POLITECNICO DI MILANO
FACOLTÀ DI INGEGNERIA

Corso di laurea in ingegneria delle telecomunicazioni
Dipartimento di elettronica e informazione
Progetto di intelligenza artificiale e robotica

Active Vision in a Collective Robotics Domain

Tesi di laurea di: Stefano Lanza
Matricola 640898

Relatore: prof. Andrea Bonarini
Correlatori: ing. Vito Trianni
prof. Marco Dorigo

Anno accademico 2004/05

ABSTRACT

This thesis discusses an innovative active vision system applied to the control of multiple robots in a coordinate motion task. The vision system of each robot consists in an omni-directional camera that acquires 360-degree views of the surrounding environment. Afterward, a virtual retina, corresponding to an area of the image of varying size and position, automatically scans the acquired images in order to extract visual information relevant to the robotic task. Artificial evolution is employed to synthesize controllers capable of exploiting the intrinsic relationship between motion and vision in the robots, and also the complex interactions between the robots and the environment where they operate.

The thesis presents three experiments of increasing complexity dealing with robots driven by the developed vision system. The obtained results and the generalization properties of the synthesized controllers are presented. In particular, the thesis shows the successful synthesis of vision-guided controllers for the coordinate motion of a group of robots. We demonstrate how the evolved vision system is capable of actively selecting, through the retina, the visual features pertaining to the accomplishment of the task expected from the robots. These results open the way to future applications in robotics, employing the virtual retina as a tool for the efficient processing of images in tasks guided by vision.

ACKNOWLEDGEMENTS

First, I would like to thank my supervisor Vito Trianni for his assistance in writing this thesis. Special thanks also to Marco Dorigo for giving me the opportunity to work on this thesis in his laboratory IRIDIA. Many thanks and greetings to all the people working there, who have become nice friends during my stay in Bruxelles. Many thanks also to my family who supported me throughout my university studies. And finally, special thanks to my nice girlfriend Dina who passed with me many special moments during the time necessary for writing this thesis. She was always ready to support my work and to comfort me when something went wrong, as Word crashing every 10 minutes… Besides, she has taught me the few French words I have learnt in Belgium.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION AND OVERVIEW

This thesis discusses an innovative approach to artificial vision for the control of single and multiple robots. We want to develop robotic controllers capable of extracting the visual information necessary for the accomplishment of a common objective, which in our case is the coordinate motion of a group of robots.

Artificial vision (*computer vision* or *computational vision*) is a research area dealing with the analysis and correct interpretation of visual information extracted from images. In robotics literature, there are two main approaches. The first approach consists in applying hard-coded algorithms to images in order to extract relevant information. Usually, this approach starts applying low-level filters that improve the quality of the images, e.g. cleaning them from noise and enhancing their contrast. Thereafter, predefined standard algorithms process the filtered images in order to extract useful characteristics. Feature extraction mechanisms include edge and motion detectors, depth from stereo disparity, shape from shading, and landmark detection [42]. The second approach to artificial vision consists in exploiting some form of unsupervised learning to reduce the great flow of information offered by images to a subset of statistically invariant and optimal characteristics. This approach requires off-line learning on a large set of available visual data and works well only on certain probability distributions of input data.

Both approaches can be applied to the control of autonomous robots. However, none takes into account the fact that vision is an intrinsic part of an organism's sensory-motor cycle [27]. In other words, the information gathered through vision partly determines, after being processed, the organism's behavior. On the other end, the performed actions affect (directly or indirectly) the successive visual acquisition, closing the analysis and response to visual stimuli into a loop that strictly characterizes the organism's behavior. Within the robotic context, the same mechanism affects the behavior of vision-driven robots throughout the whole control phase. For instance, in our experiments, robots respond to visual stimuli acquired by an on-board camera and their behavior deeply influences the successive camera acquisition. The camera indeed moves and rotates rigidly with a robot, thus constantly perceiving the environment from different points of view according to the robot's actions. Furthermore, the aforementioned approaches to vision processing do not consider the fact that the amount and type of information received by the vision

system depends also on the physical characteristics and motion capabilities of the robot, and on the environment where the robot is embedded [27].

In order to consider these aspects, the paradigm of active vision has been recently proposed [3]. The main characteristic of an active vision system is the possibility of *actively* changing both external and internal image formation parameters, e.g. fixation point, focal length, etc. The vision system adapts its behavior in order to extract the information that is important at the moment, instead of processing a huge amount of information out of every image. Active vision can be thought of as a more *task* driven approach than passive vision. Applied to robotics, an autonomous robot can actively select from the available information, through its behavior, the environmental features to directly exploit for the achievement of a certain objective. On the contrary, a passive system would process all of the data to construct a global picture before making decisions (in this sense it can be described as *data* driven).

We developed a novel active vision system for robotics control, inspired upon recent works by Marocco and Floreano [43]. Within our system, every robot is equipped with an omni-directional camera constantly acquiring panoramic, 360-degree views of the surrounding environment. Because of complex dynamic interactions between robots, vision system and environment, the acquired images constantly change unpredictably. Moreover, the optical process characterizing the robots' camera determines strong distortions of the perceived environment. Consequently, the classical approach to vision processing and features extraction through hard-coded algorithms is complex and time-demanding. Besides, unsupervised learning techniques are also unpractical because the environment where robots interact is not known a priori, especially in experiments involving many dynamic entities. We propose, as a solution, the active choice of the environmental characteristics that can be useful to accomplish a given task.

At this purpose, we provide the robots' vision system with a *virtual retina*, capable of autonomously scanning the camera images and extracting relevant information. The retina corresponds to a limited area of the image of varying size and position, capable of moving across the image in search of relevant features, and zooming in and out to extract features at different levels of detail. Kato and Floreano [37] first introduced the idea of a virtual retina "inspired upon the evidence that humans and other animals analyze images with rapid saccadic movements [39], instead of providing immediate answers based on a single snapshot of the entire image (as most computer vision techniques)" [43]. We also retain that this approach might be useful for simplifying vision-based robotic control systems. A robot may exploit the retina to direct its attention specifically on small regions which are important at various times, avoiding to waste efforts in trying always to understand the whole surroundings. Besides, the retina appears as an efficient solution to image processing in systems having low computational capabilities, also outside the context of robotics.

The robot must learn to actively control the virtual retina to gather visual information relevant at the moment to achieve its objective. One fundamental problem is still unsolved. How shall the robot move and, at the same time, control the retina in response to visual inputs received from the environment, in order to carry out a predefined task? The problem is, in the end, finding the appropriate mapping between retinal input and control output for a given task. At this purpose, we employed artificial evolution to automatically synthesize neural networks controlling robots and their active vision system according to a desired task. Within the context of vision processing, artificial evolution is a powerful method to co-evolve feature-selection mechanisms and behavior of autonomous robots because it does not differentiate visual perception from behavior as in conventional system engineering methods [15]. Moreover, artificial evolution tests robots directly in their working environment, thus considering their complex interactions and their physical characteristics in the process of synthesizing efficient controllers for the task at hand.

The work for this thesis is done entirely in simulation. The basic simulator already reproduced the dynamics between robots and environment, through an accurate simulation of the involved physical aspects. This thesis extends the simulator by adding a graphical interface that allows the study of active vision through the following features:

- Possibility of observing the environment from the point of view of an external observer.
- Possibility of observing the environment from the point of view of each robot.
- Simulation of the omni-directional camera.
- Features extraction from the camera images through a virtual retina.

We developed a new graphical engine showing the simulated environments from user-specified points of view, meeting the requirements of the first two features. The third feature is particularly challenging given the complexity of the optical process generating the camera images. In this regard, we devised a novel technique based on *ray-tracing* [25], combining several concepts and techniques taken from the field of Computer Graphics. Furthermore, we designed two artificial retinas of rectangular and circular shape respectively; the former operates on generic images while the latter better suits the omni-directional images processed by our robots, characterized by peculiar features.

The primary objective of the thesis is that of synthesizing controllers that, by exploiting the developed visual system, guide the coordinate motion of a group of robots. In particular, the robots must learn to move coordinately in the group exploiting visual information about their environment perceived by their camera and subsequently extracted through a retina. This thesis reaches that objective by presenting three experiments of increasing complexity.

The first experiment applies artificial evolution to synthesize a virtual retina capable of focusing on static objects viewed by an omni-directional camera. The

object appears in the image as a bi-dimensional figure of varying shape, position and color. The retina must localize the object in the image and then zoom in or out in order to optimally focus on the object. This experiment has the lowest complexity because the camera does not move, being detached from a robot, and acquires images of static environments. Here, the aim is studying the evolved retinal behavior for the recognition of objects in images, starting from a simple setup. We want to prove the efficiency of the virtual retina in automatic features extraction that might be useful in vision-based tasks. In particular, we are interested in the generalization properties of evolved retinas, to certify their ability to work under generic conditions and to adapt to dynamic contexts, a property strictly required if we want to exploit later the retina for robotic navigation. Therefore, this experiment serves as a basis for the more advanced tasks dealt by the other two experiments.

The second experiment improves over the first one by attaching the camera to a single, mobile robot. Here, the robot must move towards a target object randomly positioned in the environment. In doing that, the robot must learn to focus the retina on the object viewed by the omni-directional camera, and then exploit the consequent retinal information to build appropriate navigation trajectories. This experiment adds a degree of complexity over the previous one because the robot's perception of the environment changes as the robot moves in response to the retinal input, according to the sensory-motor loop we mentioned previously. We show how evolved robots are capable of actively maintaining their focus on the target object, in order to efficiently move towards it. This result is particularly interesting, showing how evolution can discover efficient solutions taking into consideration all the complex characteristics we have described so far.

The last experiment applies the developed active vision system in a collective robotics context concerning the coordinate movement of a group of robots (swarming or flocking). Robots must exploit local visual information coming from their vision system (camera and retina) to move in a coordinate fashion while exploring the environment. This behavior draws inspiration from that observed in schools of fishes and birds. We expect small groups of robots (2 – 4) to accomplish the task first, and then we will study how the evolved behaviors scale to larger groups. This experiment is the most challenging because it aims at synthesizing an intelligent vision system capable of driving independent robots towards a common objective (coordinate movement in this case). In doing that, artificial evolution must discover the interactions between vision and motion at the individual level, and among the robots inside the group, which eventually determine the desired global behavior. We show how the evolved robots control simultaneously their motion and vision in order to achieve their task, by actively tracking the other robots through the virtual retina.

We believe that the work reported in this thesis represents one of the first successful uses of evolutionary robotics to develop coordinated behavior in a collective robotics system guided by active vision.

1.1 Overview

The following sections give an overview of the fields of interest for this thesis. Section 1.1.1 discusses recent works of Active Vision, with emphasis on their application to robotics control and navigation. The other section reviews recent works on Collective Evolutionary Robotics that are closely related to the experiments described in this thesis.

1.1.1 Active Vision

The main source of inspiration for this thesis is the work done by Marocco and Floreano on active vision for automatic features selection. In their paper [43], they discuss recent work on the evolution of a simulated active retina for complex shape discrimination. This retina operates on images showing at random triangles or squares, and must recognize the correct shape by scanning the image. A simple neural network drives the retina and recognizes the correct shape, receiving in input the retinal data gives. The authors then extend this approach to a mobile robot equipped with a mobile CCD pan/tilt camera capable of independently explore the environment while the robot moves around. Here, the camera works as the active acquisition device in place of the artificial retina of the previous experiment. The robot is positioned in an arena and asked to navigate as far as possible without hitting the walls. The robot's controller receives input from the camera and determines in output the movement of the robot, of the camera, and the type of filtering applied to the camera image. The authors make use of artificial evolution to develop controllers for their experiments of active vision. They show that evolved robots are capable of selecting simple visual features and actively maintaining them on the same retinal position in order to generate efficient navigation trajectories.

Our vision system combines the main ideas introduced in those experiments. We employ an artificial retina to actively select the visual input relevant to control robots. An important difference is that, in our experimental setup, robots are not equipped with a mobile camera, but with a fixed omni-directional camera capable of perceiving the surrounding environment from every direction. Lastly, we extend the use of active vision from the single robot case to the control of multiple robots that must cooperate to achieve a swarming behavior.

Many other authors reported on works on active vision applied to the control of single robots. Harvey et al. [33] addressed the problem of navigation acquiring information about the environment through a camera. They evolved both the morphology of the visual receptive field and the architecture of the neural network. Using these settings, they successfully synthesized an individual capable of approaching a triangular shape painted on a wall while avoiding a rectangular one, guided by the vision system. In his thesis [66], Carl-Johan Westelius discusses focus of attention and gaze control of a robot active vision system. The robot is equipped

with heterogeneously sample imaging systems, foveas, resembling the space varying resolution of a human retina. It has a three-layer hierarchical gaze control system based on rotation symmetries, linear structures and disparity, estimated from images through proper filters. The author also presents a new focus of attention method based on a filtering method, normalized convolution. Andrew John Davison [16] studies mobile robot navigation using active vision. The core work of his thesis concerns simultaneous localization and map building for a robot with a stereo active head, operating in an unknown environment and using point features as visual landmarks. The robot's active camera allows the construction and maintenance of an efficient navigation map by adopting a strategy for serially fixating on different features during navigation. Li [40] built on the work of Du [22] in using an active head to detect obstacles on the ground plane in front of a mobile robot, and also performing some active tracking of features with a view to obstacle avoidance. However, only limited results were presented, and in particular the robot did not manoeuvre in response to the tracking information, and no attempt was made to link the obstacle-detection and obstacle-avoidance modules. Beardsley et al. [7] used precisely controlled motions of an active head to determine the affine structure of a dense point set of features in front of a robot, and then showed that it was possible to initiate obstacle avoidance behaviors based on the limits of free-space regions detected. At NASA, Huber and Kortenkamp [35][36] implemented a behavior-based active vision approach which enabled a robot to track and follow a moving target (usually a person). Correlation-based tracking was triggered after a period of active searching for movement, and extra vision modules helped the tracking to be stable. The system was not truly an active vision navigation system, however, since sonar sensors were used for obstacle detection and path planning.

1.1.2 Collective Evolutionary Robotics

The field of Collective Robotics studies robotic systems composed of autonomous robots cooperating for the achievement of a common goal (see [49] for an overview). The main motivation behind the study of collective robotic systems lays in the possibility to decompose the solution of a complex problem into simpler sub-problems that can be faced by simple robotic units. Within this context, the use of artificial evolution as a methodology to synthesize behaviors for groups of robots has been limited. Collective evolutionary robotics has often focused on coordinated motion in a group of robots. Trianni [63] studied coordinate motion in a group of physically linked *s-bots* sensing the traction exerted by other robots. The results showed the emergence of good coordinate behaviors, capable of generalizing to new conditions, as different size, topology and type of links in the group of robots. Reynolds [54] evolved the control system of a group of creatures, called *boids*, which were placed in an environment with static obstacles and a manually programmed predator. The control system was evolved to avoid collisions and to escape from

predators. Although the results described in the paper are rather preliminary, some evidence indicates that coordinated motion strategies emerged. In a follow-up of this work, Ward et al. [64] evolved *e-boids*, groups of artificial fishes capable of displaying schooling behavior. Two populations of predator and prey creatures were evolved and placed in a 2D environment containing randomly distributed food elements. The analysis of the distance between prey, prey and food, and predator and prey suggests that the emergence of the schooling behavior is correlated with: (i) an advantage in the ability to find food clumps, and (ii) a better protection from predation. Spector et al. [59] used genetic programming to evolve group behaviors for flying agents in a simulated environment. Overall, the above mentioned works suggest that artificial evolution can be successfully applied to synthesize effective collective behaviors. Recently, Quinn [51] explored two ways of evolving controllers for a group of robots while studying a coordinated motion task using two simulated Khepera robots. In the first approach, called *clonal*, all members of the group share the same genome. This is the same approach we used in the experiments presented in this thesis. The second approach, called *aclonal*, provides each member of the group with a different genome. In the aclonal evolution, the fitness of each robot is computed separately, whereas in the clonal evolution the fitness of a robot is calculated as the average fitness of the group. Results obtained indicated that aclonal evolution produced better performing behaviors for this rather simple task. In fact, with aclonal evolution it was possible to obtain different controllers for different roles in the performance of the task. In a very recent work, Quinn et al. [53] studied coordinated motion in small groups of real homogeneous robots provided with minimal sensors, controlled by artificially evolved neural network controllers. Analyzing the evolved behaviors, they were able to observe that robots adopt distinct roles in the group. Few other works are loosely related to the evolution of swarming behaviors. For example, Zaera et al. [66] carried out a series of experiments to study the use of evolution as a methodology to develop collective behaviors for groups of virtual fishes swimming in a rather realistic 3-D simulated environment. They were able to evolve aggregation and dispersal behaviors fairly easily, but they observed that these collective behaviors were not a result of interactions among the members of the group, but rather between the individual fish and the environment (the boundaries of the arena). Additionally, their attempts to evolve schooling behavior were not very successful.

1.2 Contributions

To the best of our knowledge, this thesis discusses, for the first time in robotics, the design and synthesis, through artificial evolution, of an active vision system for the control of a group of robots cooperating on a common goal. The main contribution of this thesis is the design of an innovative active vision system operating on small,

computationally limited mobile robots (*s-bots*). The vision system acquires images of the environment surrounding a robot through an omni-directional camera. In order to synthesize vision-based controllers for our robots in simulation, we designed a novel and highly efficient technique for the real-time simulation of the camera. The same technique can be easily adapted to the simulation of different omni-directional camera devices within other robotic projects. Moreover, the robots' vision system is equipped with an integrated software acquisition device, named *virtual retina*, capable of efficient image processing and automatic features extraction. The retinal device, previously introduced by other authors, here finds original application in the automatic extraction of visual information relevant to the navigation of single and multiple robots in the environment, according to the desired objective. In particular, we developed a special retina of circular shape to optimally process the panoramic images acquired by the omni-directional camera of each robot, in view of their peculiar characteristics. The union of the retina and the omni-directional camera form therefore a complete and efficient system of active vision for applications in robotics control.

1.3 Thesis Organization

This thesis is organized as follows:

- Chapter 2 introduces Swarm Robotics, an emergent approach to the control of multiple robots based on the collective intelligence of social insects in nature. We discuss the most important advantages of this approach and the mechanism of *self-organization*. Then, we detail the problems in controlling single and multiple robots, proposing thereafter Artificial Evolution as solution for semi-automatic controller design. The last section of chapter 2 describes the SWARM-BOTS project, wherein this work has been conducted, which aims at developing an innovative swarm robotics system.
- Chapter 3 presents the setup of our experiments. Initially, we describe the simulator with details on its components and features. Then, we introduce the hardware and simulated model of the *s-bot*, the robot developed within the SWARM-BOTS project that we used for our experiments. The vision system of these robots is then presented along with a brief description of their camera and the particular characteristics of the camera images. Finally, we describe the evolutionary algorithm that we employed to develop controllers for different tasks.
- Chapter 4 describes the simulation of the omni-directional camera through a highly optimized version of ray-tracing. The first section shows the optical process generating the camera images, and then presents ray-tracing as a method to replicate that process. The successive sections present several optimizations,

mostly inspired by various computer graphics techniques, which allow the application of the ray-tracing technique to evolutionary experiments.

- Chapter 5 discusses the virtual retina device used to extract features from the images acquired by an omni-directional camera. The first section introduces the general characteristics of a retina, and the related rendering and filtering operations. The successive sections illustrate the rectangular and the circular retina. Emphasis is given to the circular retina since it used in our experiments.
- Chapter 6, 7 and 8 describe the three experiments conducted for this thesis, dealing with a focusing, targeting and swarming task respectively. The chapters first introduce the respective task along with the motivations and challenges behind it. The experimental setup is then presented, with details on the simulated scenario, the vision system, the neural network controller and the fitness function used to evaluate controllers for the task. Finally, the last section of each chapter shows the obtained results and the analysis and generalization properties of the evolved solutions.
- Chapter 9 draws the conclusions of our work, highlighting the important aspects of the research. Finally, we indicate directions for future works.

2

SWARM ROBOTICS

This chapter introduces the emergent field of Swarm Robotics. Section 2.1 describes the main concepts of Swarm Intelligence. In particular, we describe the advantages of this approach in the control of many agents and an important related concept, *self-organization*. Next, section 2.2 discusses the problems in designing controllers for single and multiple robots. In this section, we also illustrate the solutions we adopted for our work to synthesize efficient controllers. The last section briefly describes the SWARM-BOTS project, wherein this work has been conducted, which aims at developing an innovative swarm robotics system.

2.1 Swarm Intelligence

Swarm Intelligence is the modeling of collective group behaviors found in social insects and other animal societies [9]. This field is relatively new and developed during the 90s as researchers looked for scalable and robust methods to handle problems of information control. Many social insects and animals give inspiration to scientists because they can naturally solve complex problems with no form of centralized control. For example, ants can trace the shortest path between the nest and the food source with no prior knowledge of the environment and with no single ant directing the actions of the group [17]. This problem solving behavior emerges from simple interactions among the ants.

We can apply the ideas of swarm intelligence for the control of swarms of robots tightly cooperating on a common objective. This approach naturally brings many advantages that simplify the design of control systems in collective robotics.

Decentralization: In a decentralized system, each individual takes decisions independently from the others. For example, ants and bees take most of their decisions alone without being governed by a leading individual. Even so, they are able to achieve an extremely efficient and organized behavior at the colony level. A distributed control system is much simpler in comparison to a centralized controller. The latter in fact requires reliable communication channels between each robot and the central controller. The planning of the instructions for all the robots is also demanding [44]. For these reasons, the cost and complexity of a centralized system

increases exponentially with the number of robots. On the contrary, robots in a swarm are autonomously controlled agents. This simplifies the system's design and leads to lower costs and higher robustness.

Robustness: Robustness is directly linked to decentralization. Indeed, failures in a centralized system of the controller or some components would imply in most cases the complete failure of the whole system. Instead, in a decentralized system, the loss or failure of a single agent is usually not particularly severe. However, the system must also be redundant in order be robust, otherwise the loss of specialized agents could compromise the correct functioning of the whole group. Redundancy and the consequent robustness are typical properties of insect colonies, which are able to function after the removal of many individuals. Applied to robotics, several robots may fail without affecting the task completion.

Adaptivity: A further advantage is that swarms of agents can adapt naturally and quickly to changing environmental conditions in order the preserve the best system's performance. The system can automatically create new responses to unexpected situations through complex dynamics that arise from interactions between the components and their environment.

Many potential applications of swarm-based robotics require miniaturization. Very small robots, micro-and nano-robots, with severely limited sensing and computation, may need to operate in very large groups or swarms to affect the macroworld. Approaches directly inspired or derived from swarm intelligence may be the only way to control and manage such groups of small robots.

2.1.1 Self-Organization

The concept of Self-Organization (SO) [9] is very important in many models of swarm intelligence. It explains how the behavior of a system emerges from the local interactions between its components. Most notably, this happens without any external guidance or control and without central coordination. Simply, local interactions produce emergent patterns and configurations that solve the problem faced by the swarm. There are four key aspects in the concept of self-organization.

Positive feedback: An agent influences other agents to modify their behavior. The classic example is that of an ant leaving a *pheromone* trail to a food source that other ants follow. As each ant follows the trail, it adds its own pheromone attracting even more ants to that specific trail.

Negative feedback: This feedback serves as a regulatory mechanism by somehow discouraging a type of behavior. Common forms of negative feedback are pheromone evaporation and food source exhaustion.

Random fluctuations: Theories of SO show that randomness or fluctuations in individual behavior, far from being harmful, may greatly enhance the system's ability to explore new behaviors and find new solutions. For an example, let us consider an ant colony. Ants following a path to food will sometimes wander off the path. This allows these ants to discover new food sources or possibly shorter paths to a previously discovered food source.

Multiple Interactions: Individual agents use multiple rules to govern their next decision. Often, for self-organization to appear, you need a sufficient number of tolerant agents. For instance, in some social insects, piles of bodies, or graveyards often form in predictable patterns. However, these graveyards only form when a certain number of insects are moving bodies simultaneously. Without these multiple interactions, the graveyard would not form.

The system starts in a disordered state with random fluctuations influencing individual actions and interactions. Then, self-organization may emerge from the interplay of the positive and negative feedbacks. These mechanisms lead the system to a stable state and restore its organization after any deviation caused by external forces [46][24].

We are particularly interested in self-organization because it allows a system of simple components to solve complex problems. Besides, self-organization brings two desired properties: robustness against possible failures of some components, and fast adaptation to unexpected environmental changes. In fact, the system has the natural tendency of returning to a stable configuration (even a different one) through feedback mechanisms that continuously maintain the system organized.

Apart from ants, many other biological systems offer examples of self-organization in nature. For example, the bark beetle larvae *Dendroctonus* are able to aggregate using only local mechanisms [19]. In some insect colonies there is also a particular type of self-organization called *self-assembling*, which is the creation of structures through connections among the individuals composing the system [1]. Section 2.3 shows interesting applications of this concept in the field of robotics.

2.2 Controller Design

The design of controllers for robotic systems is challenging. The general problem is determining how we should program simple robots to perform user-designed tasks. This problem is particularly difficult in the case of many robots that tightly interact and cooperate in the attempt to achieve a common goal. However, the control of a single robot working in a complex environment is also challenging.

Single Robot

The control of a single robot faces some challenges. First, we have to understand the relevant interactions between the robot and the environment that lead to the desired behavior. Then, we have to find out the rules that reproduce these interactions. This operation is difficult because we cannot easily predict which behavior results from a given set of rules, and which are the rules that determine a given behavior [48]. Once we have determined these rules, we have to encode them in an appropriate control system.

Multiple Robots

In the case of multiple robots, the field of swarm intelligence gives precious indications on the properties and characteristics of efficient control systems for many individuals. The main property, self-organization, regulates the emergence of global behaviors capable of solving complex problems, through local interactions between the system's components. At the same time, the system becomes robust against failures of some components, and capable of quickly adapting to unexpected situations.

Given these strong advantages, we want to imitate nature by designing self-organizing control systems for our robots. There are two main problems in obtaining self-organization in a group of artificial individuals [63]. First, we have to discover the important mechanisms and interactions among robots that produce the self-organizing group behavior (Figure 1, center). In doing that, we have to consider the environment where the robots are embedded because of its importance as a communication channel. After we have discovered these simple mechanisms, we have to translate them into rules for appropriate robotic controllers (Figure 1, right).

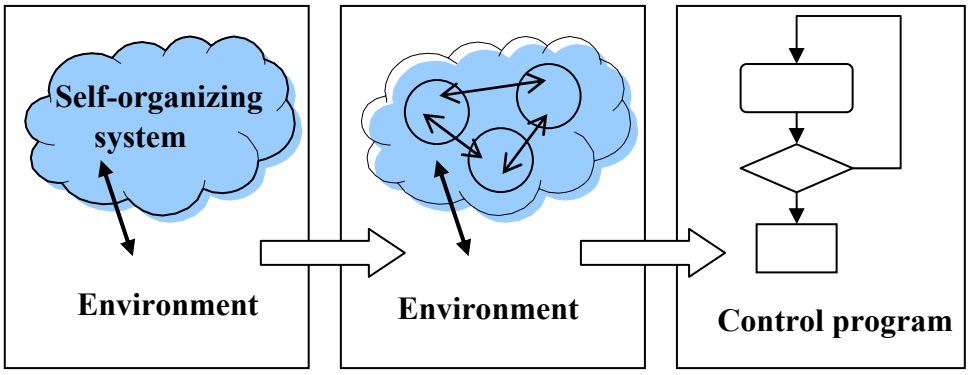

Figure 1: The design problem for multiple robots. Decomposition of the global behavior (left to center). Encoding of individual behaviors in controllers (center to right).

We note that, until now, we implicitly assumed that all robots were identical units. The situation becomes more complicated when the robots have different

characteristics, respond to different stimuli, or respond differently to the same stimuli. In this case, there is practically no theoretical guideline for the emergent design and control of heterogeneous swarms.

Summarizing, from an engineering perspective, the design of controllers for both single and multiple robots is very complex because, in both cases, we have to understand and reproduce local behaviors starting from interactions between the single robot and its environment, or interactions among multiple robots.

2.2.1 Imitating Nature

We can try to solve the aforementioned challenges by imitating the behavior of insects or other organisms in natural systems. This is actually the approach of swarm-intelligence for the control of multiple robots, but the same principle applies also to the control of single robots. In both cases, we have to understand the basic mechanisms that in nature regulate behaviors analogous to those we want to give to our robots.

This approach starts from the observation of the natural phenomenon and then proceeds with a modeling phase that tries to extract the rules that would reproduce the behavior. After that, we have to replicate the developed model into the artificial system in order to obtain dynamics similar to the natural counterpart. As shown by Figure 2, this approach requires a first decomposition step that models the phenomena observed in nature to find out the basic mechanisms and interactions. Then, in the design phase, we have to encode these mechanisms into the control program.

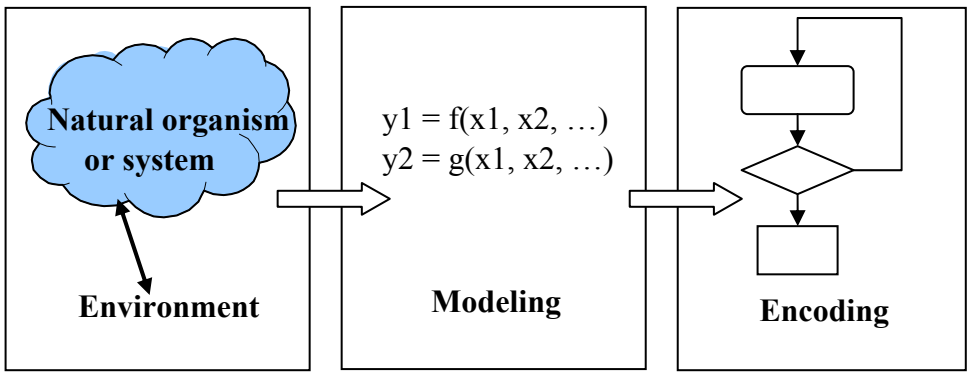

Figure 2: Imitating nature to design controllers.

The approach of imitating nature for the development of robotic controllers has some drawbacks. The first one is that it can be very hard to predict which rules and laws drive a model to produce useful behavior. Second, it is not always possible to take inspiration from natural processes because they may differ from the artificial systems in many important aspects (e.g., the physical embodiment, the type of possible interactions between individuals and so forth), or because there are no

natural systems that can be compared to the artificial one [63]. Moreover, the problem of encoding the individual behaviors into appropriate controllers for the robots remains unsolved. Our working hypothesis is that these problems can be efficiently solved relying on Artificial Evolution, as discussed in the next section.

2.2.2 Artificial Evolution

Artificial evolution can efficiently solve the problem of designing controllers for single or multiple robots. In fact, artificial evolution overcomes the two main challenges we described in the previous section: the decomposition of global behaviors in local interactions between agents and environment, and the encoding of these interactions into rules processed by a controller.

Artificial Evolution first evaluates potential controllers according to the resulting global behavior, and then it iteratively selects the best ones and discards those that perform poorly. An essential aspect of this process is that controllers are directly evaluated on robots embedded in their environment. Consequently, evolution can automatically determine those interactions between robots and environment that are relevant to obtain the desired global behavior. At the same time, evolution also discovers the controller's rules that reproduce individual behaviors. Figure 3 shows this process.

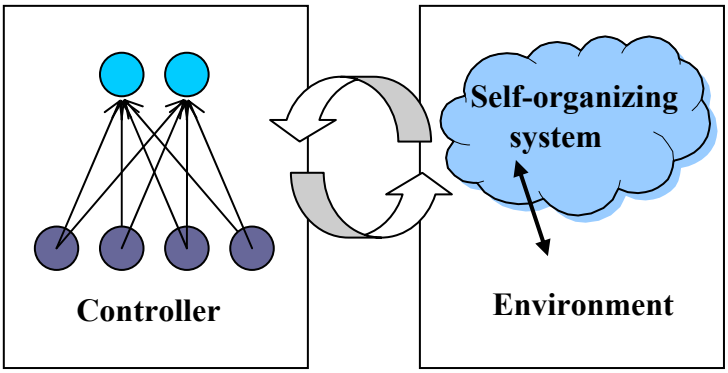

Figure 3: Design through artificial evolution. Evolution evaluates how well controllers produce the desired behavior testing them in the environment (left to right). Thereafter, the best controllers are selected and iteratively improved throughout the evolution (right to left).

Another advantage is that artificial evolution can explore the rich variety of solutions derived from the complex dynamics between the robot and the environment and, for robot swarms, among all the robots. Often, we cannot absolutely predict all these possible solutions with a hand design because the system, especially when there are many robots, is very complex. In this case, hand design is not only complex but also time-consuming and prone to errors. On the contrary, artificial evolution automates the design process.

However, artificial evolution has some costs. First, we have to identify initial conditions that guarantee the possibility of evolving solutions starting from an initial random design. Second, we have to formulate effective evaluation functions that allow significant incremental improvements of these solutions. Another important disadvantage of artificial evolution is that it requires long computation times, being a long iterative process that evaluates many potential controllers at each iteration. In addition, the evaluation of controllers on real, physical robots is often unpractical and considerably slow. For this reason, we artificially evolved controllers entirely in simulation. The simulation's accuracy is very important in order to preserve the relevant mechanisms that happen in reality. Therefore, we developed a simulator that can accurately reproduce the dynamics between the robots and the environment. Chapter 3 describes the simulator, the robot model and the evolutionary algorithms we used for our experiments.

2.3 SWARM-BOTS

The SWARM-BOTS project[1] studies the design and implementation of self-organizing and self-assembling robots. These robots are physically independent mobile robots that are able to self-assemble by connecting to and disconnecting from each other. In the project, single agents are named *s-bots*, and groups of jointed s-bots working cooperatively are named *swarm-bots*. Swarm-bots can perform tasks like exploration, transport of heavy objects and navigation on rough terrains, where a single robot has major problems. The goals of this project include [56]:

Dynamic shape formation: The *s-bots* should be able to assemble into swarm-bots when they cannot solve some tasks alone. The swarm-bots should then move coherently without centralized control, and dynamically reconfigure along the way to match environmental variability (for example, to go through a narrow passage).

Navigation on rough terrains: Swarm-bots should be able to navigate on uneven terrains presenting holes and obstacles. The group should intelligently split and assemble to overcome hostile situations, such as the passage of holes larger than a robot (Figure 4) or the climbing of a tall obstacle.

Transport of objects: The *s-bots* should be able to transport heavy object by collaborating on the transport, in a similar way of ants performing prey-retrieval.

In order to fulfill the above requirements, *s-bots* are equipped with a broad variety of sensors and actuators, limited computational resources and physical links for the connection to other *s-bots* (details on the hardware are in section 3.2.1).

[1] www.swarm-bots.org

Figure 4: Robots in rigid formation can overcome small holes.

Researchers working on the SWARM-BOTS project follow two different but complementary directions to develop controllers for the *s-bots*. The former consists in building control systems by mimicking the characteristics of biological systems such as social insects. The latter consists in building control systems loosely inspired from known mechanisms regulating real organisms, developed through a self-organization process based on artificial evolution. By following the first approach, robots able to aggregate were developed. Instead, by following the second approach, *s-bots* able to display coordinate movements, collective obstacle avoidance, and object pushing/pulling were developed. We followed the latter approach, designing controllers for coordinate movement based on innovative solutions of active vision. Chapters 6, 7, 8 describe our efforts and the obtained results.

3

EXPERIMENTAL SETUP

This chapter illustrates the general setup of our experiments, described later in their respective chapters. Section 3.1 is dedicated to the description of the simulator used by all the experiments. Section 3.2 introduces the *s-bot* hardware, the simulated model and the vision system. Finally, section 3.3 describes the evolutionary algorithm employed to synthesize controllers for different tasks within our experiments.

3.1 The Simulator

All the experiments presented in this thesis run in simulation. We completely developed the simulator in house, apart from the physics engine, in C++ language. The simulator is multi-platform working under both Windows and Linux. It features a basic scripting support, through the LUA language[2], for the configuration of the experiments. In addition, the simulator features a graphical user interface (GUI) that allows the user to interact with the simulated environment.

We can logically organize the simulator in four main components:

Dynamics engine: The dynamics engine simulates the physics of robots while they interact with the environment, made of walls, obstacles and other moving robots. At this purpose, we used the rigid body dynamics SDK Vortex™ (Critical Mass Labs, Canada), which reproduces the dynamics, friction and collision detection between physical bodies.

Graphics engine: The graphical engine displays the three-dimensional, simulated objects in real-time. It uses OpenGL[3] for multi-platform, accelerated hardware rendering, and the SDL library[4] for low-level access to input devices and 2D video. The engine supports simple lighting of materials, according to the OpenGL lighting

[2] www.lua.org
[3] www.opengl.org
[4] www.libsdl.org

model. It also makes use of standard geometrical optimization techniques, as frustum culling[5], to accelerate rendering.

All the objects handled by the graphics engine, such as physical bodies, cameras and lights, are characterized by a position and orientation in the scene, and optionally by a 3D model for display purposes. The engine supports the loading of models from file and the procedural creation of models from a set of primitive shapes, including rectangle, sphere, and cone. All the primitives managed by the physics engine are supported and thus displayable. For example, we can associate the model of a cylinder to the wheel of a robot. In addition, we can link hierarchically some objects. In our experiments, for instance, we link the omni-directional camera to the robot's turret to move and rotate it consistently with the robot.

The graphics engine interfaces with the physics engine through an intermediate layer creating the graphical object and the 3D model associated to every simulated physical body (the robots, obstacles, walls). The graphical object then automatically receives updated position and orientation from the linked physical body, realizing a perfect consistency between the physical simulation and the display of objects.

EA engine: This component of the simulator is responsible for the artificial evolution of neural network controllers. The employed algorithm is described in section 3.3.

Robotics engine: This component includes tools for the simulation of many objects, for the control of robots and for the programming of experiments. Extensive examples are given in the appendix.

3.2 The *S-bot* model

The robot employed for our research is the *s-bot*, the robot designed within the SWARM-BOTS project. This section presents the *s-bot* hardware, the simplified simulation model we used in our evolutionary experiments[6], and the characteristics of this robot's vision system.

3.2.1 Hardware

An *s-bot* is a fully autonomous robot capable of performing basic tasks such as autonomous navigation, perception of the environment and grasping of objects. One *s-bot* is also able to physical connect to other *s-bots* in flexible ways and to communicate with them. A formation of *s-bots* is called *swarm-bot*, capable of performing tasks like navigation, exploration and transportation of heavy objects,

5 Frustum culling consists in discarding invisible objects before rendering, thus saving processing power

6 Details regarding the hardware and simulation of the swarm-bot can be found on the project web-site (http://www.swarm-bots.org).

which a single robot cannot tackle without encountering major problems. Figure 5 shows the latest *s-bot* prototype produced.

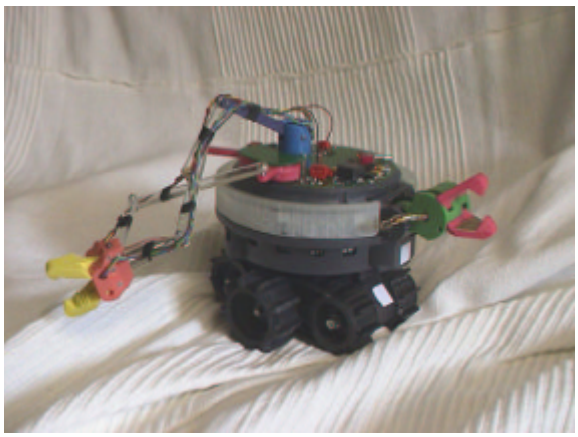

Figure 5: The first *s-bot* prototype, provided of the tracks system, the body holding the rigid and the exible grippers, and many sensor systems.

The mobility of the *s-bot* is ensured by a combination of two tracks and two wheels, labeled *Differential Treels© Drive.* The same motor drives the wheel and track on a same side, building a differential drive system controlled by two motors. This hardware design allows an efficient rotation on the spot due to the position of the wheels and makes navigation simpler on moderately rough terrains (a *swarm-bot* can tackle situations that are more complex). The chassis can rotate with respect to the main body (turret) by means of a motorized axis. This ensures an independent movement of the turret where the sensors and the grippers for physical connections to other *s-bot*s or objects are located. Sensors and actuators are controlled by electronics mainly included in the central *s-bot* body, integrated on a Linux board.

Sensors. Each *s-bot* is equipped with all the sensors necessary for navigation and communication with other *s-bots*. The available sensors are infrared proximity sensors, light and humidity sensors, accelerometers and incremental encoders on each of the nine degrees of freedom. In addition, the robots have color LEDs, local color detectors all around the body, one speaker and three microphones, as well as one wireless LAN for remote debugging and monitoring. The robots also have one omni-directional camera located over the body with the support of a transparent tube, acquiring 360-degree panoramic views of the environment around the *s-bot*. The robots can also receive information about physical contacts, efforts, and reactions at the interconnection joints with other *s-bots* through additional sensors, like torque sensors on most joints and traction sensors on the connection belt. Every sensor has a definite range: infrared proximity sensors have a limited short range, while the camera covers both short and long ranges, depending on the features extracted from the image. Indeed, many researches on collective insects showed that collective

behaviors are often based on multi-range and multi-modal sensing in order to perceive and exchange signals at multiple levels.

Actuators. The *s-bot* actuators control the two differential treels for movement, 8 RGB LEDs and 2 speakers for communications, an elevation and a rotation motor, and 2 servo grippers and an arm motor to lift or grasp rigidly other robots.

3.2.2 Simulation Model

The simulator provides *s-bot* models with the functionalities available on the real *s-bots*. Almost all the sensor and actuator devices are correctly simulated. For our experiments, we used only the omni-directional camera to acquire local information about the robots' environment. In particular, this thesis addresses in chapter 4 the problem of simulating the images acquired by omni-directional camera. There are four, different reference models with increasing levels of detail. The less detailed models have been employed to speed up the process of designing neural controllers through evolutionary algorithms. The most detailed models have been employed to validate the evolved controllers before porting them on real hardware. For our experiments, we used the model with a low level of detail (Figure 6) in order to run fast simulations still preserving the relevant features of the real *s-bot*.

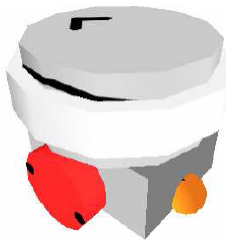

Figure 6: *S-bot* model used in our experiments.

Here, the *s-bot* turret is modeled as a cylinder, connected to the chassis by a motorized joint. The chassis is a sphere to which 4 spherical wheels are connected, two lateral and two passive wheels in the front and in the back. The wheels and the chassis are modeled as spheres to speed up the computation of dynamics. The lateral wheels are connected to the chassis with two hinge or car-wheel joints, depending on the need of the simulation (flat or rough terrain, respectively), and they simulate the *treels* system. The other two wheels are smaller and only serve as support, not being motorized. They are connected to the chassis with a ball-and-socket joint positioned at the center of the sphere, in order to leave the wheels free to rotate in every direction. The gripper is not modeled in our experiments. In this model, the omni-directional camera is an invisible object rigidly located over the robot's turret.

We disabled collision detection between all the bodies composing the model, in order to accelerate the physical simulation of the *s-bot*. Besides, all wheels and the

chassis do not collide with any other objects in the environment, except with those objects that constitute the ground. Therefore, only the turret can collide with other *s-bots*, walls and obstacles.

A unit in the model corresponds to 2 cm. All the measures henceforth mentioned throughout this thesis scale correspondingly.

3.2.3 Vision System.

Every *s-bot* acquires in real-time full images of the surrounding environment through its omni-directional camera. This camera is placed above the robot's body with the support of a transparent tube. It is made of two devices: a spherical mirror and a CCD camera. The mirror is an optical device that reflects rays of light coming from the environment towards the camera, building a panoramic view. The latter is an electrical device that transforms the received optical signals into electrical signals for further processing.

With reference to Figure 7, the acquired images have the following, main characteristics:

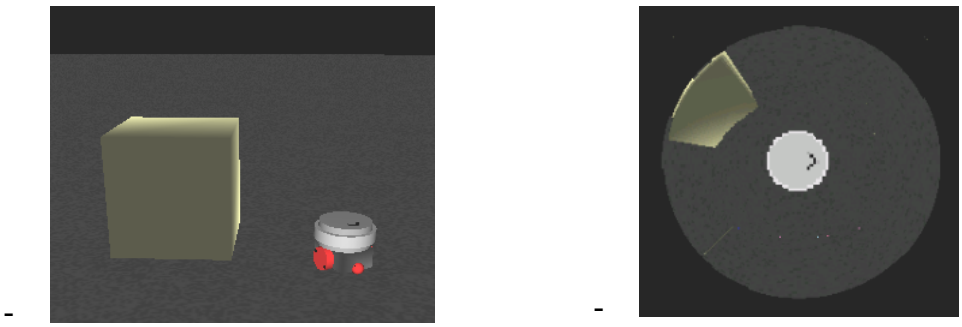

Figure 7: Left 3D scene with robot. Right: *S-bot*'s view.

Circular aspect. The spherical mirror reflects the environment towards the camera. This optical process generates images of the environment that appear circularly warped. For instance, the top and bottom edges of the cube in Figure 7 appear bent instead of straight. Mathematically, the three dimensional representation of the objects in the scene is reconstructed onto the camera image by a non-linear mapping, introduced into the optical system by the spherical mirror. This leads to the distortion of objects.

Radial stretching. Moreover, objects appear stretched in the radial direction because the mirror does not preserve the scaling aspect of the reflected objects. The local curvature of the mirror at the point of reflection determines the convergence or divergence of bunches of rays, altering the proportion between distances from world space to the image (see figure below). This process, when applied to the whole image of an object, leads to radial distortion.

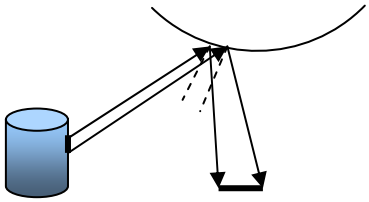

Figure 8: Radial stretching

Invariants. Finally, the image always shows the reflected image of the s-bot in the center (Figure 7, right). The reason is that the spherical mirror is rigidly located on top of the robot and thus it always sees and reflects the robot downward on the camera. Furthermore, the peripheral area of the image always shows the background (the sky or ceiling). The pixels of this area in fact correspond to light rays arriving from the background. This property breaks only in the unlikely case of one robot losing contact with the horizontal ground.

In our experiments, the optical process creating these panoramic images is simulated (chapter 4). The resulting images are colored bitmaps of 64x64 or 128x128 pixels resolution, subsequently processed by the robot's vision system.

3.3 The Evolutionary Algorithm

As anticipated in section 2.2.3, we used artificial evolution to develop the neural controllers for our experiments.

Evolutionary Algorithms (EA) are a method of finding solutions to optimization or search problems by means of simulated evolution. Some processes based on natural selection, crossover, and mutation are repeatedly applied to a population of individuals, in the form of genotypes, that represent potential solutions. Over time, the average performance of the individuals increases, until a good solution to the problem is found. In our experiments, one individual genotype stores the weights of a potential neural network, and EA must find iteratively the best neural network that produces the desired behavior.

EA start from a population of randomly generated individuals. Next, the fitness of all of the individuals in the population is evaluated, where the fitness is a criterion that quantifies how well an individual performs the desired task. After that, EA create a new population by performing operations such as *reproduction*, *crossover*, and *mutation* on the individuals whose fitness has just been measured. These genetic operations, in concert with the fitness measure, operate to improve the population. Lastly, the algorithm discards the old population and iterates using the new one

Reproduction consists in making copies of the best individuals in the new population. In our experiments, we implemented selective reproduction with the *roulette wheel* method [48].

Once selective reproduction has created the new population, offspring are randomly paired, crossed over, and mutated, providing general heuristics for the exploration of the landscape of potential solutions.

Crossover consists in swapping genetic materials between two individuals around a random point of their genotype. In one-point crossover (figure below), two parent genotypes are cut at the same point and offspring are formed by combining complementary genes from the parents (i.e., the first part of parent 1 with the second part of parent 2 and vice versa).

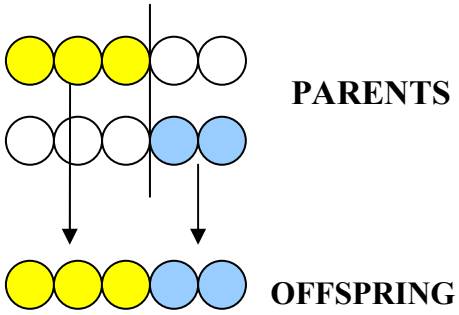

Figure 9: One-point crossover

While the crossover operation leads to a mixing of genetic material in the offspring, no new genetic material is introduced, which can lead to lack of population diversity and eventually stagnation - where the population converges on the same, non-optimal solution. The mutation operator helps to increase population diversity by introducing new genetic material. It consists in making a random change to one or more randomly chosen genes in an individual (Figure 10). For binary representations, mutation switches the value of the selected bit. For real value representations, the real value is mutated with a small random number (usually drawn from a Gaussian generator centered on zero).

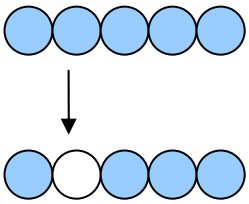

Figure 10: Mutation operator.

In our experiments, an individual corresponds to a string of real values that are the weights of a neural network under evolution. First, we clone the neural network in each robot's controller involved in the experiment. Next, we estimate the network fitness according to the desired behavior, by averaging the fitness of some trials:

$$F = \frac{1}{M} \sum_{i=1}^{M} F_e$$

where F_e is the fitness estimation obtained from a single trial, dependent on the evolved behavior, and M is the number of trials. Each trial lasts a maximum of T simulation cycles, each cycle corresponding to 100 ms of real time. The average of M trials helps in correctly estimating poor individuals that obtain high fitness only because of lucky experimental conditions.

We used neural networks controllers because they have the property that their performance gracefully degrades with respect to alterations in weights and thresholds, resulting in smoother evolutionary fitness landscapes [48]. Moreover, a variety of authors have reported successful application of artificial evolution to develop many types of neural networks for the control of robots [63][64][52][43].

The neural network controller, the fitness function and other settings of the evolutionary algorithms vary from experiment to experiment and are described in the respective chapters.

4

SIMULATION OF THE OMNI-DIRECTIONAL CAMERA

This chapter describes the simulation of the omni-directional camera. The first section of the chapter illustrates the optical process that generates the images acquired by the camera. Here, we introduce the technique of *ray-tracing*, taken from the field of Computer Graphics, which correctly reproduces the images generated by the real camera optics. In particular, the simulation uses novel techniques to heavily optimize ray-tracing for real-time purposes, which are described in section 4.2.

4.1 Ray-tracing

The process that generates the camera images follows the rules of optical geometry: rays of light arrive from the environment and eventually reach the spherical mirror that is part of the whole camera device. The mirror then reflects the rays towards the camera where the image is formed (see Figure 11).

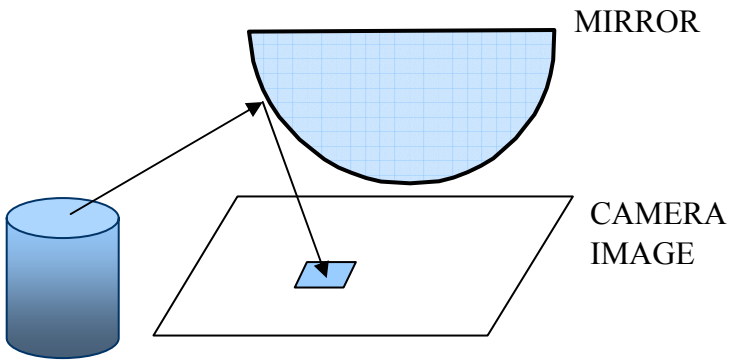

Figure 11: Optical process forming omni-directional images.

The simulation of this optical system requires the technique of *ray-tracing* [25]. Ray-tracing is a well known rendering technique in the field of Computer Graphics (CG) that naturally generates photorealistic images in the presence of reflective surfaces. This technique essentially replicates the behavior of optical geometry in

creating an image. Ray-tracing works as follows: the ray associated to every image pixel is traced from the point of view back into the scene until it intersects an object (the background in the limit). At this point, the ray is split in the corresponding reflected and refracted rays according to the laws of physics and the material properties of the intersected surface. These new rays, weighted by the surface reflectance, are then recursively shot in the scene following the same procedure, leading to a binary tree of rays (see figure below). Recursion stops when a ray reaches the background or a certain threshold (number of rays, tree depth…) is exceeded. The final tree of weighted rays determines the color of the pixel.

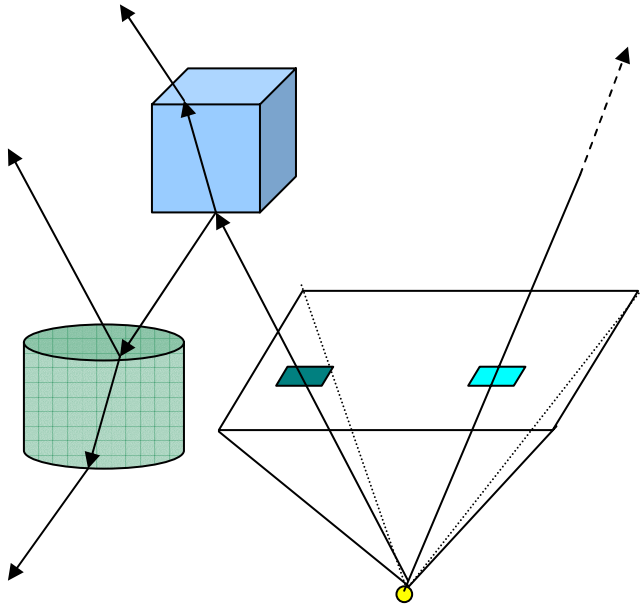

Figure 12: The principles of ray-tracing

Conventional rendering techniques running on common graphical hardware are not feasible to simulate this optical process because they involve linear mappings of geometrical representations from world space to image space. The spherical mirror introduces a non-linear mapping that only ray-tracing techniques can naturally handle.

In order to simulate the omni-directional camera of an *s-bot,* we use a simplified version of ray-tracing that takes into account only first-order reflections of rays. Every ray starting from the camera's point of view is tested for intersection against the spherical mirror and, in case, the corresponding reflected ray is traced in the scene until it reaches an object. We then assign the color of the intersected point to the image pixel. If the first or the reflected ray does not intersect any object, then we set the pixel to the color of the background. Figure 13 shows this process.

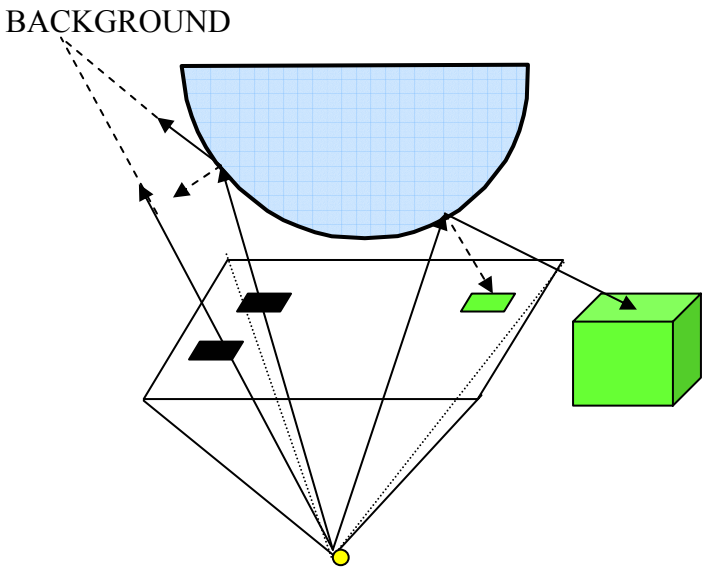

Figure 13: Simplified ray-tracing.

4.2 Optimized Ray-tracing

The above procedure has very weak points. First, the high number of intersection tests slows down the process dramatically. In the worst case, we must test every ray against the mirror and, if reflected, against all the geometrical primitives of the scene. The model of an *s-bot,* especially the detailed one, is made of thousands of primitives thus making the cost of ray-tracing prohibitive. Even with high-level optimizations based on proper organization of scene geometry in friendly data-structures, the procedure would be too demanding. Furthermore, ray-tracing needs information about the reflective properties of the intersected surfaces (only the material color in our simplified approach). In our experiments, the available information consists of geometrical meshes, texture maps and simple lighting properties, used by the graphic card to render the objects. A complete ray-tracing scheme would then replicate in software the operations done by the graphic hardware, mainly texture mapping and lighting, to retrieve the color of the intersected points. The latter constraint makes ray-tracing in software unpractical.

We have developed a hybrid technique to overcome the aforementioned limits. This technique takes advantage of common graphic hardware to critically accelerate the last, most expensive, step of ray-tracing. In addition, it relies on full pre-computation of the operations involved in tracing rays from the camera to the mirror. Further optimizations have been implemented as well. Thanks to this technique, the simulation of the omni-directional camera becomes feasible for evolutionary development of controllers for tasks of *active-vision.*

4.2.1 The Idea of Cube Map

The technique exploits the idea of *cube map.* A cube map is a cube whose faces are texture maps that show the projection on the cube of the scene around a point; each face covers a 90-degrees field of view in the horizontal and vertical. One relevant property of the cube map is that each pixel represents the scene viewed in the direction going from the center to the pixel. For this reason, a cube map is often used to store directional information about the scene around one point (the center of the cube map), like the luminance in lighting techniques, or colors in our vision system, as shown by Figure 14.

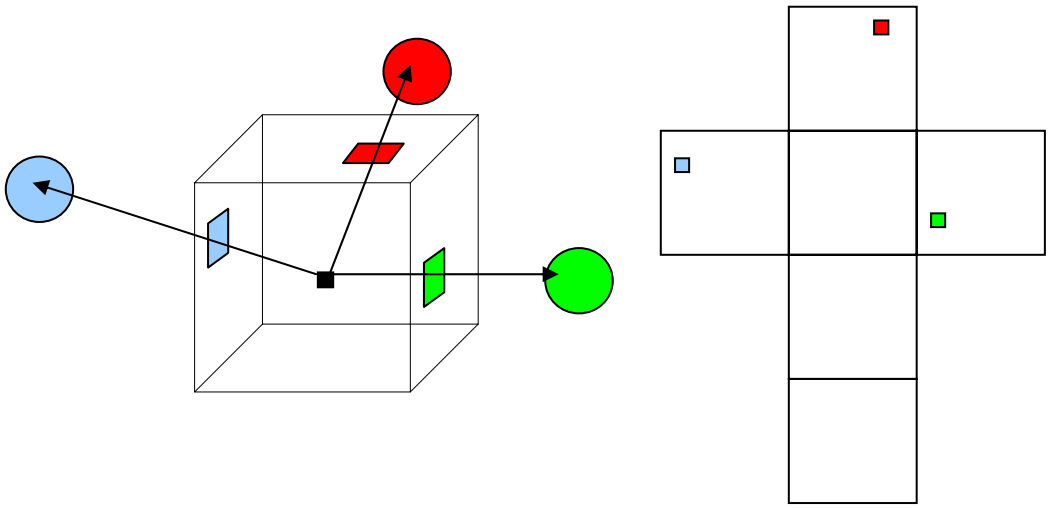

Figure 14: Left: 3D cube map of a scene. Right: unfolded version

We setup a cube map around the camera's spherical mirror such that its center coincides with the inner center of the mirror (Figure 15).

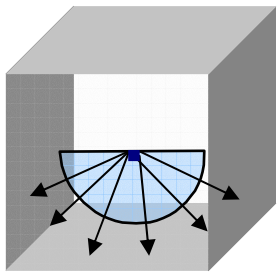

Figure 15: Cube map setup.

At each simulation step, we render the scene on each of the six textures of the cube map using a perspective projection with a field of view of 90 degrees both in the horizontal and vertical. The textures now show the scene around the mirror in all directions.

Our technique takes advantage of the cube map to significantly speed-up ray-tracing. Again, we first shoot a ray from the camera's center of projection towards the mirror. If the ray meets the mirror, it is reflected in a direction dictated by the well-known Snell's law of reflection:

$$\theta_i = \theta_r$$

where θ_i, θ_r are the angles between the mirror's normal vector at the intersection point and the incident ray and reflected ray respectively (Figure 16).

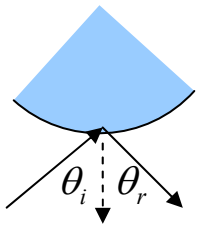

Figure 16: Snell's law of reflection

Now, instead of tracing the reflected ray into the scene, we read the cube map pixel corresponding to the direction of this ray. It is important to underline that the cube map encodes the color correctly seen by rays having origin in the mirror's center. The rays reflected by the mirror instead start from the mirror's surface, thus we have to find the ray from the center that best approximates the reflected ray[7]. We can easily obtain it by taking a point on the reflected ray far from the origin; the ray connecting this point to the mirror's center is the searched approximation (Figure 17).

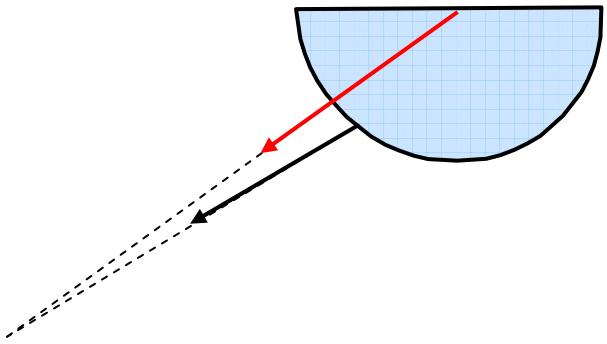

Figure 17: Reflected ray (black) and its approximation (red).

[7] Tests showed that this approximation does not affect the quality of simulation in a perceivable way.

After that, we calculate the intersection between the approximated ray and the cube map[8], and we copy the intersected pixel to the image pixel. We must repeat this process for all the pixels of the camera image.

The following algorithm summarizes the simulation of the omni-camera using the cube-map.

```
setup a cube map centered on the mirror
…
for each control step
begin
    for each side of the cube map
    begin
        render the scene on this side
        read back side's texture to system ram
    end
    perform ray tracing between the image and the cube map
end
```

The time required to update the cube map mainly depends on the speed of the employed graphic card. On modern hardware, this time ranges from one to two milliseconds on average for simple scenes with few robots. The graphical engine uses *frustum culling* to alleviate the work done by the graphic card. This basic optimization heavily reduces the amount of work necessary to render the scene six times because it discards invisible objects before rendering them on the cube map's sides. Note that the cube map textures must be read back from video to system ram to be processed by the final stage of the algorithm. We propose a solution to this problem in the next section.

4.2.2 Mapping Table

At this point, our technique introduces a strong optimization that concerns the last part of the algorithm. As already described, this part consists in finding, through ray-tracing, which pixel of the cube map corresponds to each pixel of the camera image. This is a time consuming process that involves one demanding ray-tracing for every camera pixel at each control step.

---

[8] Since the origin of the approximated ray coincides with the center of the cube, an optimized intersection test is available. The maximum absolute coordinate of the ray gives the side where the intersection occurs. The other two coordinates are then divided by this maximum and remapped from the range [-1, +1] to the range [0, *s*], where *s* is the size of the texture map associated with that side (supposed square). The resulting 2D coordinates are used to access the texture map.

Luckily, this work turns out unnecessary because the image plane and the spherical mirror have fixed relative positions. In a coordinate frame rigid with them, the geometrical path of rays starting from the center of projection of the camera, and possibly reaching the mirror and the cube map, does not change. Therefore, we can completely pre-calculate which pixel of the cube map corresponds to every image pixel. We store this information in a *mapping table.*

The mapping table has the same resolution of the camera image and tells, for every image pixel, the corresponding side and pixel of the cube map (Figure 18). If the viewing ray associated with the pixel misses the mirror, the table stores a special code that indicates the background.

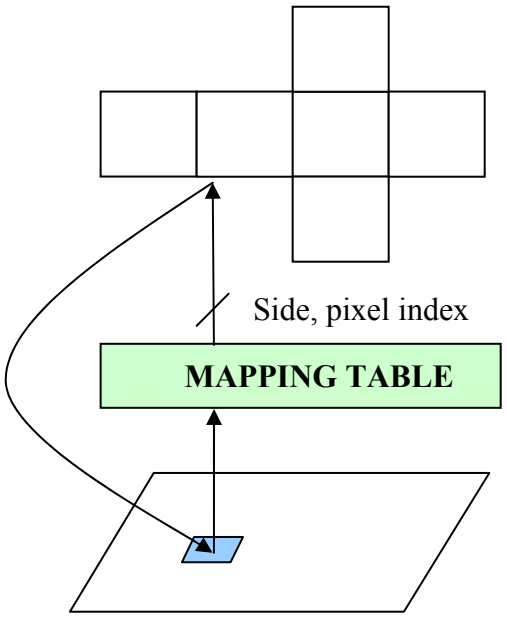

Figure 18: Using the mapping table

At run time, we update the cube map as usual by rendering the scene on its sides. After that, we build the camera image by setting each image pixel to the corresponding cube map pixel, according to the mapping table. Thanks to this optimization, the cost of ray-tracing has been substituted by the cost of a table lookup, making the algorithm already practical for evolutionary tasks requiring maximum efficiency in the simulation phase. The new algorithm is the following:

```
setup a cube map centered on the mirror
pre-calculate ray-tracing in a mapping table
…
for each control step
begin
     for each side of the cube map
     begin
```

```
        render the scene on this side
        read back side's texture to system ram
    end
    copy the cube map on the image using the mapping table
end
```

The simulation of the omni-directional camera now runs partially on the graphic card, which renders the cube map, and finally on the CPU, which performs the mapping between the cube map and the camera image. This procedure presents a serious performance bottleneck in the transfer of textures from video to system ram. This transfer is quite slow on current AGP buses due to limited bandwidth.

The problem finds solution on modern graphic cards that support rendering to textures and programmable hardware with *dependent texture fetches*. The first feature means that the graphic card is capable of outputting rendering operations to textures residing in video memory. The six textures of the cube map and the camera image require this feature. The second feature means that the graphic card can execute short programs running user-specified instructions to create a pixel. In our case, we need a program that performs the mapping between the cube map and the camera image directly in hardware. This program must read from the mapping table, stored as a texture, the ray direction corresponding to each camera pixel, and use this ray to fetch the cube map. This is an example of dependent fetching because we use data read from a texture to access another texture. The new algorithm is the following:

```
setup a cube map centered on the mirror
pre-calculate ray-tracing in a mapping texture
…
for each control step
begin
    for each side of the cube map
    begin
        render the scene on this side
    end
    render camera image using programmable hardware
    read back camera image for processing
end
```

The above process has the big advantage of running entirely on the graphic card, on hardware heavily optimized for pixel operations, and without expensive data transfers between the graphic and the system memory. Actually, we must read back the camera image to process it on the CPU side, but this cost is marginal compared to the cost of transferring six textures implied by the original algorithm. Unfortunately,

we did not exploit this optimization because the required hardware was not available on all the computers.

4.2.3 Minimal Rendering

The final high-level optimization of our technique concerns the update of the cube map at every control step. As explained, this consists in rendering the scene six times, one for each cube side. The update of *all* the sides is actually unnecessary under some circumstances.

First, some sides of the cube map may never contribute to the creation of the camera image (for example the top side). This happens when no rays reflected by the mirror intersect those sides, in which case none of their pixels is mapped on the camera image by our technique.

Besides, the rendering of the whole camera image is not always required. Vision-based controllers make use of a virtual retina, a region of the image of a certain shape and size, moving and zooming on the image. The controllers process only the pixels covered by the retina ignoring the rest of the image. Therefore, it is necessary to update only the sides of the cube map that contribute to these pixels. Figure 19 shows an example of the above situation, where only one side of the cube map influences the image area corresponding to the retina.

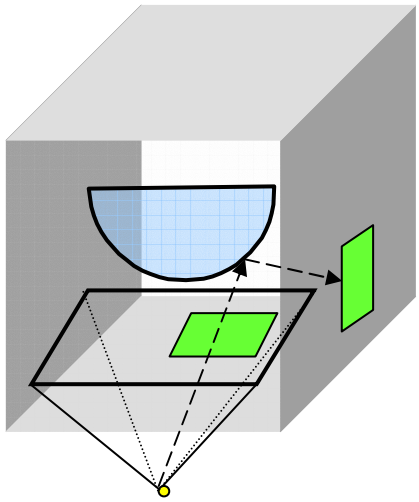

Figure 19: Example of minimal rendering. Only the right side of the cube map is necessary to render the retina region (in green).

Given a set of destination pixels (the retina or the full image), the optimization task consists in finding which sides are strictly necessary to render these pixels. A side is necessary if and only if it contains at least one pixel that will be mapped on at least one image pixel.

The solution to this problem is straightforward because the mapping table already stores the index of the cube map's side associated, through ray-tracing, to every pixel of the camera image. Therefore, we must loop over all the destination pixels and mark as active the corresponding side, read from the mapping table. Later, we will update only the active sides of the cube map instead of all the six ones. The following code describes this procedure.

```
mark all sides as non-active
for each destination pixel (x,y)
begin
    side = mappingTable(x,y):side
    side:active ← true
end
```

4.2.4 *Quadtree* Preprocessing

The operation of determining the sides of the cube map required to render the retina or the full image involves a loop over all the destination pixels, which can become rather costly for large regions. Given that the correspondence between a pixel and the associated cube side is fixed at the beginning, we can conveniently pre-calculate the necessary sides for predefined sub regions of the image.

We adopted a *quadtree* data structure to organize that information hierarchically. A quadtree is a rooted tree where every internal node has four children. In our case, every node in tree corresponds to a square region of the camera image; the root node stands for the whole image while leaf nodes correspond to the minimum regions handled by the pre-processing system.

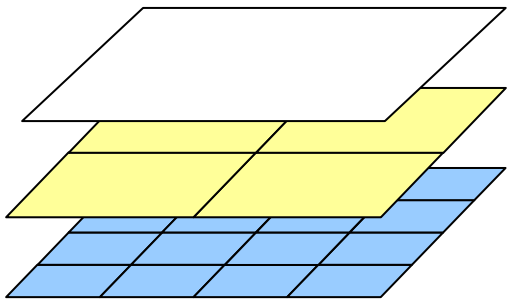

Figure 20: First three levels of the quadtree.

Each node covers all the pixels of its four children, therefore it requires all the sides required by its children together. In our implementation, every node stores a byte that tells whether the sides of the cube map are necessary or not to render the corresponding image region. The *i*-th bit of the byte is set to 1 if the *i*-th side is

necessary, 0 if not. A logical OR between the bytes of two nodes gives the union of the sides of both.

We pre-process the image in a bottom-up fashion taking into account this property. First, we calculate the byte stored with each leaf node by looping over all the node's pixels, according to the procedure described in section 4.2.2. Then, we pass this information from the leaf nodes up to the root, by calculating the byte of a parent node as the logical OR of the children bytes.

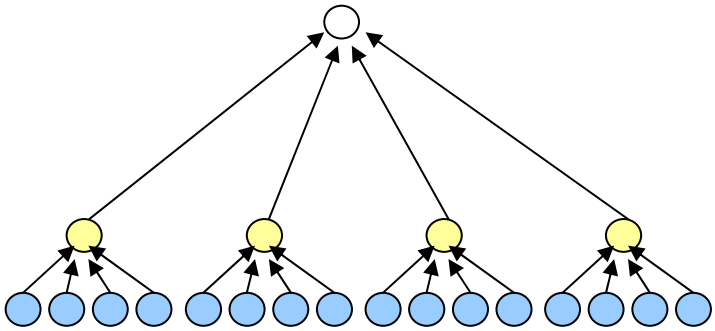

Figure 21: Quadtree pre-processing.

Now, we want to determine the sides required by regions of generic shape and size, as the virtual retina, taking advantage of the pre-calculated information stored in the quadtree. At this purpose, we first calculate the bounding rectangle of the region, which is the minimum rectangle that fully includes the region. Next, we have to retrieve the largest nodes of the quadtree fully contained in that rectangle (Figure 22, in blue), an operation achievable through a recursive splitting of the rectangle down the quadtree. The bytes stored with these nodes tell the sides necessary to render the corresponding parts of the rectangle. Finally, we must calculate the sides associated to the remaining parts of the rectangle (Figure 22, in red) using the procedure described in section 4.2.2.

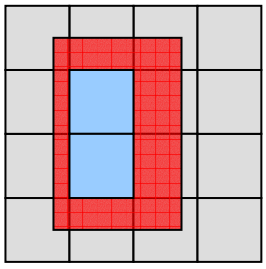

Figure 22: Quadtree splitting of a rectangle.

At the end, we have the complete set of cube-map sides necessary to simulate the omni-directional camera image in the area covered by the rectangle, which includes the virtual retina we are interested in. The final algorithm is the following:

```
setup a cube map centered on the mirror
pre-calculate ray-tracing in a mapping table
for each control step
begin
  calculate bounding rectangle of region
  determine active sides
  for each active side of the cube map
  begin
      render the scene on this side
      read back texture to system ram
  end
  copy the cube map on region using the mapping table
end
```

5

VIRTUAL RETINA

This chapter is dedicated to the description of the virtual retina device used by the vision system to extract relevant features from the images acquired by an omni-directional camera. The first section introduces the general characteristics of a retina, and the related rendering and filtering operations. Sections 5.2 and 5.3 illustrate two kinds of retina: the rectangular retina and the circular one.

5.1 General Characteristics

The vision system of every robot is equipped with a virtual retina capable of autonomously scanning the images acquired by the robot's camera. The retina corresponds to a limited area of the image, of variable position and size. It can zoom in and out and move across the image to extract visual features. For example, an intelligent retina would assume a large size when the vision task requires global information on the environment seen by the camera. On the contrary, the retina would zoom in to extract features from the image, as objects' details. Thanks to this ability, the retina can constantly detect and focus on environmental features, providing relevant information for the achievement of tasks concerning robotic navigation.

The retina is composed of an *n*-by-*m* matrix of visual cells whose receptive fields receive input from the corresponding area of the image. In our experiments, every retinal cell stores visual information about the environment and feeds one input neuron of the neural network controlling a robot.

The resolution of the retina affects the performance of the vision system in two ways. From one side, high resolutions imply a large size of the neural network fed by the retina; therefore, the artificial evolution of the network becomes more complex and time-consuming. On the other side, retinas characterized by a low resolution cannot efficiently detect small details, which might be fundamental in advanced vision tasks involving distant and small objects. As a compromise, we employed in our experiments the intermediate resolution of 4x4 cells.

Theoretically, more retinas may work simultaneously for advanced image processing. Practically, we do not suggest the use of multiple retinas because the added complexity may overcome the benefits offered by a unique, simple retina.

Additionally, the use of multiple retinas highly increases the computational time required for artificially evolving controllers capable of governing such a vision system correctly. As a result, in our experiments only one retina operates on the camera images.

5.2 Rendering

The simulator renders the images of an omni-directional camera with optimization strategies that take into account the virtual retina. Indeed, the vision system of each robot processes through filtering only the pixels covered by the retina, ignoring the rest of the image. Hence, we can optimize the camera simulation by rendering only the pixels covered by the retina, according to the procedure described in section 4.2.3. Thereafter, the retina can operate directly on the image or, alternatively, on a secondary buffer storing a copy of the retinal area (e.g. for writing operations that do not change the original image). In our implementation, the retina works directly on the simulated camera image.

5.2.1 Filtering

Every retinal cell covers a group of image pixels. The three R (Red), G (Green), B (Blue) color channels of these pixels are filtered and the resulting values associated to the cell. This triple of values (in our experiments, only their luminance) is then passed in input to the neural network for processing.

Two basic filters are available: *average* and *sampling*. The average filter calculates the arithmetical average of the filtered pixel colors:

$$cell_i(\alpha) = \frac{1}{N}\sum_{j=1}^{N} pixel_{i,j}(\alpha)$$

where N is the number of pixels belonging to the i-th cell and α is the color channel (red, green and blue).

This filter has the advantage of considering all the pixels of a cell but it loses fine details because it cuts the high frequencies of the filtered data. The sampling filter instead takes the color of the first pixel as result.

$$cell_i(\alpha) = pixel_{i,0}(\alpha)$$

This filter preserves the high frequency contents of the source image but it samples only one pixel to represent the whole cell, which may lead to aliasing problems on high-frequency images.

More filters are possible, based on different kernels. We can select one or the other filter by hand or we can let the controller select the best filtering strategy. It

would be even possible to let evolution develop the filtering kernel that best matches the task under control. This would unfortunately lead to an intolerable increase in the time and complexity of an evolution. For simplicity, we used the average filter by default in our experiments.

Before filtering, we have to find the image pixels covered by a cell. This operation is analogous to fill the interior of the cell (in our case, we have to read the pixels instead of writing) and is achievable by means of *scanline rasterization* algorithms [25]. The circular retina, described later, makes use of such an algorithm.

5.3 Rectangular Retina

The first devised retina is rectangular and aligned to the image (rotation is not supported). This retina can move over the image both in horizontal and in vertical direction in search of relevant features. It can also zoom in or out by changing its size. The retina is divided in a grid of *n*-by-*m* visual cells receiving input from the corresponding region of the image. Table 1 describes the retinal parameters, along with their range.

Table 1: Parameters of the rectangular retina.

Parameter	Description	Range
Position	Position of the top-left corner of the retina, in pixels	x: from 0 to *w*-1, where *w* is the width of the camera image y: from 0 to h-1, where h is the height of the camera image
Size	Size of the retina's sides, in pixels	From one or two pixels per cell, to the size of the whole image

The rectangular retina suits generic images. Therefore, we can use it even outside the robotic context, for instance to efficiently process images on systems limited by low-computational capabilities.

5.3.1 Filtering

The visual cells of this retina are rectangular, easing the problem of finding the corresponding image pixels. The calculus involves two nested loops, in horizontal and in vertical, over the image pixels covered by the cell. When the average filter is selected, each cell takes the average value of the three R, G, and B color channels of its pixels, calculated on the fly throughout the loop. Otherwise, when the sampling filter is selected, the cell takes the value of its first pixel at the top-left corner.

5.4 Circular Retina

The second retina we developed for our experiments has the shape of a circular sector (Figure 23). This retina can rotate around the center of the image and can translate in the radial direction. It can also zoom in or out by changing aperture and length. The circular retina is divided in a matrix of *n* X *m* visual cells, obtained by slicing the retina *n* times in the radial direction and *m* times in the angular. This kind of retina optimally adapts to the images acquired by the omni-directional camera of our robots, characterized by the circular projection of the perceived scene elements around the image center. In addition, this retina automatically excludes from processing the central area of the image, which always shows the reflection of the robot mounting the camera and it is not interesting for the control of the robot itself.

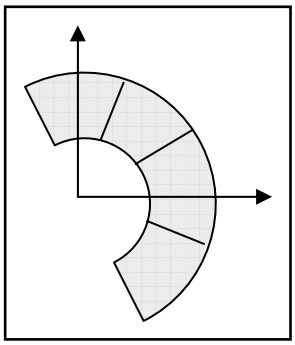

Figure 23: 4 x 1 circular retina with an aperture of 180 degrees.

Table 2 describes the circular retina's parameters and their range:

Table 2: Parameters of the circular retina.

Parameter	Description	Range
Angle	Orientation of the retina around the center of the image, in degrees	From 0 to 360 degrees
Distance	Distance from the center of the image	From 0 (nearest) to 1 (furthest)
Aperture	Angular extension of the retina	From 15 to 360 degrees
Length	Radial extension of the retina	From few pixels per cell (2, 3) to about half diagonal of the image

The retina's size and position are expressed in a coordinate system having origin in the center of the image. The retina rotates around the image following changes of the *angle* parameter. In the design of the neural network controlling the retina, we chose to preserve correlation between successive retinal acquisitions, by limiting the

variation of the *angle* parameter at every control step to 15 degrees (an empirical value). We also employed a control method that gives a stable angular movement, described in section 6.2.3.

5.4.1 Filtering

The cells of the circular retina have a convex contour defined by two straight lines and two circular arcs at the top and bottom. Given these characteristics, the problem of finding the pixels inside a cell is much more complex than in the case of the rectangular retina.

As already explained, *rasterization* algorithms were designed to draw one shape on a map of pixels (e.g. the screen). The standard algorithms determine the pixels inside the shape line-by-line (*scanline*) through a process called *scanline conversion*. This process consists in finding the intersections between horizontal scanlines and the shape's contour, which correspond to the extremes of rows of pixels (*spans*) inside the shape. Figure 24 shows these concepts applied to a triangle.

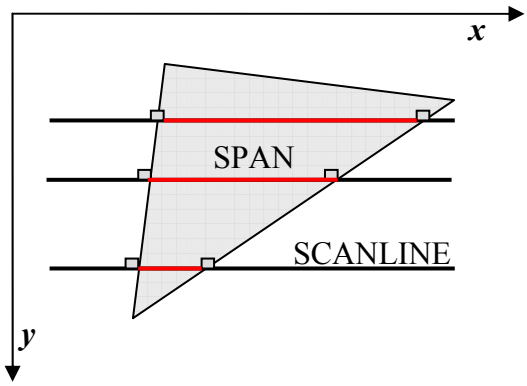

Figure 24: Scanline conversion of a triangle

Conventional rasterizers work on convex polygons, characterized by the geometrical property of having only one internal span for each scanline, whose extremes are easily calculated by drawing the polygon's edges with modified line-drawing functions. There are extensions to these algorithms supporting concave polygons and curved outlines at the cost of increased complexity.

We decided to develop our own algorithm for the circular retina. This algorithm works again on the same basis of rasterizers for convex polygon to determine the pixels inside a cell, with two differences. First, it handles the case of multiple spans per scanline introduced by the cell's concavity. Second, it adopts special drawing functions designed for circular arcs to determine the points on the top and bottom arcs of a cell.

In order to manage multiple spans per line, the algorithm maintains a table that stores the points on the cell's contour for each scanline. This table can store a

maximum of 4 points per *y* coordinate, for a number of scanlines equal to the cell height. It stores the 2D coordinates of points, along with a bit flag that classifies points in *entry* or *exit* according to their orientation with respect to the interior of the cell (see Figure 25). Internal spans are those connecting *entry* to *exit* points from left to right.

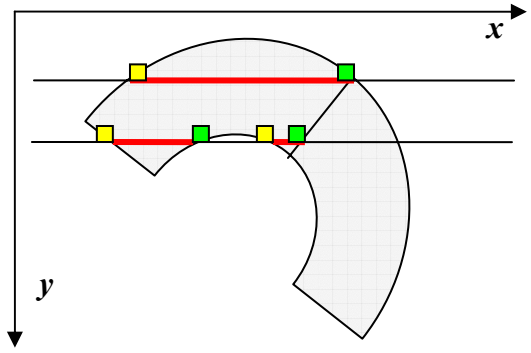

Figure 25: Scanline conversion of a cell in a 2 x 1 retina. Yellow squares represent entry points (start of spans), while green ones represent exit points (end of spans).

A modified line-drawing function calculates, for each *y* coordinate, the points lying on the two lateral lines of the cell. Thereafter, these points are classified and inserted into the table. Another function draws the top and bottom circular arcs of the cell along the *y* direction. This task is more complex that in the case of straight lines. In literature, there exists a variety of algorithms to draw circular arcs, most of them derived from the well-known circle-drawing algorithm by Bresenham [10]. We adapted one of these algorithms for our purposes. The algorithm determines, line by line, the point(s) on the arc intersected by the working scanline (a scanline can intersect an arc in one or two points).The points are then classified and inserted into the table.

At the end of this work, the table stores the points that limit the spans internal to the cell. For each scanline, we first sort these points from left to right according to their *x* coordinate. Then, we examine the sorted points and we consider spans that connect *entry* to *exit* points from left to right. When the average filter is selected, we calculate the average, for the three R, G and B channels, of the pixels belonging to all these spans. Otherwise, if the sampling filter is used, we set the cell's value to the color of its first pixel.

6

EVOLVING TASKS: FOCUS

This chapter describes our first experiment of active vision. The first section introduces the experiment along with the motivations behind it. Section 6.2 describes the experimental setup, with details on the simulated scenario, the vision system, the neural network controller and the fitness function used to evaluate controllers. The last section, 6.3, discusses the obtained results and the generalization properties of the evolved controllers.

6.1 The Focus Task

This experiment consists in evolving a virtual retina capable of focusing on static objects viewed by an omni-directional camera. Here, the camera does not move because it is not attached to a robot. It acquires at every control step images of the scene containing a random object located in a black room. The object appears in the image as a bi-dimensional figure of varying shape, position and color. The retina must localize the object in the image and then zoom in or out in order to focus on the object in the best possible way. This means that the retina must cover the whole object, and not only a fraction of it, assuming the minimal size strictly necessary to see the whole object.

This experiment represents the first step in the study of controllers capable of features extraction from camera images. We want to prove, through this experiment, that the virtual retina is an efficient tool to extract automatically from images relevant features that can be useful to solve vision-based tasks. In this case, the task coincides with the recognition and focusing of generic objects. In the next experiments, we will use the same type of retina for more advanced tasks involving mobile robots (dynamic targeting and swarming). We will show that the retina can recognize objects in dynamic environments and provide information on the objects' position. Robots can then use this information for tasks like navigation and swarming.

Moreover, this experiment serves to validate the choices taken about the retina and the controller, because we intend to adopt almost the same setup for the successive experiments.

Finally, this experiment constitutes an important ground to test the developed vision system, with regard to the omni-camera simulation for image acquisitions, and to the artificial retina for features extraction.

6.2 Experimental Setup

In the following sections, we describe the experimental setup used for the evolution of the focusing task. Section 6.2.1 illustrates the setup of the simulated scenario, including the object to focus, the background and lights. The successive section is about the setup of the vision system, with details on the omni-directional camera and on the employed virtual retina. Section 6.2.3 describes the neural network controlling the retina. Finally, section 6.2.4 explains the fitness function used to evaluate controllers for the task at hand.

6.2.1 Scene Setup

The simulated environment is very simple, being constituted only by the object to focus on and by a ground. The background around these elements, like a virtual room, is black. At the beginning of every evaluation, we randomly choose the object's shape, between a sphere and a cube, of fixed dimensions. We position the object around the camera at a random direction and distance in the range [30, 80] *cms*. We keep the distance below a maximum since very distant objects appear too small in the camera image and the retina cannot focus on them. Moreover, we randomly change the color of the object at every evaluation in order to evolve robust controllers that work under generic conditions. Finally, we place a point light near the camera to illuminate the scene, with settings that give high-contrast illuminations. Since the distance and orientation between the object and the light is random, the object's luminance slightly varies at every task evaluation. This added characteristic also contributes to develop a robust controller's behavior.

6.2.2 Vision Setup

We place an omni-directional camera in the center of the scene. In this experiment, the camera is not attached to a robot and so it cannot move. Since also the object under focus does not move, the images captured by the camera never change. We exploit this fact to optimize the simulation of the vision system during an evaluation, by creating the camera image only once and not at every control step.

Here, we use a circular retina of 4 x 4 visual cells. This number represents a good compromise between visual resolution and computational cost. On one side, lower resolutions have larger cells whose value is the average of many pixels that might belong either to the object or to the background. Consequently, these visual cells may not give a significant response when they cover small object details or distant

objects, especially in the initial control phase when the retina usually employees large cells. On the other side, higher resolutions increase the computational cost of evolution because the number of visual cells affects the size of the neural network under evolution (details in the section about the controller setup). Table 3 shows the retinal parameters used in this experiment:

Table 3: Retinal parameters used in focusing experiment.

Parameter	Min Range	Max Range
Aperture	15 [degrees]	360 [degrees]
Radius	0 [pixels]	63 [pixels]
Length	4 [pixels]	49 [pixels]

The retina operates on simulated images of 128 x 128 pixels resolution. We initialize the retina at full size at the beginning of every experiment to give it an immediate global view of the image. Note that here the camera does not see the reflected image of the robot in the center of the image, as it happens in the other experiments where the camera is attached to a robot. For this reason, we give the retina complete freedom in the radial movement, expressed by the full range (0, 63 [pixels]) of the parameter *radius*. In addition, the retinal *aperture* and *length* have a lower bound to keep the size over a minimum.

6.2.3 Controller Setup

The neural network is a *perceptron* having 17 input neurons that encode the state of 16 sensors and a bias unit (i.e. a unit whose activation state is always 1). Each sensory neuron is directly connected to 5 output neurons controlling the artificial retina. The transfer function of these neurons is a standard sigmoid function. The connection weights are 85, ranging from -1 to +1, and they are genetically determined.

Input neurons: The 16 sensors in input to the neural network encode the average luminance of the 4 x 4 visual cells of the artificial retina. We update the color value of these cells at each image acquisition by averaging the pixels of each cell (see section 5.2.1). Then, we map the resulting color to grey scale, equal to the cell's luminance:

$$cell_i(l) = 0.299 * cell_i(r) + 0.587 * cell_i(g) + 0.114 * cell_i(b)$$

where the luminance (l) of the i-th cell is obtained by properly weighting the three color components (r,g,b) associated to the cell (the above formula is standard in Computer Graphic's literature [25]).

Operating on monochromatic data reduces the amount of information by a factor of 3 and greatly simplifies the neural network controller. Anyway, we plan to run experiments using colored, instead of monochromatic, information in future (for instance, colors may be useful for advanced discrimination tasks).

Output neurons: The output neurons of the *perceptron* control the following retinal parameters:

Parameter	Range
Radial distance from the center	0 - 63 [pixels]. The retina cannot move beyond the image borders.
Angular shift	0 - 15 [degrees]. We limit the maximum shift to preserve the correlation between successive sensorial inputs.
Angular shift direction (clockwise or anticlockwise)	0 – 1 [units]. When the output is < 0.5, the direction is clockwise, otherwise it is counterclockwise.
Length	0 – 1 [units]. 0 represents the minimum length, 1 the maximum.
Aperture	0 – 360 [degrees]. At 360 degrees the retina sees in all directions.

Every output neuron ranges from 0 to 1 and is remapped to the corresponding parameter range before the control phase takes place. Note that we used two distinct neurons to control the retina's angular movement for a better stability in the movement. This stability is necessary in order to produce solid focus actions without noisy oscillations of the retina. The reason is that if we use a single neuron encoding the whole range of angular movement (-15, +15 [degrees]), a null movement (0 degrees) would correspond to the neural value 0.5. The precise value of 0.5 is difficult to obtain using a *perceptron* network whose output neurons quickly saturate to 0 or 1 in the presence of high synaptic values. Earlier experiments using a single neuron to encode the retina's angular shift confirmed this problem, producing focus actions with undesired, high oscillations.

6.2.4 Fitness Estimation

The fitness function is a criterion that evaluates the performance of a neural network in achieving the desired focusing task. At this purpose, we take into account the different luminance between the object and the background. In the following, we

consider one image pixel to belong to the object if its luminance exceeds a given threshold, equal to the ground plane's luminance (0.3 typically).

We calculate the fitness by combining two components evaluated at every control step. The first component, F_i^l, rewards the positioning and focusing of the retina on the object, while the second component seeks to adjust the retinal dimensions for optimal coverage.

With regard to component F_i^l, we must calculate the average luminance of all the pixels seen by the retina (L_r) and of the object's pixels in the whole image (L_i). The ratio between these values expresses the retina's performance in targeting the object in the image. For instance, when the retina misses the object or it is too large, its average luminance is low because it sees mainly the dark background. Instead, when the retina centers the object, it detects a higher luminance that may also exceed that of the object. Fitness component F_i^l is then:

$$F_i^l = \min(1, L_r / L_i)$$

This component is not enough to evolve a good focusing behavior because it cannot discriminate, through the parameter L_r, the whole object from an inner part of it, in case they have the same average luminance. This fact allows the retina to focus on small portions of the object, instead of optimally focusing on the entire object. In order to overcome this problem, we introduce a second fitness component that constraints the retina to cover the whole object. At this purpose, we must calculate the number of object's pixels both under the retina (C_r) and in the whole image (C_i). The ratio between these two numbers gives the fitness component F_i^c for the i-th control step:

$$F_i^c = C_r / C_i$$

According to the above formula, fitness component F_i^c is low when the retina does not cover all the object's pixels in the image, which prevents the retina to become smaller than the object. On the contrary, F_i^c equals one when the retina covers the whole object.

The product between the two fitness components gives the complete fitness of the focus action for the i-th control step, having all the required properties.

$$F_i^f = F_i^l * F_i^c$$

When the retina covers the whole object employing an excessive size, fitness F_i^l is low and fitness F_i^c is high (unitary in the limit), giving an overall low fitness F_i^f.

Instead, when the retina focuses on part of the object, component F_i^l is almost unitary while component F_i^c is small. The fitness F_i^f reaches a maximum value when the retina optimally outlines the object's image. In any case, it is virtually impossible to reach a unitary fitness because the retina cannot match perfectly the shape of every generic object seen by the camera.

Finally, we want the retina to maintain a stable focus on the object over time. At this purpose, we average the fitness F_i^f over all the control steps in order to evolve a stable focusing behavior. The final fitness evaluating the controller at a certain trial is thus:

$$F_e = \frac{1}{N}\sum_{i=1}^{N} F_i^f$$

where N is the number of control steps and F_i^f is the fitness of the focus action calculated at the i-th control step.

The number of control steps for an evaluation is only 60 in order to generate a fast behavior.

6.3 Results

We performed 10 replications of this experiment, starting with different randomly generated populations at each replication. Table 4 summarizes the parameters of the evolutionary algorithm used for this experiment.

Table 4: EA parameters for focusing experiment.

Parameter	Value
Trials	5
Population size	100
Selected out	20
Mutation rate	0.15
Recombination rate	0.3
Genotype length	105
Genotype encoding	REAL
Initial range of genes	[-1, +1]
Standard deviation	0.3

In the end, evolution produced successful controllers capable of focusing the retina on generic objects. Figure 26 shows the average and best fitness over the 10 replications of the experiment.

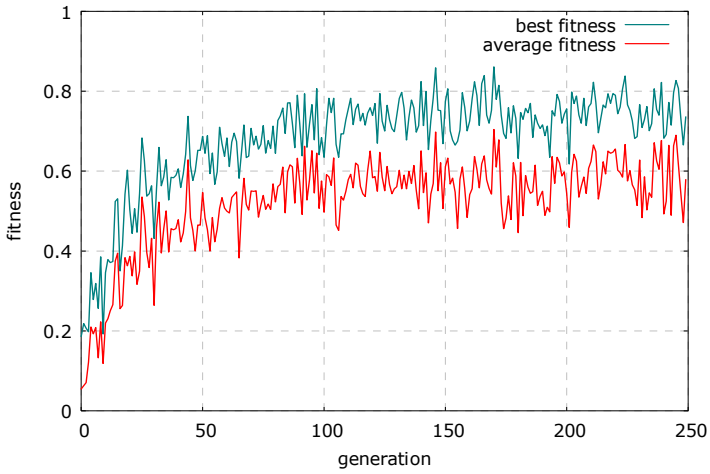

Figure 26: Evolution of the focusing ability. The average and best fitness of the population are plotted against the generation number.

We tested the best controllers produced in each replication, evaluating them for 500 trials of 100 control steps. The average fitness values are presented in Table 5.

Table 5: Average performance of the best controllers evolved in each replication of the experiment.

Replication	Fitness	Replication	Fitness
1	0.572	6	0.560
2	0.464	7	0.558
3	0.529	8	0.593
4	0.529	9	0.578
5	0.539	10	0.565

The evolved retinal behavior does well in almost all situations, with fluctuations of the performance that depend only on the random initial conditions of the experiment. This behavior is practically the same in all the evolved controllers and consists in the following steps. Initially, the retina takes an average opening, of about 45 degrees, and a nearly zero degree orientation. Then, it starts rotating clockwise until it detects the object. At this point, the retina adjusts its position and size in order to cover optimally the object; the latter process takes about 10 cycles. Finally, the retina keeps the focus on the object, with minor oscillations around it. In some cases, depending on the object's luminance in the image, the retina stays perfectly still without oscillations. Figure 27 shows an example of the described behavior.

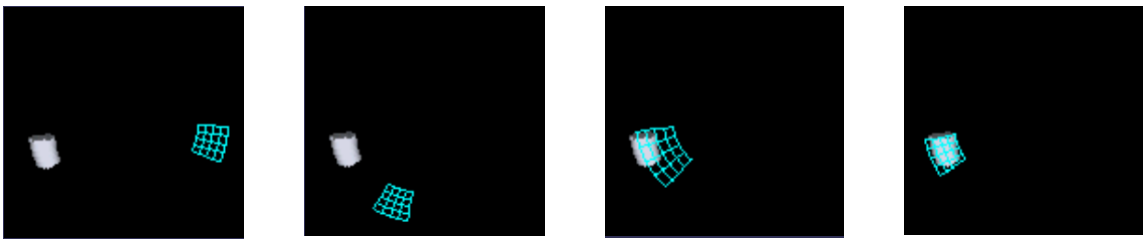

Figure 27: An example of evolved focusing behavior.

6.3.1 Robustness

We devised two tests to study the robustness of an evolved retina.

The first test consists in measuring the retina's performance varying the luminance of an object from dark to bright. At this purpose, we measure for 50 times the fitness value on a set of 10 luminances, uniformly spanning the range [0.3, 1]; we do not test luminances below the value 0.3, corresponding to the ground's luminance, because our fitness formulation cannot discriminate the object from the ground in that case. At each evaluation, lasting 60 control steps, we initialize the object with a random, short distance from the camera and with a random orientation. Figure 28 shows the average and variance of the fitness values for each the 10 luminances. As shown, the performance of the retina is satisfactory over all the possible values of luminance. The variance slightly increases on dark objects because, being less distinguishable from the background, the retina sometimes cannot focus on them optimally. On the contrary, the retina displays a better robustness when operating on objects characterized by higher luminances; in the plot, this is reflected as a smaller variance.

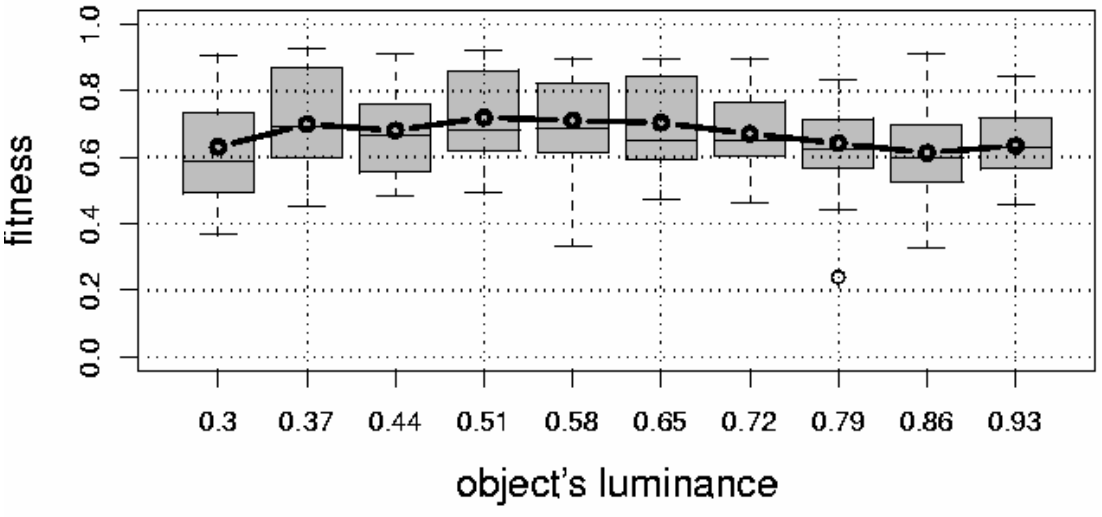

Figure 28: Fitness in function of the object's luminance.

The second test studies the performance of the retina varying the distance between the object and the omni-directional camera. The test measures the fitness value 50 times on each of 10 distances in the range [20, 200] *cms*. As in the previous test, each evaluation lasts 60 control steps and each time we randomly initialize the other parameters of the object, in this case its luminance and its orientation around the camera. The resulting graph (Figure 29) shows that the retina is capable of focusing on the object until a distance of about 100 *cms*, slightly more than the maximum distance used during evolution. After that, the fitness drops because the object appears very small in the image acquired by the camera, and the retina cannot easily focus on it. The ultimate reason is that the retina, when passing on far objects, receives a visual input that is not sufficient to discriminate them from the background. This result confirms the statement, often appearing throughout this thesis, that the retina can hardly focus on far objects.

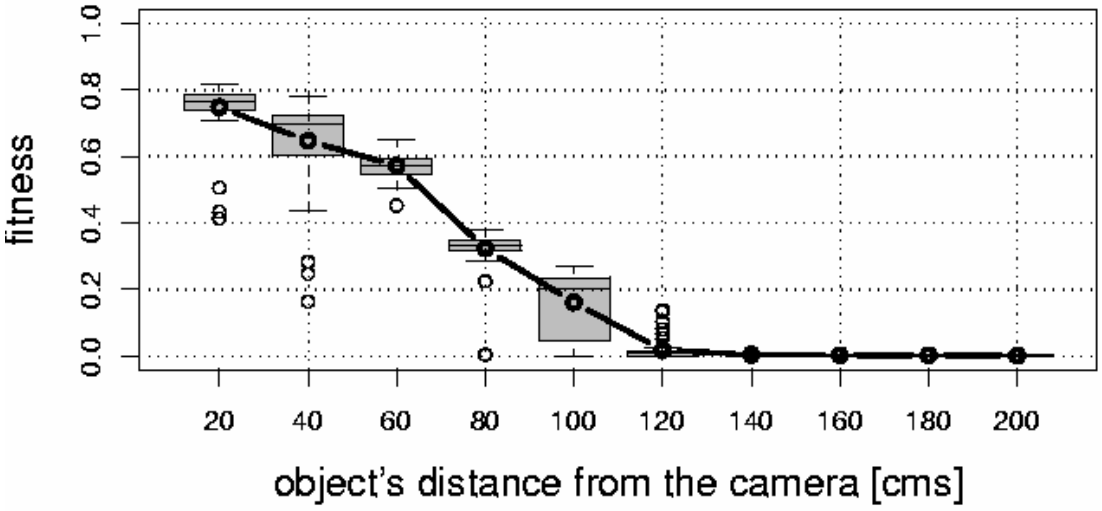

Figure 29: Fitness in function of the object's distance from the camera.

6.3.2 Generalization Properties

The evolved behavior generalizes particularly well on moving objects. We studied this property with a third test that measures the retina's ability in focusing on moving objects. We expect the retina to follow well objects moving not "too" fast. Indeed, recall that the retina can move at each control step for a maximum of 15 degrees (Table 2) in order to preserve correlation between successive visual inputs. Objects moving too fast break this correlation and consequently the retina's ability in keeping its focus on them. The test evaluates the fitness value for 50 times on each of 10 distinct angular velocities of the object, spanning the range $[-\pi, \pi]$ degrees per control step. At each evaluation, running for 60 control steps, the object is initialized at a random distance from the camera and with a random color. Figure 30 plots the

obtained fitness values. As shown, the retina's performance decreases as the object moves faster but it still maintains a satisfactory level. This result is particularly interesting because demonstrates how evolution can successfully produce controllers that greatly adapt to different situations. The graph shows also an asymmetry in the performance between negative and positive rotations. This actually depends on the evolved strategy to focus on objects, sometimes implying clockwise rotations that better suit objects moving in the same way, and other time counterclockwise rotations.

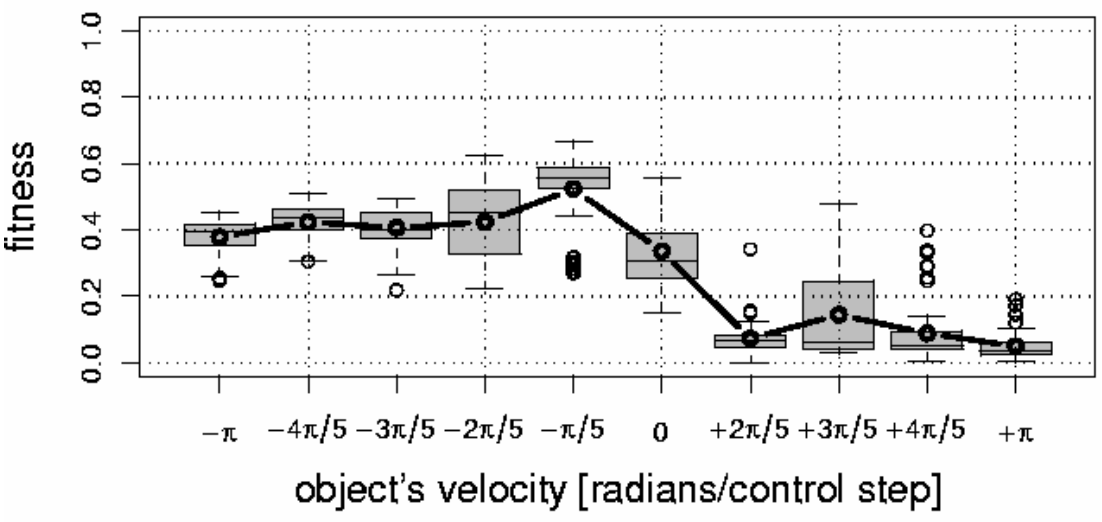

Figure 30: Fitness in function of the object's velocity around the camera.

Summarizing, the evolved retinas successfully focus on generic objects perceived by an omni-directional camera. The behavior is very robust against different conditions, as the object's luminance, position and distance from the camera. Besides, the retina can also focus on moving objects, thus demonstrating high capacities of adaptation. The latter property is particularly interesting in the view of our second experiment, described in the next chapter, which deals with a mobile robot that must extract relevant information through the retina in a navigation task.

7

EVOLVING TASKS: TARGETING

This chapter describes our second experiment, which concerns the evolution of a targeting task guided by vision. The first section introduces the task and explains the motivations and challenges behind it. Section 7.2 describes the experimental setup, including the simulated environment, the vision system, the neural network controller and the fitness function that evaluates potential controllers. Finally, the last section discusses the results obtained in this experiment and the generalization properties of the evolved controllers.

7.1 The Targeting Task

The second experiment of this thesis consists in developing a vision-guided controller capable of driving a robot towards a target object. The robot perceives the environment, consisting in the target object randomly placed over a ground, by means of its omni-directional camera. The camera captures images that show 360-degree views of the surrounding environment, and represent the only sensorial information available to the robot to accomplish its task. At this purpose, the robot must possess two abilities. The first ability consists in localizing the target object in the scene. The artificial retina integrated in the robot's vision system serves this purpose through the recognition of the object in the camera images acquired. In analogy to the previous experiment, the retina must find the object in the image and then focus on it. As second ability, the robot must move towards the target object as fast as possible. In doing this, the robot should constantly estimate the object's position in the scene by exploiting the visual information gathered by the retina.

This experiment has a fundamental difference with respect to the previous one: the images processed by the retina are dynamic because the camera, moving with the robot, regularly sees the surrounding environment from different points of view and orientations. This fact alone adds a level of complexity over the previous experiment, where evolution operated on static images of the environment. In addition, motion and vision in the robot are related through a sensory-motor loop that strictly characterizes the robot's behavior during the whole control phase. On one side, the motion of the robot highly influences its visual perception of the environment, often in unpredictable ways in consequence of complex dynamics with the environment

where it moves. On the other side, the information gathered by the retina operating on the acquired images determines, through the controller, the motion of the robot. Artificial evolution must therefore co-evolve the robot's motion and vision, taking into account their intrinsic and mutual relationship.

7.2 Experimental Setup

The following sections describe the experimental setup used for the evolution of the targeting task. Section 7.2.1 is about the simulated environment. Section 7.2.2 deals with the robot's vision system. Section 7.2.3 describes the neural network controlling the robot's vision and motion. Section 7.2.4 then discusses the choice of the fitness function we used to evaluate controllers for this task.

7.2.1 Scene Setup

The simulated environment includes one robot, the target object and the ground, surrounded by a black background (an example of the simulated environment is shown in Figure 31).

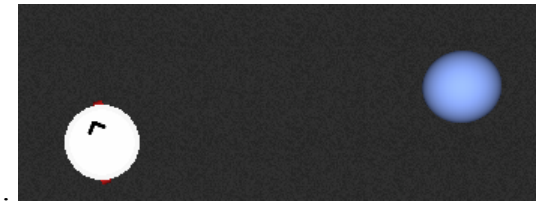

Figure 31: Sample scene for the targeting experiment, showing the target object (in blue) and the robot (in white) over a ground plane.

At the beginning of a controller evaluation, we randomly place the robot on the ground. In the simulation, the ground plane is rendered with a dark, highly detailed texture, which is perceived as small noise by the robot's vision. Next, we position a sphere, representing the target object, at a random distance and orientation from the robot. The distance varies in the range [60, 80] *cms*. It is limited because distant objects appear too small (few pixels) in the image and cannot be easily recognized by the virtual retina. As in the previous experiment, the color of the object randomly varies at every evaluation in order to evolve robust controllers that work under generic conditions. Still, the object's luminance is greater than the background's luminance to make it possible the discrimination between these different entities. Finally, we place a light in the centre of the scene to illuminate the object, the robot and the ground. The luminance of these objects slightly varies at every task evaluation because their distance and orientation from the light is random. This characteristic further enhances the controller's robustness.

7.2.2 Vision Setup

The robot is provided with an omni-directional camera placed over its turret, which acquires images of the surrounding environment in real-time. The turret, and thus the camera, cannot freely rotate with respect to the chassis to prevent directional aliasing. This means that the orientation between the camera and the chassis must be consistent throughout the whole control phase. This is fundamental because, as we show in the results, the evolved robots exploit the retina's orientation for a correct navigation towards the target object.

In this experiment, the retina has 4 x 4 visual cells and operates on images of 128x128 pixels. A higher-resolution retina would allow a more precise targeting of distant objects appearing small in the image. However, it would also increase the time and complexity of evolution, given the relationship between the number of visual cells and the neural network's size. The adopted resolution proved successful in targeting objects placed within the distance range mentioned in the scene setup. The retinal parameters are more or less the same used in the previous experiment, as shown by Table 6.

Table 6: Retinal parameters in targeting experiment.

Parameter	Min Range	Max Range
Aperture	15 [degrees]	360 [degrees]
Radius	14 [pixels]	63 [pixels]
Length	4 [pixels]	49 [pixels]

Note that the radius of the bottom arc defining the circular retina starts from 14 pixels, equal to the robot's radius in the image. This way, the retina never processes the central area of the image, which always shows the reflection of the robot mounting the camera and it is not interesting for the navigation of the robot itself.

7.2.3 Controller Setup

The neural network controlling the robot is a single-layer *perceptron* having 21 input neurons, including a bias unit. The output neurons are 7 and control the artificial retina and the robot's movement. Overall, the connection weights are 144, ranging from -1 to +1.

Input neurons: The first 16 neurons in input to the neural network encode the average luminance of the 4 x 4 visual cells of the artificial retina, as in the previous experiment. The remaining 4 neurons are associated to 4 directions $\theta_i = \{ 0, \pi/2, \pi, 3\pi/2 \}$ degrees, and encode the retina's orientation according to the formula:

$$\begin{cases} \cos(\theta_i - \gamma) & |\theta_i - \gamma| <= \pi/2 \\ 0 & \text{otherwise} \end{cases}$$

where θ_i is the angle associated to the i-th neuron, and γ is the retina's angle in degrees. Figure 32 shows the activation state of the four neurons in function of the retina's angle.

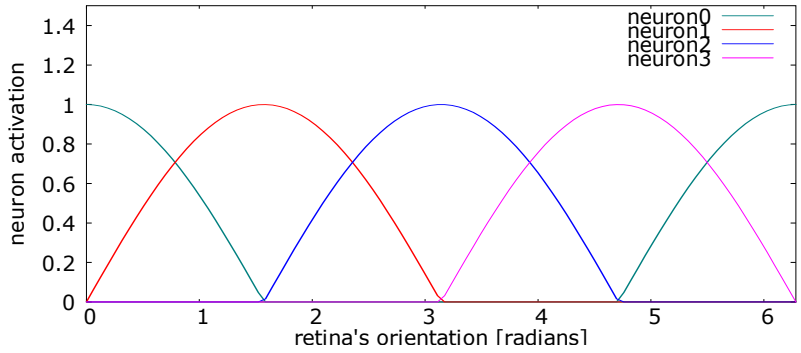

Figure 32: Neurons activation in function of the retina's angle.

The above encoding has the property that the four directional neurons receive a positive activation state only when the retina spans the associated angular range. Consequently, the controller receives clearly distinct inputs in correspondence of different retina's orientations, which as shown later in the results, provide an estimate of the object's position in the scene. With reference to Figure 33, the retina might focus on an object appearing on the right side of the camera image.

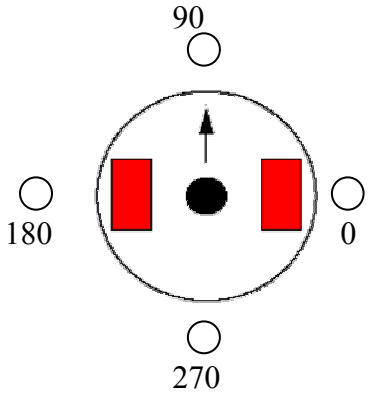

Figure 33: Retina's angles for encoding.

The neuron corresponding to the direction of 0 degrees would then receive a high activation state, while the other neurons would receive a null value. At this point, an evolved robot would know how to exploit this particular input to correctly move in the right direction towards the object. We argue that the adopted encoding, providing

a fine distinction between different directions, eases navigation tasks driven by our simple and reactive neural networks.

Output neurons: The output neurons control both the retina, like in the previous experiment, and the robot's wheels. The following are the controlled parameters along with their range.

Parameter	Range
Radial distance from the center	14 - 63 [pixels]. The retina cannot move beyond the image borders.
Angular shift	0 - 15 [degrees]. We limit the maximum shift to preserve the correlation between successive sensorial inputs.
Angular shift direction (clockwise or anticlockwise)	0 – 1 [units]. When the output is < 0.5, the direction is clockwise, otherwise it is anticlockwise.
Length	0 – 1 [units]. 0 represents the minimum length, 1 the maximum.
Aperture	15 – 360 [degrees]. At 360 degrees the retina sees in all directions.
Left wheel's speed	-6,5 – +6,5 [radians/sec]
Right wheel's speed	-6,5 – +6,5 [radians/sec]

Every output neuron ranges from 0 to 1 and is remapped to the corresponding parameter range before the control phase takes place. Note that the range of the controlled retinal parameters is more or less the same as the previous experiment. The only difference concerns the minimum value of the retina's distance from the center, which in this case starts from 14, equal to the robot's radius in the image. Again, we limit the maximum angular movement at each control step in order to preserve correlation between successive visual inputs. This is a good property that allows the robot to constantly keep its focus on the target object during navigation, without accidental losses of focus that would interrupt its correct navigation for some control steps.

7.2.4 Fitness Estimation

The fitness function evaluates the ability of a controlled robot in the targeting task. We evaluate the required abilities of target focusing and approaching with two dedicated fitness components.

The fitness component evaluating the focusing ability is the same as the previous experiment. An evolution guided by this fitness produces a retinal behavior that constantly locates and focuses on the target object in the camera image. As already explained, this ability allows the robot to estimate the object's position in the scene, before and while moving towards it.

We combine this fitness with a second component that rewards the movement of the robot towards the target object. This component, called F_i^d, is calculated at each i-th control step as:

$$\begin{cases} F_i^d = 1 - D_0 / D_i & D_i < D_0 \\ F_i^d = 0 & \text{otherwise} \end{cases}$$

where D_0 is the initial distance between the robot and the object, and D_i is their distance at the i-th control step. The component F_i^d grows as the robot approaches the target object, reaching a maximum value when their mutual distance is minimum.

Finally, we combine the two fitness components according to the formula:

$$F_i^t = F_i^f * \left(0.5 + 0.5 * F_i^d\right)$$

The previous formula implies a constant focusing ability and rewards the movement of the robot in the direction of the focused object. The component F_i^t is then averaged over all control steps in order to evolve a robust behavior lasting in time. The final fitness evaluating the controller is thus:

$$F_e = \frac{1}{N} \sum_{i=1}^{N} F_i^t$$

where N is the number of control steps and F_i^t is the fitness of the targeting action evaluated at the i-th control step.

In this experiment, each controller is evaluated for 200 control steps, which are enough to let the robot reach the target object from any possible distance in the initialization range. This number of control steps also influences the evaluation of a controller because the robot should find, within the short, given time, efficient strategies of movement in order to receive a high fitness.

7.3 Results

We replicated the experiment 10 times, using the parameters of the evolutionary algorithms presented in Table 7.

Table 7: EA parameters for targeting experiment.

Parameter	Value
Trials	5
Population size	80
Selected out	20
Mutation rate	0.15
Recombination rate	0.3
Genotype length	147
Genotype encoding	REAL
Initial range of genes	[-1, +1]
Standard Deviation	0.3

Figure 34 shows the average and best fitness over the 10 evolutionary runs. As shown, evolution reaches after approximately 300 generations a satisfactory fitness value of nearly 0.64. A higher fitness is not realistic because the retina cannot perfectly focus on the target object at every control step, which determines a fitness component F_i^f always below one. In addition, the robot necessarily takes some time in reaching the target object, giving a sub-optimal fitness F_i^d.

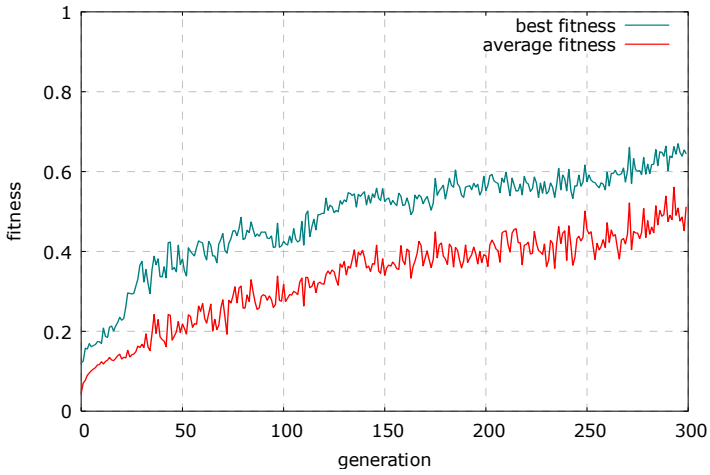

Figure 34: Evolution of the targeting ability.

We tested the best controllers produced in each replication, evaluating them for 500 trials. The average fitness values are presented in Table 8.

Table 8: Average performance of the best controllers evolved in each replication of the experiment.

Replication	Fitness	Replication	Fitness
1	0.5625	6	0.6234
2	0.6345	7	0.6019
3	0.6119	8	0.5716
4	0.5988	9	0.5970
5	0.5865	10	0.6145

The evolved behaviors present differences in the initial control phase. According to one controller, the retina initially assumes an aperture of about 110 degrees and an orientation of 0 degrees. Next, the retina in the image and the robot in the environment rotate clockwise (Figure 35, left). The retina stops rotating upon reaching an orientation of nearly 180 degrees, which correspond to the left side of the image. The robot instead stops rotating when the object appears under the retina, which means on the left side of the camera image (Figure 35, center). At this point, the retina starts focusing on the object and the robot begins to approach it, moving forwards while constantly estimating its position through the retina. Once the robot has reached the target object, sometimes it stops in front of it and sometimes it rotates around the object, depending on the evolved controller (Figure 35, right). In both cases, the robot maintains an orientation such that the object appears on the bottom side of the image. Lastly, the retina always keeps a good focus on the object, which allows the robot to constantly estimate the object's position and never run away from it.

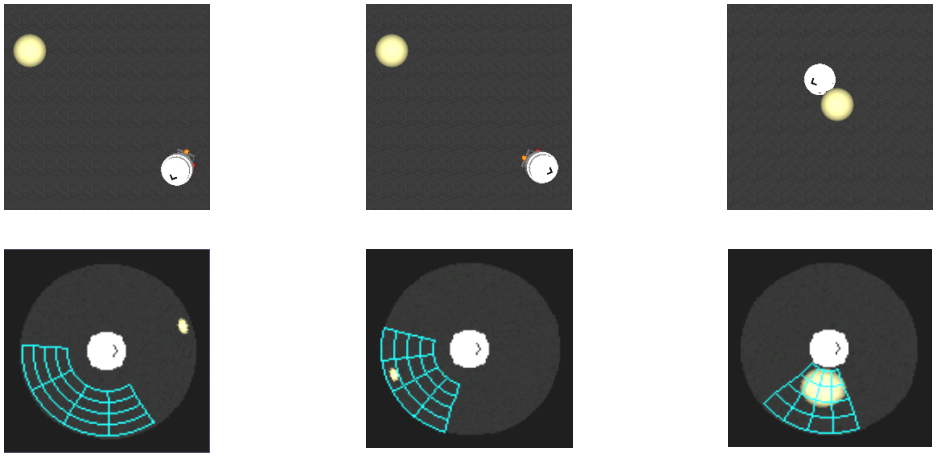

Figure 35: An example of evolved targeting behavior. *Left:* The robot rotates until the object appears on the left side. *Center:* The robot approaches the object and the retina focuses on it. *Right:* The robot stays close to the object, maintaining the focus on it.

Another controller generates a different behavior at the beginning. Here, the robot and the retina initially behave in order to bring the target object in an image position that triggers the targeting action. From the right side of the image, the retina starts rotating clockwise in search of the target object. Meanwhile, the robot stands still. When the retina detects the object, it keeps rotating around the image, but also the robot starts rotating clockwise in place. The robot stops rotating when the target object appears on the left side of the camera image. Hereafter, the behavior is the same as that previously illustrated.

7.3.1 Robustness

In order to study the robustness of the evolved controllers, we devised a series of tests that measure the performance of the robot in targeting objects with different properties.

The first test evaluates the performance of the robot varying the initial distance between the robot and the target object. We evaluate the fitness value for 50 times on each of 10 different distances, spanning the range [20, 200] *cms*. Each evaluation lasts 300 control steps, more than during evolution, because the robot naturally needs more time to reach distant objects. At every evaluation, we also initialize the object's color and shape at random. For this test, we measure the ability of the robot in targeting the object through the fitness F_i^d of the last control step. This way, we do not penalize the analysis of robots operating on far object because they naturally require more time to eventually reach the object. Figure 36 plots the measured fitness values.

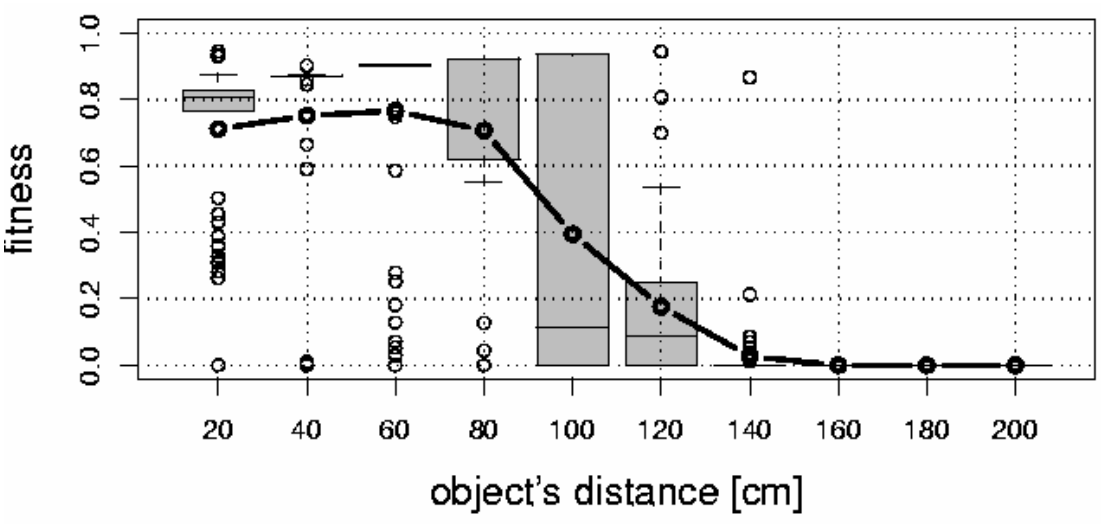

Figure 36: Targeting performance in function of the robot's distance from the object.

The fitness value maintains a high value up to a distance of nearly 100 *cms*. This is a good result because the distance 100 corresponds to about 8 times the robot's diameter. After that, the fitness drastically drops because the robot is not capable of targeting very distant objects. The primary reason is that the retina cannot detect and focus on far objects and consequently the robot cannot estimate their position in the environment while moving towards them. In future, we will study solutions that might improve the robot's performance with distant objects, perhaps using higher retinal resolutions, different controllers (multi-layer) and better tuned evolutions.

We devised a second test to verify the robustness of the robot's performance under different colors of the target object, in the range [0.3, 1]. In view of the results of the previous experiment, we expected a good performance in targeting bright objects because they clearly differ in luminance from the ground. Moreover, the performance should drop for dark objects because the retina cannot easily discriminate them from the ground and consequently it cannot drive the robot towards them. To verify this hypothesis, we average the value of the fitness component F_i^d over 500 times, testing a controller on 10 distinct luminances of the object. Each evaluation lasts 200 control steps. The results are shown in Figure 37.

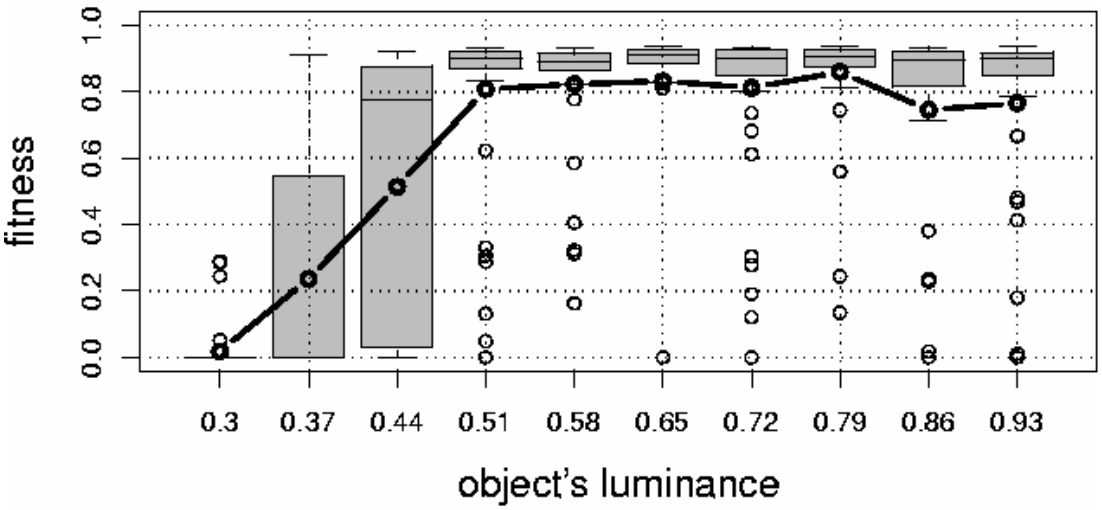

Figure 37: Fitness in function of the object's luminance.

As shown, the robot maintains a high performance in targeting the object until a luminance value around 0.5, demonstrating a good robustness in the evolved controller. Below that value, the robot encounters difficulties in detecting and moving towards the object because the object becomes less and less distinguishable in luminance from the ground. The high variance of the fitness value for low luminances further demonstrate that, in that range, the robot's performance is not particularly robust. In proximity of the value 0.3, the performance drops completely because the target object practically has the same color of the ground and the robot

cannot detect the object relying on vision only. For these situations, integration with additional sensors (e.g. infrared sensors) may be a solution.

7.3.2 Generalization Properties

Preliminary tests showed that the evolved robots possess great capabilities in targeting moving objects, demonstrating once more the flexibility of artificial evolution in synthesizing an active vision system that can operate efficiently under dynamic conditions.

In order to measure the targeting performance under such conditions, we devised an additional test that measures the performance of the robot in targeting objects moving at different velocities. At each evaluation of this test, we initially place the target object at a short, random distance from the robot. We then let the robot target the object. Whenever the robot arrives at a certain distance from the object, we assign a velocity to the latter to move it away from the robot in the successive control steps. The velocity vector points in the robot's direction, perturbed by a slight random orientation. From now on, the object will move away from the robot at the imposed velocity and direction. The robot must thereafter follow and reach again the object. According to this procedure, the object always moves away from the robot at a certain velocity. We repeat this test for 10 distinct velocities, in the range [0, 2.6] *cms* per control step, evaluating each 50 times. The upper limit is the highest velocity the robot can move in one control step. Naturally, it is pointless to let the object move faster than the robot is able to, because the robot must have the possibility of reaching (following at least) the object. Figure 38 shows the obtained results.

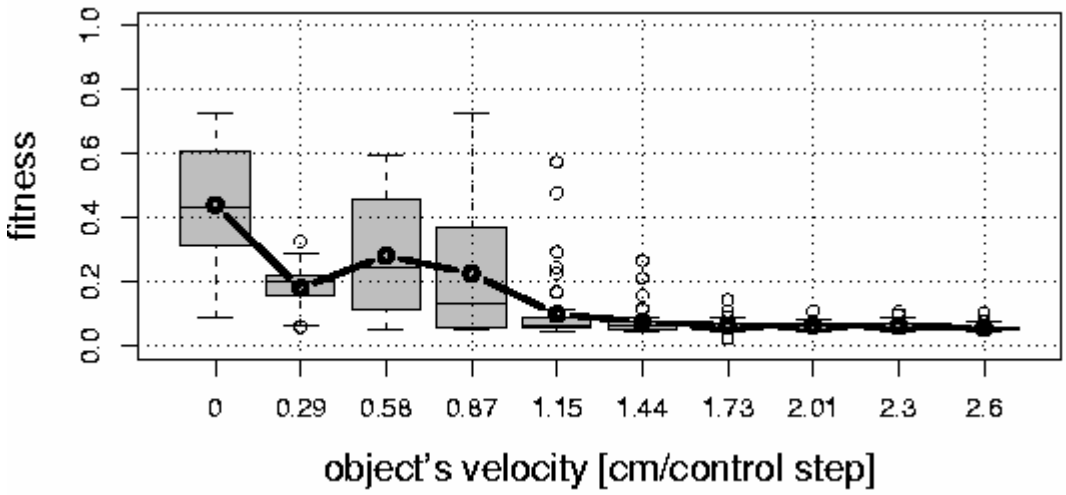

Figure 38: Targeting fitness in function of the object's velocity.

The graph shows that the robot is capable of following objects up to a velocity of nearly 0.8 *cms* per control step. After this value, the average fitness drops essentially

because the robot can maintain the retinal focus on the moving object less efficiently and thus it cannot estimate correctly the object's position in the environment. This result agrees with that obtained in the previous experiment, where we showed how the retina's performance drops for increasing velocities of the object to focus. Nevertheless, the graph demonstrates that the robot is capable of navigating towards moving objects, although within a limited range of velocities. This behavior is possible because the retina, by dynamically tracking the moving object, allows the robot to constantly update its trajectory in direction of the object.

The described generalization property of the evolved controllers confirms the efficiency of our active vision system in adapting to dynamic and unexpected conditions. The most important result here is that the vision system can operate efficiently on mobile robots perceiving dynamic objects. This is exactly the experimental setup of the successive experiment that concludes our thesis. In that experiment, we apply active vision to control a group of robots, having the task of moving together in a coordinate fashion by exploiting visual information extracted by each robot through a retina. The results here reported rise our expectations on the success of the last experiment.

8

EVOLVING TASKS: SWARMING

The final experiment of this thesis concerns the evolution of coordinate motion in a group of *s-bots* driven by vision. Section 8.1 introduces the experiment along with its motivations and challenges. Section 8.2 is about the experimental setup, with details on the simulated scene, on the robots' vision system and neural controller, and the choice of an appropriate fitness function. Lastly, section 8.3 discusses the obtained results and the generalization properties of the evolved solutions.

8.1 The Swarming Task

Coordinate motion is a basic capability strictly necessary in groups of autonomous robots working collectively on some task. In order to overcome particular challenges, as avoiding collisions or moving compactly to a certain location, the robots should move as a single entity, implying a form of coordination. Nature provides many examples of coordinate motion, as flocks of starlings, schools of cods. These examples are particularly interesting because they represent cases of self-organizing behaviors creating patterns of individuals moving coordinately. Many researchers have provided models for the replication of flocking (swarming) behaviors in artificial life simulations. These models are often based on hard-coded rules controlling the behavior of individuals according to internal parameters and the state of neighboring individuals. We follow a different approach, synthesizing controllers for coordinate motion through artificial evolution on simple and purely reactive robots.

In this work, we study coordinate motion in a group of physically independent *s-bots* driven by the vision system developed throughout this thesis. Our purpose is not that of replicating a perfect swarming as it happens in nature. Instead, we want to study how evolved robots exploit vision to move together in a coordinate fashion. As shown in section 8.3, the obtained results are promising and can be considered the starting point for future research and applications.

This experiment, involving multiple robots, presents some of the characteristics and challenges previously discussed in chapter 2.

The main characteristic is that robots act as autonomous entities, not explicitly communicating information (i.e. via radio) about their internal state or behavior to

other robots. Moreover, the system is decentralized, which means that robots do not receive essential global information (e.g. the position of the nearest robot) from a supervising system coordinating the group. The swarming behavior must arise, and be maintained, solely through local interactions between individual robots.

Robots have a minimal perception of the environment through their vision system, which acquires information through a retina operating on 360-degree views of the surrounding environment, including the other robots and the ground. On the contrary, many biological models of self-organizing coordinate movement assume that agents are presented with significantly more information about their local environment. For example, in models of flocking and shoaling, agents have ideal sensors that provide the location, velocity and orientation of the nearest neighbors [64],[12].

Additionally, every robot has the same reactive controller and the same design. However, we expect robots to undertake functionally different roles in the group in order to solve the swarming task. For example, one robot may undertake the role of group leader and the other robots of followers. This specialization cannot arise from differences in the controller or morphology of robots, but must be the result of interactions among the robots and their environment.

As in the previous experiment, each robot extracts information from the acquired camera images through a virtual retina. For instance, one robot may exploit the retina to focus its visual attention on a near robot, in order to stay close to it. A major difference between this experiment and the previous ones is that the fitness function (section 8.2.4) does not explicitly define the behavior expected from the retina. On the contrary, in the targeting and focusing tasks, the retina was evolved with the purpose of optimally focusing on objects. Here, we cannot establish a priori an optimal retinal behavior because there are many robots in the environment and constraining the focus of the retina may not be necessarily the best solution. Consequently, we let evolution discover the best strategies to using the retina for the accomplishment of the swarming task.

8.2 Experimental Setup

The following sections describe the setup of the simulated environment (8.2.1), the robots' vision system (8.2.2) and the setup of the robots' controller (8.2.3). The last section, 8.2.4, discusses an appropriate fitness function for the evolution of controllers capable of producing a good swarming behavior.

8.2.1 Scene Setup

The simulated environment includes some *s-bots*, from three to eight, placed over a ground plane free from obstacles. The background is black. The ground is rendered with a high-frequency grey texture that is perceived as noise by the robots' artificial retina. Similarly to the previous experiment, the robots are brighter than the ground

plane to help the discrimination between these different objects. At the beginning of every evaluation, we randomly place the robots on the ground within a short, reciprocal distance, such that each robot is in the sensorial range of one another. This distance ranges from a minimum of 24 *cms*, to prevent initial overlapping positions, to a maximum of 72 *cms*. The distance is limited because robots cannot easily recognize other far robots through their vision system, as demonstrated by the previous experiments. Robots are given the same initial orientation perturbed by a random deviation in the range [-90, +90] degrees. Finally, we place a light in the centre of the scene to illuminate the robots and the ground. Figure 39 shows an example of the simulated environment:

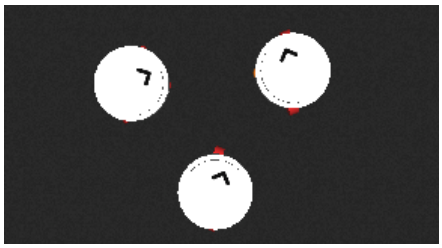

Figure 39: Simulated environment during evolution in the swarming experiment.

8.2.2 Vision System

The vision system of each robot does not differ from that described in section 7.2.2 for the targeting task. Again, robots acquire panoramic views of the environment through their omni-directional camera. An artificial retina then operates on the acquired images in order to extract visual information about the environment. Here, we still use a retina of 4x4 cells. However, the resolution of 3x3 might prove good enough to solve this task with success, and we will study this possibility in future. Instead, a higher resolution (e.g. 5x4 or 5x5) would be useful to target distant robots but it would increase the cost of evolution because the retinal resolution directly affects the neural network's size. Besides, a good swarming behavior should keep the robots in a tight formation, in which case there is no need for a high-resolution retina. Table 9 shows the parameters of the retina.

Table 9: Retinal parameters in swarming experiment.

Parameter	Min Range	Max Range
Aperture	15 [degrees]	360 [degrees]
Radius	7 [pixels]	31 [pixels]
Length	6 [pixels]	24 [pixels]

In this experiment, we simulate camera images of 64 x 64 pixels resolution, lower than in the other experiments. This greatly accelerates the simulation phase, involving at least three robots, and thus artificial evolution.

8.2.3 Controller Setup

The robots have the same neural network controller as in the targeting experiment, a single-layer *perceptron* with 21 input neurons (including a bias) and 7 output neurons. The input neurons encode the luminance of the 16 retinal cells and the retina's orientation. The output neurons control the retinal parameters and the robot's movement. The encoding of the input and output neurons is the same as the targeting experiment, described in section 7.2.3.

8.2.4 Fitness Estimation

The fitness function evaluates the swarming ability of a group of *s-bots*. It is expressed through two components that evaluate the group's aggregation (F_i^a) and its coordinate movement (F_i^m) respectively.

In our early attempts, we expressed the first component as the average distance of robots from the center of mass of the group:

$$F_i^a = \frac{1}{N}\sum_{1}^{N}\left(1 - D_i/\tau\right)$$

where N is the number of robots (3 in our case), D_i is the distance of the i-th robot from the center of mass and τ is an upper bound for the distances.

Evolution driven by this function produced robots capable of aggregating in circular formations, but unable to move towards a common direction. Actually, other robotic configurations, as a chain (each robot following the previous) or eagle formation (one leader and two followers), might better suit the swarming task, but were not favored by this fitness function.

The important characteristic for a successful swarming behavior is that every robot stays in the visual range of at least another robot because, in this way, no robots will get lost. This property characterizes many compact formations and seems particular appropriate for our needs.

We decided to use the fitness formulation devised by Quinn et al. [53] for the evolution of coordinate motion in robots guided by infrared sensors. We associate every robot with a sensor range, defining a circle centered on the robot (Figure 40). The radius corresponds to the maximum range within which a robot can successfully perceive other robots. Indeed, as demonstrated by the previous experiments, the retina cannot effectively see objects after a certain distance. After some trials, we tuned the sensor range to the value 18 units (equal to 36 *cms*).

First, we calculate the distance D_i of every robot from the sensorial range of the nearest robot (see Figure 40). The quality of aggregation is then expressed by:

$$F_i^a = 1 - \tanh[\max(D_i)]$$

The hyperbolic tangent sharply degrades as its argument increases, penalizing the dispersion of robots. Note that when all robots are in the range of at least another robot, the values D_i are all null and F_i^a equals 1.

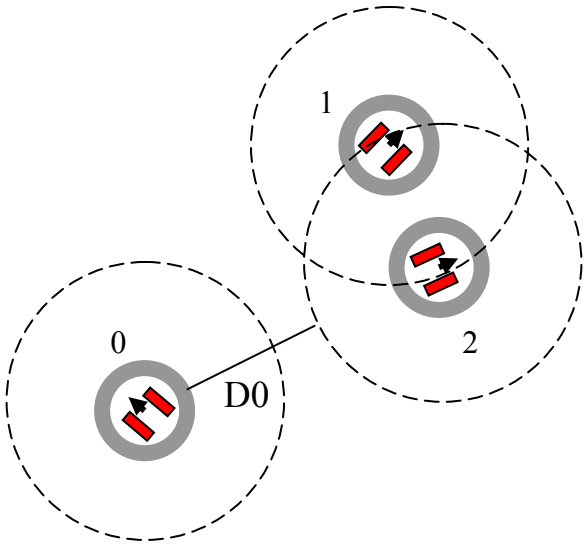

Figure 40: Sensorial range of swarming robots. Robot 0 is lost being outside the range of the other two robots. D_0 is its distance from the range of its nearest robot (robot 2).

The second fitness component rewards the movement of the group in a certain direction. It is calculated at each control step as:

$$F_i^m = \begin{cases} (d - D)/10 & d > D \\ 0 & otherwise \end{cases}$$

where d is the distance between the current center of mass and the initial one, and D is the maximum d reached so far; 10 corresponds to the maximum distance (in units) that the group of robots can travel in one control step. This formula rewards movement in a definite direction, constantly away from the previous position. We also tried other formulas, for example the normalized distance from the previous center of mass, but they often produced circular, sub-optimal trajectories.

The product of the two fitness components gives the complete swarming fitness at the i-th control step.

$$F_i = F_i^a * F_i^m$$

The final fitness F_e of a trial is the average of F_i over all control steps:

$$F_e = \frac{1}{N} \sum_{i=1}^{N} F_i$$

where N is the number of control steps.

Finally, when F_i^a equals 0 in a certain control step, we immediately terminate the evaluation and assign 0 to F_e in order to penalize controllers that separate robots.

Every trial in this experiment lasts 200 control steps in order to evaluate the performance of the group of robots for a sufficiently long time. Naturally, we hope that coordinate motion will evolve robust and consequently last even longer.

8.3 Results

Table 10 presents the parameters of the evolutionary algorithm used in this experiment.

Table 10: EA parameters in the swarming experiment.

Parameter	Value
Trials	5
Population size	80
Selected out	20
Mutation rate	0.15
Recombination rate	0.1
Genotype length	147
Genotype encoding	REAL
Initial range of genes	[-1, +1]
Bound range of genes	unbounded
Standard Deviation	0.2

We replicated the experiment 10 times with different randomly generated populations. The fitness of an evolutionary run is shown in Figure 41.

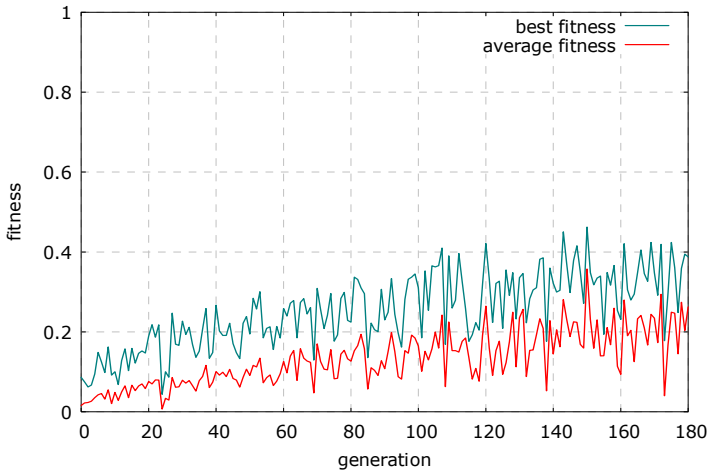

Figure 41: Evolution of the swarming ability.

The plot indicates that the evolutionary experiment is partly successful, as a maximum fitness of nearly 0.42 was achieved in all the replications after 140 generations. The reasons behind the low value achieved are essentially two: on one end, the evolved controllers are not very robust because robots sometimes move in opposite directions, thus determining an overall null fitness of a trial. On the other end, robots moving in a coordinate fashion proceed at a speed inferior to the theoretical maximum, determining a low value of the fitness component *Fm*.

The plot also shows high oscillations of the fitness against the generation number. The main cause is that the swarming performance evaluated during evolution depends on the initial conditions of the tested robots. For instance, let us consider an initialization that creates three robots pointing in exactly the same direction. In this case, a bad controller that simply moves the robots straight and unintelligently would receive a high fitness. Evolution would then select this controller, which would perform badly in the successive generations characterized by different conditions. This mechanism explains the frequent drops in fitness shown by the graph. The previous experiments lacked these oscillations because they employed fitness formulations that were almost independent of the test initial conditions. In order to solve this problem, we tried to force the initialization of robots to opposite directions, in which case robots had to behave intelligently to agree on a common direction of movement. Unfortunately, evolution with these settings did not start (bootstrap problem), producing robots that immediately separated and were unable to aggregate. In future, we will try other possible solutions. Probably, we should calculate the fitness as the average of many more trials, not only 5, at the cost of a more time-consuming evolution. The average would then mitigate the excessive evaluation of lucky individuals in some trials because they would perform poorly in other trials with different initial conditions for the robots.

We tested the best controllers produced in each replication, evaluating them for 500 trials. The average fitness values are presented in Table 11.

Table 11: Average performance of the best controllers evolved in each replication of the experiment.

Replication	Fitness	Replication	Fitness
1	0.2493	6	0.2420
2	0.2687	7	0.2376
3	0.2118	8	0.3017
4	0.2934	9	0.2733
5	0.2358	10	0.2243

Nonetheless, direct observation on three robots has shown that the evolved behaviors are often satisfactory. At the beginning, the robots start moving forwards following their initial random orientation. Meanwhile, their retina takes an aperture of nearly 270 degrees and an orientation in the image of 45 degrees. We interpret this behavior saying that a wide retina, involving the perception of the environment all around a robot, eases the detection of other robots in the environment and thus the accomplishment of the task. Henceforth, different input patterns determine distinct roles in the successive coordinate motion of the group. The robot perceiving all other robots on the back undertakes the role of leader and autonomously proceeds, driving the group in the exploration of the environment (Figure 42, a). Instead, the individual that detects one robot on the front and the other on the back maintains thereafter the role of middle robot in the formation; the retina correspondently perceives input on the left and right side (Figure 42, b).

Figure 42: Evolved swarming configuration in the form of a chain. Leader (a), middle (b) and rear robot's view (c).

Lastly, the rear robot is the one viewing the other two robots on the front, which are perceived by the retina as positive inputs in the right side of the image (Figure 42, c). The three robots proceed afterwards in the exploration of the environment, always maintaining a chain configuration. In some cases, some robots acquire similar sensory patterns and accordingly behave in analogous ways; as discussed later, this fact is partly responsible for the low robustness of coordinate motion under general

initial conditions. Occasional collisions or random rotations can then break this similarity and move the robots to the described configuration. Individuals initialized at a certain distance from the center of mass of the group are also capable of accelerating to reach the group.

Figure 43 plots the angular distance between the chassis of each *s-bot* and the average orientation of the group, in radians. It shows that after a transition period, the *s-bots* converge towards a common direction of movement. Thereafter, all the robots maintain their orientation aligned and proceed in the same direction, with occasional variations in an individual's orientation when the group steers. The alignment strategy is determined by the retinal input and depends on the role of each robot in the group. For instance, the rear robot always adjusts its orientation in order to maintain the perceived robots at the boundary between adjacent retinal cells on the right side of the retina (Figure 42, c).

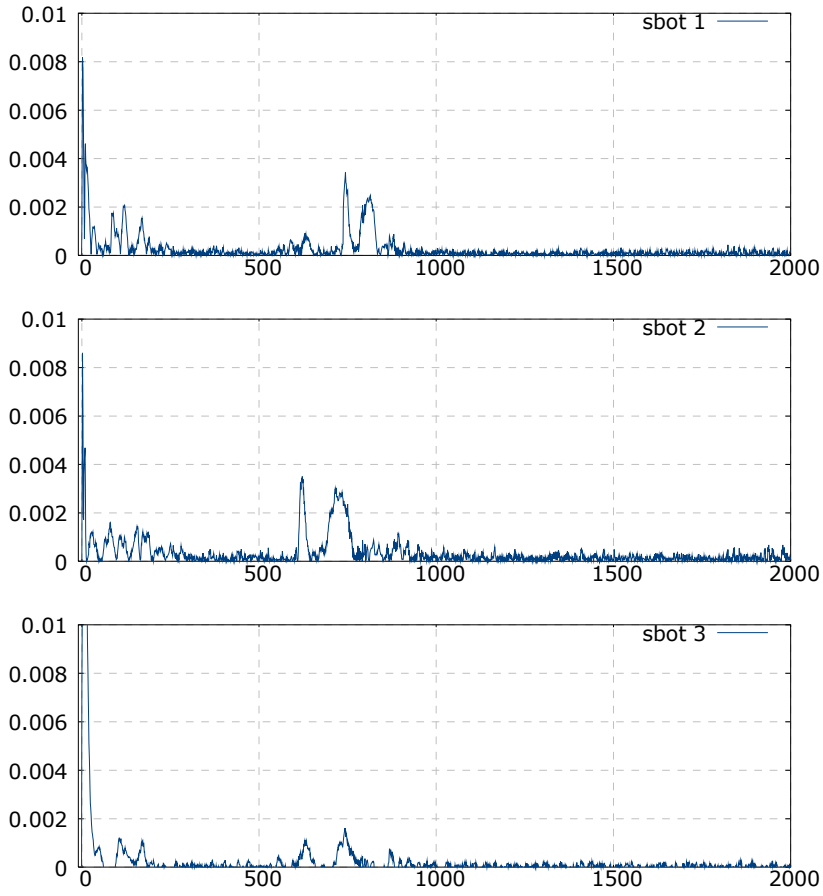

Figure 43: Coordination dynamics: angular distance between the chassis of each *s-bot* and the average orientation of the group.

8.3.1 Robustness

The main limit of the evolved behavior is that coordinate motion occurs only when the robots are initialized with similar orientations. In the opposite case, the robots often separate in the first control steps and by moving away they lose visual contact with each other.

We tried to synthesize robust controllers by initializing the robots during artificial evolution to completely random directions. Unfortunately, evolution with these settings displayed bootstrap problems, being unable to synthesize controllers that kept the robots together. Probably, the main cause of this limit is the use of a purely reactive neural network controller. Since the *perceptron* moving the robots lacks memory, it cannot easily discriminate the initial control phase, where robots may need to establish a common direction of movement, from the secondary phase when robots must swarm. Therefore, robots receiving similar inputs at the beginning of the task behave in the same way, as leaders for instance, and often compromise the swarming of the group. In future, we will replicate our studies in coordinate motion using different neural architectures with memory. We hope that different controllers will produce behaviors characterized by a greater robustness. The main disadvantage over a simple reactive controller is that artificial evolution would require many more generations to eventually produce acceptable solutions.

8.3.2 Generalization Properties

One of the objectives of this thesis is studying how controllers for the coordinate motion of some *s-bots* scale to groups of different sizes.

At this purpose, we measure the swarming fitness on six groups with a different number of robots, from 3 to 8. The evaluation is repeated 50 times for each group and then averaged. The robots are always initialized with similar orientations because, as mentioned in section 8.3, the evolved controllers do not work efficiently when the robots initially point to divergent directions. We also randomly place the robot at a short distance among each other of 24 *cms* (4 times a *s-bot*'s radius). Testing a number of s-bots superior to three with the fitness function described in 8.2.4 introduces a problem: that fitness cannot detect the possibility that robots divide in sub-groups that independently explore the environment. This particular behavior is actually correct according to our fitness formulation since every robot follows at least another one and no robots are lost. Moreover, this mechanism may be useful in more advanced tasks involving the exploration of complex environments, in which case it might be advantageous for robots to separate in small groups, each thereafter exploring independently an area. Nevertheless, we decided to penalize the separation in small groups. Therefore, we modified the fitness function illustrated in 8.2.4 as following: we additionally calculate the distance D_i' of each robot from the sensorial range of the *furthest* robot. When this distance exceeds a maximum, dependent on the number of robots, the robots receive a binary penalty according to:

$$P_i = \begin{cases} 0 & \exists\, D_i' \mid D_i' > \tau \\ 1 & otherwise \end{cases}$$

where τ *is a threshold* equal to *(N-1)*R*, *N* is the number of *s-bots* and *R* is the sensorial range of each robot. In other words, the penalty is null when one robot cannot be linked, directly or indirectly through a chain of other robots, to the furthest one, which happens when some robots separate from the group.

The fitness F_i of the group now takes into account the penalty:

$$F_i = F_i^a * F_i^m * P_i$$

Robots that divide in sub-groups always receive a null penalty P_i and consequently a null swarming fitness F_i.

Note that in calculating the fitness, we use a larger sensory range (36, equal to 72 *cms*) than the one used during evolution, in order to give robots a higher freedom of movement. Furthermore, we do not interrupt the evaluation when one or more robot s get lost, to give them the possibility to reunite with the main group.

Table 12 compares the performance of coordinate motion on different numbers of robots, penalizing the division in sub-groups.

Table 12: Average performance for different group sizes

Number of *s-bots*	**Average performance**
3	0.3722
4	0.2346
5	0.1468
6	0.1164
7	0.1059
8	0.1023

As shown, the performance decreases as we add more robots to the group. The reason is that the evolved controllers are not sufficiently robust to handle all the possible initial conditions determined by a large number of robots. Therefore, some robots often separate from the group, especially in the initial phase, and the fitness takes a null value. Anyway, direct observation of the group's behavior showed that in many cases groups of many *s-bots* are capable of swarming for long distances maintaining a complete aggregation. This fact encourages further studies in this area in the attempt to improve the robustness of the controllers. As already explained, different controller architectures and better tuned evolutions might contribute to he achieve an improved robustness over the one obtained so far. We will investigate this possibility in the near future.

The coordinate motion behavior displays another interesting generalization property in some of the evolved controllers: the robots steer to stay on the ground when they detect the ground's boundaries, corresponding to a black visual input. This result suggests that controllers evolved the capability to discriminate three different colors (white for robots, grey for ground and black for the background) and react to them in different ways while respecting the assigned task. For instance, we tried to place black obstacles in a small arena delimited by black walls. The robots were able to coordinately explore the arena while avoiding in many cases collisions with the obstacles and the walls. This behavior, though not explicitly requested during evolution, confirms the flexibility of the retina device in advanced tasks guided by vision. In the conclusions, chapter 9, we describe future applications to extend our research.

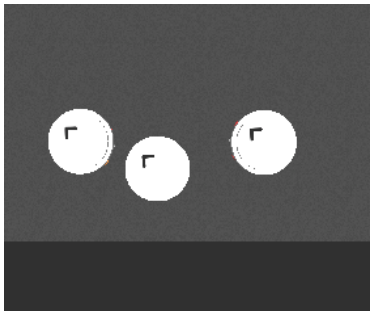

Figure 44: Robots steer when they detect the black visual inputs.

9

CONCLUSIONS

In this thesis, we have presented a novel active vision system applied to the control of single and multiple robots. The vision system includes an omni-directional camera acquiring panoramic, 360-degree views of the surrounding environment. Robots are equipped with this camera in order to constantly acquire local, visual information about their environment. The resulting camera images represent an extremely rich source of information available to the robots' controllers for the accomplishment of predefined tasks. Hard-coded algorithms for image processing and features extraction are not feasible because the acquired images have complex characteristics. On one end, the optical process characterizing the omni-directional camera produces images with strong distortions, in comparison to images acquired by common linear cameras. Therefore, standard features extraction mechanisms, as edge detection, depth from shading and so forth, do not immediately apply to these images. On the other end, the images are highly unpredictable being the result of complex, dynamic interactions between the perceiving robot and the environment, comprehensive of other mobile robots in collective robotics tasks. In order to overcome these challenges, we processed the images through a *virtual retina*, an acquisition device corresponding to an area of the image of variable size and position, capable of scanning the images and automatically extracting relevant information. This information enters the robots' neural network controller and eventually determines the successive robots' behavior in the form of actions affecting the robot's motion and vision system, according the active vision paradigm.

We applied a generational evolutionary algorithm to synthesize automatically efficient controllers for different vision-based tasks. Artificial evolution, by testing robots and their vision system directly in the working environment, allowed the design of controllers considering the complex mechanisms that affect the robot in the whole control phase. The solutions found by evolution are efficient and often generalize to different environmental situations.

9.1 Obtained Results

We studied the developed vision system in three experiments of increasing complexity. The first consisted in evolving a virtual retina capable of focusing on

static objects placed in the environment and perceived by an omni-directional camera. The results demonstrated the efficiency of the virtual retina in detecting and focusing on objects under generic conditions using a minimal, purely reactive neural network controller. Additionally, the evolved retinas are capable of tracking moving objects, even if they were evolved in a static context.

These results served as a premise for the successive experiment, dealing with a mobile robot with the task of targeting a distant object. The robot is equipped with the developed active vision system, consisting in an omni-directional camera for the perception of the environment, and in the virtual retina for image processing. We have observed that the evolved robots can maintain the retinal focus on the object in order to estimate its relative position in the environment and develop efficient navigation trajectories towards it. Moreover, further analysis confirmed the flexibility of the retina in the detection and tracking of objects. In fact, the evolved robots are particularly robust, being capable of targeting objects of different colors, distances and shapes. As a generalization property, robot can also follow and possibly reach moving objects, despite artificial evolution evaluated robots in a static context. The latter result is particularly important because it demonstrates that the retina could work efficiently on mobile robots perceiving dynamic objects, exactly the setup of the last experiment.

The third and last experiment is the most important of the thesis because, to the best of our knowledge, it applies techniques of active vision in a collective robotics context. The objective was that of evolving coordinate motion in a group of robots exploiting only local, visual information, actively extracted from the environment. The robot in particular had the task of moving in a coordinate fashion while maintaining a compact formation, a behavior known as swarming or flocking in some animal societies as fishes and birds. The task was particularly challenging because the robots are autonomous and independent units, yet requested to behave coherently as a group. Every robot is equipped with an omni-directional camera to perceive the other robots around, and with the virtual retina to extract visual information related to the other robots that could be useful for the swarming. We evolved some reactive neural controllers that, despite their simplicity, produced promising results. Robots starting from nearly the same orientation are capable of coordinately moving towards a common direction while maintaining visual contact with near robots. In particular, each robot exploits a wide retina to detect near robots all around, in order to constantly align its trajectory and follow the group. This behavior allows the formation of mobile robotic configurations maintained linked only through visual information locally acquired by each robot composing the group. By analyzing the evolved controllers, we have observed that the evolved strategy allows the coordinate motion of groups of different sizes. Actually, the generalization is not very robust because some robots often depart from the group, especially in the initial control phase. However, groups of numerous s-bots are often able to move in a

compact formation for long times by adopting the aforementioned strategy. In view of the simple controllers here employed, we retain this result particularly brilliant. In the next section we discuss our plans to improve and extend our research.

9.2 Future Work

The work done in this thesis opens the way to many possible future enhancements and applications.

Our research will mainly focus on improving the synthesis of controllers for groups of robots guided by the developed vision system. In particular, we want to investigate improved solutions to the accomplishment of cooperative tasks by extending the research done for the last experiment of this thesis. We will study different design choices (e.g. non-reactive neural networks) to synthesize better controllers, not only for a task of coordinate motion but also for more advanced tasks. An idea is replicating the experiment by Trianni [63] concerning the synthesis of controllers for hole avoidance in groups of robots. We would like to repeat that experiment employing unlinked robots guided by vision. We will also study the integration of the vision system with additional sensorial capabilities, which might prove necessary in complex environments.

Within the robotic context, we also plan to devise other experiments with a single robot. As an extension of the targeting experiment, we would like to evolve robots capable of detecting and approaching specific objects in the environment, characterized by particular colors or shapes, while at the same time avoiding collision with obstacles and walls. The retina in this case would be useful for tracking the target objects in the scene and for detecting the obstacles along the robot's trajectory.

Apart from the control of robots, the virtual retina can be used for the efficient analysis of images in systems having low computational capabilities. In [43], the retina was employed in a simple discrimination task. As a hint for future activity, other researchers may extend that approach by exploiting our vision system to solve complex tasks as recognition of facial features in photos or tracking of moving objects in surveillance cameras, traditionally solved by massive neural networks or by hard-coded image processing algorithms.

BIBLIOGRAPHY

[1] C. Anderson, G. Theraulaz, and J.-L. Deneubourg. Self-assemblage in insects societies. *Insectes Sociaux*, 49:99-110, 2002.

[2] N. Ayache. Artificial Vision for Mobile Robots: Stereo Vision and Multisensory Perception. MIT Press, Cambridge MA, 1991.

[3] R. Bajcsy. Active Perception. *Proceedings of the IEEE*, 76:996-1005, 1998.

[4] T. Balch and R.C. Arkin. Communication in reactive multiagent robotic systems. Autonomous Robots, 1(1):27-52, 1994.

[5] G. Baldassarre, S. Nolfi, and D. Parisi. Evolving Mobile Robots Able to Display Collective Behaviors. In C.K. Hemelrijk and E. Bonabeau, editors, *Proceedings of the International Workshop on Self-organization and Evolution of Social Behavior*, pages 11-22, Monte Verità, Ascona, Switzerland, September 8-13, 2002.

[6] D. Ballard. Animate vision. *Artificial Intelligence*, 48:57-86, 1991.

[7] P. A. Beardsley, I. D. Reid, A. Zisserman, and D. W. Murray. Active visual navigation using non-metric structure. In *Proceedings of the 5th International Conference on Computer Vision*, Boston, pages 58-65. IEEE Computer Society Press, 1995.

[8] A. Blake, A. Zisserman, and R. Cipolla. Visual exploration of free-space. In A. Blake and A. Yuille, editors, *Active Vision*. MIT Press, Cambridge, MA, 1992.

[9] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, NY, 1999.

[10] J. E. Bresenham. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*. Volume 20, issue 2, pages 100-106. ISSN 0001-0782, 1977.

[11] S. Camazine, J.-L. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, Princeton, 2001.

[12] S. Camazine and J. Sneyd. A model of collective nectar source selection by honey bees: self-organization through simple individual rules. *Journal of Theoretical Biology*, 149:547-571, 1991.

[13] S. Camazine, J. Sneyd, M.J. Jenkins, and J.D. Murray. A mathematical model of self-organized pattern formation the combs of honeybees colonies. *Journal of Theoretical Biology*, 1:295-311, 1990.

[14] G.S. Chirikjian. Kinematics of a Metamorphic Robotic System. In E. Straub and R. Spencer Sipple, editors, *Proceedings of the International Conference on Robotics and Automation*. Volume 1, pages 449-455. IEEE Computer Society Press, 1994.

[15] D.T. Cliff and J.Noble. Knowledge-based vision and simple vision machines. *Philosophical Transactions of the Royal Society of London*: Series B, 352:1165-1175, 1997.

[16] A. J. Davison, I. D. Reid, and D. W. Murray. The active camera as a projective pointing device. In *Proceedings of the 6th British Machine Vision Conference.* Birmingham, pages 453-462, 1995.

[17] J.-L. Deneubourg, S. Aron, S. Goss, and J.M. Pasteels. The self-organizing exploratory patterns of the argentine ant. *Journal of Insect Behavior*, 3:159-168, 1990.

[18] J.-L. Deneubourg and S. Goss. Collective patterns and decision making. *Ethology Ecology Evolution*, pages 295-311, 1989.

[19] J.-L. Deneubourg, J. C. Gregoire, and E. Le Fort. Kinetics of the larval gregarious behavior in the bark beetle Dendroctonus micans. *Journal of Insect Behavior*, 3:169-182, 1990.

[20] C. Detrain. Field study on foraging by the polymorphic ant species pheidole pallidula. *Insectes Sociaux*, 37(4):315-332, 1990.

[21] M. Dill, R. Wolf, and M.Heisenberg. Visual pattern recognition in drosophila involves retino-topic matching. *Nature*, 355:751-753, 1993.

[22] F. Du. Navigation in tight clearances using active vision. Technical Report OUEL 2041/94, Department of Engineering Science, University of Oxford, 1994. First year transfer report.

[23] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 3(26):396-407, 1996.

[24] H. von Foerster. On Self-Organizing Systems and Their Environments. In M.C. Yovits and S. Cameron, editors, *Self-Organizing Systems*, pages 31-50. Pergamon Press, London, 1960.

[25] J. D. Foley, A. van Dam, A., S. K. Feiner, and J. F. Hughes. *Computer Graphics. Principles and Practice*. Second edition. Addison-Wesley Publishing Company. ISBN 0-201-12110-7, 1990.

[26] B.P. Gerkey and M.J. Matari_c. Sold!: Auction methods for multi-robot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758-768, 2002.

[27] J.J. Gibson. The Ecological Approach to Visual Perception. Houghton Miffin, Boston. Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, Redwood City, CA, 1979.

[28] J. Gluckman and S. K. Nayar. Ego-motion and omnidirectional cameras. In *Proceedings of the 6th International Conference on Computer Vision*, Bombay, pages 999-1005, 1998.

[29] D. Goldberg and M. Matari_c. Design and evaluation of robust behavior-based controllers. In T. Balch and L.E. Parker, editors, *Robot Teams: From Diversity to Polymorphism.* A K Peters, 2002.

[30] C. G. Harris and J. M. Pike. 3D positional integration from image sequences. In *Proc. 3rd Alvey Vision Conference*, Cambridge, pages 233-236, 1987.

[31] C. G. Harris and M. Stephens. A combined corner and edge detector. In *Proc. 4th Alvey Vision Conference*, Manchester, pages 147-151, 1988.

[32] C. G. Harris. Tracking with rigid models. In A. Blake and A. Yuille, editors, *Active Vision*. MIT Press, Cambridge, MA, 1992.

[33] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: Artificial evolution, real vision. In D. Cliff, P. Husbands, J.-A. Meyer, and S.W.Wilson, editors, *From Animals to Animats 3: Proc. of the Third Int. Conf. on Simulation of Adaptive Behavior*, pages 392-401. The MIT Press, 1994.

[34] F. Heylighen. The Science of Self-Organization and Adaptivity. In The Encyclopedia of Life Support Systems (EOLSS), Knowledge Management,

Organizational Intelligence and Learning, and Complexity. Developed under the Auspices of the UNESCO, Eolss Publishers, Oxford, UK, 2003.

[35] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[36] E. Huber and D. Kortenkamp. A behavior-based approach to active stereo vision for mobile robots. To appear in *Engineering Applications of Artificial Intelligence Journal*, 1998.

[37] E. Huber and D. Kortenkamp. Using stereo vision to pursue moving agents with a mobile robot. In *IEEE International Conference on Robotics and Automation*, May 1995.

[38] T. Kato and D.Floreano. An evolutionary active-vision system. In *Proceedings of the Congress on Evolutionary Computation (CEC01)*, Piscataway. IEEE Press, 2001.

[39] E. A. Krupinski and R. M. Nishikawa. Comparison of eye position versus computer identified microcalcification clusters on mammograms. *Medical Physics*, 24:17-23, 1997.

[40] F. Li. Active Stereo for AGV Navigation. *PhD thesis*, University of Oxford, 1996.

[41] A. Lioni, C. Sauwens, G. Theraulaz, and J.-L. Deneubourg. Chain formation in Œcophilla longinoda. *Journal of Insect Behavior*, 15:679-696, 2001.

[42] H. A. Mallot. *Computational Vision*. MIT Press, Cambridge, MA, 2000.

[43] D. Marocco and D. Floreano. Active Vision and Feature Selection in Evolutionary Behavorial Systems. In *Proceedings of the Seventh Conference on Simulation of Adaptive Behavior*, pages 247-255. MIT Press, 2002.

[44] M.J. Matarić. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems*, 16:321-331, 1995.

[45] F. Mondada, G. C. Pettinaro, I. Kwee, A. Guignard, L. Gambardella, D. Floreano, S. Nolfi, J.-L. Deneubourg, and M. Dorigo. SWARM-BOT: A swarm of autonomous mobile robots with self-assembling capabilities. In C.K. Hemelrijk and E. Bonabeau, editors, *Proceedings of the International Workshop on Self-organization and Evolution of Social Behavior*, pages 307-312, Monte Verità, Ascona, Switzerland, September 8-13, 2002.

[46] G. Nicolis and I. Prigogine. *Self-Organization in Non equilibrium Systems*. John Wiley & Sons, New York, 1977.

[47] S. Nolfi. Evolving non-trivial behavior on autonomous robots: Adaptation is more powerful than decomposition and integration. In T.Gomi, editor, *Evolutionary Robotics*, pages 21-48. AAI Books, Ontario (Canada), 1997.

[48] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press/Bradford Books, Cambridge, MA, USA, 2000.

[49] L.E. Parker, G Bekey, and J Barhen, editors. *Distributed Autonomous Robotic Systems 4*. Springer, Tokyo, Japan, 2000.

[50] G.C. Pettinaro, I.W. Kwee, L.M. Gambardella, F. Mondada, D. Floreano, S. Nolfi, J.-L. Deneubourg and M. Dorigo. Swarm robotics: A different approach to service robotics. In *Proceedings of the 33rd International Symposium on Robotics*, Stockholm, Sweden, October 7-11 2002. International Federation of Robotics.

[51] M. Quinn. A comparison of approaches to the evolution of homogeneous multi-robot teams. In *Proceedings of the 2001 Congress on Evolutionary Computation (CEC2001)*, pages 128–135. IEEE Press, Piscataway, NJ, 2001.

[52] M. Quinn. Evolving communication without dedicated communication channels. In J. Kelemen and P. Sosik, editors, *Advances in Artificial Life: Sixth European Conference on Artificial Life (ECAL 2001)*, pages 357-366, Berlin, 2001. Springer-Verlag.

[53] M. Quinn, L. Smith, G. Mayley, and P. Husband. Evolving teamwork and role allocation with real robots. In R.K. Standish, M.A. Bedau, and H.A. Abbass, editors, *Proceedings of the 8th International Conference on Artificial Life*, pages 302-311. MIT Press, 2002.

[54] C.W. Reynolds. An evolved, vision-based behavioral model of coordinated group motion. In J.-A. Meyer, H. Roitblat, and S.W.Wilson, editors, From *Animals to Animats 2. Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92)*, pages 384–392. MIT Press, Cambridge, MA, 1993.

[55] S. Rougeaux and Y. Kuniyoshi. Robust real-time tracking on an active vision head. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1997.

[56] E. Şahin, T.H. Labella, V. Trianni, J.-L. Deneubourg, P. Rasse, D. Floreano, L.M. Gambardella, F. Mondada, S. Nolfi, and M. Dorigo. SWARM-BOTS: Pattern formation in a swarm of self-assembling mobile robots. In A. El Kamel, K. Mellouli, and P. Borne, editors, *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisia, October 6-9, 2002. Piscataway, NJ: IEEE Press.

[57] E. Şahin and P. Gaudiano. Visual Looming as a range sensor for mobile robots. In Pfeifer, R., Blumberg, B., Meyer, J., and Wilson, S., (Eds.), From Animals to Animats V: *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*. MIT Press-Bradford Books, Cambridge, MA, 1998.

[58] T.D. Seeley. The Wisdom of Hive, pages 277-290. Harvard University Press, Cambridge, 1995. T.D. Seeley, S. Camazine, and J. Sneyd. Collective decision-making in honey bees: how colonies choose among nectar source. *Behavioral Ecology and Sociobiology*, 28:277-290, 1991.

[59] L. Spector, J. Klein, C. Perry, and M.D Feinstein. Emergence of collective behavior in evolving populations of flying agents. In E. Cantù-Paz et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, volume 2723 of Lecture Notes in Computer Science, pages 61–73. Springer-Verlag, Berlin, Germany, 2003.

[60] K. Støy, W.-M. Shen, and P. Will. Global locomotion from local interaction in self-reconfigurable robots. In W.M. Shen, C. Torras, and H. Yuasa, editors, *Proceedings of the 7th international conference on intelligent autonomous systems (IAS-7)*, pages 309-316. IOS Press, 2002.

[61] J. I. Thomas and J. Oliensis. Automatic position estimation of a mobile robot. In *IEEE Conference on AI Applications*, pages 438-444, 1993.

[62] V. Trianni, R. Gross, T.H. Labella, E. Şahin, P. Rasse, J.-L.Deneubourg, and M. Dorigo. Evolving Aggregation Behaviors in a Swarm of Robots. Technical Report TR/IRIDIA/2003-07, IRIDIA -Université Libre de Bruxelles, Belgium, February 2003.

[63] V. Trianni. Evolution of Coordinate Motion Behaviors in a Group of Self-Assembled Robots. Technical Report TR/IRIDIA/2003-25, IRIDIA-Université Libre de Bruxelles, Belgium, May 2003.

[64] C.R. Ward, F. Gobet, and G. Kendall. Evolving collective behavior in an artificial ecology. *Artificial Life*, 7(2):191-209, 2001.

[65] G. M. Werner and M. G. Dyer. Evolution of herding behavior in artificial animals. In J.-A. Meyer, H. L. Roitblat, and S. W. Wilson, editors, From *Animals to Animats 2. Proceedings of the Second International Conference on Simulation of Adaptive Behavior (SAB92)*, pages 393–399. MIT Press, Cambridge, MA, 1992.

[66] C. J. Westelius. Focus of Attention and Gaze Control for Robot Vision. Department of Electrical Engineering. Linköping University, S-581 83 LINKÖPING, Sweden, 1995.

[67] N. Zaera, D. Cliff, and J. Bruten. (Not) evolving collective behaviors in synthetic fish. In P. Maes, M. Matarić, J.-A. Meyer, J. Pollack, and S. Wilson, editors, *From Animals to Animats 4. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB96)*, pages 635–644. MIT Press, Cambridge, MA, 1996.

APPENDIX

Simulator Guide

The following is a guide to the programming of experiments within the simulator developed for this thesis. Examples and extracts from working code are given to illustrate the programming of new functionalities and the use of the available classes. The simulator is written in C++, so knowledge of this programming language is required. The whole software code is available upon request to the author at steflanz@tin.it

Classes overview

The following diagram shows the organization of classes available at the moment within the simulator. Extension of these classes is possible through the mechanism of objects inheritance offered by the C++ language.

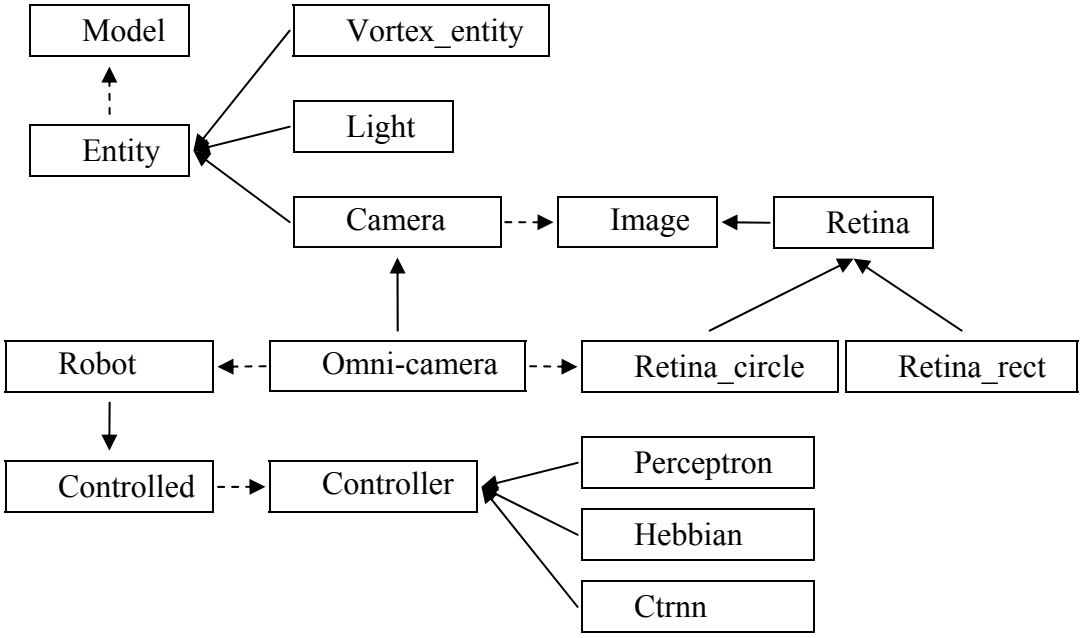

Figure: Classes organization in the simulator. A thick arrow means inherited from, a dotted arrow means the class contains the other as member.

We will now explain the meaning of each class, along with the list of the most significant functions for their manipulation, practical examples about their use, and

hints on extensions or alternative uses. For details on the implementation of each class and of its member functions, please refer to the simulator's source code.

Model stores geometrical information about a 3D model displayed by the simulator's graphical engine. The simulator can procedurally generate primitive shapes, as cone, square, sphere, and load models from file.

Entity is a base class representing generic objects, as Vortex physical bodies, lights, cameras, characterized by a position and orientation in the virtual environment. Optionally, an entity can be associated to a 3D model to be displayed during the simulation.

Main functions:
- set_origin (position): move the entity to a certain position.
- set_orientation (angles): rotate the entity according to the given angles.
- set_model (model_name): associate a 3D model to the entity for display purposes.
- link_entity (child): link the entity to another one, establishing a hierarchy.
- show (): show the entity during rendering.
- hide (): hide the entity during rendering.
- update (frame_time, application_time): update the entity's state.
- render (): render the entity if there is an associated 3D model.

Examples:

```
entity = new entity_t;
entity->set_origin( vec3(0,0,0) );
entity->set_angles( vec3(0, 90, 0) );
entity->set_model((“cone”);
```

This code generates a cone placed in the center of the scene and pointing upwards (90 degrees). In order to define particular behaviors during the update of the object, we have to inherit a new class from the basic entity and redefine the contents of the function update, and then use this class when creating the object.

Vortex_entity represents physical bodies simulated by the Vortex (Critical Mass Labs, Canada) physics engine. These bodies automatically update their position and orientation by interfacing to the corresponding Vortex model, thus realizing a perfect consistency between Vortex and our simulator.

Main functions:
- attach_Vortex_model (model): attach a Vortex model to this entity.
- attach_Vortex_body (body): attach a Vortex physical body to this entity.

Examples:

```
robot_wheel = new vortex_entity_t();
```

```
robot_wheel->set_model ("sphere");
robot_wheel->attach_Vortex_model (s-bot_wheel_modelID);
```

This code creates a wheel, rendered as a sphere, linked to a Vortex model representing the wheel of an s-bot.

Camera represents virtual cameras acquiring three dimensional views of the simulated environment. The basic class is a linear camera (CCD camera) acquiring images through perspective projection, the standard in computer rendering.

Main functions:
- acquire_image (): render and project the scene on the camera's image from the camera's point of view (POV).

Examples:

```
robot_chassis = new vortex_entity_t();
camera = new camera_t();
robot_chassis->link(camera);
…
camera->acquire_image();
```

The above code links a camera to the chassis of one robot; thereafter, the camera will automatically acquire images of the environment from the point of view of the chassis at each render call.

Omni_camera is the simulated omni-directional camera of the *s-bots*, whose simulation is accurately described in chapter 4 of the thesis.

Main functions:
- acquire_image (): acquire panoramic images of the surrounding environment.
- add_retina (retina): add a virtual retina operating on the acquired images.
- setup_mirror (height, radius): setup the camera's mirror with given parameters.

Examples:

```
robot_turret = new vortex_entity_t();
omnicamera = new omni_camera_t();
robot_turret->link(omnicamera);
```

This code creates an omni-directional camera and attaches it to the turret of one robot, in order to acquire 360-degree views of the environment around the robot, as used in our experiments.

Light represents lights with a position in the scene (even infinitely far in case of directional lights), simulated by OpenGL lighting calls. Point, spot and directional lights are supported.

Main functions:
- set_diffuse (color): set the light's diffusive color.
- set_specular(color) : set the light's specular color.
- set_attenuation (value): set the light's attenuation.
- set_ambient(color): set the light's ambient color.
- render (): activate OpenGL's lighting using this light's parameters.

Examples:

```
robot_LED = new vortex_entity_t();
light = new light_t();
light->set_diffuse( vec3(1, 0, 0) );  // red
robot_LED->link(light);
```

This code attaches a red light to the LED of one robot. The result is that the environment will be dynamically illuminated by the LED as the robot moves around.

Image represents images acquired by a camera, or sub-regions of other images (as the virtual retina).

Main functions:
- link (parent): make this image a region of a parent image

Examples:

```
retina = new retina_rect _t();
omnicamera = new omni_camera_t();
retina->link( omnicamera->get_image() );
```

This code creates a square retina and links it to the images acquired by an omni-directional camera. This way, the retina always automatically updates its contents by reading them from the corresponding area of the camera image.

Retina is a generic virtual retina, used for the extraction of data from an associated *image*.

Main functions:
- set_filter (filter): set the retina filtering method to sampling or averaging (box filter).
- capture_image (): copy the retinal region from the associated image to an internal buffer.
- filter_cells() : calculate the value of the retinal cells by filtering the corresponding pixels.
- pan(param1, param2): pan the retina according to two parameters.
- zoom (param1, param2): zoom the retina according to two parameters.

Retina_rect is a rectangular (usually square) retina, operating for instance on static bitmaps or on images acquired by a linear camera.

Main functions:
- hit_border (): tells whether the retina has hit the image border or not.
- pan (radius, angle): move the retina from the current top-left corner.
- zoom (zoom_factor): zoom the square retina.

Examples:

```
retina =  new retina_rect_t(64, 64, 32, 4, 4);
retina->pan(10, 0);  // to the right
if ( retina->hit_border())
  cout << "Retina hitted image border!";
retina->zoom(0.5);
```

This code creates a square retina of 64x64 pixels, 32 bits per pixel, divided into 4x4 cells. Then it translates the retina to the right, check if the retina has hit the image border and then halves the retina's size by zooming in.

Retina_circle is a circular retina, used for the extraction of data from images acquired by an omni-directional camera.

Main functions:
- pan (radius, angle): move the retina along the radial direction and rotate it of a certain angle.
- zoom (length, aperture): modify the retina's length and aperture.

Controlled is an abstract class representing objects, as robots or independent cameras, controlled during the simulation.

Main functions:
- set_controller (controller): associate a neural network controller to this object.
- control (): control the object.
- sense (): acquire sensory input by "sensing" the environment.
- set_inputs (num_inputs, inputs): fill the input neurons of the network.
- apply_outputs (num_outputs, outputs): map the output neurons to the corresponding actuators.

Controller is a base neural network, with input and output neurons, and weights.

Main functions:
- init (filename): initialize the neural network from file.
- init (genes, length): initialize the neural network with a corresponding genotype.
- step (): calculate the output neurons

- reset (): reset the neural network.

Percepton, Hebbian, Ctrnn are three types of available neural networks, inherited from the base class *controller*.

Example:

```
robot = new robot_t();  // inherited from controlled
… // read controller's parameters from XML
robot->set_controller(new perceptron_t(num_inputs, num_outputs));
```

This code creates a new robot's controller in the form of a *perceptron*.

Robot is the class for the simulation of *s-bots*. It contains as members a set of *vortex_entities* representing robotic components as wheels, the turret, the bulk… Moreover, it contains as member an *omni-camera* for images acquisition in our experiments. *Robot* is inherited from *controlled* and defines its own control functions.

Main functions:

- set_position (position): move the robot to a certain position.
- set_orientation (angles): rotate the robot.
- get_position (): return the robot's position, useful in experiments.
- get_orientation(): return the robot's orientation.
- get_params(): return parameters relative to the s-bot.
- attach_camera (camera): attach a camera to the *s-bot* for images acquisition.
- update (): update the robot's state at every control step.
- control (): control the robot, by reading sensory inputs and then calculating and passing the outputs to the actuators.
- sense (): sense the environment.
- set_inputs (num_inputs, inputs): passes the acquired sensorial data to the neural inputs controlling the robot.
- apply_outputs (num_outputs, outputs): maps the neural outputs to the simulated actuators moving the robot and its components.

Examples:

```
robot_t::control() {
  sense();
  // set the controller inputs
  inputs = controller->get_inputs();
  inputs.clear();
  set_inputs(controller->get_num_inputs(), inputs);
  // calculate outputs of the neural network
  controller->step();
```

```
    // apply outputs
    outputs = controller->get_outputs().begin();
    apply_outputs(controller->get_num_outputs(), outputs);
  }
```

The above code is the control phase of a simulated robot. The robot acquires images through its omni-directional camera, calculates visual input through a virtual retina, passes this input to the neural network, calculates the neural outputs and maps them to the corresponding motor actuators.

Programming Experiments

Programming new experiments within the simulator is quite easy. In essence, experiments are defined inside a corresponding C++ class inherited from a base class *task.* Every experiment must redefine the following virtual functions.

Trial evaluation

- start_evaluation (): operations done at the beginning of an experiment, e.g. reading from script some parameters, creating robots and their omni-camera, initializing controllers, collision detection…
- close_evaluation(): operations done at the end of an experiment.
- start_trial (trial): operations done at the beginning of a trial, e.g. random positioning of robots and obstacles in the scene.
- run_trial (trial, num_control_steps): operations done at each control step, as updating the simulator's entities (lights, cameras…), running Vortex simulation, updating the fitness and rendering the environment.

Fitness tools

- compute_fitness (trial): compute the fitness of a trial. Refer to the thesis for examples on the fitness formulations used for our experiments.
- compute_final_fitness (): compute the fitness that evaluates the experiment, usually by averaging the fitness of each trial.
- dump_data(): save useful data to file.

Test and data analysis

- start_test (): initialize data used during tests (e.g. opening output file and reading from script the desired type of test).
- close_test (): close data used for testing the experiment (e.g. closing the output file).
- initialize_test (trial, num_control_steps): initialize trial-dependent tests, e.g. varying the color or position of objects in function of the trial number.

- control_test (trial, control_step, num_control_steps): perform operations at each control step for testing the experiment. For example, in the focusing experiment, here we rotate the camera to study its ability to focus on dynamic images.
- analyze_test (trial, num_control_steps): analyze data gathered throughout testing. Here, we usually write to file the result of a test in multiple-columns format, for successive plotting under Matlab or gnuplot.

Example

We will now illustrate the programming of an experiment concerning a robot navigating in the environment driven by visual information acquired through a retina. First, we have to inherit a new robot class, from robot, specifying the actions performed at each control step and optionally additional components not already present in the base robot class.

```
class navigator_robot_t : public robot_t {
  virtual void sense();
  virtual void set_input(…);
  virtual void set_output(…);
  …
}
```

The robot senses the environment by acquiring 360-degree images of the surrounding environment, and then by extracting data through the retina.

```
sense(){
  omnicamera->acquire_image();
  omnicamera->get_retina(0)->filter_cells();
}
```

The robot then passes the grey value of the retinal cells as input to the neural network.

```
set_input(…){
  for (int i = 0; i < num_retina_cells; i++) {
    neural_network->inputs[i] = grey_value( retina->cells[i] );
  }
}
```

Finally, the robot maps the neural outputs into the corresponding actuators activation states, controlling in this case the robot's wheels.

```
set_ouput(…){
  float lwheel = neural_network->remap(outputs[0], -max_speed,
                                                   +max_speed);
  float rwheel = neural_network->remap(outputs[1], -max_speed,
                                                   +max_speed);
  set_wheel_speed(0, lwheel);  // left wheel
```

```
    set_wheel_speed(1, rwheel);  // right wheel
  }
```

At this point, we have to program an experiment using the new robot, for example to move it without collision with obstacles randomly placed in the environment. The first step consists in initializing the experiment, by creating the involved objects:

```
  start_evaluation () {
    robot = new navigator_robot_t();
    omni_camera = new omni_camera_t();
    omni_camera->setup_mirror(12, 0.2);
    robot->attach_camera(omni_camera);
    // create a circular virtual retina
    retina = new retina_circle_t(32, 4, 4);  // 4x4 cells
    omni_camera->add_retina(retina);
    // create a number of spherical obstacles
    // NOTE: this is actually done automatically when reading the
    Vortex XML files describing the simulated world
    num_obstacles = X;
    for (int i = 0; i < num_obstacles; i++) {
      obstacles[i] = new vortex_entity_t();
      obstacles[i]->set_model("sphere");
    }
    …
  }
```

The second step consists in describing the operations performed at the end of the experiment, as deleting objects.

```
  close_evaluation () {
    delete robot;
    delete omni_camera;
    …
  }
```

Now, we describe some operations done at the beginning of each trial that vary from trial to trial. In our case, we randomly position the robot and the obstacles in the environment.

```
  start_trial (int trial) {
    for (int i = 0; i < num_obstacles; i++)
      obstacles[i]->set_origin( some random position );
    robot->set_position( some random position );
    robot->set_orientation ( a random orientation );
  }
```

The most important function describes the operations done within the experiment at each control step, as rendering the virtual environment, updating the physical simulation of bodies and controlling the robot.

```
run_trial (int trial, int num_control_steps) {
  fitness[trial] = 0.0;
  for (int i = 0; i < num_control_steps; i++) {
    viewer().update_frame();
    // run simulation for one step
    simulation().run(1);
    // update fitness
    fitness[trial] += compute_navigation_fitness();
    // render the virtual environment
    viewer().render();
  }
  // normalize fitness
  fitness[trial] /= (float)(num_control_steps);
}
```

The fitness function evaluates somehow the robot's ability in navigating across the environment while avoiding the obstacles.

```
compute_navigation_fitness() {
  fitness = …
  return fitness;
}
```

Next, we have to define a function returning the robot's fitness. In our example, we have already calculated the fitness throughout the control phase, so we simply return the calculated fitness:

```
compute_fitness (int trial) {
  return fitness[trial];
}
```

Now, in order to analyze the evolved robot's behavior, we can design some tests through the following procedure: initially, we initialize the test, by opening a file to write data to.

```
start_test(){
  output_file.open("navigation.test");
}
```

We must also close the file at the end of the test.

```
close_test){
  output_file.close();
}
```

For this example, we propose a test consisting in randomly moving the obstacles in the environment, in order to test the robot's ability in avoiding collisions with them. Initially, we arrange the obstacles in predefined positions.

```
initialize_test(int trial, int num_control_steps) {
  for (int i = 0; i < num_obstacles; i++) {
    obstacles[i]->set_origin( obstacle_pos[i] );
  }
}
```

Second, we move the obstacles at each control step:

```
control_test(int trial, int control_step, int num_control_steps) {
  for (int i = 0; i < num_obstacles; i++) {
    obstacles[i]->translate( random );  }
  }
}
```

Note that we have to insert a call to `control_test` inside the loop defined in `run_trial`. The loop then calculates the fitness taking into account the operations defined for this test.

The last step regards the analysis of data. For example, we calculate and save to file the number of collisions between the robot and the obstacles in function of the number of control steps.

```
analyze_test(int trial, int control_step) {
  num_collisions += check_collision(robot, obstacle);
  output_file << control_step << " " << num_collisions << endl;
}
```

At the end of the experiment's test, we can plot the output file "navigation.test" inside any common plotting program, as gnuplot or Matlab.

ESTRATTO

Questa tesi discute un innovativo sistema di visione attiva applicato al controllo di un gruppo di robot. Vogliamo creare dei robot che siano in grado di estrarre autonomamente dall' ambiente le informazioni visive necessarie al completamento di un compito comune, che nel nostro caso consiste nel moto coordinato di un gruppo di robot.

La visione artificiale è un campo dell'intelligenza artificiale e dell'elaborazione delle immagini che si occupa dell'analisi e della corretta interpretazione di informazioni estratte dalle immagini. In letteratura esistono principalmente due approcci: il primo consiste nell' applicare algoritmi predefiniti per estrarre dalle immagini caratteristiche di interesse. Il secondo approccio applica invece tecniche di apprendimento nel tentativo di ridurre le immagini ad un insieme di caratteristiche ottimali e statisticamente invarianti.

Entrambi gli approcci possono essere applicati al controllo di robot autonomi. Tuttavia, essi non considerano il fatto che, in analogia con organismi viventi, la visione e il comportamento di un robot sono intimamente legati in un ciclo senso-motorio. Da una parte, le azioni svolte dal robot influenzano, direttamente o indirettamente, la percezione che il robot ha dell'ambiente circostante. Si pensi ad esempio a una telecamera montata sul robot; essa naturalmente vede l'ambiente sempre da posizioni diverse in conseguenza dei movimenti del robot. Dall'altra parte, le informazioni provenienti dalle immagini acquisite determinano, una volta elaborate da un controllore, il successivo comportamento del robot. In aggiunta, i due approcci menzionati precedentemente non considerano il fatto che la quantità e il tipo di informazioni acquisite dipendono fortemente dalle caratteristiche fisiche del robot e dalle sue capacità motorie.

In risposta a questi limiti, è stato recentemente proposto il paradigma della visione attiva. Un sistema di visione attiva è in grado, a differenza di un sistema passivo, di scegliere attivamente le informazioni visive che sono importanti al momento, invece di elaborare tutte le informazioni disponibili ma non necessariamente utili. Nel nostro caso, un robot dotato di visione attiva è in grado di scegliere quelle informazioni visive relative al proprio ambiente che gli sono utili per il completamento di un determinato compito.

Il sistema visivo sviluppato in questa tesi consiste innanzitutto in una telecamera omni-direzionale per l' acquisizione di immagini a 360 gradi dell' ambiente circostante. Successivamente, una retina virtuale, corrispondente a una regione dell' immagine di dimensione e posizione variabile, può analizzare autonomamente l'

immagine per estrarre informazioni che sono importanti per il compito del robot. Ad esempio, la retina può cercare nelle immagini alcune caratteristiche di interesse, e quindi zoomare per estrarle a diversi livelli di dettaglio.

L' approccio adottato implica un problema fondamentale: come sintetizzare controllori che siano in grado di guidare allo stesso tempo il sistema visivo e il movimento dei robot, in modo che essi svolgano un compito predefinito? A questo proposito, abbiamo adottato tecniche di evoluzione artificiale per creare dei controllori che considerino l'intrinsico legame tra movimento e visione caratterizzante il ciclo di controllo di un robot, ed inoltre le complesse interazioni tra i robot e l'ambiente in cui operano (vedi la sezione 2.2.2). La sezione 3.3 della tesi fornisce una breve introduzione agli algoritmi evoluzionari impiegati.

Nel processo di sintesi di un controllore robotico bisogna tenere in considerazione alcuni aspetti, quali decentralizzazione, robustezza e adattatività. Queste proprietà caratterizzano in generale il comportamento di insetti sociali ed altri animali, individui semplici che lavorando coordinatamente in gruppo sono in grado di svolgere compiti complessi. Il campo della Swarm Intelligence (introdotto nella sezione 2.1) studia i meccanismi che in natura regolano il comportamento degli insetti ed altri animali per poi applicarli al controllo di individui artificiali come i robot. Noi abbiamo cercato di applicare gli stessi principi al fine di riprodurre, mediante evoluzione artificiale, comportamenti individuali e globali caratterizzati dalle proprietà sopra menzionate.

Il lavoro discusso in questa tesi è svolto interamente in simulazione in modo da accelerare notevolmente la sintesi dei controllori. Il robot impiegato nei nostri esperimenti è l's-bot, sviluppato all' interno del progetto SWARM-BOTS, di cui la sezione 2.3 fornisce una breve introduzione. L' hardware, il modello simulato e il sistema visivo dell' s-bot sono descritti nella sezione 3.2 della tesi.

Abbiamo sviluppato un simulatore con molteplici funzionalità (capitolo 3). Innanzitutto, grazie all'integrazione con il simulatore fisico Vortex (Critical Mass Labs, Canada), è possibile simulare le dinamiche e le collisioni tra corpi fisici (robot, ostacoli) con grande accuratezza, in modo tale da ottenere comportamenti fedeli alla controparte reale. Abbiamo inoltre sviluppato un motore grafico per la visualizzazione 3D in tempo reale dell' ambiente simulato dal punto di vista di osservatori esterni e dal punto di vista di ogni robot. Queste ultime caratteristiche erano parte degli obbiettivi della tesi in quanto agevolano fortemente lo studio della visione nei nostri esperimenti.

La parte decisamente più problematica consiste nel simulare il processo che, nella telecamera omni-direzionale dei robot, genera immagini a 360 gradi dell'ambiente circostante. Da un punto di vista ottico questo processo è relativamente semplice: i raggi di luce provienti dall' ambiente da tutte le direzioni sono riflessi da uno specchio sferico in direzione della telecamera, formando immagini panoramiche con caratteristiche peculiari (ampiamente descritte nella sezione 3.2.3). Questo processo

può essere simulato in maniera naturale con la tecnica del ray-tracing presa dal campo della computer graphics, come descritto nella sezione 4.1 della tesi. Tuttavia, una implementazione immediata del ray-tracing in questo caso risulta essere estramemente lenta, e quindi del tutto inadatta ai nostri scopi. Ricordiamo infatti che i nostri esperimenti, utilizzando l'evoluzione artificiale, comportano lunghe e reiterate valutazioni dei robot e quindi richiedono una simulazione il più veloce possibile. A questo proposito, sono state introdotte numerose ottimizzazioni, anche essere ispirate a tecniche di computer graphics, per accelerare criticamente la simulazione di una telecamera omni-direzionale tramite ray-tracing. Queste tecniche, descritte nel capitolo 4, sfruttano hardware grafico comune per simulare la visuale omni-direzionale dello specchio che fa parte della telecamera. Successivamente, ricostruiscono velocemente l'immagine della telecamera partendo dalla visuale dello specchio sfruttando dati precalcolati. In aggiunta, è possibile ottimizzare ulteriormente la simulazione dell'immagine in presenza di una retina virtuale.

Il simulatore sviluppato infine comprende strumenti per la sintesi di controllori tramite algoritmi evoluzionari e altri strumenti per il controllo e la manipolazione ad alto livello di vari oggetti simulati e per la programmazione di nuovi esperimenti. L'appendice alla tesi fornisce esempi estesi sull' utilizzo di questi strumenti.

Come anticipato, la retina virtuale è lo strumento tramite il quale i robot possono estrarre informazioni dalle loro immagini dell'ambiente. Essa corrisponde a una regione dell'immagine di dimensioni e posizione variabili, in grado di muoversi nell'immagine e di zoomare. La retina è suddivisa in una matrice di celle che ricevono input visivo dalla corrispondente area dell' immagine. In particolare, ogni cella assume come valore la media dei corrispondenti pixel dell' immagine; questo calcolo implica in generale determinare quali pixel dell'immagine sono coperti dalla cella. Abbiamo sviluppato due tipi di retina: una rettangolare e una a settore circolare, descritte rispettivamente nelle sezioni 5.3 e 5.4 della tesi. Il primo tipo risulta di semplice implementazione ma poco si adatta alle immagini acquisite da una telecamera omni-direzionale, caratterizzate da proprietà specifiche (sezione 3.2.3). L' altro tipo di retina invece è stato sviluppato ad hoc per operare su quel tipo di immagini. Questa retina ha la forma di un settore circolare, di apertura e lunghezza variabile. Può inoltre ruotare nell' immagine in modo da estrarre informazioni sull' ambiente che circonda un robot sotto varie direzioni. La forma particolare di questo tipo di retina, convessa e con lati curvilinei, rende molto più difficile il calcolo del valore di ogni cella. A questo proposito abbiamo sviluppato una tecnica basata su algoritmi per il disegno di forme curve su schermo (sezione 5.4.1).

L' obbiettivo finale della tesi è quello di generare dei controllori che, sfruttando il sistema visivo sviluppato, guidino il moto coordinato in un gruppo di più robot (capitolo 8). In particolare, i robot devono imparare a muoversi in gruppo sfruttando le informazioni visive relative all' ambiente percepite tramite la loro telecamera e

successivamente estratte dalla retina virtuale. La tesi giunge al suo obbiettivo presentando tre esperimenti di complessità crescente.

Il primo esperimento consiste nel sintetizzare tramite evoluzione artificiale una retina capace di focalizzare un oggetto posto casualmente nell' ambiente (focusing task, trattato nel capitolo 6). Una telecamera omni-direzionale fissa cattura immagini dell' ambiente comprendente l'oggetto. Esso appare nell' immagine come una figura bidimensionale di dimensioni e posizione casuale. La retina deve cercare l'oggetto nell' immagine e quindi adattare le sue dimensioni per focalizzarlo al meglio. Il controllore adottato qua e nei successivi esperimenti è un perceptron, cioè una rete neurale puramente reattiva. Esso riceve in input il livello di grigio delle celle della retina, e controlla in output il comportamento della retina, cioè la sua posizione e le sue dimensioni.

Questo esperimento serve a studiare le proprietà della retina nell' estrarre caratteristiche dalle immagini partendo da un contesto semplice. I risultati sono incoraggianti perché la retina mostra ottime prestazioni nel focalizzare oggetti di varie caratteristiche, come diverse luminosità e distanze dalla telecamera. Un risultato particolarmente interessante è il fatto che la retina è in grado di focalizzare dinamicamente oggetti in movimento (sezione 6.4), sebbene essa sia stata evoluta su oggetti statici. Questo risultato dimostra le grandi capacità di adattamento della retina, e serve come solida premessa all' esperimento successivo

Il secondo esperimento consiste nell' evolvere in un robot mobile la capacità di riconoscere nell' ambiente un oggetto e quindi di muoversi verso di esso (targeting task, capitolo 7). Il robot percepisce l'ambiente tramite la sua telecamera omni-direzionale, e poi estrae informazioni dalle immagini acquisite tramite una retina virtuale. Il robot deve imparare a focalizzare la retina sull' oggetto e successivamente a sfruttare l'input visivo fornito dalla retina per muoversi verso l'oggetto. In questo esperimento il controllo del robot è caratterizzato dal ciclo senso-motorio discusso in precedenza. L' evoluzione artificiale deve dunque sintetizzare validi controllori (ancora dei perceptron) co-evolvendo la visione e il movimento del robot.

I risultati ottenuti, descritti nella sezione 7.4, sono brillanti: la retina focalizza in pochi passi di controllo l'oggetto target nell' immagine. Il robot a questo punto si muove velocemente verso l'oggetto stimandone la posizione tramite la retina, e sempre mantenendo la sua attenzione visiva sull'oggetto. Le analisi dei controllori ottenuti dimostrano un buon grado di robustezza nel comportamento del robot rispetto a oggetti con diverse proprietà, come colore o distanza dal robot. Il risultato però più interessante è la capacità del robot di inseguire oggetti in movimento. Questa proprietà di generalizzazione anticipa la capacità del sistema visivo sviluppato di operare su robot mobili che percepiscono altri oggetti in movimento, esattamente lo scenario dell' ultimo esperimento della tesi.

Come anticipato, il terzo e ultimo esperimento della tesi consiste nel sintetizzare dei controllori basati sulla visione che guidino il moto coordinato di un gruppo di

robot (swarming task, capitolo 8). In particolare, alcuni robot devono esplorare l'ambiente procedendo compatti in una certa direzione comune. Ogni robot è una unità autonoma che percepisce l'ambiente, comprensivo degli altri robot, tramite una telecamera. Ancora una volta il robot deve imparare ad estrarre dalle immagini acquisite le informazioni utili al suo compito, in questo caso muoversi coordinatamente col gruppo. Alla luce dei risultati precedentemente conseguiti, usiamo per ogni robot lo stesso controllore dell' esperimento di targeting, un perceptron che riceve in ingresso le informazioni estratte dalla retina e controlla in uscita il movimento e il sistema visivo del robot.

Abbiamo sintetizzato controllori in grado di coordinare il moto di un gruppo di robot in maniera soddisfacente. I robot sfruttano una retina di grande apertura per estrarre informazioni da varie direzioni sugli altri robot posti nelle vicinanze. A seconda dell' input visivo acquisito, i robot assumono ruoli distinti nel successivo moto coordinato del gruppo. Di solito, i robot procedono nell' esplorazione dell' ambiente in una configurazione a trenino, mantenendo, tramite la retina, contatto visivo coi robot vicini per non perdersi. Ulteriori analisi (sezione 8.4) confermano discrete proprietà di generalizzazione nei controllori evoluti. In particolare, abbiamo studiato le prestazioni dei controllori usando di volta in volta un numero diverso di robot. Sebbene i controllori non siano robustissimi, dato che spesso alcuni robot si allontanano dal gruppo, riescono in molti casi a guidare con successo e per lungo tempo lo swarming di molti robot. In futuro cercheremo di migliorare le prestazioni del gruppo, magari impiegando controllori di diversa architettura (con memoria) per superare i limiti di un comportamento puramente reattivo.

I risultati conseguiti in questa tesi aprono la strada a future applicazioni nel campo della robotica che utilizzino la retina virtuale per l'analisi e l'elaborazione efficiente di immagini in sistemi di controllo basati sulla visione. Il capitolo 9 della tesi fornisce numerosi spunti per lavori futuri. In breve, la nostra ricerca si impegnerà a migliorare la sintesi di controllori per gruppi di molti robot guidati dalla visione. In particolare, vogliamo estendere l'ultimo esperimento della tesi a compiti più avanzati, come il superamento di ostacoli. Studieremo inoltre l'integrazione del sistema visivo con altre capacità sensoriali, in quanto essa potrebbe essere necessaria per operare in ambienti complessi. Nuovi esperimenti saranno inoltre condotti su robot singoli. Infine, un lavoro futuro potrebbe essere l'applicazione della retina virtuale nell' elaborazione di immagini in sistemi con limitate capacità computazionali.