. This fact is due to the new communication capabilities: the robots now have a new signal to interpret (the color change of nearby robots) and they will have to learn how to take advantage of it. However, and like Maynard Smith said "*It's not good making a signal unless it is understood and a signal will not be understood the first time it is made*" [27], so the robots will start to understand the signals they receive and to explicitly send understandable signals by the result of a fluke. In fact, in evolutionary runs 3, 5 and 6 the robots did not use the color actuator to communicate. evolutionary run 3 also shows a significantly worse performance when compared to the remaining ones. This can indicate that when an explicit means of communication is available, the chances that evolution does not achieve an optimal solution within the limit number of generations increase. However, 10 evolutionary runs are not enough to conclude this with an acceptable degree of certainty.

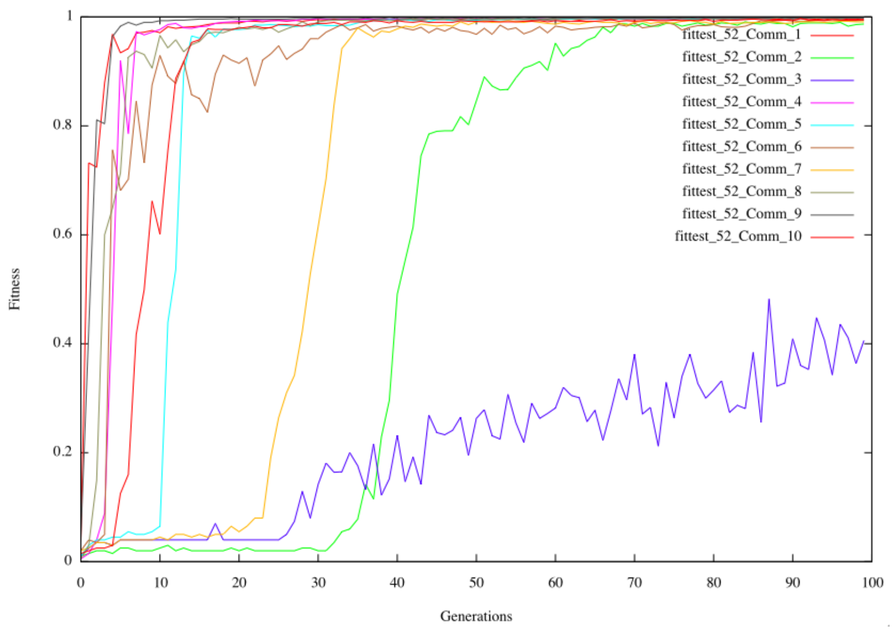
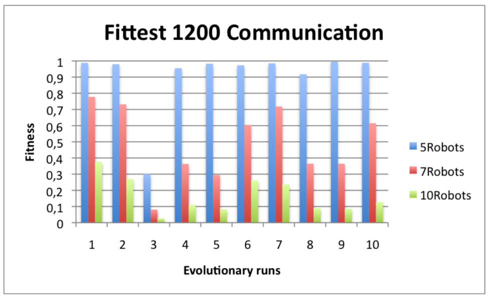


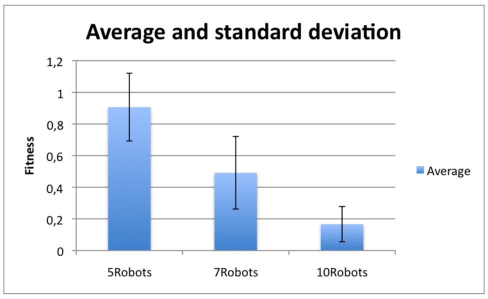
Figure 6.4: fitness trajectory of all 10 evolutionary runs

In terms of strategy, except on evolutionary run 3, the robots split in two groups (a group of 2 and a group of 3) and head for an energy landmark, like in the previous experiment, the group with 3 robots starts to move in circles in the border of the energy landmark, so there are never 3 robots inside simultaneously. On evolutionary run 3, all 5 robots head to a landmark and start to move in circles in the border of the energy landmark. This can explain the poor performance achieved.

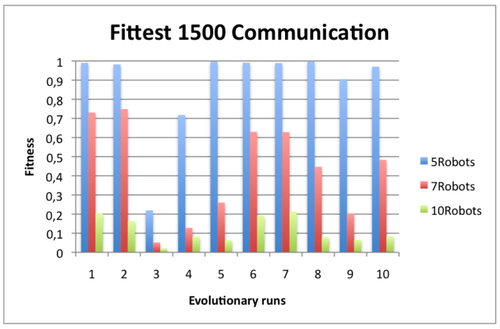
Scalability



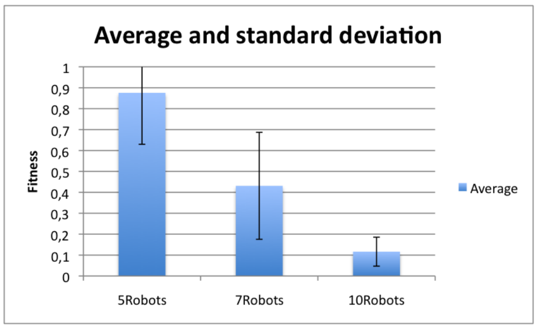
(a) 1200 steps



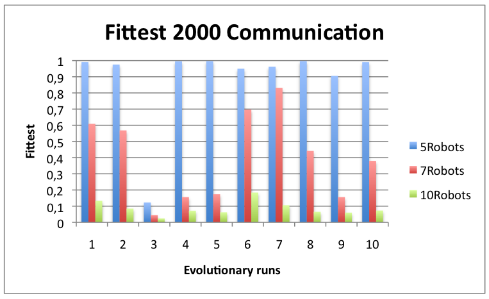
(b) Average and standard deviation 1200 steps



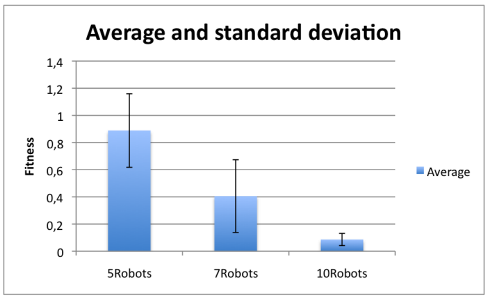
(c) 1500 steps



(d) Average and standard deviation 1500 steps



(e) 2000 steps



(f) Average and standard deviation 2000 steps

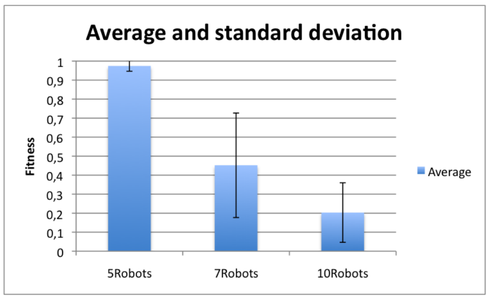
Figure 6.5: Scalability study

Like in the experiment described in section 6.1, to study scalability 10 samples of the last generation of each evolutionary run were analyzed. For each evolutionary run, populations of 5, 7 and 10 robots were used, during 1200, 1500 and 2000 control steps. The results are shown in figure group 6.5. Like in the experiment described in 6.1, populations of 5 robots scaled well in terms of time and with less success in terms of population size. The percentage of populations that did not lose any member until the end of the simulation was of 90% for 1200 steps simulations, 90% for 1500 steps simulations and 90% for 2000 steps simulations. When the number of robots is increased to 10 the percentage dropped to 60% for 1200 steps, 40% for 1500 steps and 30% for 2000 steps. The results suggest that there is not much difference in terms of scalability between populations with and without explicit communication capabilities. However, the energy levels shown in this experiment are slightly higher, which means an increase of energy management efficiency. That difference may be related to the way the robots apply the strategy described in 6.2.2: in groups of robots where communication did emerge, it is possible to observe robots explicitly waiting for others to come out of the energy source. A change of color also occurs when the robots are low in energy and also a change of behavior: the robots when reach a critical low will force their entrance to the source and will adopt a different color as well, thus warning the robots inside to leave. This behavior was not observed in the experiment described in 6.1, where the strategy to manage more robots with the same energy is just to rotate faster in the border of the energy landmarks.

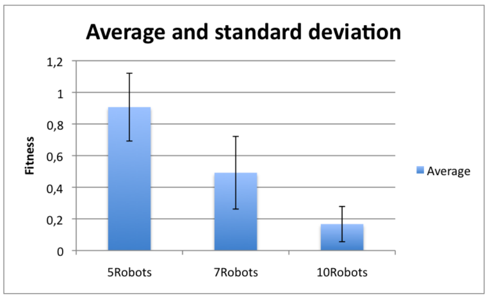
6.2.3 Discussion

The aim of this experiment was to study the emergence of communication protocols in the energy management problem and if it would show an increase of performance when compared with the experiment described in section 6.1, where the robots did not have the ability to explicitly communicate. Nevertheless the robots in this experiment performed slightly better (the fitness values were slightly higher in some cases) it is not enough to conclude that the simple fact the robots are able to explicitly communicate will make them perform better. In fact, the results suggested that when a communication mean is available, the populations will take longer to reach a good solution. evolutionary runs that perform significantly worse than the average seem to appear more often when the robots are able to explicitly communicate but this issue should be studied more in detail to draw a conclusion. A comparison between the results obtained in both experiments can be found in figure 6.6.

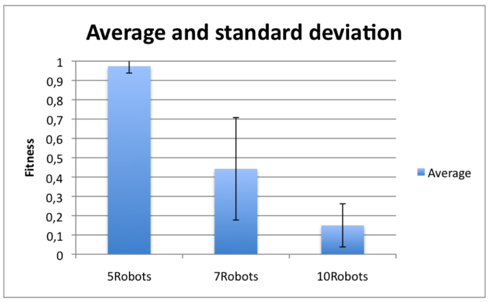
The emergence of communication seems to be the result of a fluke. The robots will start to emit signals and to react to those signals because, at a given point in the evolutionary process, a robot emitted a random signal and another robot reacted randomly to that signal and by mere chance the result contributed to an increase of the overall fitness and that set of behaviors was passed to the next generation. Like in the experiments in section 5, it is the junction between coordinated movement and communication that will lead to a good fitness, however, in the particular case described in this section, communication seemed to have a small contribution. In fact, communication made the evolutionary process slower (the experiments took up to 50% more time to run when the robots had the color actuator and sensors), the optimal solution (if any) was reached significantly later and in some cases communication protocols did not emerge at all. The decision of evolving communication protocols for the robots to perform a given task is something that has to be studied thoroughly, because it is time and resource consuming and it might not add performance at all.



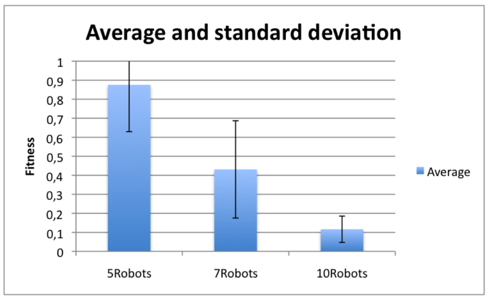
(a) 1200 steps no communication



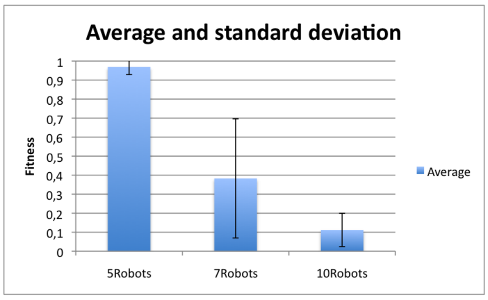
(b) 1200 steps with communication



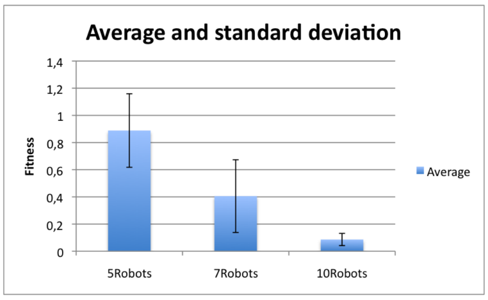
(c) 1500 steps no communication



(d) 1500 steps with communication



(e) 2000 steps no communication



(f) 2000 steps with communication

Figure 6.6: Scalability study

Chapter 7

Conclusions

This report consisted of a set of 5 experiments regarding self-organized aggregation, collective choice and energy management, using artificial evolution techniques, in the JBotEvolver simulator. The main goal of the experiments was to study de emergent strategies the robots evolved to solve the problems and if those strategies would still be valid in different environmental setups or different number of robots.

In the self-organized aggregation experiments, the results showed that the behaviors emerged fast and were efficient. Even though the evolutionary process only counted with 5 robots, it was proved that the solutions achieved by each one of the 10 epochs performed could scale at least up to 100 robots, so we can conclude that the resulting controllers have a very high scalability potential. Nevertheless, there is still space to improvements: when the groups are big enough, the distance from the robots in the far side of the cluster are far away from its center of mass, even though they are a single cluster. An evaluation function able to count the number of clusters available was developed like is referred in section 1.3 but due to time limitations it has not been tested in the self-organizing aggregation problem. Another approach would be to find the relationship between maximum cluster radius and population size and apply it to an evaluation function. The advantages of using this approach instead of counting clusters is that will be less resource consuming, which is a very important factor to take in consideration if this technique for producing efficient robot controllers is to be used at a large scale.

In the study of emergent collective choice behaviors, the robots had to collectively find and choose a single cluster to aggregate in. The controllers produced evaluated in terms of efficiency and scalability. The results showed that collective choice behaviors did emerge and 2 different strategies to choose a landmark and cluster in it were identified: *Scouting* and *Following*. The *Scouting* strategy proved to be more efficient than the *Following* strategy: when a group of robots use the *Scouting* strategy, only a few elements of the population will actively leave the group to find a landmark. The remaining robots are clustered and following, as a flock and at a distance, the scout robots. When the scout robots find a landmark, they will stop inside, thus forcing the remaining group to join them and the collective choice is made. On the other hand, the *Following* strategy relies on brute force: the robots start to move in circles, with a variable radius, until a landmark is found by one of them. When a robot finds a landmark, it will stop and create a chain reaction where the robots moving behind the one who found the landmark will reach the landmark and stop and so forth until all robots are inside the landmark. However, by using this strategy, two or more robots can find a different landmark simultaneously, thus forming two or more clusters. The results showed that only between 30% and 40% of all epochs developed *Scouting* strategies, but they have also showed that it is possible to evolve a controller general enough to solve other problems than the ones faced during evolution: the controllers that developed a *Scouting* strategy performed well with bigger groups of robots and with more landmarks present in the environment.

The next step was to repeat the same experiment but with populations of robots capable of communicate through a color actuator. The purpose of this experiment was to study if communication protocols would emerge and if so, if they would increase the performance of the robots when compared with groups of robots without the color actuator. The results showed that communication protocols did not emerge in all epochs. They also showed that the simple fact the robots are able to communicate through a color actuator does not mean that they will perform better. The increase of performance only happened when communication protocols and coordinated movement was observed. In fact, the results also showed that the simple fact the robots were able to detect each other is a communication protocol per se and it should be taken in consideration when more explicit communication protocols are expected to emerge. The main problem with this experiment is the percentage of epochs that showed a good combination between movement coordination and explicit communication through the color actuator, which was of 10%.

Finally, experiments where the robots had to develop strategies to manage two energy sources that can recharge only a limited number of robots at a time were performed and described. The controllers produced were evaluated in terms of efficiency and scalability like the previous experiments. Since the management of an energy source is the problem to be solved, the lifespan of the population (the amount of time the population can survive) is also a question to be studied, so the scalability study has 2 dimensions: the size of the population and the time the population can survive without losing an element. The results showed that, in terms of time, the solutions achieved scaled very well: even though only 1200 control steps were used during evolution, a population of 5 robots could be maintained up to 2000 control steps without losing any robot, and since the energy level of the weakest robot by the 2000th control step was above 90%, it is expectable that the population survives more time. In terms of population size, the solutions achieved were significantly less effective, since the fitness dropped significantly when populations of 7 and 10 robots were used. This fitness drop was expected, because there are more robots to recharge and therefore the waiting times to enter an energy source are higher and the time a robot spends recharging is lower. When the number of robots was increased, the fitness was also more sensitive to the number of control steps the population had to survive. The results showed a significant drop of fitness, and even some cases were the population loses robots, when the number of control steps is increased. Despite that, some controllers emerged that showed that it is possible to manage the energy landmarks with more robots than the ones used during evolution and for a longer period of time, without losses and with energy levels of the weakest robots between 30% and 40%.

Like in the collective choice experiment, an explicit communication channel was created, by equipping the robots with a color actuator and color sensors in the same experiment. The results showed that the populations took significantly more time to achieve a good solution, when compared to populations without the color actuator. It also suggested that populations with a significantly worse performance than the average are more prone to appear, but this fact should be studied further. Even thought the results appeared to be slightly better than the ones obtained with robots without the color actuator, it is not enough to assume that robots able to explicitly communicate perform better. In fact, populations of robots able to communicate through the color actuator will take longer to reach a good solution and that is an important factor to take in consideration if this technique is to be used in a large scale. The emergence of communication seems to be the result of a fluke, like suggested by Maynard Smith [27]. Both collective choice with communication and energy management with communication experiments showed that the simple fact the robots have a communication mean does not mean evolution will exploit that. Sometimes it happens, sometimes it does not, like the results suggest. The decision of evolving communication protocols to solve a given problem or to perform a given task has to be studied thoroughly, because of its time and resource consuming nature and also because it might not add performance at all.

From a broader perspective, all the experiments gave good results and interesting information about self-organized aggregation, collective choice and energy management. Self-organized aggregation proved to be a behavior that easily emerges, in a very short period of time. The controllers produced to have self-organized aggregation behaviors performed well, especially in terms of scalability, as the results showed. Collective choice behaviors seemed to be more difficult to evolve. Not all the evolutionary runs produced controllers able to make the group of robots to collectively choose and aggregate in a single landmark. However, some interesting strategies were observed that might inspire new evaluation functions to evaluate the populations in a way that will promote more efficient controllers. Regarding the energy management experiments, since similar experiments were not found in literature, I did not know what to expect in terms of results. Nevertheless, the controllers produced developed strategies to manage the energy landmarks and make the group of robots to live for long periods of time. When an explicit communication means was added to the robots and the experiments were re run, the results showed a slight improvement but not enough to reach solid conclusions.

The results presented in this report suggest that it is possible to use artificial evolution to produce efficient controllers. Evolution proved to be a way to produce complex controllers in a relative short period of time. However, to produce quality controllers, able to perform efficiently and scalable enough, the evaluation functions used to score the generations during evolution have to be carefully designed. The task to be done by the robots has to be absolutely clear in order to be translated precisely into an evaluation function, otherwise evolution will always follow the path of least resistance, over simplifying the problem and producing controllers that will not show the desirable behavior. For the set of experiments shown in this report, several evaluation functions were tested and discarded until an optimal solution was found. This was one of the most time consuming tasks I had, only second to the time the evolutionary process took. Nevertheless, if the problem is carefully approached and translated to an evaluation function and if the environment translates precisely the problem to be solved, the results shown in this report suggest that efficient controllers can be produced.

7.1 Final report vs. preliminary report

The objectives defined in the preliminary report were partially achieved. The *Double Landmark* environment has been used to perform experiments about collective choice. Based on that environment, the *Landmark* environment was also created, which allowed performing experiments with any number of landmarks, placed in a circle or randomly in the environment. Experiments with the *Landmark* environment were also performed and described in this report. The influence of the size of the population used in the experiments, or scalability, was also studied, as suggested in the preliminary report. The *Energy Management* environment was an addition to the experiments that were suggested in the preliminary report, and came as a natural following of the collective choice experiments.

Unfortunately, the controllers produced were not applied to the *e-puck* [34] robots, as suggested in the preliminary report. While performing the experiment in the simulator, my coordinators and I decided that the time and effort that we would spend to adapt the controllers produced to work on the *e-puck* robots would be best applied on studying in more detail some other aspects of collective choice and energy management. Also, a bug on a sensor was discovered and lots of results had to be discarded.

Overall, my opinion is that the main objectives of this research were achieved, despite the fact that none of the controllers were adapted to work on the real world.

7.2 Future work

The confirmation of the results obtained in the real world, with real robots would be the most important step to take from here. Some other issues pointed in the discussion sections of each experiment should also be studied in more detail.

In the self-organized aggregation experiments, a new evaluation function should be developed to take in consideration the maximum radio of a cluster formed by a given the number of robots. That way, the robots in the outer part of the cluster would still be considered as at a distance 0 of the center of mass. Another approach to this problem would be to repeat the experiments but with the *CenterOfMassAndClusters* evaluation function that counts the number of clusters at a given control step.

Regarding the collective choice experiments, a way to increase the percentage of epochs that show a *Scouting* behavior should be studied. Controllers that developed *Scouting* behaviors proved to scale very well but, with the approach taken in the experiments performed they appear in only 30% or 40% of the epochs. Also, a way to increase the percentage of epochs that can combine movement coordination and communication through the color actuator should be studied, since in the results presented it is only of 10% to 20%.

Energy management experiments with robots, using an evolutionary algorithm are a novelty and so far I could not find any relevant literature about that. Therefore it has a lot of space for improvements. A new environment could be developed where the robots should perform a simple task, a foraging task for example, where in order to survive the robots should also learn how to search for energy and recharge. The fact the robots are recharging would not contribute for the final fitness, just the foraging task. Also, an explicit communication mean, like the color actuator used in the experiments in this report, could be used to study if the fact the robots are able to communicate helps in some way the foraging task and/or the strategy the robots take to recharge themselves.

Experiments using the artificial pheromone developed for the simulator could also be performed to study if that is a more efficient mean of communication.

Appendix A

Evaluation functions

To create a new evaluation function class, a class in the *evolutionaryrobotics.evaluationfunctions* package, extending the *EvaluationFunction* class has to be created. Each evaluation function has a *step* method which calculates the fitness value in each control step and a *getFitness* method to return the fitness value. The final fitness value of a chromosome is the average value obtained during the simulation.

A.1 Example of an evaluation function

This evaluation function evaluates the distance between each robot and the group's center of mass and the number of neighbors of the ith robot:

```
package evolutionaryrobotics.evaluationfunctions;

import mathutils.Vector2d;
import simulation.Simulator;
import simulation.physicalobjects.ArtificialPheromoneObject;
import simulation.robot.Robot;
import simulation.robot.sensors.NearRobotSensor;

/**
 *
 * @author Andre Bastos
 */

public class CenterOfMassEvaluationFunction extends
    EvaluationFunction {

  /**
   * Evaluation function that evaluates the distance between each
      robot and the group's center of mass
   */
  private int steps = 0;
    private double fitness;

  public CenterOfMassEvaluationFunction(Simulator simulator){
    super(simulator);
  }
```

```
    public void step(){
      steps++;
        Vector2d coord = new Vector2d();
        double x=0.0, y=0.0;
        double distance = 0.0;
        double auxDist = 0.0;
        double finalDist = 0.0;
        int near = 0;
        int nRobots = simulator.getEnvironment().getRobots().size();


    for(Robot r: simulator.getEnvironment().getRobots()){
      x+=r.getPosition().x;
      y+=r.getPosition().y;
    }
    x=x/simulator.getEnvironment().getRobots().size();
    y=y/simulator.getEnvironment().getRobots().size();
    coord.set(x, y);

    for(Robot t: simulator.getEnvironment().getRobots()){
      distance = coord.distanceTo(t.getPosition());
      if(distance <0.3)
        distance = 0;
      if(distance >((NearRobotSensor) t.getSensorWithId(1)).getRange
          ()){
        distance =((NearRobotSensor) t.getSensorWithId(1)).getRange
            ();
      }
      auxDist += (1-(distance/((NearRobotSensor) t.getSensorWithId
          (1)).getRange()));
    }

    finalDist = (auxDist/nRobots);
    fitness +=finalDist;

  }

  public double getFitness() {
    return fitness/simulator.getExperiment().getNumberOfStepsPerRun
        ();
  }
}
```

Appendix B

Example of a configuration file

Here is a configuration file example where robots, actuators, sensors, controllers, environments and experiments are defined correctly.

```
    –robots robotconfigid=1,name=bee,radius=.05,sensors=(energysourcesensor=
    (id=1,numbersensors=8,orientation=0,angle=135,range=5.0),energysensor=(id=2),nearrobot=
    (id=3,numbersensors=8,orientation=0,angle=135,range=5)),placement=nest,actuators=
    (twowheelact=(id=1))
    –controllers name=CTRNNMultilayer,robotconfigid=1,hiddennodes=5,inputs=auto,outputs=auto
    –evaluation name=survive
    –environment name=Energy,foragelimit=2,forbiddenarea=5,
    energySourceDistance=2.0,energySourceRadius=0.
    30,energySourceNumber=1,energy=2,varyexperiments=false,israndom=false,initialEnergy=800,
consumption=1,gain=2
    –random-seed 1563
    –experiment name=aggregation,numberofrobots=5,varynumberofrobots=false
    –population name=mulambda, size=100, samples=2, stepsperrun=1200, generations=500,
mutationrate=0.15
    –gui name=none
    –output results
```

B.1 Options

Description of the parameter used in the example:

* –robots: type and parameters of the robots to be used in the simulation.

* –controllers: type and parameters of the controller to be evolved in the simulation.

* –environment: type and parameters of the environment to be used in the simulation.

* – output: output folder where the simulation output files, such as populations, arguments, best generations configuration files, fitness log and number of generations defined are generated.

* –evaluation: name of the evaluation function to be used.

* –random-seed: the random seed to be used in the simulation.

* –population: the type of population to be used, size of the genome, number of control steps (virtual time), number of generations to be created and the mutation rate.

* –gui: type of gui to be used

Appendix C

Physical objects

Each physical object type is defined by a class in the *simulation.physicalobjects* package that extends the *PhysicalObject* class. The first step to create a new physical object is to define its name in the *PhysicalObjectType* enum class. The next step is to create the physical object class, in the *simulation.physicalobjects* package, extending the *PhysicalObject* class. The new physical object class will have 1) a constructor which will receive the simulator instance and the object's parameters, such as name, coordinates in the environment, size and the correspondent *PhysicalObjectType* element and 2) a *getPosition* method that will return the object's position in the form of a 2D vector.

In order for the sensors to detect an object, a checker class has to be created in the *simulator.physicalobjects.checkers* package. A checker class extends the *AllowedObjectsChecker* class and consists on a boolean value that defines that a given type of object is allowed to be detected by the sensor.

Finally, to make the object visible in the simulator environment, the *TwoDRenderer* class, in the *gui.renderer* package has to be modified. A method has to be created to define the way the object will be displayed. Parameters such as color, color changes and so forth are defined in it.

C.1 Example of a physical object

This is the *LightPole* object or *Landmark* used in the collective choice experiments, chapter 5.

```
package simulation.physicalobjects;

import mathutils.Vector2d;
import simulation.collisionhandling.knotsandbolts.
   CircleCollisionObject;
```

```
public class LightPole extends PhysicalObject {

  public LightPole( String name, double x, double y, double
     radius) {
    super(name, x, y, 0, 0, radius, PhysicalObjectType.
       LIGHTPOLE);
    this.collisionObject = new CircleCollisionObject(name + "
       CollisionObject", this, x, y, 2 * radius, radius);
  }

  public Vector2d getPosition() {
    return super.getPosition();
  }


}
```

Appendix D

Sensors

To create a new sensor, a class extending the Sensor class has to be created in the *simulation.robot.sensors* package. Each sensor will have at least 1) a constructor which will receive an instance of the simulator, an id, and the robot instance where it will be used, and a 2) *getSensorReading* method which will return the sensor readings. If the sensor is to detect a specific object, a sensor class extending the ConeTypeSensor has to be created. In this case, the constructor will also have the positions in the robot the sensors will be placed, the sensor range value and the type of object the sensor will detect (the physical object checker class). A *calculateContribuitonToSensor* method will be created instead of the *getSensorReading*, where the rules to decide whether a given physical object is within range are defined. After the sensor class is created, it has to be included on the *RobotFactory* class, where the name that will be used in the configuration file to create the new sensor is defined, as well as its default parameters, such as number of sensors in a robot, orientation, opening angle and range. The sensor also has to be connected to an input neuron of the neural network, so an input class has to be created. The input classes are in the *evolutionaryrobotics.neuralnetworks.inputs* package and implement the NNInput interface. The input classes will have a 1) constructor method, where the sensor instance will be received, 2) a *getNumberOfInputValues* where the number of sensors of the given kind is returned and a 3) *getValue* method to return the readings from the ith sensor are returned. These input classes will be instantiated in the *ControllerFactory* (in the *createInput* method) and will be identified by the corresponding sensor's name, defined in the robot factory class.

Finally, to use the sensor, in the robot section of the configuration file under the sensors subsection, the name that was defined in the robot factory class can be used and its parameters values defined. If a sensor name that is not defined in the robot factory class is used, an exception will occur and the simulator will not start.

D.1 Example of a sensor

Proximity sensor used by the robots

```
package simulation.robot.sensors;

import java.util.Arrays;

import mathutils.Vector2d;
import simulation.Simulator;
import simulation.environment.Environment;
import simulation.physicalobjects.GeometricInfo;
import simulation.physicalobjects.PhysicalObjectDistance;
import simulation.physicalobjects.checkers.
   AllowedObjectsChecker;
import simulation.robot.Robot;
import simulation.util.SimRandom;

/**
 * Proximity sensor
 * @author Andre Bastos
 *
 */
public class NearRobotSensor extends ConeTypeSensor {
  private double openingAngle;
  private double maxDistance;
  private double r;
  public NearRobotSensor(int id, Robot robot, Simulator
      simulator,
      int numbersensors, double orientation, double range,
      AllowedObjectsChecker allowObjectChecker, double
          openingAngle) {
    super(id,robot,simulator,numbersensors,orientation,range,
       allowObjectChecker);
    this.robot = robot;
    this.openingAngle = openingAngle/2;
    maxDistance = range - robot.getRadius();
    r=(Math.PI/2)/openingAngle;

  }



  /**
   * Calculates the distance between the robot and a given
      PhisicalObject if in range
   * @return distance or 0 if not in range
   */
    @Override
    protected double calculateContributionToSensor(int
       sensorNumber,
      PhysicalObjectDistance source) {
    GeometricInfo sensorInfo = getSensorGeometricInfo(
       sensorNumber, source);
    if(sensorInfo.getDistance()<getRange() && (sensorInfo.
       getAngle()< (openingAngle)) &&
       (sensorInfo.getAngle()>(-openingAngle))){
      double val = ((getRange()-sensorInfo.getDistance())/
          getRange())*Math.cos((sensorInfo.getAngle())*r) +
          SimRandom.getInstance().nextGaussian() * NOISESTDEV;
      if (val > 1.0)
        val = 1.0;
      else if (val < 0.0)
        val = 0.0;

      return val;
    }
    return 0;
  }


  /**
   * Sensor readings
   */
    @Override
  public String toString() {
    for(int i=0;i<numberOfSensors;i++)
      getSensorReading(i);
    return "NearRobot [readings=" + Arrays.toString(readings) +
        "]";
  }

}
```

Appendix E

Actuators

The process of creating a new actuator is similar to the process of creating a new sensor. An actuator class extending the Actuator class has to be created in the *simulation.robot.sensors* package. Each actuator will have 1) a constructor which will receive an instance of the simulator and an id and 2) an *apply* method which will define the behavior of the actuator. After creating the actuator class, it has to be included in the *RobotFactory* class, where the name that will be used in the configuration file to create the new actuator is defined, as well as its default parameters. Actuators have to be connected to an output neuron of the neural network, so an output class has to be created also. The output classes are in the *evolutionaryrobotics.neuralnetworks.inputs* package and they implement the *NNOutput* interface. The output classes will have 1) a constructor method which will receive the instance of the actuator and a list of arguments that define the actuator's parameters, 2) a *getNumberOfOutputValues* where the number of instances of the actuator present in a robot is returned, 3) a *setValue* method which will define the actuator's behavior according to the value the neural network returns and an *apply* method that will make the needed changes to the actuator's parameters. These output classes will be instantiated in the *ControllerFactory* (in the *createOutput* method) and will be identified by the corresponding actuator's name, defined in the robot factory class.

Finally, to use the actuator, in the robot section of the configuration file under the actuators subsection, the name that was defined in the robot factory class can be used and its parameters values defined. If an actuator name that is not defined in the robot factory class is used, an exception will occur and the simulator will not start.

E.1 Example of an actuator

Wheel actuator used by the robots

```
package simulation.robot.actuators;

import java.util.Random;

import simulation.robot.Robot;
import simulation.util.SimRandom;

public class TwoWheelActuator extends Actuator {

  public static float   NOISESTDEV  = 0.05f;
  private Random      random    = SimRandom.getInstance();

  private double      leftSpeed   = 0;
  private double      rightSpeed  = 0;

  public TwoWheelActuator(int id) {
    super(id);
  }

  public void setLeftWheelSpeed(double speed) {
    leftSpeed = speed;
  }

  public void setRightWheelSpeed(double speed) {
    rightSpeed = speed;
  }

  public void setWheelSpeed(double leftSpeed, double rightSpeed
     ) {
    this.leftSpeed  = leftSpeed;
    this.rightSpeed = rightSpeed;
  }

  @Override
  public void apply(Robot robot) {
    leftSpeed*=(1 +random .nextGaussian() * NOISESTDEV);
    rightSpeed*=(1 +random .nextGaussian() * NOISESTDEV);

    if (leftSpeed < -Robot.MAXIMUMSPEED)
      leftSpeed = -Robot.MAXIMUMSPEED;
    else if (leftSpeed > Robot.MAXIMUMSPEED)
      leftSpeed = Robot.MAXIMUMSPEED;

    if (rightSpeed < -Robot.MAXIMUMSPEED)
      rightSpeed = -Robot.MAXIMUMSPEED;
    else if (rightSpeed > Robot.MAXIMUMSPEED)
      rightSpeed = Robot.MAXIMUMSPEED;
    robot.setWheelSpeed(leftSpeed, rightSpeed);

  }

  @Override
  public String toString() {
    return "TwoWheelActuator [leftSpeed=" + leftSpeed + ",
       rightSpeed="
        + rightSpeed + "]";
    }
}
```

Appendix F

Environments

To create a new environment, a class in the *simulation.environment* package that extends the *Environment* class. Each environment class will have a 1) constructor where parameters such as type and placement of physical objects are received, as well as the simulation instance and a 2) *draw* method to draw the environment on the screen. The physical objects are all created in the environment class. After creating the environment class, it has to be included in the *EnvironmentFactory*, where the name that will be used in the configuration file to create the new environment is defined.

F.1 Example of an environment

Landmark environment used in chapter 5.

```
package simulation.environment;

import gui.renderer.Renderer;
import java.util.ArrayList;
import simulation.physicalobjects.LightPole;
import simulation.util.Arguments;

/**
 * Class that defines a lightpole environment
 * @author Andre Bastos
 */
public class LightPoleEnvironment extends Environment {


  private double lightPoleRadius;
  private double lightPoleDistance;
  private double forageLimit, forbiddenArea;
  private int currentSample;
  private int maxNumberRobots;
        private int lightPoleNumber;
        private ArrayList<LightPole> poles;
        /**
         * Constructs a lightpole environment
         *
         * @param arguments
         */
  public LightPoleEnvironment(Arguments arguments){
    super(arguments.getArgumentIsDefined("forbiddenarea") ?
        arguments.getArgumentAsDouble("forbiddenarea")        :
        5.0,
        arguments.getArgumentIsDefined("forbiddenarea") ?
            arguments.getArgumentAsDouble("forbiddenarea") :
            5.0);
                lightPoleDistance = arguments.
                    getArgumentIsDefined("lightpoledistance") ?
                    arguments.getArgumentAsDouble("
                    lightpoledistance"): 2.0;
                lightPoleRadius = arguments.
                    getArgumentIsDefined("lightpoleradius") ?
                    arguments.getArgumentAsDouble("
                    lightpoleradius") : 0.10;
                lightPoleNumber = arguments.
                    getArgumentIsDefined("lightPoleNumber") ?
                    arguments.getArgumentAsInt("lightPoleNumber
                    ") : 0;
                String vary = arguments.getArgumentIsDefined("
                    varyexperiments") ? arguments.
                    getArgumentAsString("varyexperiments") : "
                    false";
                currentSample = arguments.getArgumentIsDefined
                    ("fitnesssample") ? arguments.
                    getArgumentAsInt("fitnesssample") : 0;
                if(vary.equalsIgnoreCase("true")){

                  poles = createMultipleEnvironments(
                      lightPoleRadius);
                }
                else{
                  poles = createPoles(getPositions(
                      lightPoleNumber), lightPoleRadius);
                }
                currentSample = arguments.getArgumentIsDefined
                    ("fitnesssample") ? arguments.
                    getArgumentAsInt("fitnesssample") : 0;
                maxNumberRobots = arguments.
                    getArgumentIsDefined("maxnumberrobots") ?
                    arguments.getArgumentAsInt("maxnumberrobots
                    ") : Environment.getInstance().robots.size()
                    ;

                for(LightPole p: poles)
                    addObject(p);

  }
```

```
/**
 * Generates the poles
 * @param list, the positions list of the poles
 * @param radius
 * @return list of poles
 */
private ArrayList<LightPole> createPoles(ArrayList<
    Double> list, double radius){
     LightPole light;
     ArrayList<LightPole> polesToAdd = new ArrayList<
         LightPole>();
     for(int i = 0; i<list.size(); i++){
          light = new LightPole("pole "+Integer.toString(
              i), list.get(i), list.get(i+1), radius);
          polesToAdd.add(light);
          i = i+1;
     }
     return polesToAdd;
}

private ArrayList<LightPole> createMultipleEnvironments
    (double radius){
  LightPole light;
  ArrayList<LightPole> polesToAdd = new ArrayList<
      LightPole>();

switch(currentSample%4){
case 0: lightPoleNumber = 0;break;
case 1: lightPoleNumber = 1;break;
case 2: lightPoleNumber = 2;break;
case 3: lightPoleNumber = 5;break;
}
ArrayList<Double> positions = getPositions(
    lightPoleNumber);
  for(int i=0; i<positions.size();i++){
     light = new LightPole("pole "+Integer.toString(i),
         positions.get(i), positions.get(i+1), radius);
          polesToAdd.add(light);
          i = i+1;
  }
  return polesToAdd;
}

/**
 * Calculates the position of the poles
 * @param numberOfPoles
 * @return positions
 */
private ArrayList<Double> getPositions(int
    numberOfPoles){
     ArrayList<Double> list = new ArrayList<Double>();
     if(numberOfPoles == 1){
          list.add(0.0);
        list.add(0.0);
     }
     else{
```

```
                    for (int i = 1; i<= numberOfPoles; i++){
                        list.add(lightPoleDistance*Math.cos(2*(Math.PI
                            )*i/numberOfPoles));
                        list.add(lightPoleDistance*Math.sin(2*(Math.PI
                            )*i/numberOfPoles));
                    }

                }
                return list;
            }

    public double getLightPolesRadius(){
      return lightPoleRadius;
    }

    public double getLightPoleDistance(){
      return lightPoleDistance;
    }

    public double getForageRadius() {
      return forageLimit;
    }

    public double getForbiddenArea() {
      return forbiddenArea;
    }

    @Override
    public void draw(Renderer renderer){
                for(LightPole p: poles){
                    renderer.drawCircle(p.getPosition(), p.
                        getRadius());
                }
    }



  }
```

Appendix G

Experiments

To create a new experiment, a class in the *experiment* package that extends the *Experiment* class has to be created. Each experiment class will only have a constructor where the experiment, environment, controller, robots arguments are received, as well as the simulator instance. After creating the experiment class, it has to be included in the *ExperimentFactory* where where the name that will be used in the configuration file to create the new environment is defined.

G.1 Example of an experiment

Aggregation experiment used in all experiments.

```
package experiments;

import java.util.Vector;

import simulation.robot.Robot;
import simulation.util.Arguments;
import evolutionaryrobotics.Experiment;


/**
 * Aggregation Experiment
 * @author Borga
 *
 */
public class AggregationExperiment extends Experiment {

  public AggregationExperiment(Arguments experimentArguments,
      Arguments environmentArguments, Arguments robotArguments,
      Arguments controllerArguments) {
    super(experimentArguments, environmentArguments,
        robotArguments,
         controllerArguments);

  }
  /**
   * Create robots. The number of robots can vary in each
      sample
   */
  public Vector<Robot> createRobots() {
    int numSamples=experimentArguments.getArgumentAsInt("
       fitnesssample");
    int numRobots = experimentArguments.getArgumentAsInt("
       numberofrobots");
    String varyNumberRobots = experimentArguments.
       getArgumentAsString("varynumberofrobots");
    if(varyNumberRobots.equalsIgnoreCase("true")){
      robots = super.createRobots();
      if (experimentArguments != null) {
        numSamples = experimentArguments.getArgumentAsInt("
           fitnesssample");
      for (int i = 0; i < (numSamples%3)*numRobots; i++)
        robots.add(createOneRobot(robotArguments,
           controllerArguments));
      }
    }
    else{
      robots = super.createRobots();
    }
    //System.out.println("Robots "+robots.size());
    return robots;
  }
}
```

Bibliography

[1] Jean-Marc Ame, Colette Rivault, and Jean-Louis Deneubourg. Cockroach aggregation based on strain odour recognition. *Animal Behaviour*, 68(4):793–801, October 2004.

[2] T. Back, U. Hammel, and H.P. Schwefel. Evolutionary computation: Comments on the history and current state. *Evolutionary Computation, IEEE Transactions on*, 1(1):3–17, 1997.

[3] G. Baldassarre, S. Nolfi, and D. Parisi. Evolving mobile robots able to display collective behaviors. *Artificial Life*, 9(3):255–267, 2003.

[4] R.D. Beer. On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, 3(4):469, 1995.

[5] S. Camazine. *Self-organization in biological systems.* Princeton Univ Pr, 2003.

[6] G. Caprari, P. Balmer, R. Piguet, and R. Siegwart. The autonomous micro robot ÒAliceÓ: a platform for scientific and commercial applications. In *Micromechatronics and Human Science, 1998. MHS'98. Proceedings of the 1998 International Symposium on*, pages 231–235. IEEE, 1998.

[7] N. Correll and A. Martinoli. Modeling self-organized aggregation in a swarm of miniature robots. In *Proceedings of the IEEE 2007 International Conference on Robotics and Automation Workshop on Collective Behaviors Inspired by Biological and Biochemical Systems*, 2007.

[8] C. Darwin. *Origin of species.* Wilder Pubns Ltd, 2008.

[9] J. De Greeff and S. Nolfi. Evolution of implicit and explicit communication in mobile robots. *Evolution of Communication and Language in Embodied Agents*, 1:179, 2010.

[10] J L Deneubourg, a Lioni, and C Detrain. Dynamics of aggregation and emergence of cooperation. *The Biological bulletin*, 202(3):262–7, June 2002.

[11] D.B. Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence*, volume 1. Wiley-IEEE Press, 2006.

[12] L.J. Fogel, A.J. Owens, and M.J. Walsh. Artificial intelligence through simulated evolution. 1966.

[13] R. M. Friedberg, B. Dunham, and J. H. North. A learning machine: Part ii. *Ibm Journal of Research and Development*, 3:282–287, 1959.

[14] R.M. Friedberg. A learning machine: Part i. *IBM Journal of Research and Development*, 2(1):2–13, 1958.

[15] Simon Garnier, Christian Jost, Jacques Gautrais, M Asadpour, G Caprari, and R. The embodiment of cockroach aggregation behavior in a group of micro-robots. *Artificial Life*, 408:387 – 408, 2008.

[16] Simon Garnier, Christian Jost, R Jeanson, and Jacques Gautrais. Aggregation behaviour as a source of collective decision in a group of cockroach-like-robots. *Advances in Artificial*, pages 169–178, 2005.

[17] D.E. Goldberg and J.H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[18] E. Hagen, P. Hammerstein, and N. Hess. Strategic aspects of communication. *Evolution of Communication and Language in Embodied Agents*, 1:55, 2010.

[19] G. Hardin. The tragedy of the commons. *Nature's web: rethinking our place on earth*, page 173, 1993.

[20] JH Holland. Adaptation in natural and artificial systems: An introductory analysis with applications to biology. *Control, and Artificial Intelligence (MIT Press, Cambridge, MA, 1992).*

[21] D.E. Jackson and FL Ratnieks. Communication in ants. *Current biology: CB*, 16(15):R570, 2006.

[22] Raphael Jeanson, Colette Rivault, Jean-Louis Deneubourg, Stephane Blanco, Richard Fournier, Christian Jost, and Guy Theraulaz. Self-organized aggregation in cockroaches. *Animal Behaviour*, 69(1):169–180, January 2005.

[23] G. Jones. Genetic and evolutionary algorithms, 1998.

[24] H. Kwasnicka, U. Markowska-Kaczmar, and M. Mikosik. Flocking behaviour in simple ecosystems as a result of artificial evolution. *Applied Soft Computing*, 2010.

[25] V. Marmol, M.B. Dias, and B. Kannan. Market-based coordination of recharging robots.

[26] C. Mautner and R.K. Belew. Evolving robot morphology and control. *Artificial Life and Robotics*, 4(3):130–136, 2000.

[27] J. Maynard Smith. The theory of evolution, 1975.

[28] J. Maynard Smith and E. Szathmáry. The major transitions in evolution. *Nature*, 1995.

[29] C. Melhuish, O. Holland, and S. Hoddell. Convoying: using chorusing to form travelling groups of minimal agents. *Robotics and Autonomous Systems*, 28(2-3):207–216, 1999.

[30] M. Mirolli and S. Nolfi. Evolving communication in embodied agents: Theory, methods, and evaluation. *Evolution of Communication and Language in Embodied Agents*, 105, 2010.

[31] S. Mitri, D. Floreano, and L. Keller. The evolution of information suppression in communicating robots with conflicting interests. *Proceedings of the National Academy of Sciences*, 106(37):15786, 2009.

[32] S. Mitri, D. Floreano, and L. Keller. Evolutionary conditions for the emergence of communication. *Evolution of Communication and Language in Embodied Agents*, 1:123, 2010.

[33] C. Moeslinger, T. Schmickl, and K. Crailsheim. A minimalist flocking algorithm for swarm robots. In *Proceedings of ECAL*, volume 9. Citeseer, 2009.

[34] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65, 2009.

[35] 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.

[36] S. Nolfi and D. Floreano. Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines. 2000.

[37] D. Parisi and A. Cangelosi. The emergence of a" language" in an evolving population of neural networks. In *Connection Science*. Citeseer, 1996.

[38] M. Quinn, L. Smith, G. Mayley, and P. Husbands. Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1811):2321, 2003.

[39] I. Rechenberg. Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution. 1973. *Frommann-Holzboog, Stuttgart*.

[40] H.P. Schwefel. Numerical optimization of computer models. 1981.

[41] F. Sempe, A. Munoz, and A. Drogoul. Autonomous robots sharing a charging station with no communication: a case study. *Distributed Autonomous Robotic Systems*, 5:91–100.

[42] O. Soysal, E. BAHCECI, and E. Sahin. Aggregation in swarm robotic systems: Evolution and probabilistic control. *Turk J Elec Engin*, 15(2), 2007.

[43] Luc Steels. Modeling the formation of language in embodied agents: Methods and open challenges. 2010.

[44] V. Trianni, R. Groß, T.H. Labella, E. Sahin, and M. Dorigo. Evolving aggregation behaviors in a swarm of robots. *Advances in Artificial Life*, pages 865–874, 2003.

[45] G. Werner and M. Dyer. Evolution of communication in artificial organisms. In C. Langton, C. Taylor, D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 659–687, Redwood City, CA, 1992. Addison-Wesley Pub.

[46] E. WILEYTO, G. BOUSH, and L.M. GAWIN. Function of cockroach (Orthoptera: Blattidae) aggregation behavior. *Environmental entomology*, 13(6):1557–1560, 1984.