

Institut für Informatik
Lehrstuhl für Programmierung und Softwaretechnik

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN



Diploma Thesis

Mechanisms for Task Allocation in Swarm Robotics

Martin Burger

Aufgabensteller: Prof. Dr. Martin Wirsing

Betreuer: Dr. Matthias Hözl, Annabelle Klarl, Christian Kroiß

Abgabetermin: 30. März 2012

Ich versichere hiermit eidesstattlich, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, den 30. März 2012

.....
(*Unterschrift des Kandidaten*)

Zusammenfassung

Die Forschungsrichtung der *Schwarmrobotik* stellt Ingenieure vor die Aufgabe, die Organisationsstruktur ihres mechanischen Schwarms festzulegen. Eine der wichtigsten Entscheidungen ist dabei, welcher Roboter welcher Aufgabe nachgehen sollte, um eine vorliegende Mission zu erfüllen. Das zugehörige Optimierungsproblem nennt sich *Task Allocation* (Aufgabenzuweisung) und kann mit verschiedenen Mechanismen in Angriff genommen werden, beispielsweise mit auktionsbasierten Ansätzen, die auf Ergebnissen der Spieltheorie beruhen.

Diese Arbeit gibt einen Überblick über Mechanismen zur Aufgabenzuweisung in der Schwarmrobotik. Zu diesem Zweck wird ein neues Klassifizierungsschema vorgestellt, das Lösungsstrategien in die Kategorien *Heteronomous Task Allocation* (fremdbestimmte Aufgabenzuweisung), *Autonomous Task Allocation* (selbstbestimmte Aufgabenzuweisung) und *Hybrid Task Allocation* (hybride Aufgabenzuweisung) einteilt.

Der experimentelle Teil dieser Arbeit nutzt diese Systematik, um Lösungsansätze für ein konkretes Szenario zu entwickeln, das sich an die Natur anlehnt: Ähnlich einem Bienenschwarm sind mehrere Roboter einem Nest zugeordnet, das als Sammelstelle für Futter genutzt wird. Neben der Futtersuche hat der künstliche Schwarm die Aufgabe, das Nest nahe einer optimalen Temperatur zu halten. Andernfalls kann das gesammelte Futter nicht in Energie umgesetzt werden.

Diese Arbeit fokussiert sich auf einen wahrscheinlichkeits- beziehungsweise motivationsbasierten Ansatz sowie verschiedene Varianten von zentralem und dezentralem Reinforcement Learning, eine Form des maschinellen Lernens, bei der die Selektion von Aktionen durch Belohnungen bekräftigt wird. Zum Vergleich der verschiedenen Lösungsstrategien werden entsprechende Schwärme in statischen und dynamischen Umgebungen simuliert. Statisch bedeutet in diesem Fall, dass sich die Außentemperatur und das Futteraufkommen nicht ändern.

Zur Durchführung der Experimente wurde der *Swarmulator* entwickelt, eine Simulationsplattform, die zum Testen von Zuweisungsalgorithmen in der Schwarmrobotik besonders geeignet ist. Als Features bietet der Swarmulator ein modulares Design sowie Stapelverarbeitung von Simulationen. Statistische Daten werden dabei tabellarisch gespeichert und dienen als Grundlage für die abschließende Analyse der vorgestellten Lösungsansätze.

Abstract

In the research field of *Swarm Robotics*, engineers have to decide how to organize their robotic swarm. In this context, one of the most important decisions is which robot should execute which task in order to achieve a given global mission. The corresponding optimization problem is called *Task Allocation* and can be tackled by various mechanisms, e.g. auctions, which are based on research in game theory.

This thesis gives an overview of mechanisms for Task Allocation in Swarm Robotics. For this purpose, a new taxonomy is proposed that divides solution strategies into *Heteronomous Task Allocation*, *Autonomous Task Allocation* and *Hybrid Task Allocation*.

The experimental part of this thesis uses this system to develop solution methods for a concrete mission that is inspired by nature: similar to a swarm of bees, multiple robots are attached to a nest that is used to deposit food. Besides foraging, the artificial swarm needs to keep the nest's temperature close to an optimal value. Otherwise gathered food cannot be processed to energy.

This thesis focuses on one probabilistic, motivation-based approach and variants of centralized and decentralized reinforcement learning, which is a kind of machine learning that emphasizes the selection of actions under the observation of rewards. For comparison of the approaches, corresponding swarms are simulated in static and dynamic environments. In this context, static means that the aerial temperature and the food density are fixed.

For the execution of these experiments, the *Swarmulator* was developed, a simulation platform that is well suited for testing mechanisms for Task Allocation in Swarm Robotics. The Swarmulator features a modular design and batch processing. Statistical data is saved in tabular form and serves as a basis for the concluding analysis of the proposed solution methods.

Acknowledgements

I would like to thank Prof. Dr. Martin Wirsing for the opportunity to investigate a really interesting field of research and for the excellent support that I experienced at the Research Unit of Programming and Software Engineering.

Special thanks go to the advisors of my thesis: Dr. Matthias Hölzl, who proposed the topic and gave me all freedom to bring it to fruition, Annabelle Klarl, who sedulously counterchecked my thesis to find failures and to give further advice, and Christian Kroiß, who continuously motivated me to bring the thesis to a level that makes me proud of it and who counterchecked it in the final days. All of them helped me to channel my pursuit of perfection.

Finally, I would like to thank my friends and family for their widespread support in my course of studies. They formed an environment that I do not want to miss. Special thanks go to my girlfriend Katharina for her continuous encouragement.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Outline	3
2	Background	5
2.1	Definitions and Taxonomy	5
2.1.1	Global Mission and Tasks	5
2.1.2	Swarm Robotics (SR)	7
2.1.3	Multi Robot Task Allocation (MRTA)	13
2.2	The Swarm Robotics Research Field	14
2.2.1	Research Axes	15
2.2.2	Canonical testbeds in Swarm Robotics	16
2.2.3	Swarm Robotic Projects	17
2.3	Reinforcement Learning	20
2.3.1	Basics	21
2.3.2	Learning Methods	23
2.3.3	Improved Techniques	26
2.4	Summary	28
3	Mechanisms for Task Allocation in Swarm Robotics	31
3.1	Proposed Taxonomy	31
3.2	Heteronomous Task Allocation	32
3.2.1	Basic Concepts	33
3.2.2	Centralized Task Allocation	35
3.2.3	Distributed Task Allocation	38
3.2.4	Relevance for Swarm Robotics	41
3.3	Autonomous Task Allocation	42
3.3.1	Basic Concepts	42
3.3.2	Rule-based Control	44
3.3.3	Threshold-based Control	47
3.3.4	Probabilistic Control	51
3.3.5	Decentralized Reinforced Control	55
3.3.6	Relevance for Swarm Robotics	56
3.4	Hybrid Task Allocation	57
3.4.1	Interlaced Control	57
3.4.2	Side-by-Side Control	58
3.5	Summary	60

4	Swarmulator - A Simulator for Swarm Robotics	63
4.1	Simulation Platform	63
4.1.1	Architecture	63
4.1.2	Modular Design	66
4.1.3	Batch Processing	67
4.2	Applicability for Task Allocation in Swarm Robotics	68
4.2.1	World of Components	68
4.2.2	Tasks for Active Components	69
4.2.3	Tasks for Task Allocation	72
4.3	Summary	73
5	Experiments	77
5.1	Scenery	77
5.1.1	World and Components	77
5.1.2	Global Mission	79
5.1.3	Tasks	80
5.2	Approaches	81
5.2.1	Heteronomous Task Allocation	81
5.2.2	Autonomous Task Allocation	84
5.2.3	Hybrid Task Allocation	87
5.3	Simulation and Comparison	88
5.3.1	Static Scenarios	89
5.3.2	Dynamic Scenarios	96
5.3.3	Discussion	100
5.4	Summary	102
6	Conclusion and Outlook	103
	List of Figures	105
	List of Tables	107
	Content of the enclosed CD	109
	Bibliography	111

Chapter 1

Introduction

Since the middle of the 20th century, robots are used to help humans in industrial fabrication. These robotic agents are generally optimized to accomplish a local task precisely and accurately, like welding two pieces of metal. Up until today, the application range of robots has expanded very much. Mechanical agents can be specialized in a way that supports humans in nearly all areas of life.

There is a desire to increase the artificial intelligence of robots, often driven by the wish to make them more human-like. This leads to the development of increasingly complex single robots that may even look like humans. The disadvantage in constructing multi-functional single robots is obvious: a complex and capable machine is very expensive, and failure of the individual robot is an absolute catastrophe. To prevent such a single point of failure and to limit costs, multiple simpler robots could be deployed. These robots are equipped with less abilities but are able to cooperate in order to achieve a given complex mission that an individual robot is unable to fulfill on its own.

The research field of Swarm Robotics is mainly inspired by the observation of social insects. Swarms in nature show efficient behavior although their individual members are comparably incapable. This observed swarm intelligence motivates researchers to design robotic systems that use a swarm of physical robots to accomplish complex missions in cooperation. Because such missions are composed of several tasks, one of the most important challenges in Swarm Robotics is to define which robot should execute which task under which circumstances. The corresponding optimization problem is called *Task Allocation* and can be faced by various mechanisms.

1.1 Motivation

Robotic swarms offer great potential. The deployment of a multiplicity of robots does not only improve robustness by redundancy, it may also improve efficiency by parallel and coordinated execution of tasks. Ideally, the swarm's members cooperate in order to fulfill the given mission as best as possible.

For example: assume a swarm of robots that should explore the unknown environment of Mars. Of course, this mission could also be achieved by a single rover but a collective of robots is able to cover the area much faster. In order to be efficient, the robots need to coordinate their movement. Otherwise, target points are visited more often than necessary or processed in an inefficient order. This coordination has to be achieved by some kind of mechanism that solves the problem of Task Allocation.

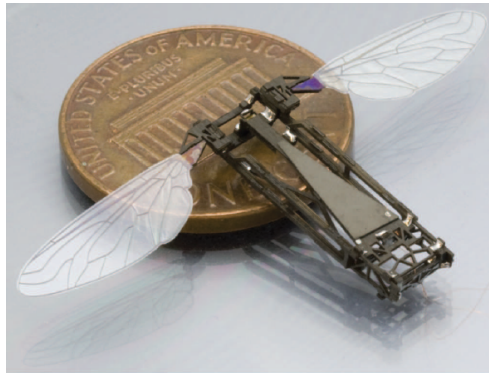


Figure 1.1: *Artificial bee. Image is taken from [SWSW12].*

Robotic swarms can be used in a wide variety of missions, including warehouse management [Dia04], mine countermeasure [Sar07], fire fighting [NGTR08, PCdDB10] and search & rescue [MFG⁺02].

Mechanical swarms could even take over tasks that would normally be achieved by natural swarms. Between 2007 and 2011, about 30 % of all honey bee colonies in the United States failed to survive until spring due to the scourge of Colony Collapse Disorder [RT12]. Because honey bees are one of the most important animals for pollination of agricultural crops, this disease is a serious threat that needs to be faced. Although robots are not the first choice to fight Colony Collapse Disorder, robotic pollination is a very interesting vision that features the integration of robotic swarms into nature. Today, it is already possible to construct light-weight flying robots that could be used in missions like exploration, search & rescue and agricultural assistance. Figure 1.1 shows such an artificial bee that is in discussion to be used for pollination.

Swarm Robotics does not only profit from results in biology and robotic research. It also has the potential to influence backwards. By experiments with robotic swarms, behavior of social insects – and natural swarms in general – can be understood better. Furthermore, the emergence of coordinated behavior through swarm intelligence can lead to new concepts in artificial intelligence and robotic design.

In order to tell the robots what to do next, designers of swarm robotic systems must face the problem of Task Allocation. Finding an appropriate mechanism for task assignment is a challenging problem because Task Allocation is influenced by many factors.

First of all, the complexity of tasks plays a major role. Tasks can be constrained by arbitrarily difficult dependencies. An example for such a dependency is a sequential order. In this case, robots have to wait for the completion of some tasks before others can be allocated. Another example is the definition of special demands that specify which number and which kinds of robots are needed to accomplish a task.

Furthermore, a robotic swarm has to adapt its behavior to the environment. If the environment is not static but dynamic, robots will have even more difficulties to adapt properly. The faster and the more abrupt an environment changes, the harder it gets to design an efficient swarm.

Last but not least, in Swarm Robotics the physical robots will likely influence each other, at least by their presence. If space is narrow, interference will almost certainly occur and constrain beneficial allocation patterns.

In order to support designers of swarm robotic systems, this thesis wants to enlighten the field of Task Allocation in Swarm Robotics. Because of the diversity of application areas, there is no silver bullet for Task Allocation. In some cases, very easy control rules may be sufficient, in others, complex negotiation between robots has to be preferred.

Ideally, the robots are able to learn on their own which task to select under which conditions. One way to achieve such adaptivity is the utilization of reinforcement learning, which updates an action selection policy dependent on the observation of rewards. Because of its high potential, this thesis sets a focus on this approach.

In order to compare reinforcement learning approaches and other mechanisms against each other, this thesis aims at the simulation of robotic swarms that follow corresponding control methods. To increase the informative value, the approaches should be tested in both static and dynamic scenarios. For this purpose, a simulator that fulfills the requirements of such experiments with robotic swarms needs to be developed.

1.2 Outline

This thesis provides an overview of mechanisms to solve Task Allocation in Swarm Robotics. It is aimed at giving designers an idea of how to organize their robotic swarm. Additionally, a concrete mission is investigated that needs a swarm to assign four different tasks to its members. In order to compare approaches to this problem, simulations in static and dynamic scenarios are carried out. For this purpose, a simulation environment, called *Swarmulator*, was developed. In detail, the thesis is structured as follows.

Chapter 2 enlightens the background needed to understand the following investigation of Task Allocation in Swarm Robotics. First, basic definitions and taxonomy are given, especially regarding the term *Swarm Robotics*. In addition, the corresponding research field is presented. Finally, an introduction to reinforcement learning is given, as it is a promising tool for both adapting and controlling mechanisms for Task Allocation.

Chapter 3 describes different mechanisms for Task Allocation in Swarm Robotics. In order to clearly arrange the overview, a taxonomy is proposed first. According to the resulting classification into Heteronomous, Autonomous and Hybrid Task Allocation, various approaches are sketched.

Chapter 4 presents the *Swarmulator*, a simulation platform that was developed in the scope of this thesis. Although the *Swarmulator* is designed for testing mechanisms for Task Allocation in Swarm Robotics, it features a modular architecture that allows to simulate arbitrary experiments that rely on stepwise execution of a virtual world.

Chapter 5 constructs a foraging scenario that forces a swarm to maintain an effective allocation of four different tasks. Following the taxonomy given in chapter 3, diverse approaches to the problem are sketched. Some of them, in particular those that follow the focus on reinforcement learning, are simulated in the *Swarmulator* and compared in static and dynamic environments.

Finally, chapter 6 briefly summarizes the results of this thesis and gives a short outlook on some current research directions that intend to bring Task Allocation to the next level.

Chapter 2

Background

Before a discussion about Task Allocation in Swarm Robotics is possible, the environment of this topic has to be illuminated.

First, important basic definitions and taxonomy are given. Especially the winged word “swarm” needs some argumentation to delimit its bounds in the context of robotic systems. Second, the research field of Swarm Robotics with its various research axes is presented, concluded by an overview of swarm robotic projects. Third, a short introduction to reinforcement learning is given, as it is a promising tool for adapting action selection and, thus, task allocation.

2.1 Definitions and Taxonomy

This section defines some basic terms needed for the discussion about Task Allocation in Swarm Robotics: First the term *task* in the context of a *global mission* is defined, second the term *Swarm Robotics* is concretized and finally a definition of *Multi Robot Task Allocation* is given, relating it to Task Allocation in Swarm Robotics.

2.1.1 Global Mission and Tasks

The first thing a designer of swarm robotic systems has to worry about is the definition of goals, which have to be accomplished by the physical robots in the system. The aggregation of all such goals is called the *global mission* of the swarm, where “global” emphasizes that the swarm *as a whole* is responsible for achieving the mission. There is only one single global mission and from this point of view it does not matter which goal is pursued by which robot. In the context of this thesis, an informal description of the global mission and its goals shall be sufficient.

Definition 1 (Global Mission). *The global mission of a swarm defines the mission the swarm as a whole has to accomplish. This mission consists of an arbitrarily large number of goals, defining the circumstances in which the swarm is successful.*

According to [Mat95], there are two types of goals: *attainment-* and *maintenance-goals*. An attainment-goal can be removed from the mission when the goal is fulfilled, whereas a maintenance-goal lasts forever and needs constant attention.

For example: “find and retrieve the black box of the crashed airplane” is an attainment-goal, whereas “prevent the airplane from crashing” is a maintenance-goal. Note that goals often (implicitly) define constraints, like only being valid if there is a crashed airplane or if the airplane has not crashed yet. By this, multiple conflicting

goals can coexist. Goals additionally may be ranked or rated, giving a clue about how well a swarm performs.

Neither the global mission nor the goals specify *how* to achieve them. To enable swarms of robots to handle a given mission, the mission has to be broken up into smaller units that each can be executed by individual robots. Gerkey and Mataric [GM01, p. 356] define a task as follows:

Definition 2 (Task). “A task is an atomic unit of computation and control that a single robot will execute.”

Note that this definition emphasizes that a task must be executable by a *single* robot. At first glance this seems to limit cooperation. At second glance a multi-robot-task – a task that needs multiple robots for being successfully executed – is just a form of *compound task*, which consists of multiple (atomic) tasks “complementing” one another. For instance: the task “hand over object x ” is a multi-robot task and can be broken into the tasks “offer x ” and “take offered x ”.

The point in defining the terms *task* and *global mission* is to emphasize the difference between the job of a single robot and the job of the swarm. This helps to distinguish two basic views in the design of swarm robotic systems:

The global view abstracts from individual robots and deals with the swarm as a whole. In multi-agent control, this view is applied onto the *collective level* mentioned in [Mat95].

The local view concentrates on individual robots and their actions. In multi-agent control, this view is applied onto the *individual agent level* mentioned in [Mat95].

At global view, the swarm has exactly one task: accomplish the global mission. The process of breaking this task into smaller units is called *task decomposition* and results in a set of more or less complex tasks individual robots can execute to achieve the goals of the global mission.

As goals can be interrelated and constrained, tasks can be interrelated and constrained, too. One of the most important dependencies between tasks are sequential and parallel dependency. Sequential dependent tasks have to be executed one after the other, whereas parallel dependent tasks can be executed in parallel, but are linked in some way (e.g. requiring synchronized execution). Tasks consisting of various dependent subtasks are called *complex tasks*. The most complex task in a swarm robotic system is the achievement of the global mission.

Complex tasks can be described by graphical models. One recent model is the *task dependency graph*, proposed by Brutschy in [Bru09], which focuses on sequential and parallel dependency. Although the reason for each dependency is not mentioned in the graph, a good overview of the tasks complexity can be gained.

Another recent representation of complex tasks is proposed by Cao et al. [CLIA10]. A tree structure is used to decompose a complex task into simpler subtasks, while allowing to add some constraints into the graph, like giving a choice of two possible decompositions or adding sequential interrelation.

2.1.2 Swarm Robotics (SR)

Swarm Robotics (SR) is a research area which can be seen as a part of the research in *Multi Robot Systems (MRS)*. One step further, MRS are a subset of *Multi Agent Systems (MAS)*.

Defining the boundary of MRS in MAS is straightforward. As the name implies, MRS focuses on *robots*: embodied agents which are placed in a physical environment. Where most work in MAS concentrates on software agents, MRS give hardware a shot and realize multi agent techniques in multi robot scenarios.

Research in MAS inspires both MRS and SR, especially in terms of communication, cooperation and decision making (containing negotiation about scarce resources). An introduction to MAS can be found in the book [Woo09].

Defining the boundary of SR in MRS is not as easy. It needs discussion of the term *swarm* which has to concretize the vague category of *multiple robots*. This is done in the following section. After that definition of SR, characteristics and advantages of the swarm approach are given, followed by a taxonomy for SR.

2.1.2.1 Definition

A simple definition of Swarm Robotics can be formulated as follows:

Definition 3 (Swarm Robotics). *Swarm Robotics (SR) is a research field studying the design of systems consisting of a swarm of embodied agents (robots).*

This definition is deliberately broad to give SR enough space in the realm of MRS. Although SR is often seen as an independent research field, the only thing setting SR apart from MRS is – in the above definition – the focus on a *swarm* of robots. Unfortunately, the *swarm* characteristic consists of multiple interlaced attributes that make its bounds fluid and a clear-cut definition difficult to impossible.

Yardsticks for “Swarm” Robotics. According to Dorigo and Şahin [DŞ04], there are four “yardsticks” for how “swarm robotic” a system is.

- (1) **Relevance for large numbers of robots:** The first attribute is the most obvious one: The more individuals a group of robots can consist of, the more swarm robotic it is. According to this, scalability should be a core issue in SR.
- (2) **Relatively few, large homogeneous groups:** Few homogeneous groups make a system more swarm robotic. Additionally each homogeneous group should consist of a large number of robots. In the best case all robots have equal capabilities, both in hardware and in software. This enables every individual to take on tasks of other individuals, which makes the system more robust when single robots drop out. Additionally the swarm gets more flexible, because it can be deployed in different environments and missions without having to think about the abilities of the individuals.
- (3) **Performance improvement by cooperation:** A swarm robotic system should be able to profit from cooperation, resulting in a higher performance than a single robot could achieve. This performance boost can be gained both by concurrent execution of simple tasks and by the execution of (complex) multi robot tasks, which require a group of robots for successful execution.

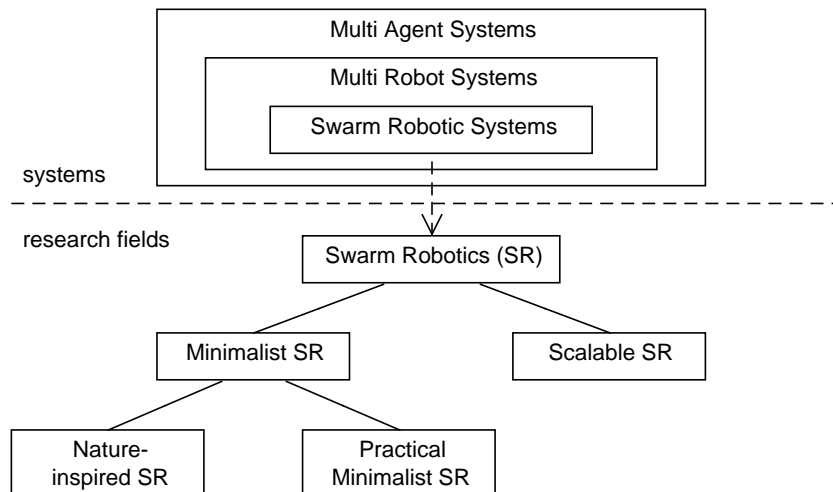


Figure 2.1: Affiliation of SR to MAS / MRS and classification of SR studies with respect to the inclusion of and the motivation for minimalism (of individuals).

- (4) **Local and limited sensing and communication:** Robotic swarms should consist of individuals with local and limited sensing and communication. This minimalism in perceptual and communicational abilities lifts the importance of cooperation because it gets harder to achieve tasks without communication.

Dorigo and Şahin emphasize that these four points cannot be seen as a checklist defining if a study is swarm robotic or not. They are just yardsticks that give an idea of the magnitude of “swarm roboticness”. According to Dorigo and Şahin, the new term Swarm Robotics is still emerging and lacks a better definition.

Discussion on Minimalism. Minimalism of individuals, yardstick (4), is an arguable attribute, which SR has mainly derived from biological self-organized systems. Although a single ant does not seem to have high perceptual, computational and communicational capabilities, a swarm of ants is able to show complex intelligent behavior, often referred to as *swarm intelligence* (like cooperation transportation of heavy objects and collective building of a highly structured nest). Many researchers in SR start at this point and try to focus on simple robots with emerging intelligent behavior.

Sharkey argues in [Sha07] that “despite its short history, swarm robotics has developed and moved away from its roots. As a result there is currently a lack of clarity about its defining features, particularly about the continued importance of and reasons for minimalism or simplicity of the individual robots in a swarm.”

To overcome this lack of clarity, Sharkey suggests the division of SR into two categories: *Scalable SR*, which allows arbitrarily complex individuals but still emphasizes the need of scalability, and *Minimalist SR*, which emphasizes minimalist individuals. Minimalist SR is further separated into *Practical Minimalist SR* and *Nature-inspired SR*, reflecting the motivation of the researcher.

Figure 2.1 gives an overview of both the affiliation of SR to MRS / MAS and the classification of different SR studies, derived from the discussion on minimalism.

Refinement of yardsticks (1) & (2). This thesis refines the four yardsticks to get a more precise picture. Especially yardstick (2) is rather vague in its formulation and, in our opinion, needs some further adjustment. How many different robots are allowed in a swarm robotic system and how many of each kind are needed?

The only reason for having few, but large groups of homogeneous individuals, is raising robustness by *redundancy*: If one individual fails, there are enough left to do its job. This prevents the system from single points of failure.

This situation can also be found in biological swarms, which are a natural source of inspiration for Swarm Robotics. If, for example, in a bee colony [MAA12] a worker bee dies, the swarm of bees will still survive due to its large number of remaining worker bees. On a closer look, there is not only the caste of workers, but two additional types of bees: drones and queens. Drones are bred as they are needed, their number is season-dependent and they are not constantly needed. In contrast, queens are the only type of bee that can mate and thus are required to get new worker bees. A swarm of bees without a queen will die off, making queens an essential part of the swarm. Surprisingly the number of queens in a swarm of bees is exactly one, turning the queen into a single point of failure.

According to yardstick (2), a swarm of bees with only one single queen is less “swarm robotic” than if there was a large number of queens. Nevertheless a swarm of bees is very stable and robust and does not need more queens. A queenless hive is very rare, because of two precaution mechanisms:

1. Supersedure process: If a queen gets old and lays less eggs, worker bees start to breed new queens in cells called “queen cups”.
2. Emergency queens: If a queen dies unexpectedly, worker larvae are set to a royal jelly diet, which turns them into queen larvae. Emergency queens are less productive than queens raised in the supersedure process, but better than complete failure of the swarm.

In both cases the newly raised queen will kill all present queens: the remaining ones in the queen cups and, if present, the old queen. Additionally to the precaution mechanisms, the queen resides inside the hive, which protects her from environmental threats, like birds. This renders the chance of swarm failure in case of queen death very low.

The queen bee example illustrates that the number of individuals in homogeneous groups is not an essential point. Overall robustness, which limits the risk of swarm failure, is much more important. Low numbers in concrete groups can be balanced by a reduced chance of individual failure and proper emergency techniques. In some cases, low numbers in a special group are even desired, like only having one single queen bee (a central beacon of indirect control).

From this point of view, we want to answer the above questions:

How many robots of each kind are needed? As much as the concrete SR system can robustly maintain.

How many different kinds of robots are allowed? Even if we have got the extreme case where all robots are heterogeneous, the swarm as a whole can be efficient, independently of the concrete set of robots available. Different combinations of robots can all have a solid behavior in the concrete context, making

the success of the swarm independent from the individual. So the answer is: There are as many different kinds of robots allowed as the concrete SR system can robustly maintain.

To conclude, yardstick (2) can be removed if an appropriate formulation of “as long as the concrete SR system can robustly maintain” is added. Because both yardstick (1) and (2) focus on the number of individuals, a combined yardstick is suggested. The concrete number of individuals is not relevant if success of the swarm does not depend on it. By this combination, three basic yardsticks remain: one about the numbers, one about cooperation and one about the abilities of individuals.

This thesis suggests the following formulation of three yardsticks how “swarm robotic” a Multi Robot System (MRS) is:

- (I) **Independence from addition and removal of individuals:** MRS that do not rely on concrete numbers of individuals, do not easily break on individual failure and are scalable to large numbers are more swarm robotic. The way of increasing independence is not prescribed. One prominent solution is maintaining redundancy in all different kinds of robots. This yardstick is an extended combination of yardstick (1) and (2).
- (II) **Profit from cooperation:** MRS that make use of cooperation to increase performance, are more swarm robotic. The way the performance boost is achieved is not prescribed. Multi-robot tasks can require cooperation, single-robot tasks can be executed concurrently. This yardstick equals yardstick (3).
- (III) **Minimalism of individuals:** MRS with minimalist robots are more swarm robotic. Minimalism covers perceptual, computational and communicational abilities of robots in the swarm. This yardstick extends yardstick (4).

Each yardstick is motivated by some benefits expected from a swarm. The next section covers those advantages.

2.1.2.2 Key advantages

The development of swarm robotic systems is driven by some desired benefits. [MP10] lists the following advantages:

- Parallelism (quicker accomplishment of tasks),
- Robustness (no single point of failure),
- Scalability (outperform centralized systems by swarm size),
- Heterogeneousness (utilization of specialists, which can even appear in homogeneous swarms as a result of imperfect hardware components or learning),
- Flexibility (adaptivity to different applications),
- Complex Tasks (ability to perform complex tasks, single robots are not able to execute) and
- Cheap Alternative (cheapness, simplicity and flexibility of simple robots).

As most of these benefits are directly backed by the three yardsticks in this thesis, a reduced set of *key advantages* can be formulated:

- Scalability and Robustness (favored by yardstick (I), proclaiming the *independence from individual count and failure*)
- Performance (favored by yardstick (II), proclaiming the *profit from cooperation*)
- Cheapness and Simplicity (favored by yardstick (III), proclaiming the *minimalism of individuals*)

As Parallelism, Heterogeneousness, Complex Tasks and Flexibility can be considered to be concrete benefits in the broad term of Performance, all benefits from [MP10] are incorporated into the given core advantages. Note that Cheapness and Simplicity are nice-to-have features that may not be needed in every swarm application due to the availability of complex hardware. This aspect highlights the optionality of yardstick (III), which demands the *minimalism of individuals*.

2.1.2.3 Taxonomy

This section gives a brief overview of taxonomy in Swarm Robotics, which helps to classify existing studies and to formulate new ones.

First a well-known taxonomy for *swarm robots* is given, followed by a more recent taxonomy of *cooperative Multi Robot Systems*.

Taxonomy of Swarm Robots. Groups of mobile robots can be designed in various ways. In order to highlight differences in design, Dudek et al. [DJMW93] define a set of properties for swarms of robots, shown in table 2.1.

Dudek’s taxonomy describes a robotic swarm in its structure (size, composition and reconfigurability) and its communicational and computational abilities. By limiting the amount of robot abilities, the *minimalism of individuals* is amplified, while a robust (most likely redundant, potentially reconfigurable) structure tries to maintain the swarms *independence from individuals*. Since the remaining characteristic (yardstick) of SR, *profit from cooperation*, builds upon structure, computation and communication and thus is more complex, cooperation is given its own taxonomy.

Taxonomy of Cooperative MRS. As swarm robotic systems are a particular form of Multi Robot Systems, much of MRS taxonomy can be applied to SR. Since non-cooperative MRS have little in common with SR, a focus on cooperative MRS is sufficient. As shown in figure 2.2, Iocchi et al. [INS01] present a cooperation-focused categorization of MRS along the following four related levels.

Cooperation: This level simply defines whether an MRS is cooperative or not. Since Iocchi et al. [INS01] – as well as SR – focus on cooperation, only cooperative MRS are categorized further by the following levels.

Knowledge: This level describes how much knowledge each robot has about the presence of other robots, in order to achieve cooperation. The corresponding attribute for an individual robot is called *Awareness*.

Coordination: This level categorizes the amount of coordination in *aware* systems. There are three levels of coordination: *Strong Coordination*, which relies on a coordination protocol (a fixed set of interaction rules), *Weak Coordination*, which does not rely on a coordination protocol, and *No Coordination*.

property	attribute value	description
swarm size	<i>ALONE</i>	one robot
	<i>PAIR</i>	two robots
	<i>LIM-GROUP</i>	small number (relative to mission or environment size)
	<i>INF-GROUP</i>	an effectively infinite number (with respect to environment and mission size)
communication range	<i>COM-NONE</i>	no communication by hardware (e.g. radio or infrared)
	<i>COM-NEAR</i>	communication with limited range
	<i>COM-INF</i>	communication with practically infinite range
communication topology	<i>TOP-BROAD</i>	broadcasting
	<i>TOP-ADD</i>	unicasting (address based)
	(...)	
communication bandwidth	<i>BAND-HIGH</i>	practically free communication
	<i>BAND-MOTION</i>	communication cost equals motion cost
	<i>BAND-LOW</i>	expensive communication, in comparison to motion
swarm reconfigurability	<i>ARR-STATIC</i>	fixed topology
	<i>ARR-COMM</i>	rearrangement based on communication relationship of members can change arbitrarily
	<i>ARR-DYN</i>	
swarm unit processing ability	<i>PROC-PDA</i>	push-down automaton equivalent processing capabilities
	<i>PROC-TME</i>	turing machine equivalent processing capabilities
	(...)	
swarm composition	<i>homogeneous</i>	all members are equal
	<i>heterogeneous</i>	swarm consists of different types of robots

Table 2.1: Properties of a robotic swarm, as defined by Dudek et al. [DJMW93].

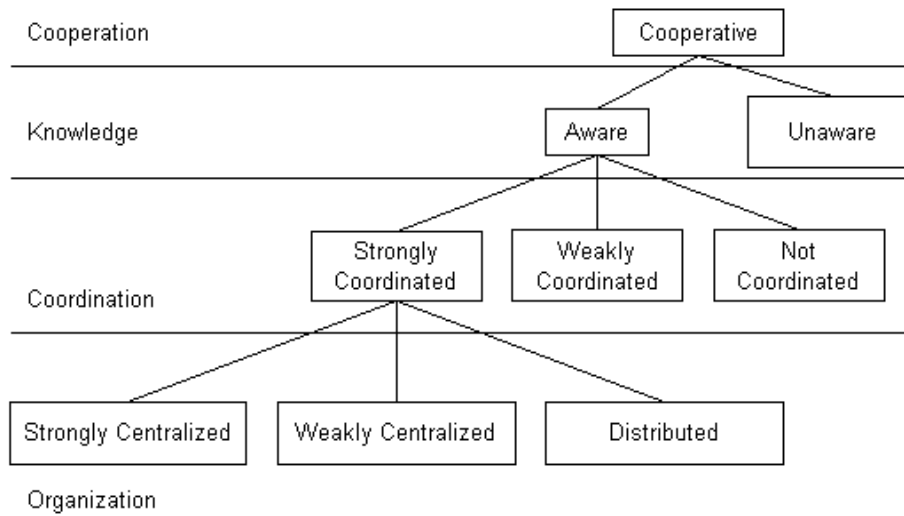


Figure 2.2: MRS taxonomy proposed by Iocchi et al., figure is taken from [INS01].

Organization: This level further specifies how *strong coordination* is achieved, regarding the organizational structure. *Centralization* relies on at least one leader giving orders. *Distribution* concentrates on autonomous decision making and declines hierarchical structures. *Centralization* can be *strong* or *weak* dependent on whether the leader role is fixed (in the course of an entire mission) or not.

Additionally Iocchi et al. [INS01] define two dimensions which are orthogonal to the various levels of cooperation:

Communication: Systems can use *direct* or *indirect* communication. *Direct Communication* is an explicit way of information exchange that makes use of some sort of hardware (e.g. radio or infrared). *Indirect Communication* makes use of stigmergy, a form of communication via the environment (e.g. by laying pheromones affecting the sensory input of other team members).

System Composition: Robotic teams can either be *homogeneous* or *heterogeneous*, dependent on whether all robots are equal (in both hardware and controlling software) or not. This dimension is entirely the same as the property *swarm composition* in Dudek’s taxonomy of swarm robots [DJMW93] above.

The taxonomy presented in this section serves as a basic vocabulary in the research of swarm robotic systems.

2.1.3 Multi Robot Task Allocation (MRTA)

Assuming that the complex task of achieving the global mission is already decomposed into atomic tasks individual robots are capable of, the most important question in a cooperative Multi Robot System (MRS) is: When should which robot execute which task?

2.1.3.1 Definition

Generally the process of mapping tasks to robots is called *Task Allocation*. In the context of MRS, Dahl et al. [DMS09] define *Multi Robot Task Allocation (MRTA)* as follows.

Definition 4 (Multi Robot Task Allocation). “*MRTA is the problem of optimizing the allocation of tasks to robots over time, with respect to some given criteria.*”

Depending on the goals in the global mission, there are various possible criteria for optimization: One might expect the minimization of costs, like energy or time. Others might emphasize the maximization of social welfare, for example by maximizing the number of achieved goals. Often many aspects have to be balanced to achieve an optimal trade-off.

This thesis investigates mechanisms for *Task Allocation in Swarm Robotics*, which is MRTA in the context of SR. Remember that SR is a subclass of MRS using a *swarm* of robots (according to the three yardsticks from the definition of SR in section 2.1.2.1).

MRTA is a problem closely related to planning and scheduling. Many well-known theoretical problems are equivalent to concrete instances of MRTA, e.g. the Optimal Assignment Problem [GM03] or the Traveling Salesman Problem [STBE09].

2.1.3.2 Taxonomy

There are different kinds of MRTA problems. To discriminate MRTA problems, Gerkey and Mataric [GM04] propose a taxonomy along the following three axis:

Robot ability: single-task (ST) vs. multi-task (MT). This attribute tells if all robots are able to perform only one single task at a time (ST) or if at least one robot is potent of executing multiple tasks concurrently (MT).

Task character: single-robot (SR) vs. multi-robot (MR). This attribute clears whether there are tasks that require multiple robots (MR) or not (SR).

Assignment: instantaneous (IA) vs. time-extended (TA). The Assignment attribute defines if there is information available that enables planning. IA means that only instantaneous task allocation is possible. Every robot only plans for the next task to do. In TA, more information about robots, tasks and / or environment (e.g. a model) is available that enables planning of future task execution extending the current one.

MRTA problems of the class ST-SR-IA are most likely the easiest to handle, as they are actually instances of the Optimal Assignment Problem (OAP) [GM04]. On the other side, multi-tasking robots (MT) may be able to perform much more tasks in the same time, multi-robot tasks (MR) may be more efficient than single-robot tasks and time-extended assignment (TA) offers the potential of better multi robot coordination through planning and scheduling.

2.2 The Swarm Robotics Research Field

This section gives a brief overview of the field of research in SR, especially regarding the context of Task Allocation.

2.2.1 Research Axes

Although SR is a young field of research, many research axes have emerged that explore the possibilities of swarms in MRS. The following propositions of research axes contribute to this thesis in two ways: On the one hand they give an entry point for finding literature in the field of Swarm Robotics. On the other hand they show that Task Allocation is one of the youngest research axes.

One of the first categorizations of research relevant for swarms is presented by Cao et al. [CFK97] who classify research in cooperative mobile robots along five axes:

- group architecture,
- resource conflicts,
- origins of cooperation and
- geometric problems.

Group architecture defines the organizational structure of the mobile robots, for example a swarm. *Resource conflicts* arise when multiple robots share the same environment. The search for *origins of cooperation* is another aspect of research, whereas learning is seen as a key component for adaptive cooperation. Finally *geometric problems* bother with path planning, formation and pattern generation.

The paper does already mention Task Allocation: it is seen as a basic mechanism for generating cooperation, but up to this point it did not get enough attention on its own to justify a new axis.

Recent reviews of SR propose new axes to better represent the amount of ongoing research in each category. Bayindir and Şahin [BŞ07] define five main axes, namely *Modeling, Behavior Design, Communication, Analytical Studies* and *Problems*. Each axis, except Analytical Studies, has got some subgroups, getting into more detail: Behavior Design, for instance, contains the axis of learning. The authors focus on an extensive study of present research, so topics without proper attention by the swarm robotic community did not get an axis. Again Task Allocation is just a side issue, without even getting a sub-group.

One of the most recent reviews of SR is written by Mohan and Ponnambalam [MP10]. They present a state of the art survey along nine research axes:

1. biological inspiration
2. communication
3. control approach
4. mapping and localization
5. object transportation and manipulation
6. reconfigurable robotics
7. motion coordination
8. learning
9. task allocation

Although most papers listed in the axis of task allocation are investigating Multi Robot Task Allocation without a focus on SR, they give a good starting point for various mechanisms useful both in MRS and SR.

2.2.2 Canonical testbeds in Swarm Robotics

Most studies in SR define their own test scenario, which is then used to build a concrete swarm robotic system capable of solving the problem. This leads to a huge amount of differently designed global missions and as a result to many different solutions which are hard to compare. Fortunately, most missions are instances of basic mission types, already known from MRS. Because of their frequent use in literature, these types can be seen as typical, canonical testbeds in MRS and SR.

According to [INS01] there are five representative domains for the study of cooperative MRS:

Foraging is about finding and retrieving some sort of prey. This is one of the most prominent testbeds in SR due to its relation to social insects and the potential use in complex missions like search & rescue, toxic waste cleanup or mine countermeasure.

Multi target observation is about finding some static or moving targets and holding them in sensory range as long as possible.

Box pushing generally is a form of transportation mission. In SR, box pushing most likely is about collectively pushing boxes that are larger or heavier than the robots. This is basically a sub-category of collective transportation, where some objects have to be transported by more than one robot in a cooperative, hopefully coordinated, manner.

Exploration and flocking are about moving through the environment, either to explore the world or to arrange and move in some kind of shape.

Soccer is about cooperation in a competitive scenario. The research in this domain is mostly driven by competitions like the RoboCup [RF12].

A more recent study [BS07] lists some typical missions in current research as sub-categories of the main axis *Problems*:

Pattern formation is about positioning in a way to form a specific shape. This kind of mission is closely related to flocking.

Aggregation is about gathering in one location.

Chain-forming is a special kind of pattern formation.

Self-assembly is about the formation of complex structures out of multiple simple units.

Coordinated movement is about keeping a global pattern while moving. This kind of mission is closely related to flocking.

Hole avoidance is a form of coordinated movement that prevents robots from stepping into holes.

Foraging is, like above, about finding and retrieving some sort of prey.

Self-deployment is about covering the environment. This is a combination of exploration and observation of space.

This list of research axes in SR shows the diversity of current research. All missions described are relevant for task allocation, because every kind of cooperation can be decomposed into atomic tasks that have to be allocated properly. Nevertheless, the most relevant mission for this thesis is foraging.



Figure 2.3: Robots of the Centibots Project in action. The image, also appearing in [OVM05], is taken from [SRI02].

2.2.3 Swarm Robotic Projects

The robotic community has started various projects for the research of swarm robotic systems. This section introduces the most relevant ones for this thesis.

2.2.3.1 The Centibots Project

The Centibots Project [KOV⁺03, SRI02] is funded by the Defense Advanced Research Projects Agency (DARPA) and aimed on missions like urban surveillance. The project is supported by SRI International’s Artificial Intelligence Center (AIC), Stanford University, the University of Washington and the robot manufacturer ActivMedia Robotics. As the name implies, Centibots is focused on the cooperation of 100 robots (“centum” Latin for “one hundred”).

The results of the Centibots Project are presented in [OVM05].

Hardware. Two types of robots are used in the Centibots Project: the ActivMedia Pioneer II (AT or DX) and the ActivMedia Amigobot. The Pioneer is equipped with a laser range finder and meant for exploration. After Pioneer surveillance, a team of Amigobots is following the Pioneers exploiting gained knowledge about the environment to efficiently search for objects of interest, track intruders and share information among themselves and a command center. Figure 2.3 shows both Pioneers and Amigobots in action.

2.2.3.2 The iRobot Swarm

The iRobot Swarm is developed by the Multi-Robot Systems Lab at Rice University (Houston, Texas, USA), in cooperation with the robot manufacturer iRobot. According to [McL04], the ultimate goal of research with the iRobot swarm is to program group behaviors at the swarm level. To achieve this goal, [McL04] proposes a library of basic and more complex (combined) behavior. An example for basic behavior would be *moveForward* or *moveStop*, whereas *followRobot* and *avoidRobot* are complex.

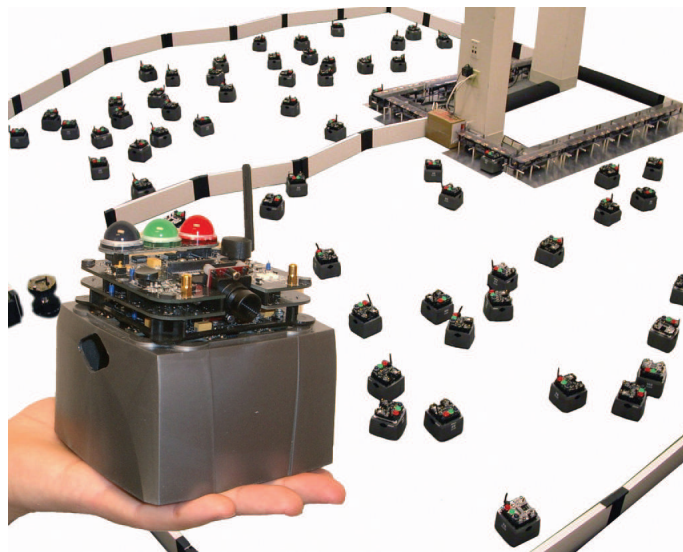


Figure 2.4: The iRobot Swarm consists of about 100 iRobot SwarmBotsTM, charging stations and navigational beacons. The image is taken from [MY05].

Hardware. The iRobot Swarm consists of about 100 iRobot SwarmBots^{TM1}. SwarmBots are small, palm-sized, cubical-shaped autonomous robots, able to recharge themselves in special docking stations (cf. figure 2.4). Besides bump sensors, light sensors and a camera, the robot has got some infra-red transceivers, enabling it to communicate over short distances.

2.2.3.3 The I-SWARM project

The I-SWARM project [SSB⁺05, K⁺08], funded by the European Commission and coordinated by the University of Karlsruhe (Germany), aims for the development of a swarm of up to 1000 micro-robots, $2 \times 2 \times 1 \text{ mm}^3$ in size. I-SWARM stands for “Intelligent Small-World Autonomous Robots for Micro-manipulation”.

Due to the very small size of single robots, which limits the sensory capabilities of individuals, collective perception becomes a key component for cooperative behavior. For example, obstacles which are much larger than the robots, can only be recognized by the accumulation of sensor data [KKC⁺05]. Another example for collective perception (with minimal communicational and computational efforts) is the use of trophallaxis-inspired strategies for spreading information about target location [SMC07]. Trophallaxis is the transmission of food from one animal to another, e.g. between ants.

Hardware. On the way to mm-sized robots, larger prototypes have been developed. The most recent coin-sized prototype, a little smaller than a cube with edge length 3 cm, is called Jasmine III. It is part of the open-source project *Swarmrobot* [SK11], which is basically supported by the University of Stuttgart (Germany). The Jasmine III is intended to be a cheap swarm robot with a “sandwich-layer” design. Various extension boards can be used to add sensor capabilities, like GPS. Figure 2.5 shows a small swarm of Jasmine robots collectively sensing an obstacle.

¹SwarmBot is a trademarks of iRobot, inc.

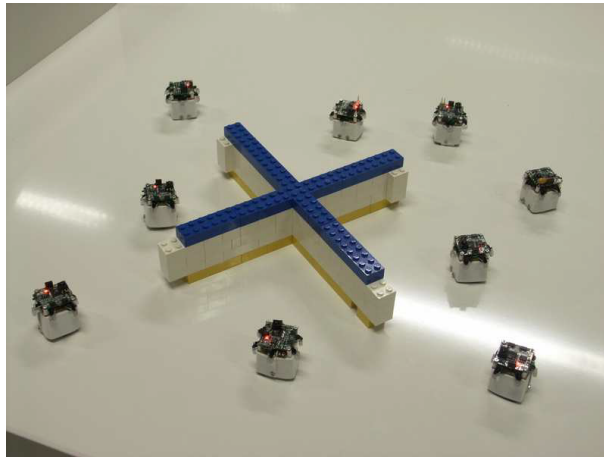


Figure 2.5: *Jasmine robots in action, collectively sensing an obstacle. The image is taken from [KKC⁺05].*



Figure 2.6: *Two swarm-bots, each consisting of six s-bots, in simulation: the left one is moving down a stair, the right one is crossing over a hole. Both images are taken from [MPG⁺04].*

2.2.3.4 The Swarm-bots Project

The Swarm-bots Project [PKG⁺02, IRI06], funded by the European Commission and coordinated by Prof. Dr. Marco Dorigo (Université Libre de Bruxelles, Belgium), concentrates on the development of the *swarm-bot*, an “artefact composed of a number of simpler, insect-like, robots” [IRI06]. Beside the construction of hardware, this includes the implementation of a simulator and the integration of swarm-intelligence-based control mechanisms.

Hardware. A *swarm-bot* consists of 30-35 homogeneous robots, called *s-bots*. The *s-bot* [MPG⁺04] is a fully autonomous robot with the ability to connect to each other. Two different kinds of grippers can be used to connect the robots either in a rigid or flexible way. This enables the self-assembled robot to move in rough terrain, featuring different heights and big holes (bigger than single robots). Each robot is equipped with an omnidirectional camera and infra-red proximity sensors and s-bots are able to communicate via LED-lights. Figure 2.6 shows a simulation of two swarm-bots (each consisting of six s-bots): the first one is moving down a stair, the second one is crossing over a hole.

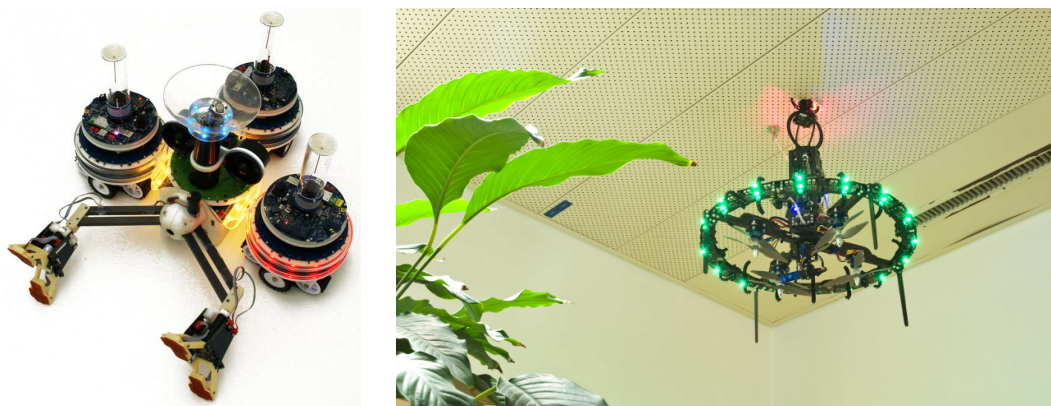


Figure 2.7: The Swarmanoid robots: the left picture shows the connection of three foot-bots and one hand-bot, the right picture shows an eye-bot attached to the ceiling. Both images are taken from [DFG⁺11].

2.2.3.5 The Swarmanoid Project

The Swarmanoid Project [DFG⁺11, IRI11] is the successor project of the Swarm-bots project, also funded by the European Commission and coordinated by Prof. Dr. Marco Dorigo. A *swarmanoid* is a swarm consisting of multiple heterogeneous robots, able to self-assemble into more complex forms, competent of achieving tasks the robots cannot perform on their own. The project includes the implementation of a simulator, called *ARGoS* [PTO⁺11, P⁺12], which is used to develop and test distributed control mechanisms.

Hardware. A *swarmanoid* consists of about 60 autonomous robots of three different types: *eye-bot*, *foot-bot* and *hand-bot*. The eye-bot is a flying robot, which can attach itself to a magnetic ceiling, survey the local environment from an aerial view and provide coordination support. The foot-bot is basically an advanced version of the s-bot, equipped with a smaller gripper for rigid connections. In contrast to the foot-bot, the hand-bot is not able to move on the ground, but it is equipped with two gripper arms and a magnetic ceiling attachment, enabling it to climb shelves. Because the hand-bot is not able to move itself, it needs to connect with foot-bots. All three types of robots are able to communicate via LED-lights. Figure 2.7 shows the Swarmanoid robots.

2.3 Reinforcement Learning

Reinforcement learning is the iterative process of improving an agent’s behavior by observing goal-relevant attributes of a real or simulated environment.

Due to its adaptive nature, reinforcement learning is well suited for acclimatizing to a previously unknown – and possibly dynamic – environment. As the composition and size of a robotic swarm most likely is both unknown and dynamic, reinforcement learning is a promising tool for controlling and modulating task allocation mechanisms in Swarm Robotics.

This section is based on the book “Reinforcement Learning: An Introduction” by Sutton and Barto [SB98] and gives a compact overview of some reinforcement learning techniques relevant for this thesis.

2.3.1 Basics

Like in task allocation, one central question in reinforcement learning is which *action* to take next. To answer this question properly, the *agent* takes into account, in which *state* the environment (all outside the agent's control) currently is. The learner and decision maker selects the best suited action by maximizing the amount of *reward* it can expect. After one or more steps of interaction with the environment, the knowledge about rewards is updated and thus the action selection changes.

2.3.1.1 Elements

A reinforcement learning system basically consists of four elements: a *policy*, a *reward function*, a *value function* and a *model* of the environment.

The policy defines the behavior of an agent. This is implemented by a probabilistic function π which defines $\pi(s, a)$, the probability of selecting action a in state s .

The reward function either defines the desirability of a state or the desirability of a state-action pair (selecting an action in a specific state). After each step of execution, the agent observes the corresponding reward. Reward values are specified a priori, should be directly related to the goals of the agent and thus have to be designed very carefully. The agent is not able to alter the reward function, but it may alter its policy to optimize the amount of reward accumulated in the long run. Achieving a greater amount of reward should mean that the agent is performing better with respect to its goals.

The value function V^π is a mapping of states to numerical values $V^\pi(s)$, defining the amount of reward the agent can expect to accumulate in the future, starting from state s and using policy π . In contrast to the reward function, which defines the immediate gratification for the transition into a state, the value function indicates the desirability of a state in the long run. Alternatively, the action-value function Q^π can be defined. $Q^\pi(s, a)$ defines the amount of reward the agent can expect when taking action a in state s and following policy π afterwards.

The model of the environment is an optional tool used to simulate changes in the environment. This enables the agent to plan a course of actions by forecasting its resulting states and rewards.

Since every function in this model works with the state of the environment, it is very important that the state information includes all sensible data relevant for decision making. In a game of chess, it is sufficient to know the current board configuration to make proper decisions. In contrast, in the memory card game the current configuration of cards does not help, the results of previous actions are the core information. If an environment's state signal contains all relevant information from the past states and actions, the signal has the *Markov property*.

Definition 5 (The Markov Property). *A state signal s_t has the Markov property if the probability for every following state s_{t+1} and every following reward r_{t+1} is solely dependent on the action a_t taken in s_t , rendering the history of preceding states and actions $s_0, a_0, \dots, s_{t-1}, a_{t-1}$ irrelevant.*

A state signal including the complete history of past actions and events always fulfills the Markov property. Fortunately, in most cases, like in the example of a game of chess, this is not necessary and a much more compact, reusable state representation is sufficient.

2.3.1.2 Markov Decision Processes

A *Markov decision process (MDP)* is a reinforcement learning problem satisfying the Markov property. An MDP with a finite number of states and actions is called *finite Markov decision process (finite MDP)*.

Finite MDPs can be described by a finite set of transition probabilities, covering all state transitions $s \xrightarrow{a} s'$. In finite MDPs, agents are able to manage both the policy π and its value function V^π in finite space, because all possible values $\pi(s, a)$ and $V^\pi(s)$ can be stored explicitly, e.g. in a table or an array.

2.3.1.3 Generalized Policy Iteration

Initially an agent does not know the optimal policy π^* and its corresponding value function V^* . To overcome this lack of knowledge, the agent starts with an arbitrary initial policy (potent of selecting every action in every state by chance) and improves it in an iterative manner. This process is called *policy iteration*.

Policy iteration consists of two steps: *policy evaluation* and *policy improvement*. Policy evaluation is the process of constructing the value function V^π which reflects the desirability of each state when following policy π . Policy improvement is the process of analyzing the value function V^π to build a greedy policy π' exploiting knowledge about expected rewards. The resulting policy π' then becomes the new π for policy evaluation, which starts the next iteration.

The creation of a policy π' from a value function V^π is not very intuitive. In fact, the value function on its own is not sufficient for the construction of π' , because of the nonavailability of information about how to reach a desired state. To build a greedy policy, the transition probabilities of the Markov decision process have to be evaluated. By the calculation of expected rewards depending on both state and action, the greedy action for a specific state, which is the one with the highest expected reward, can easily be identified. If transition probabilities are not known, the agent has to estimate them by itself. This can be done by managing an action-value function instead of a value function. The action-value function Q^π defines numerical values $Q^\pi(s, a)$ which is the expected accumulated reward for choosing action a in state s , when following policy π later on. This expected reward inherently includes transition probabilities.

In policy iteration both policy evaluation and policy improvement are completed before proceeding to the next step. As policy evaluation itself needs multiple sweeps to construct the value function (or action-value function) consistent with the current policy, the process can get very costly and time consuming. Fortunately, it is possible to partially execute policy evaluation and policy improvement, as long as both interacting processes continue to update all reachable states.

Generalized policy iteration refers to the general idea of interacting policy evaluation and policy improvement, without constraining the amount both processes update their target parameters. One process is making the (action-)value function more consistent with the policy, the other one is making the policy more greedy with respect to the (action-)value function. Finally, when the policy π is greedy to its own evaluation function V^π or Q^π , the processes stabilize and the optimal policy $\pi = \pi' = \pi^*$ has been found.

2.3.1.4 Exploration vs. Exploitation

As (generalized) policy iteration needs to update all states continuously, it is very important to constantly explore the state space by allowing the selection of every available action. In contrast, knowledge about rewards should be exploited to prefer actions leading to “good” states and evade actions leading to “bad” states. To find an optimal policy, no action may be banned, but without focusing on profitable actions, the optimal, greedy policy will never be in charge. This conflict in reinforcement learning is called *exploration vs. exploitation*.

Policy iteration has to balance exploration and exploitation: without exploration “good” states may never be found, without exploitation there will be no learning effect. The trick is to grant the action selection policy the ability, to select every action in every state at least by minimal chance. Moreover, by continuously decreasing this probability without hitting zero, the policy can be made increasingly greedy, asymptotically reaching the optimal policy.

ϵ -greedy action selection: a policy is called *ϵ -greedy* if it returns a random action with probability ϵ and the greedy action otherwise.

softmax action selection: a policy with *softmax action selection* uses the action-value estimates to weight and rank all actions. The probability of selecting an action is dependent on its weight which still grants the greedy action the highest chance of being selected, but grades the probabilities of all actions, especially of the non-greedy ones. Note that every probability must remain greater zero to enable constant exploration.

It may depend on the concrete reinforcement problem which method of action selection performs better. If there are many actions that reveal to be “bad” very fast, softmax action selection might be the better choice, because it prefers less “bad” actions for further exploration. But if those actions appearing to be “bad” at the start tend to be better in the long run, ϵ -greedy action selection will perform better, because it gives equal chances for exploration, regardless of past experience.

2.3.2 Learning Methods

This sections introduces three basic classes of methods in reinforcement learning: dynamic programming, Monte Carlo methods and temporal-difference learning.

2.3.2.1 Dynamic Programming

Dynamic programming aims to calculate the best policy by the use of a perfect model of the environment. This model is given by a complete finite Markov decision process, defining the dynamics of the environment.

Generally dynamic programming finds the optimal policy by strict policy iteration: starting with a random policy the model is used to calculate its corresponding value function or a good approximation of it. After that, the value function is analyzed to create a greedy policy, which is subject for the next iteration. This process continues until the policy does not change anymore.

As already mentioned, policy evaluation can be a costly process and, if realized iteratively, converges to V^π only in the limit. This is a result of the iterative structure of V^π which defines the estimated accumulated reward following policy π :

$$V^\pi(s) = E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\}, \quad (2.1)$$

where $E_\pi\{\dots \mid s_t = s\}$ is the expected value when starting at any time t in state s and following policy π . Each value of the value function can be calculated from the sum of the immediately following reward r_{t+1} and an – eventually discounted – amount of expected reward following in the reached state s_{t+1} , which is already known as the value $V^\pi(s_{t+1})$. The *discount factor* γ is used to control the influence of future rewards.

Since dynamic programming features a perfect model defining transition probabilities $\mathcal{P}_{ss'}^a$ and expected rewards $\mathcal{R}_{ss'}^a$, the value function can be iteratively computed by the following update rule:

$$V_{k+1}(s) = \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')], \quad (2.2)$$

where $\pi(s, a)$ is the probability of selecting action a in state s following policy π , $\mathcal{P}_{ss'}^a$ is the transition probability of transition $s \xrightarrow{a} s'$ and $\mathcal{R}_{ss'}^a$ is its expected immediate reward. This update rule uses the already known value function of step k to calculate the next iteration $k + 1$. As soon as no further changes occur the computation is finished.

Update rule 2.2 implements formula 2.1 by including the transition probabilities $\mathcal{P}_{ss'}^a$ and the expected immediate rewards $\mathcal{R}_{ss'}^a$, known from the environment's model. The inner sum $\sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')]$ weights the expected accumulated rewards for all possible following states s' , which may result from taking action a in s , by the transition probabilities $\mathcal{P}_{ss'}^a$. These averages, which are dependant on the chosen action a , are weighted by the outer sum $\sum_a \pi(s, a)$ covering all possible actions a in state s .

The computation of $V_{k+1}(s)$ for all states s is called a *sweep* through the state space. Fortunately, it is not necessary to wait for convergence, as it is possible to rearrange sweeps and even interactively combine them with policy improvement.

Value Iteration. An example for such a rearrangement is *value iteration*. In value iteration every sweep of the value function includes the selection of the most rewarding action. One sweep is computed by the following update rule:

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k(s')] \quad (2.3)$$

In contrast to update rule 2.2, the value function does not cover expected rewards for all actions that could be taken by an arbitrary policy π . Instead of weighting by the probabilities $\pi(s, a)$ for all possible actions a , only the one action maximizing expected reward is used to define the value function.

By this definition, the value function specifies the expected accumulated reward of a state, assuming that the next action taken is the greedy one, which is equal to following a greedy policy derived from the last sweep. As soon as $V_{k+1} \approx V_k$, value iteration can be aborted and V_{k+1} can be used to derive a greedy policy which is near or equal to the optimal one.

Regarding Task Allocation in Swarm Robotics, dynamic programming can be used to calculate a policy for the swarm as a whole. This can be compared to a game of chess,

where a player uses a model (chess rules and expected behavior of the opponent) to compute a greedy strategy for his swarm of chessmen. Unfortunately, even the relatively small swarm-environment of chess offers so many different states that the calculation of a perfect policy is infeasible. After all, it may be sufficient to limit computation to the next few turns. Its depth can be efficiently extended by various techniques, e.g. by pruning (omitting unlikely branches in the state tree).

Beside the computational expense, the most limiting downside of dynamic programming is the need for a model. Because of imperfect hardware and unpredictable dynamics of the swarm and the environment, it may be impossible to design an appropriate model.

2.3.2.2 Monte Carlo Methods

In contrast to dynamic programming, *Monte Carlo methods* do not need a perfect model of the environment's dynamics. Instead, they experience transition probabilities and rewards either in simulation or on-line.

The central idea of Monte Carlo methods is to execute each experimental run until termination, before analyzing the observations and adjusting a value function. In practice this implies that tasks for Monte Carlo methods need to work with *episodic* tasks, which always terminate at some time.

After each episode generated by a concrete policy π , policy evaluation takes place, generally by averaging the observed rewards for each state. The next step is the construction of a new policy π' based on the previously calculated value function V^π . As Monte Carlo methods lack a perfect model and thus transition probabilities are not known, the value function V^π is not sufficient for policy improvement. Therefore, as already proposed for generalized policy iteration in section 2.3.1.3, an action-value function Q^π is managed instead: after each episode k evaluating policy π , for each state-action pair (s, a) , all rewards following the first occurrence of (s, a) are summed up. The approximated Q -value for π after episode k , $Q_k^\pi(s, a)$, is then defined as the average of this episode's sum and all sums from previous episodes evaluating π . In the limit ($k \rightarrow \infty$), Q_k^π converges to Q^π . By iterating over all states and selecting the action which maximizes the corresponding Q -value, a greedy policy can easily be derived.

As Monte Carlo methods rely on experiencing all states and rewards, exploration is a main issue. To ensure that exploration continuously takes place in all episodes, a *soft* policy should be used, e.g. ϵ -greedy.

Regarding Task Allocation in Swarm Robotics, Monte Carlo methods can be used to improve the robot's efficiency in episodic tasks. The main disadvantage of this approach is the necessity to wait for a terminal state before adaptation can take place. The Q -values and the derived greedy strategy of an agent is not updated until the end of each episode. This delays adaptation unnecessarily. Nevertheless, Monte Carlo methods may be useful for prototyping policies in simulation, which can be refined in live environments later on.

2.3.2.3 Temporal-Difference Learning

Another solution method for reinforcement learning solely relying on experience is *temporal-difference (TD) learning*. In contrast to Monte Carlo methods, TD learning does not wait for a terminal state before policy evaluation takes place. Instead, TD learns on-line during the experience of a – possibly infinite – episode.

Like Monte Carlo methods, TD learning abandons the use of a perfect model of the environment's dynamics. As a result, TD learning generally does not manage a value function V , but an action-value function Q that can be used to derive a policy from.

After each transition $s_t \xrightarrow{a_t} s_{t+1}$, the resulting rewards r_{t+1} is observed and the Q -value for action a_t in state s_t is updated as follows:

$$\begin{aligned} Q(s_t, a_t) &= Q(s_t, a_t) + \alpha * \Delta \\ &= Q(s_t, a_t) + \alpha * [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)], \end{aligned} \quad (2.4)$$

where γ is the discount rate, Δ is the change in reward expectation and α is a constant factor that defines, how much of Δ is used to adapt the Q -value. $\alpha = 1$ would mean that $Q(s_t, a_t)$ is completely driven to the new reward expectation $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$. Smaller values of α make Q more robust against the influence of the observed reward r_{t+1} . Thus, α steers the speed of Q adapting to new situations, rendering it an important control parameter.

Sarsa. Update rule 2.4 can be extended to a complete TD control algorithm: in each step $t + 1$, s_t and a_t are known from the last step t , r_{t+1} and s_{t+1} are observed after taking action a_t in s_t , and a_{t+1} is chosen from the available actions in s_{t+1} using a policy derived from Q (e.g. ϵ -greedy). Because this algorithm is based on the quintuple of events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, it is called *Sarsa*.

Regarding Task Allocation in Swarm Robotics, Sarsa – and TD learning in general – is a promising algorithm. If rewards are defined in a way that guides robots to the desired behavior, Sarsa seems to be useful for the selection of tasks.

2.3.3 Improved Techniques

In reinforcement learning many techniques can improve the quality of the learned policy. This section covers some of them: off-policy control, eligibility traces and function approximation.

2.3.3.1 Off-Policy control

Up to this point, we have concentrated on on-policy control. The policy used for action selection was the same one that should be optimized. On-policy control can only find a policy that still explores.

Off-policy control uses a *behavior* policy for action selection, while optimizing an *estimation* policy. By this, the estimation policy can be greedy, without abandoning exploration.

One favorite example for off-policy control is *Q-learning*. *Q-learning* is a temporal-difference control algorithm very similar to Sarsa. In its simplest form, *one-step Q-learning*, it uses the following update rule:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha * [r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)] \quad (2.5)$$

The only difference to Sarsa is the addition of the max-operator which conforms the use of a greedy policy derived from Q for the selection of a_{t+1} . By this, the Q -values approximate Q^{π^*} , the action-value function corresponding to the optimal policy π^* . Nevertheless, a soft policy (e.g. ϵ -greedy) is used for action selection, which enables constant exploration.

There is one disadvantage in off-policy control: the on-line performance may be worse than when using on-policy control. This is a result of Q -values being more advanced than the exploring behavior policy. For example: if the optimal path in a world is risky and false steps are punished, e.g. when walking along a cliff, exploration based on the optimal path is deadly. Sarsa would learn Q -values that include the fuzzy exploring walk, while Q -learning ignores the danger for an explorer and lays down breadcrumbs on the dangerous edge of the cliff.

2.3.3.2 Eligibility Traces

When an agent using one-step temporal-difference learning observes a reward r_{t+1} , only the last state-action pair (s_t, a_t) is benefited (cf. update rule 2.4). In the limit, of course, this reward will be propagated through all Q -values by the discount rate γ , but the immediate update is entirely focused on $Q(s_t, a_t)$. Thus, the learning of delayed rewards, which are related to actions some steps ago, is very slow. To extend the limited last-step view to a diminishing trail of recent state-action pairs, eligibility traces are used.

An eligibility value $e(s, a)$ defines the qualification of $Q(s, a)$ for being influenced by an immediate reward. It is raised when the state-action pair is observed and it is continuously decreased later on. This builds an eligibility trace, quantifying the influence of rewards on recent state-actions pairs.

Sarsa(λ). The one-step TD control method Sarsa from section 2.3.2.3 can be extended by eligibility traces. Because the diminishing rate of a trace is controlled by the parameter λ , the resulting algorithm is called Sarsa(λ). The following modifications have to be made.

First the eligibility values for all state-action pairs have to be initialized by zero. In each step, before the Q -values are updated, the eligibility value for the state-action pair (s_t, a_t) is increased by one (or alternatively set to one).

Second the update rule 2.4 has to be extended to modify all Q -values with respect to their eligibility value:

$$Q_{t+1}(s, a) = Q_t(s, a) + e_t(s, a) * \alpha * \Delta_t, \quad \text{for all } s, a, \quad (2.6)$$

where Δ_t is the familiar change in expected reward $\Delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$. The indexes t and $t+1$ of Q highlight the iterative process of the action-value function: every time step changes Q as a whole, not only single Q -values.

Third the eligibility values for all state-action pairs have to be decreased after each update of the Q -values:

$$e_{t+1}(s, a) = \gamma * \lambda * e_t(s, a), \quad \text{for all } s, a,$$

where γ is the familiar discount factor and λ is a constant parameter controlling the stability of the discounted eligibility values.

By the use of eligibility traces the gap between one-step temporal-difference learning and the use of full episodes in Monte Carlo methods can be closed.

2.3.3.3 Function Approximation

All methods for reinforcement learning presented in this thesis rely on a finite state and action space, which allows to store complete policies, value functions and eligibility traces in finite space.

Unfortunately, the number of states and actions may be infinite. This section exemplarily concentrates on states. A state consisting of a continuous variable like temperature allows to define an infinite number of states. Managing a table with values for each temperature state is neither possible nor meaningful. If an agent knows the desirability of temperature x , it still doesn't know the desirability of immediate neighbors, because every little change in temperature $x + \epsilon$ will result in a freshly new state with unknown desirability.

To overcome this problem, *function approximation* can be used. This changes the interpretation of a state s when used as a parameter, for example in a value function V : instead of using s to look up the return value $V(s)$ in a table, all available sample points in V are used to approximate $V(s)$.

Function approximation enables the agent to generalize from sample values. Even if an agent gets into a state it has never seen before, it can profit from its knowledge and take an appropriate action based on experience with similar states.

2.4 Summary

This chapter has described background knowledge needed to understand the following investigation of Task Allocation in Swarm Robotics. First, some basic definitions and taxonomy were presented. Second, the research field of Swarm Robotics was discussed. Third, an introduction to reinforcement learning was given.

For a start, the swarm's global mission, which consists of multiple goals, has been defined. To achieve these goals and to fulfill the global mission, robots in the swarm have to execute tasks. In this context, two essential points of view have been introduced: the global view, which focuses on the swarm as a whole, and the local view, which concentrates on single individuals.

After that, the term "swarm" has been concretized by the definition of three basic yardsticks telling how "swarm robotic" a system is. This description helped to classify Task Allocation in Swarm Robotics, which is a special case of the Multi Robot Task Allocation problem focusing on the use of a swarm of robots.

The following presentation of the Swarm Robotics research field showed the context of this thesis. Recently, Task Allocation was credited to be a research axis on its own. Although Swarm Robotics is still a young research field, some canonical testbeds can be identified that appear in a high number of papers. In the experimental part (chapter 5), this thesis will focus on the foraging domain, taking inspiration from the Swarmanoid project, which features very well designed mobile robots and was presented in this chapter beside other well-known swarm robotic projects.

Finally, reinforcement learning was demonstrated to be a powerful tool for adapting to a previously unknown environment. This seems to be very valuable in the context of Swarm Robotics because it can improve Task Allocation by rewarding a robot for selecting the most efficient tasks.

From global view, dynamic programming could be used to compute an optimal solution for the swarm as a whole. Unfortunately, dynamic programming needs a perfect model of the environment which can only rarely be found in Swarm Robotics.

From local view, both Monte Carlo methods and temporal-difference learning are promising. Because temporal-difference learning works better in on-line scenarios and may be used to adapt behavior in dynamic environments, this technique will be of greater interest for this thesis. Some improved techniques, like the use of eligibility traces and function approximation, may even boost the utility of temporal-difference learning.

The next chapter is based upon the background presented in this chapter. It targets on an overview of existing mechanisms for Task Allocation in Swarm Robotics and on the inclusion of new mechanisms using reinforcement learning techniques.

Chapter 3

Mechanisms for Task Allocation in Swarm Robotics

After defining Swarm Robotics and Multi Robot Task Allocation (MRTA) in chapter 2, we can continue our investigation of Task Allocation in Swarm Robotics. This chapter describes different mechanisms that can be used to solve the problem of MRTA in the context of Swarm Robotics.

Because Task Allocation in Swarm Robotics is a subclass of MRTA, existing solutions for this problem can also be used in Swarm Robotics, as long as they can be successfully transferred into a swarm robotic context. Baghaei and Agah [BA02], for example, present methodologies for MRTA that may be relevant for swarms. Unfortunately, their technical report just enumerates different solutions in literature and lacks a decent categorization of them. This makes it hard for designers to choose a solution that fits their requirements. Creators of swarm robotic systems should at least know if a mechanism focuses on the design of individual robots or on the coordination of the swarm as a whole.

To overcome this lack of classification and to additionally relate mechanisms to Swarm Robotics, the next section presents a new taxonomy, providing three fundamental design-classes for Task Allocation in the context of Swarm Robotics: *Heteronomous*, *Autonomous* and *Hybrid Task Allocation*. Following these categories, the subsequent sections present various mechanisms that can be found in the literature. Additionally, this thesis suggests to use reinforcement learning for Task Allocation and proposes a motivation-based approach.

3.1 Proposed Taxonomy

Finding a decent categorization of mechanisms for Task Allocation in Swarm Robotics is not a trivial problem. At first sight, mechanisms could be classified by the type of swarm needed for execution. Unfortunately, the taxonomy of Swarm Robotics, presented in section 2.1.2.3, focuses on robotic abilities, like awareness and communication range, but does not give a clue about how a mechanism utilizes the swarm.

From an *architectural perspective*, system designers discriminate between centralized and decentralized approaches. Since a swarm of robots always has a strong decentralized component, the terms centralized and decentralized are not very effective in the context of Swarm Robotics.

From a *control architectural perspective*, Lueth and Laengle [LL94] distinguish between *centralized*, *distributed* and *decentralized* control architectures in MRS:

Centralized: A central component makes decisions and transmits them to executing components.

Distributed: The executing components follow a negotiation process to make a collective decision.

Decentralized: Every executing component decides on its own.

Unfortunately, the terms *centralized*, *distributed* and *decentralized* are not self-explanatory in the context of Task Allocation but ambiguous. For instance: in global view, which is examining the swarm as a whole, the term centralized could describe the fact that one single system component decides for all robots. In local view, which is focusing on an individual robot, the same term could define that the robot decides on its own and does not distribute decision making about itself to other system components.

From the perspective of decision making, this thesis proposes three categories for Task Allocation in Swarm Robotics: *Heteronomous*, *Autonomous* and *Hybrid Task Allocation*.

Heteronomous Task Allocation: An executing agent in the swarm cannot decide on its own which tasks to execute. From local or robot view, the allocation of tasks is heteronomous.

Autonomous Task Allocation: Every robot in the swarm is able to decide on its own which tasks to execute. From local or robot view, the allocation of tasks is self-determined, making the robot autonomous.

Hybrid Task Allocation: Any combination of Autonomous and Heteronomous Task Allocation is considered to be hybrid.

As shown in figure 3.1, the main categories of this taxonomy are derived from the design-classes of MRS proposed by Lueth and Laengle [LL94]. The additional category of Hybrid Task Allocation opens a new dimension, enabling the design of complex systems using a combination of mechanisms from both Autonomous and Heteronomous Task Allocation.

Remember that Multi Robot Task Allocation (MRTA) is an optimization problem (cf. section 2.1.3), whereas Autonomous, Heteronomous and Hybrid Task Allocation refer to principal design-classes for the solution of MRTA in Swarm Robotics.

In the following, Heteronomous Task Allocation is discussed first, because it is the best explored category with respect to literature available. Second, Autonomous Task Allocation, which basically is inspired by nature and subject of ongoing research, is examined. Finally, Hybrid Task Allocation is considered, which offers great potential for future research.

3.2 Heteronomous Task Allocation

Firstly, this section describes the basic concept of Heteronomous Task Allocation. After that, it focuses on corresponding mechanisms and discusses both centralized and distributed solutions.

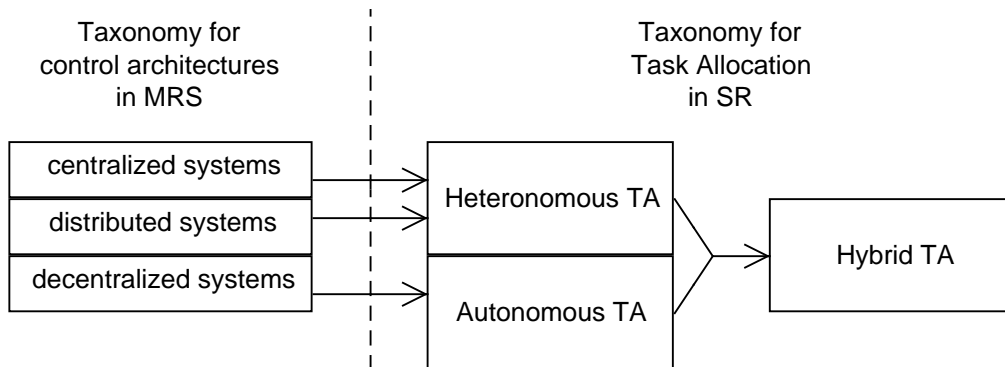


Figure 3.1: Proposed taxonomy for Task Allocation in Swarm Robotics, showing its relation to design-classes in MRS.

3.2.1 Basic Concepts

In Heteronomous Task Allocation, robots generally execute tasks that another entity of the system allocated to them. An individual member of the swarm is not allowed to decide on its own what to do next.

The following summarizes the keystones of this type of Task Allocation: *top-down coordination*, *essential communication* and the conflict of *quality of solution vs. computational complexity*.

3.2.1.1 Top-Down Coordination

All solution methods in Heteronomous Task Allocation share a hierarchical structure, which lasts for the allocation of at least one task. Coordination of robots is achieved by using this hierarchy, defining two roles:

- leaders and
- workers.

Leaders are decision makers, who try to get a complete picture of the situation, compute a hopefully optimal solution and allocate tasks to workers. Workers accept the allocations and try to achieve the given tasks.

Designers creating a swarm of robots using Heteronomous Task Allocation basically work in global view. Instead of focusing on single individuals, the swarm of potential workers is observed as a whole and tasks are allocated in a top-down manner to the best suited robots.

3.2.1.2 Essential Communication

As a leader has to communicate its decision to the workers, at least the leaders must have the ability to send messages. This communication, of course, does not have to be direct, it may be indirect, too. In the taxonomy of Iocchi et al. [INS01], direct

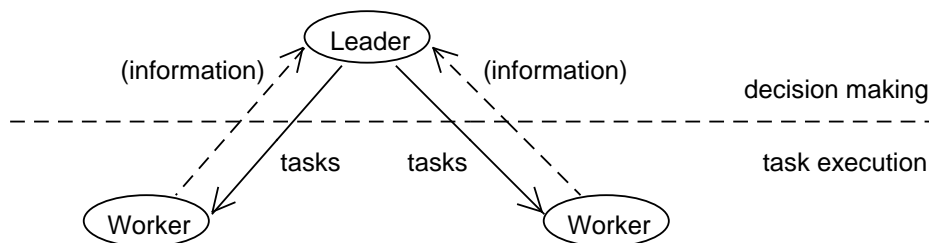


Figure 3.2: *Communication in Heteronomous Task Allocation: leaders are responsible for decision making and provide tasks, whereas workers are responsible for executing allocated tasks and optionally provide information.*

communication refers to an explicit way of information exchange that makes use of some sort of hardware (e.g. radio or infrared). Indirect communications refers to implicit information transmission via the environment. For example: laying done a red stone representing the task “go to the red zone” is indirect communication.

If the workers are also able to send messages, they can transmit relevant information to the leader, e.g. a score defining the robot’s fitness for a specific task.

Figure 3.2 shows the basic communication ways in Heteronomous Task Allocation: leaders provide the allocation of tasks, whereas workers (optionally) provide information. Note that both leaders and workers are just roles. Thus, some of the depicted roles may belong to the same physical robot.

3.2.1.3 Quality of Solution vs. Computational Complexity

Assuming that a leading robot possesses all relevant information about tasks, robots and the environment’s dynamics, it can compute an optimal solution, e.g. via dynamic programming (cf. section 2.3.2.1). As already mentioned in the context of reinforcement learning, the computational effort may be infeasible, especially if the space of allocation patterns (states) and intermediate (re)allocations (actions) is large.

Fortunately, the simplest form of MRTA, ST-SR-IA, which works with single-task robots, single-robot tasks and instantaneous assignment, can be seen as an unweighted scheduling problem, which is polynomial solvable [GM03]. Unfortunately, many variants of MRTA are \mathcal{NP} -hard, especially when incorporating time-extended assignment. For example: the Multiple Traveling Robot Problem [STBE09], which appears in multi-robot exploration, is a variant of the well-known \mathcal{NP} -hard Multiple Traveling Salesman Problem. As a trade-off between quality and complexity, such problems are often approached by the use of heuristics, like estimating utility of each robot for each task based on their distance [SB06, MDJ07, Das09].

In literature, much work that falls into Heteronomous Task Allocation is classified as “explicit MRTA” [GM03], as the allocation problem of multiple robots is considered explicitly. Top-down coordination and communication allow Heteronomous Task Allocation to profit from centralized decision making, enabling leaders to calculate optimal solutions. On the downside, the solution quality comes with high computational effort, which is often faced by heuristics.

The following sections discuss centralized and distributed mechanisms for Heteronomous Task Allocation.

3.2.2 Centralized Task Allocation

This category contains strategies for Task Allocation that rely on a single central decision maker. In general, strong centralization is used, which defines systems where the leader role is fixed for an entire mission. Nevertheless, it is possible to weaken centralization and allow re-assignment of the leader role in case of failure, e.g. by election. For the sake of simplicity, this thesis concentrates on strong centralization. Further information about elections can be found in [Woo09, chapter 12].

As the hierarchy in Centralized Task Allocation is absolutely clear, the following sub-categories concentrate on the accumulation of knowledge by the leader and on the decision making process.

3.2.2.1 Omniscient Control

Omniscient control assumes that the decision making entity has all relevant information on its own and does not need the workers for knowledge accumulation. This enables the leader to compute an optimal allocation of tasks at every point of time. In chess, for instance, a player has omniscient control, although he is only able to estimate the opponent's strategy.

In Swarm Robotics, omniscient control is almost absent, because embodied agents are not expected to be omniscient. Robotic swarms generally consist of physical robots that have limited sensory capabilities. Usually, no single robot is equipped with sensors that provide an overview of the whole swarm.

Nevertheless, there may be systems, where a swarm is controlled by a single omniscient entity. For example: imagine a swarm robotic warehouse consisting of robotic workers and a control center, which has access to an all-embracing sensory system covering all robots and to a communication systems for the allocation of tasks. Here, the executing agents are not involved in decision making and fully reliant on a single entity. Nevertheless, this single point of failure makes the system less "swarm robotic" (cf. the yardsticks in section 2.1.2).

3.2.2.2 Blackboard Control

Blackboard control assumes that workers contribute to decision making by the addition of relevant information to a global platform, called *blackboard*. The blackboard serves as a knowledge base for the leader, who allocates tasks either directly by communication with concrete workers or indirectly by publishing allocations on the blackboard. In the latter case, workers need to continuously read public blackboard information.

In general, blackboard systems [Cor91] are a powerful tool for the distributed solving of problems, like planning and scheduling. Blackboard systems do not limit the processing of information to a central decision maker.

In the context of Centralized Task Allocation, the blackboard features the most accurate picture of the environment's state currently available. Additionally it may contain proposals from the workers, which helps a central entity to solve Task Allocation. Both blackboard and decision maker can be seen as one entity providing "blackboard control", which is shown in figure 3.3.

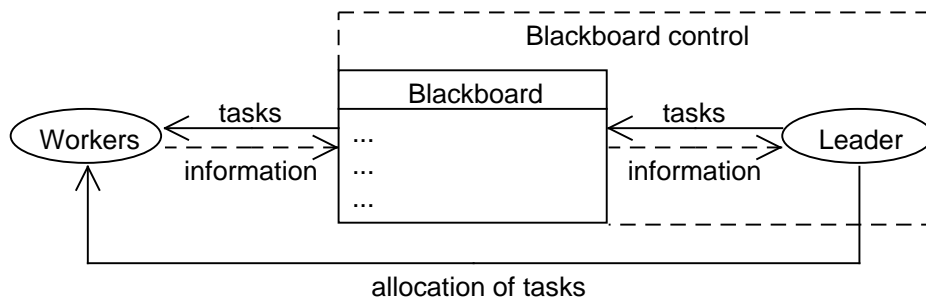


Figure 3.3: *Blackboard control: Workers add information to the blackboard, which is processed by the leader and results in the allocation of tasks. Additionally the workers may receive information from the blackboard to get a clue which information is needed.*

An example for the use of blackboard control is the GOFER project [CCL⁺90], which aims at the control of several dozen robots in an indoor environment. In GOFER, each worker communicates with a central task planning and scheduling system (CTPS). The CTPS knows the goals of the mission, generates a plan structure to achieve these goals and supplies all available robots with pending goals and plan structures. In reaction to this announcement, workers make proposals which are iteratively reviewed by the decision maker. The CTPS keeps track of an optimal solution and finally, when no more proposals appear, allocates the tasks accordingly.

If accurate information about goals, tasks, robots and the environment is available, Centralized Task Allocation can basically focus on planning and scheduling. In real world applications, environments tend to be dynamic which results in unexpected task execution times. The offset between expected and actual execution times leads to sub-optimal performance. By gathering information about the progress of each task the decision maker is able to proactively re-plan in order to update previous allocations that became inefficient. Further information about proactive re-planning for multi-robot teams can be found in [Sel09].

3.2.2.3 Centralized Reinforced Control

Up to this point, all presented centralized control solutions rely on the existence of an algorithm for mapping a rich state signal to an optimal allocation of tasks. In many scenarios, such an algorithm cannot be defined a priori because the leader is not able to sense all information needed for an optimal allocation.

For example: in a foraging scenario, the global mission is to gather food from the environment as fast as possible, but without unnecessarily wasting energy by moving around. Assume that the leader robot stays in the delivery zone, where it is able to observe the amount of food collected by the swarm, but unable to sense food in the environment. In other words: the leader does not know how many robots can efficiently forage but it is able to learn from the observed collection rate. In this case, reinforcement learning can serve as an adaptation mechanisms for the number of allocated foragers.

Adaptation. In many cases, reinforcement learning is used to adjust parameters of an a priori imperfect allocation strategy. Learning in general is a powerful tool for adapting decision making to the actual environment, including group dynamics like physical robot interference, which is often hard to predict in advance.

Strens and Windelinckx [SW05] investigate the inclusion of reinforcement learning into a scheduling system. In their work, a central planner has to allocate robots to tasks that arise continuously at arbitrary locations. Their hybrid planning/learning approach extends planning quality by the incorporation of robot positioning. Plans that cluster robots are learnt to be less efficient because the robots give up coverage of the environment.

Paquet et al. [PCdDB10] use reinforcement learning in the context of a RoboCup-Rescue mission. In their work, *FireBrigade* agents are commanded by a central *FireStation* agent to efficiently extinguish fire at different locations. The *FireStation* initially does not know how many robots are needed at each fire area. By observing which kind of building, e.g. a wooden one, demands more attention than others, the central decision maker is able to adapt its allocation strategy, allocating more robots to difficult fire sites.

Control. This thesis goes one step further and suggests to use reinforcement learning directly for Task Allocation. By explicitly learning which tasks to instruct under which environmental circumstances, the central entity is able to optimize the process of allocating tasks on its own. As long as the number of actions (i.e. ordering tasks) and the number of experienced states (i.e. some kind of processed sensory data) is sufficiently low, the leader is able to explore its possibilities in an acceptable amount of time. This approach seems to be very promising, especially for division of labor, which aims at managing an efficient distribution of a small set of tasks to a high number of workers.

The idea of learning Task Allocation is not new. Research in Multi Agent Systems already investigated the use of reinforcement learning as a mechanisms for task assignment. For example: Abdallah and Lesser [AL06] consider Task Allocation as a game where mediator agents have to allocate tasks to different servers. Although their learning algorithm is aimed at the coordination of multiple leaders, this example demonstrates that learning from observed rewards is a powerful mechanism to control a leaders task allocation strategy.

This thesis proposes to use Sarsa(λ), which is a temporal difference learning control algorithm using eligibility traces (cf. section 2.3). By this, the decision maker is able to adapt its allocation strategy on-line. The most difficult part in using this type of Task Allocation mechanism is the definition of proper rewards, which have to ensure that the swarm is driven to the accomplishment of the mission.

Omniscient control, blackboard control and centralized reinforced control are example strategies for solving MRTA from the perspective of a central leader. Because centralized solutions are relatively uninteresting for Swarm Robotics due to the creation of a single point of failure, this overview of different approaches shall be sufficient, although there may exist more strategies for specific scenarios.

Blackboard control already opened a path to distributed problem solving. The idea of all robots contributing to decision making can be extended by spreading the power of actual decision making to more than one robot. This approach is considered in Distributed Task Allocation.

3.2.3 Distributed Task Allocation

In order to avoid a single point of failure, Distributed Task Allocation allows the swarm of robots to negotiate about the allocation of tasks. Generally, this is done by dynamically granting the leader role when needed. From the perspective of the executing agent, the allocation of tasks remains heteronomous, because the worker has to comply with the leader's decision.

3.2.3.1 Market-based Approaches

The most prominent approach for Distributed Task Allocation is the use of a market-based system, where tasks are considered to be goods and robots bargain for who will get each task by bidding with virtual money.

The following paragraphs present the auction mechanism and some well-known frameworks using a market-based approach. A comprehensive survey and analysis of market-based multi-robot coordination can be found in [KZDS05].

Auctions. In market-based systems, the most important mechanism for negotiation is the *auction*. Every auction consists of three basic phases:

1. announcement,
2. bidding and
3. assignment.

In the announcement phase, a temporal leader, called the *auctioneer*, offers one or more tasks for negotiation. After that, workers bid on the announced tasks. Finally, when all bids are gathered, the auctioneer assigns the tasks by comparing the received bids and by maximizing the total profit.

Depending on the number of items announced, the following different kinds of auctions can be identified:

Single-item auctions offer one task at a time.

Multi-item auctions offer a set of tasks that is auctioned as a whole.

Combinatorial auctions offer a set of tasks that may be split to multiple bidders.

Bids can be made for any subset of the offer.

In general, single-item auctions are preferred, because multi-item auctions produce inferior solutions and combinatorial auctions are often intractable with respect to computational and communicational requirements. [KZDS05]

For example: The Centibots system [OVM05] (cf. section 2.2.3.1) organizes its robots in teams featuring a leader, called dispatcher. Ready team members get a job list from their dispatcher, who manages an auction about all available tasks. In response to the job list, each robot makes a bid representing its preferences with respect to its location and battery level. The auctioneer can then allocate the tasks in preference order.

The contract net protocol (CNP) [Smi80] is a high-level protocol for negotiation among agents, providing distributed control of cooperative task execution. Basically it describes an auction as presented above and thus serves as a basis for most concrete implementations of market-based systems in literature.

MURDOCH. Gerkey and Mataric [GM01] present a publish / subscribe system using a simple auction. In MURDOCH, all robots are listening for tasks. As soon as a robot finds a task it starts a one-round single-item auction by announcing the task. In response, every individual capable of performing the task calculates a score defining its fitness for the task and sends an appropriate broadcast message. As MURDOCH relies on a sufficient communication system, the usually following announcement message can be skipped: the participating robots already know who is best suited for the task. The winner simply starts to do the job without waiting for an acknowledgement.

MURDOCH can also be described as an “instantaneous blackboard system”. By broadcasting, tasks are published on a virtual blackboard, allowing all relevant robots to subscribe for task execution. As soon as no more bids can be expected, the robots check their accumulated blackboard information and the winner executes the task. Afterwards, all information about the task can be deleted, because the winner is expected to successfully accomplish the task.

TraderBots. Dias [Dia04] describes another market-based approach. In TraderBots, the robots bid on tasks on the basis of cost, like the expected time to accomplish a task. The bidder that is able to offer the lowest cost is contracted to fulfill the task. In contrast to MURDOCH, robots are allowed to trade assigned tasks. By frequent re-contracting, imperfect plans can be resolved and emerge to an optimal solution. This is especially interesting in dynamic environments, where it is likely that contracts cannot be hold, because real costs turn out to be higher than the estimated costs.

In TraderBots, every robot tries to maximize its personal profit. The accomplishment of tasks costs some virtual money, but includes a revenue. By continuously competing for tasks and possibly cooperating to mutually achieve higher profit, binding contracts are made and Distributed Task Allocation emerges from the virtual economy. This strategy is known as the *free market approach*, which originally was introduced by Stentz and Dias [SD99].

DEMiR-CF. Sariel et al. [SBS06, Sar07, STBE11] propose a generic framework, called DEMiR-CF, which is designed for distributed multi-robot cooperation and basically uses a single-item auction. DEMiR-CF, which is a shortcut for “**D**istributed and **E**fficient **M**ulti **R**obot - **C**ooperation **F**ramework”, is able to reallocate tasks and reorganize team members if necessary. Additionally, precaution routines are used to respond to various failure cases, like robot breakdown. Many conflicts resulting from missing knowledge are resolved by observing the standard auctioning process. If, for instance, a robot does not know that a task is already achieved, it will announce an auction resulting in other robots informing the first one about its fault.

The basic case study of DEMiR-CF is a naval mine countermeasures mission. There are two types of robots in the system: UUVs, which are able to detect mines in a radius of 30 feet but unable to identify them, and crawlers, who can both detect and identify but only in a radius of 20 feet. In consequence, UUVs are used for exploration and crawlers are used for identification. As a result, tasks of crawlers are created and announced by explorers.

Market-based approaches for Task Allocation are still subject of ongoing research. For example: Dasgupta [Das09] recently proposed the introduction of dynamic bids, which consist of lower and upper bound prizes robots are willing to pay. By not paying the full prize in advance, like not promising to achieve a task in the minimal time possi-

ble, the robot may be able to re-contract without breaking its made agreements. Even more recently, Cao et al. [CLIA10] considered complex tasks that can be organized in a tree-based formalism containing task-ordering. Because bidding for all possible portions of the tree may cost too much bandwidth, they propose to limit the number of bids by comparing approximated lower bounds of the tasks with reserve prizes and skipping irrelevant bids. De Weerd and van der Krogt [dWvdK06] showed that auctions can result in arbitrarily bad solutions in the worst case when recontracting and multilateral deals are not allowed. Although, in this case, they proofed that egoistic decisions or imprecise information about task costs of other agents can result in poor allocations in theory, this circumstances seem to be very rare in practice.

3.2.3.2 Other Distributed Approaches

Beside the outstanding market-based approach, literature features many more interesting methodologies for Distributed Task Allocation.

Virtual Blackboard. As already mentioned, blackboard systems [Cor91] are a powerful tool for distributed problem solving. Blackboard control, which is a centralized approach presented in section 3.2.2.2, needs one central leader to manage a global blackboard. By giving every robot a private blackboard and using broadcast messages to influence other blackboards, a global virtual blackboard can be formed. If all individuals in the system use the same algorithm for decision making (with respect to who is allowed to execute which task) and communication is reliable, then all robots will come to the same conclusion and successfully allocate tasks without having a concrete leader. In this case, the blackboard system itself takes the leader role, since every robot has to comply with its inherent logic. Examples for such control can be found in [ØMS01] and [BC07].

Opinion Dynamics. Montes de Oca et al. [MdOFM⁺10, MdOSBD10] investigate how an opinion dynamics model can be used to achieve decentralized decision making. In their work, robots have to decide frequently between two different foraging sources without knowing which one can be reached faster and thus is the better one for efficient mission accomplishment. Every robot has an individual opinion about which task is better. At start, this opinion is purely random. Before allocating one of the two possible tasks, each robot has to participate in a negotiation process: a group of three ready robots is picked at random and their opinion is aggregated by a decision rule, e.g. majority. After that, each robot of the group allocates the task corresponding to its new opinion. Because robots with the correct opinion, which is choosing the shorter path, return sooner, they are included in opinion aggregation more often. Thus, they have greater influence on the swarm's opinion.

This technique is very powerful for spreading opinions about time efficiency without actively observing any durations. Although opinion dynamics are not sufficient for controlling a complex task allocation process on their own, they can contribute to a better task allocation in a very elegant way.

Note that the introduction of opinions is already a step towards Autonomous Task Allocation. Robots that follow an opinion about which task is best to take are able to allocate tasks for themselves. But as long as opinions are a result of a dictated negotiation process, the use of opinion dynamics falls into Heteronomous Task Allocation.

BLE. Werger and Mataric [WM00] present a formalism called BLE, which is a shortcut for **B**roadcast of **L**ocal **E**ligibility. In BLE, robots broadcast local eligibility values for all their different behaviors. The robot who is locally best suited for a specific behavior will notice that it has got the highest eligibility value of all robots in communication range. Therefore, it will execute the behavior. Up to this point, this is very much the same as a local version of MURDOCH with its auction-algorithm providing an “instantaneous blackboard”. In contrast to MURDOCH, BLE features the active inhibition of a behavior in other robots. Instead of relying on all robots knowing that one is already executing a behavior, an explicit message which deactivates the behavior in other robots is broadcasted locally.

The use of a behavior-based approach like BLE is another step towards Autonomous Task Allocation. By allowing robots to control their behavior they are able to decide their next step on their own. In BLE, Task Allocation still remains heteronomous, because the received combination of local eligibility values and inhibition messages from other robots explicitly dictates behavior activation. The robots are still unable to decide their next action autonomously.

3.2.4 Relevance for Swarm Robotics

Heteronomous Task Allocation has some strengths and weaknesses. On the one hand, the use of leaders offers the potential to find (local) optimal solutions for Task Allocation. Additionally, from the designer’s perspective, it is relatively easy to program coordinated behavior, because coordination is achieved in a top-down manner and leaders are allowed to control other robots directly. On the other hand, in most cases optimal solutions are bought dearly, as they require high computational and / or communicational effort.

Centralized Task Allocation seems to be the least relevant approach for Task Allocation in Swarm Robotics. *Omniscient control* is unrealistic, because even if it is possible to equip a central leader with all-embracing sensors, algorithms heading for an optimal solution will not be scalable and therefore fail for a large swarm of workers. Although *blackboard control* distributes the knowledge accumulation, it still emphasizes the central processing of massive amounts of data. Additionally, it is very difficult to provide algorithms that include all possible situations appearing in a dynamic environment. Those dynamics are especially hard to predict, when the swarm is contributing to changes in the environment and featuring group dynamics.

To take up the cudgels for Centralized Task Allocation, this thesis proposes to use on-line reinforcement learning in a central entity. Instead of directly targeting optimal solutions, *centralized reinforced control* encourages adaptation at run-time, driven by live experience made in a previously unknown environment. By limiting learning to a single robot, designers can still concentrate on centralized decision making and do not have to worry about all robots making decisions in parallel.

All centralized approaches have in common that they create a single point of failure. Using the yardsticks presented in section 2.1.2, which are defining how swarm robotic a system is, this risky design conflicts with yardstick (I), postulating the *independence from addition and removal of individuals*. Nevertheless, Centralized Task Allocation can still be used in Swarm Robotics if the central decision maker can be replaced easily and thus the chance of swarm failure is reduced.

Distributed Task Allocation tries to increase scalability by distributing control in the whole swarm. This reduces the computational effort for single robots. On the downside, negotiation about task allocation increases the amount of communication needed, and the quality of solution will most likely suffer, at least initially.

The most basic mechanism in Distributed Task Allocation is the *auction*. Auctions are used to announce tasks and to accumulate task utility and / or cost values to make a proper allocation afterwards. In general, market-based approaches are very well suited for single-execution tasks that originate from accomplishment goals: every time such a task emerges from the environment, robots negotiate about who will do the job. In contrast, maintenance goals define tasks that often have to be accomplished frequently by a certain proportion of the whole swarm. In this division of labor case, auctions are less efficient, because they overact by continuously announcing the same task.

Distributed Task Allocation can be very efficient in clearly arranged scenarios. Especially when tasks are concrete one-time jobs and the best suited robot can be derived by comparing scores, market-based approaches are very promising. Unfortunately, the number of robots in the swarm – or at least in communication range of one robot – has still a great influence on performance. Although Distributed Task Allocation may not be applicable for large swarms as a whole, it still has high potential as a local subroutine.

3.3 Autonomous Task Allocation

In contrast to Heteronomous Task Allocation, where an individual is controlled by some kind of leader, Autonomous Task Allocation gives every single robot the ability to allocate its tasks on its own. This section describes the basic concepts of this approach and presents different mechanisms that can be used.

3.3.1 Basic Concepts

Most mechanisms for autonomous coordination of multiple robots are inspired by nature. Especially when focusing on swarms, nature offers interesting subjects for study. Typically, designers emulate techniques from ant or bee colonies because social insects are perfect role models for artificial swarms.

Autonomous Task Allocation is based on the keystones *bottom-up coordination*, *limited communication* and the conflict of *simplicity vs. quality of solution* which are presented in the following.

3.3.1.1 Bottom-Up Coordination

In Autonomous Task Allocation, system design is limited to a local view, which focuses on single entities. Although individuals select their tasks autonomously and therefore do not follow a leader, complex behavior can still appear. From designer's perspective, the emergence of coordinated behavior from individual actions is called *bottom-up coordination*. The designer has to program each robot's action selection in a way that results in the desired behavior at swarm level.

Even very simplistic rules can result in complex behavior. Langton's ant, first appearing in [Lan86], is a perfect example for an individual showing complex behavior while following very simple rules. Langton's ant lives in a cellular world, consisting of white and black cells. When the ant stands on a white cell, it turns 90° to the right, switches the color of the cell to black and moves to the cell in front of it. In contrast,



Figure 3.4: Langton’s ant after 11000 steps. The ant has already started to build the “highway” pattern out of the apparently chaotic behavior before. Image is taken from Wikimedia Commons*.

* <http://commons.wikimedia.org/wiki/File:LangtonsAnt.png>

standing on a black cell results in turning 90° to the left, switching the color to white and again leaving the cell by taking a step forward. Starting on a completely white grid, the ant shows seemingly chaotic behavior for about 10000 steps. After that, it appears to build a “highway” out of the chaos (cf. figure 3.4). The repetitive pattern constructing the “highway” consists of 104 steps that repeat infinitely.

Similar to Langton’s ant, the movement of single real ants can be very confusing at first sight. Watching an ant colony for some time reveals that the simple small actions of single ants form to reasonable patterns on the swarm level. For example, ant trails can be spotted that lead to food sources.

In the context of swarms, the emergence of complex behavior from simple individual actions is often called the result of *swarm intelligence (SI)*. Even if every single member of the swarm is “dumb” with respect to planning and coordination capabilities, the swarm as a whole can still appear to be intelligent. Labella [Lab07] states that “the major contribution from [swarm intelligence], or better its application to robotics, Swarm Robotics [...], is to clearly show that the use of communication, planning, mapping (or any other explicit representation of the environment) are not a necessity.”

3.3.1.2 Limited Communication

In Autonomous Task Allocation, communication plays a minor role. Many mechanisms solely rely on indirect communication via the environment or no communication at all. Each robot selects actions based on its own perception of the environment.

Nevertheless, some form of indirect communication will always be present. Since physical robots are manipulating their environment by the execution of tasks or simply by their presence, the environment’s state includes signs of each robot’s actions, which can be sensed by other robots. If the trace of previous actions stimulates the selection of future actions, the resulting behavior is called *stigmergic*. *Stigmergy* is a form of indirect coordination that can be found in many social insects. For example: termites form surprisingly complex nests by iteratively placing available building material on top of each other. By putting some pheromone into the building material, other termites are animated to continue building at the same structure. Each individual perceives the signs of other workers and reacts to them. This local behavior emerges the construction of a typical termite nest although no planning is involved and communication is limited.

3.3.1.3 Simplicity vs. Quality of Solution

Inspired by social insects, Autonomous Task Allocation counts on simplistic individuals that do not necessarily know the goals of the swarm. The global mission is achieved by coordinated behavior that emerges from multiple robots acting autonomously.

Although nature evolved very efficient solutions, the solution quality is never optimal. This is a direct result of relying on individuals with a limited perceptual range that does not allow to oversee the whole swarm. Inevitably, the number of workers executing a specific task will not be minimal. If there was a single controlling entity, the swarm could perform better. In fact, the absence of a single point of failure and the inherent redundancy are intended features that make the swarm more robust, especially in dynamic environments.

In Autonomous Task Allocation, the quality of solution is very dependent on the subtlety of the individuals' design. If designed properly, an efficient solution for the Task Allocation problem can emerge. As already demonstrated by the example of Langton's ant, even very simple rules for action selection can result in unexpected behavior. Because of this, simulation is a very important tool in the design of mechanisms for Autonomous Task Allocation. As most methods are driven by a set of variables that have to fit the concrete mission of the swarm, parameters are first adapted in a multitude of simulation runs before using them in a real world experiment.

In some cases, the outcome of autonomous mechanisms can also be predicted by the use of mathematical models. These can help to understand dynamic task allocation before implementation and therefore save costs in development. Unfortunately, for complex systems, mathematical models are not easy to define and are often based on assumptions that are not valid for real world scenarios. Nevertheless, some mathematical models have already been proposed, at least for the prominent domain of foraging [LJGM06, CD07].

Autonomous Task Allocation relies on the emergence of coordinated behavior from decentralized decision making. In consequence, every single robot has to allocate its tasks in a way that inherently drives the whole swarm to mission accomplishment. The following sections present different mechanisms for Autonomous Task Allocation by focusing on the type of control used in each autonomous robot.

3.3.2 Rule-based Control

In rule-based control, each agent is equipped with a set of rules defining which action to take under which circumstances. Those rules are bounded to the robot's view which includes sensory data and potentially accumulated knowledge or belief. By following its personal rules, each robot autonomously allocates tasks for itself.

In literature, approaches in this category are often related to *behavior-based systems*. As the name suggests, each robot manages a set of behaviors. In the context of this thesis, behaviors are very much the same as tasks. There is only a small difference that results from the point of view: tasks are considered to be abstract descriptions what a robot should do without describing how to do it, whereas behaviors are very much related to a course of concrete actions. In other words: the term task is used from the viewpoint of a leader, who wants to assign a job, whereas behavior defines a strategy that is followed by a robot to execute a given task. The term behavior is also used from the viewpoint of a watcher, who observes different kinds of activity.

3.3.2.1 Basic and Complex Behavior

There are two different kinds of behavior: basic and complex behavior. Each robot is primarily described by its basic behavior set. These behaviors are unique, independent from each other and none of them can be described by the others. From the viewpoint of rule-based control, which aims at Task Allocation, basic behaviors are analog to atomic tasks: there are no smaller units of control.

Complex behaviors are pre-described strategies that use a combination of basic behaviors. Instead of having to deal with atomic tasks, rule-based control is then able to allocate complex tasks that can be accomplished by the activation of a corresponding complex behavior.

For example: Mataric [Mat95] implements the complex high-level behavior *foraging* by switching between the basic behaviors *avoidance*, *dispersion*, *following*, *homing* and *wandering* under appropriate sensory conditions. Additionally, the rudimentary abilities to pick up and drop items were used.

The repertoire of abilities that rule-based control works with is defined by both basic and complex behaviors. Because the assignment target is evident, rule-based control – and Autonomous Task Allocation in general – allocates tasks by the activation of corresponding behaviors.

Rule-based control builds upon a given set of strategies to choose from, where each strategy is a combination of basic behaviors. For example: a foraging robot might be able to apply a foraging or resting behavior. Rule-based control defines when to switch between these two strategies. From this point of view, it is irrelevant that foraging is a complex behavior consisting of multiple basic behaviors. It is assumed that the strategy is already implemented, possibly by a rule-based approach.

3.3.2.2 Examples

This section presents some examples from literature that use a rule-based approach to control Task Allocation.

Aggregation. Schmickl et al. [STM⁺09] present a very simplistic rule-based approach. In their work, a robot swarm has the mission to aggregate in areas with high light intensity. Each robot follows a simple algorithm, called BEECLUST. By default, a robot performs a random walk. Every time it collides with another robot, it stops, checks the light intensity and waits for a time dependent on the measured illuminance before switching to random walk again. This small set of rules is sufficient for clustering the swarm in areas with high light intensity. As the name suggests, the BEECLUST algorithm is inspired by the behavior of bees. Bees tend to aggregate in spots with comfortable temperature. However, this behavior can only be observed in groups of bees: single bees do not stop at any location on their own. Clustering can only emerge if multiple bees are present for some kind of interaction. Collision with other bees seems to be the only form of communication that drives this behavior.

Chain Formation in Foraging. Nouyan et al. [NGB⁺09] demonstrate that a fixed rule set can be used to create complex behavior. In their foraging scenario, robots have to find a heavy food item and collectively retrieve it to the nest. This mission is achieved by the formation of a robot chain that originates in the nest. Robots follow a set of conditions that trigger transitions between different behaviors, like *search chain*,



Figure 3.5: Foraging via chain forming: each robot follows a set of conditions that trigger transitions between different behaviors. In the left picture, robots have built two short chains that both do not yet reach the food item in the left corner of the area. In the right picture, one long chain to the food item could be created and two robots are already attached for transportation. Both images are taken from [NGB⁺09].

join chain, explore chain, assemble and transport target. Figure 3.5 shows two stages in a trial with twelve robots. The transitions between some behaviors related to chain maintenance are driven by the probabilistic parameters P_{in} and P_{out} , which define the probability for joining and leaving a chain respectively. Because of their influence on chain formation and stability, these parameters have to be tuned very carefully to fit the specific environment.

Light-based Task Allocation. Ducatelle et al. [DFDCG09b, DFDCG09a] use a rule-based approach as a sub-strategy in a larger task allocation mission. In order to allocate a multi-robot task, special robots, called eye-bots, have to gather a specific number of foot-bots around them. Beside an approach using infra-red communication, Ducatelle et al. propose a light-based mechanism that controls the aggregation of a sufficient number of foot-bots. The eye-bot announces the multi-robot task by emitting a number of yellow lights proportional to the task size.

The light-based approach utilizes *attraction* and *repulsion* to achieve appropriate clustering. Each foot-bot is driven by simple rules: if the robot senses yellow lights, it heads towards the nearest one, if it senses green lights in a short range, it is repulsed from them. Because foot-bots emit a green light, they indirectly disadvise other robot to head to the same target. At the eye-bots location, all present robots spread out and “dock” at different yellow lights. This helps to accumulate the correct number of robots needed for the task.

Whenever attraction and repulsion is sensed at the same time, an internal frustration level is raised. If the frustration reaches a fixed threshold, the foot-bot executes an escape movement, which drives it away from the attraction source and enables it to explore other parts of the arena. Figure 3.6 demonstrates the influence of attraction, repulsion and frustration in a simulated example.

Rule-based control is a very powerful mechanism to determine Task Allocation for single robots. This is advantageous and disadvantageous at the same time. On the one hand, designers have full control over the robots’ behavior. Thus, they can program

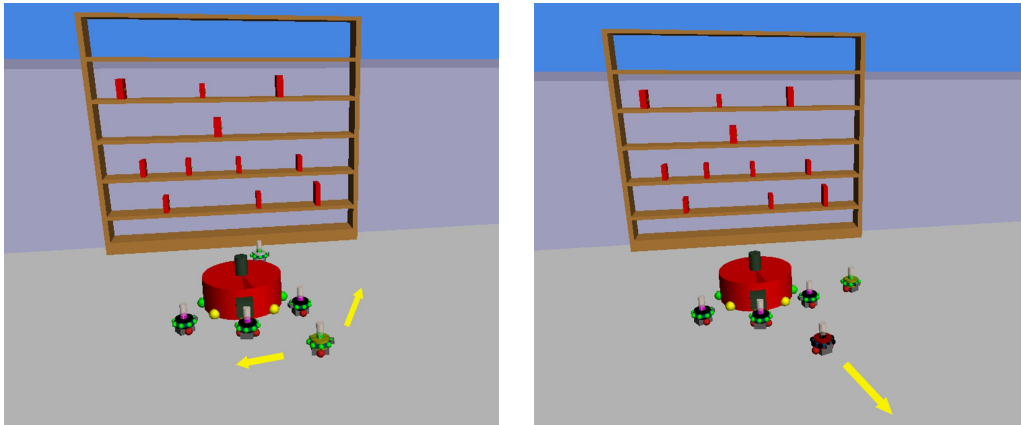


Figure 3.6: *Light-based control: robots are attracted by yellow lights and repulsed from green lights. In the left picture, the robot is attracted and repulsed at the same time. In the right picture, the robot’s frustration level for sensing attraction and repulsion at the same time has reached a fixed threshold, which forces the robot to execute an escape movement away from the attracting yellow lights. Both images are taken from [DFDCG09a].*

interacting rules that emerge a desired swarm behavior. On the other hand, desired behavior is hard to achieve by the definition of simple rules. In most cases, the rules are tailored to the concrete global mission and include the specification of complex behaviors. As a result, rule-based systems as a whole are generally highly specialized and rarely reusable in other missions and scenarios. Note that sub-sets of the rules, like the descriptions of complex behavior, are very well reusable.

Rule-based control can be seen as a superclass for all forms of Autonomous Task Allocation, because every mechanism can be described by a set of rules. It is even possible to simulate Heteronomous Task Allocation by the definition of communication rules. Therefore, this thesis proposes to use the term rule-based control only for rule-based systems that are designed from scratch without using a mechanism that automatically defines a set of rules. From designer’s perspective, the definition of rules has to remain tractable, of course.

Since rule-based control is a very unspecific approach to Autonomous Task Allocation, the following sections cover more concrete mechanisms. First, threshold-based control is discussed, which switches between two tasks by the use of a special parameter called threshold. Second, probabilistic control is presented, which chooses from a set of tasks based with respect to given probabilities. Finally, decentralized reinforced control is proposed, which adapts action selection policies based on experienced rewards.

3.3.3 Threshold-based Control

In threshold-based control, the self-allocation of a task basically depends on a specific parameter called threshold. In general, some stimulus is frequently incremented by task-relevant events. As soon as the stimulus reaches or passes the threshold, the execution of a corresponding task is triggered.

An example for the use of a threshold was already given in rule-based-control: in the light-based approach of Ducatelle et al. [DFDCG09b, DFDCG09a], each mobile robot is equipped with a frustration level that is raised when the robot is synchronously

exposed to attraction by yellow lights and repulsion from green lights. Passing a fixed threshold triggers an escape movement that drives the robot away from the source of attraction (cf. figure 3.6).

Although the principle of threshold-based approaches always remains the same, various implementations and interpretations are possible. In some cases, the threshold is an absolute value that controls which exact stimulus is needed to activate the execution of the corresponding task. In other cases, the threshold appears as a parameter of a function that defines probabilities dependent on some stimulus. This enables the threshold to control the influence of stimuli at a finer grain. The first category is based on the *activation threshold model* and is described next.

3.3.3.1 Activation Threshold Model

In the activation threshold model [KB00], each robot manages private fixed thresholds that define the amount of stimulus needed for task activation. After activation the stimulus eventually drops below the threshold enabling the robot to switch to other tasks that demand attention.

For example: Krieger and Billeter [KB00] apply a simple form of this task allocation method to a foraging scenario. The global mission of the swarm is to keep the nest-energy at a safe level by gathering items that increase the nest-energy on retrieval. There are two tasks robots can select from: foraging and resting. Both tasks need energy but resting is comparably low in cost. As a result, resting robots are draining less energy than unsuccessful foragers. To avoid unnecessary use of energy, robots follow a passive strategy: each robot continuously checks the amount of energy available in the nest and starts foraging as soon as the energy drops below an activation threshold.

By giving each robot a private activation threshold, not all robots start to execute a specific task at the same time. In the foraging example, a proper dispersion of activation thresholds results in robots having different opinions about which nest-energy value is critical and therefore needs their attention. This technique enables the swarm to maintain a proper division of labor although each threshold is fixed, equally interpreted and never changed.

Activation thresholds are very easy to use in scenarios with only two tasks to choose from. By declaring one task to be the default task and introducing a single activation threshold for the temporally constrained activation of the other task, the task allocation process is straightforward.

If there are more than two tasks controlled by activation thresholds, some additional rules have to be defined. Otherwise it would be unclear which task to execute if multiple thresholds are passed concurrently. One way to make things clear is to define a precedence order for task activation. This ordering can be either fixed or dynamic. By comparison of the exaggerated stimuli, the most urgent activity may be found.

3.3.3.2 Adaptive Threshold Functions

As already mentioned, thresholds can be used as control parameters for probabilistic functions. Strictly speaking, Task Allocation is then controlled by the function and not by the threshold. But as long as the threshold controls the reactivity to a stimulus and thus describes some vague point where the allocation of a task gets very likely, it is still a threshold-based approach. In the context of this thesis, a threshold function maps a stimulus s to a probability $p(s)$ by the incorporation of a corresponding threshold t .

One of the simplest threshold functions can be defined by translating the activation threshold model into a probabilistic function. Since probability switches from 0 to 1 at the fixed activation threshold t , the function can be formulated as follows:

$$p(s) = 1 - \max(\text{sgn}(t - s), 0) = \begin{cases} 1, & s \geq t \\ 0, & \text{else} \end{cases} \quad (3.1)$$

By allowing a robot to change the activation threshold t , the threshold function gets adaptive. For example: assume a blind baby bird that gets more and more hungry and eventually begins demanding food. The biddy does neither know how many other birds are in the nest, nor does it know how long its mother will need to bring a sufficient amount of food. In this scenario, the stimulus s increases every time step the bird feels any amount of hunger while not demanding food. The threshold t then represents the time to wait before demanding food. After mother bird has come back and has fed her children, the biddy checks if it is still in a hungry state. In this case the biddy reduces its waiting time, starting to demand food more early next time. Otherwise it may increase the threshold, because it might have been overfed. This scenario covers both dynamic environments and group dynamics.

Inspired by the work of Brutschy [Bru09], this thesis proposes to use a shifted sigmoid curve, which can be seen as a flattened version of the strict activation function 3.1. By using the logistic function, which is the most common sigmoid curve, as a basis, the probability function can be defined as follows:

$$p(s) = \frac{1}{1 + e^{-\theta(s)}} \quad \text{with} \quad \theta(s) = \frac{6}{r}(s - t), \quad (3.2)$$

where t is the threshold defining the amount of stimulus s needed to get a probability of 0.5. The constant r is used to control the steepness of the curve. Because the logistic function basically changes in the interval $[-6, +6]$, r can be seen as the radius around the threshold where the probability rises from approximately 0 to approximately 1.

Figure 3.7 shows two sigmoid curves with example values for r in comparison to the original logistic function. For low values of r , the sigmoid curve behaves very much the same as the strict activation function 3.1. For high values of r , the transition gets more fluid and robots will manage a small probability for starting the task even when the stimulus s is low. Note that the probability will never reach 0 or 1. In theory, there is always a small probability left to execute or ignore the task respectively.

By increasing the threshold t , the logistic curve is shifted to the right. As a result, the stimulus has to reach higher values to have the same influence on probability. As soon as the threshold is passed, the task activates with a chance of at least 50%. Thus its execution gets likely, which corresponds to the general idea of threshold control.

In their study of ants, Bonabeau et al. [BTD96] have chosen a similar function to model the probability of reacting to a stimulus:

$$p(s) = \frac{s^2}{s^2 + t^2}. \quad (3.3)$$

Again, the threshold value t defines the amount of stimulus s needed to activate the task with probability $p(s) = 0.5$. Bonabeau et al. use this model to explain the behavior of ant colonies consisting of two castes: minors and majors. As long as enough minors are present to execute a specific task frequently, the majors remain inactive with respect to the task. As soon as the minors count is reduced, the task is not executed at

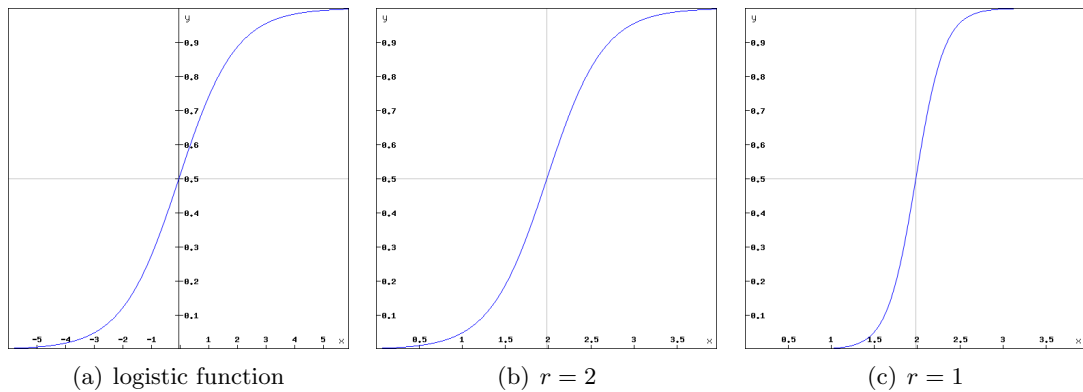


Figure 3.7: (a) The logistic function. (b) + (c) Two logistic probability functions for task activation (cf. equation 3.2) with different values for the radius r . The threshold parameter $t = 2$ defines that probability 0.5 is reached at $s = 2$.

a sufficient rate, which raises the stimulus and results in the majors getting involved. This is regarded as a result of different threshold levels in the two castes.

3.3.3.3 ALLIANCE

ALLIANCE [Par98] is a very prominent framework using a threshold-based approach. Each robot has a set of behaviors – tasks – that can each be activated by a corresponding stimulus called *motivation*. Motivation is basically increased by an *impatience rate*, which is fast by default. If any robot has already started the task, the impatience rate is low, at least as long as the other robot fulfills the task in reasonable time bounds.

While the robot increases its motivation for a task, each time another robot starts to execute the task, the motivation is reset to zero. Other conditions for resetting the stimulus are missing sensory feedback, another activity that suppresses activation and *acquiescence*. Acquiescence indicates that the robot gives up, either because it wasn't able to achieve the task in a reasonable time or because another robot took the task assuming that a standard achievement time was exceeded. In summary, the motivation for a specific task is updated as follows:

$$\begin{aligned}
 m_t &= [m_{t-1} + \textit{impatience}_t] \\
 &\quad * \textit{sensoryFeedback}_t \quad // \text{ 0 if task is not applicable} \\
 &\quad * \textit{activitySuppression}_t \quad // \text{ 0 if other task is active} \\
 &\quad * \textit{impatienceReset}_t \quad // \text{ 0 if other robot has started the task} \\
 &\quad * \textit{acquiescence}_t, \quad // \text{ 0 if there is a reason to give up}
 \end{aligned}$$

where the index t denotes the current time step, m_{t-1} is the motivation from the last time step and $\textit{impatience}_t$ is either a slow or fast impatience rate. In order to avoid a reset of motivation to 0, all binary variables – $\textit{sensoryFeedback}_t$, $\textit{activitySuppression}_t$, $\textit{impatienceReset}_t$ and $\textit{acquiescence}_t$ – need to be 1.

ALLIANCE is designed to work in fault-prone environments. Each robot watches the performance of other robots and takes the task if the robot does not perform well. By this, failures in robots are detected and all tasks are continuously executed as long as the sensory feedback allows the tasks' execution.

On the downside, ALLIANCE introduces many variables that have to be adapted carefully to the specific mission. In particular, the impatience rates must conform the

task execution times of the robots. To get a grip on this problem, the ALLIANCE framework is extended by a learning and adapting approach. In L-ALLIANCE [Par97], all robots monitor their own and the other robots' performance times. Based on these values, each robot is able to estimate how long a task takes by default.

L-ALLIANCE features two distinct phases: the active learning phase for training missions and the adaptive learning phase for live missions. In training, the robots are maximally patient and minimally acquiescent. This ensures that they can learn their own and each others' task execution times without disturbance. In live missions, the training data is then used to estimate proper impatience rates. Of course, monitoring remains active and the robots are still able to adapt their estimates.

With respect to Swarm Robotics, one of the most constraining properties of ALLIANCE is that robots have to be able to sense what other robots are doing. This degree of awareness can easily be reached by allowing robots to broadcast their status. Unfortunately, broadcasting does not scale very well and may be limited by the medium's bandwidth. To overcome this problem, communication range may be decreased. In such a "local version" of ALLIANCE, the robots will still be able to perform well, but the solution quality will most likely suffer due to the missing knowledge about robots out of range.

3.3.3.4 Summary of Threshold-based Control

Threshold-based control is a very intuitive mechanism for activating a single task. In activation threshold models, the threshold absolutely defines when to switch to the task. To avoid that all individuals of a swarm are switching at the same point of time, the threshold values can be dispersed. Alternatively, the robots can interpret the threshold and probably switch earlier or later. This is achieved by the use of a threshold function, mapping a stimulus to an activation probability. The point where activation gets likely is controlled by the threshold.

ALLIANCE is a prominent framework using activation thresholds. Its robustness is based on the availability of information about the states of other robots. As a result, its usefulness is limited to scenarios where either communication of states is possible or sensory data is rich enough to derive other robots' performance from.

If more than one task demands activation at the same time, the robot has to make a decision. By the use of strict rules, like a precedence order, tasks with low demands will never be served as long as higher demands are present. This undesired behavior can be countered by probabilistic control, which may be based on thresholds but doesn't have to.

3.3.4 Probabilistic Control

In probabilistic control, each robot randomly allocates tasks based on a discrete probability distribution which maps each task i to a probability value p_i ($\sum_i p_i = 1$). This probability distribution, which may be different for each robot, controls individual behavior.

As already mentioned in threshold-based approaches, adaptation of control parameters – like thresholds – is a very powerful mechanism to deal with previously unknown and potentially dynamic environments. In probabilistic control, the change of one probability has to involve a change of at least one other probability to ensure that the sum of all values still equals 1.

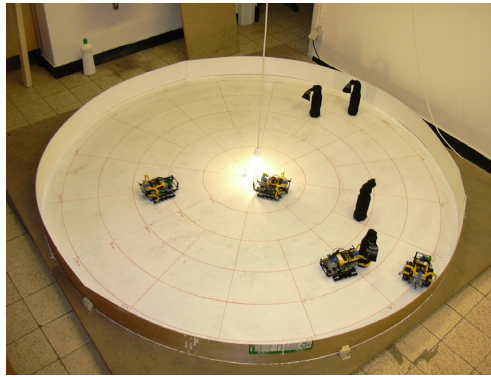


Figure 3.8: Snapshot of an experiment using the Variable Delta Rule. Four robots are either foraging or resting. The nest, which is the retrieval zone for prey, is situated in the center and indicated by a light source. The image is taken from [LDD04a].

In the following, adaptation in the simple case of two tasks to choose from is discussed first. After that, a more general solution for adaptive selection from an arbitrary number of tasks is presented.

3.3.4.1 Variable Delta Rule

In the case of two tasks, managing a single probability value p_1 is sufficient because the other one can be derived from p_1 : $p_2 = (1 - p_1)$. Labella et al. [LDD04a, LDD04b, Lab07] investigate a foraging mission where each robot manages a value p_1 defining the probability to stop resting and to activate foraging.

Figure 3.8 shows a snapshot of an experimental scenario where four *MindS-bots* are deployed for foraging. A *MindS-bot* is a LEGO MINDSTORMS™ robot that emulates the abilities of an s-bot from the Swarm-bots Project (cf. section 2.2.3.4).

The success of foraging is dependent on the amount of food available in the environment. After some time without finding food, the robot will return to the nest, abort foraging and activate resting. If foraging was successful, p_1 is raised by Δ_s , and if foraging failed, p_1 is reduced by Δ_f . Δ_s and Δ_f are dependent on the consecutive number of successes and failures respectively: $\Delta_s = \text{successes} * \Delta$ and $\Delta_f = \text{failures} * \Delta$. Additionally, p_1 is constrained to a minimum and maximum value, which ensures that foraging is never completely abandoned.

This adaptation mechanism is called *Variable Delta Rule*. Although it is very simple, it is able to efficiently adapt to dynamic environments. Additionally, a proper division of labor can be achieved. If the swarm is heterogeneous and some robots are more successful in foraging than others, the swarm will even select the best individuals more often.

Liu et al. [LWS⁺07a, LWS⁺07b] propose some modifications of the Variable Delta Rule that can make adaptation even more efficient. In their work, each robot is not only influenced by its own outcome but also by the other robots' observations and some environmental cues. In contrast to the work of Labella et al., Liu et al. do not adapt probabilities but thresholds: instead of adjusting the probability for starting a time-constrained foraging task, the times for resting and searching are adapted. Nevertheless, the robots behave very similar and both approaches are appropriate solutions. In fact, the Variable Delta Rule is an all-purpose mechanism.

3.3.4.2 Motivation-based Task Allocation

This thesis proposes a motivation-based approach for Task Allocation. Each task is related to a specific motivation level in the range $[0, 1]$. In contrast to ALLIANCE, the motivation value does not have to reach a given threshold before the corresponding task is activated. Only a motivation level of 0 means that the task will definitely not be selected. In all other cases, the selection is possible by chance. Note that the highest motivation level of 1 does assure selection but indicates that there is no other task that is selected with a higher probability.

Individual robots allocate their next task by comparing all motivation levels and building a simple discrete probability distribution where the weight of each probability conforms the weight of each motivation. As a result, each probability for selecting task i can be calculated from the corresponding motivation m_i as follows:

$$p_i = \frac{m_i}{\sum_k m_k}. \quad (3.4)$$

This definition assumes that at least one motivation m_i is greater than 0. Basically, the corresponding action selection algorithm is a simple form of softmax action selection where actions – tasks – are weighted by their motivation values (cf. section 2.3.1.4 in the introduction of reinforcement learning).

In fact, limiting the motivation values to the range $[0, 1]$ is not necessary. Tasks that manage a minimum motivation level greater than 0 will always be selected by chance. Tasks with increased upper motivation bound may accumulate more weight in comparison to other tasks.

For the modification of motivation values, the designer may use any kind of adaptation rules. The corresponding probabilities will always be adjusted automatically. Nevertheless, the design of proper rules remains difficult. In most cases, common sense helps to find efficient adaptation rules. For example: Momen and Sharkey [MS09] present some fixed adaptation rules in the context of a swarm consisting of foraging and brood caring ants. Although they are working with thresholds defining some waiting time, the concept of adaptation is valid for motivations, too: if brood carers observe a lack of food, their threshold value for foraging is reduced, which conforms the increase of a corresponding motivation.

Note that motivation-based Task Allocation forces the selection of a task. To enable robots not to activate any task, an additional “do nothing” task has to be introduced. By the use of proper adaptation rules, the motivation for this empty task can be raised if no other task demands activation.

Motivational Adaptive Threshold Functions. Motivation-based Task Allocation can be combined with adaptive threshold functions. In this case, each former probability for activating a task, which is dependent on a stimulus and a threshold, is interpreted as a motivation value for Task Allocation. This paragraph defines the corresponding discrete probability distribution that is used to control a single robot.

Given that n is the number of active tasks a robot can choose from, task 0 is defined as the *idle task*. If the idle task is activated, the robot waits for a fixed amount of time. This waiting time defines the rate of an idle robot bothering with Task Allocation. In contrast, the execution times for task 1 to n are task dependent. The next task is not selected until the currently active task has been completed. For example: a foraging task can last until a food item is successfully retrieved or until some internally defined search time is exceeded.

For each active task $i \in \{1, \dots, n\}$, the robot under consideration manages a stimulus s_i . The vector $s = (s_1, \dots, s_n)$ covers those stimuli and is calculated from the robot's perception, which may consist of both sensory input and accumulated knowledge. Additionally, the vector $t = (t_1, \dots, t_n)$ defines threshold values for each active task.

The probability for selecting task i , which is dependent on the observed stimuli vector s and the current set of thresholds t , can then be defined as follows:

$$p_i(s, t) = \frac{m_i(s, t)}{\sum_{k=0}^n m_k(s, t)}, \quad (3.5)$$

where the motivation value $m_j(s, t)$ is defined by a threshold-based function that returns a provisional probability for activating task j – including the idle task 0. By using a threshold function based on the logistic function, m_j can be defined as follows:

$$m_0(s, t) = \max(0, 1 - \sum_{k=1}^n m_k(s, t)),$$

$$m_i(s, t) = \frac{1}{1 + e^{-\theta_i(s, t)}}, \quad \theta_i(s, t) = \frac{6}{r_i}(s_i - t_i), \quad i \in \{1, \dots, n\},$$

where r_i is the control parameter that adjusts the steepness of the sigmoid curve. The definition of m_0 ensures that the provisional probabilities remain the same if their sum is smaller than 1. The idle task will simply adopt the remaining probability. As a result, the threshold t_j still indicates the level of stimulus s_j needed to get a probability of 0.5 to select the task. As soon as the sum of provisional probabilities exceeds 1, the idle task adopts a probability of 0 and the provisional probabilities are weighted due to their interpretation as motivation levels.

The incorporation of the idle task is a tradeoff between full-time engagement and selective engagement. As long as there is a sufficient demand to do something, the robot will definitely activate a task unequal to the idle task. As soon as the sum of motivating probabilities drops below 1, the robot will probably stay idle. For example: given two tasks with $m_1 = 0.3$ and $m_2 = 0.2$, the idle task gets probability $p_0 = 0.5$ and the two other tasks adopt the provisional probabilities: $p_1 = m_1 = 0.3$ and $p_2 = m_2 = 0.2$. Given two tasks with $m_1 = 1$ and $m_2 = 0.5$, the idle task is rendered impossible and the remaining probabilities are weighted: $p_0 = 0$, $p_1 = \frac{1}{1+0.5} = \frac{2}{3}$ and $p_2 = \frac{0.5}{1.5} = \frac{1}{3}$.

Note that each motivation value is influenced by a perceived stimulus and a threshold. As before, the threshold may be adapted, e.g. by the Variable Delta Rule.

3.3.4.3 Summary of Probabilistic Control

Probabilistic control follows a simple principle: let chance decide which task to allocate. Each robot manages a private discrete probability distribution and acts accordingly.

The Variable Delta Rule is a possible mechanism for adapting the probability distribution. It was presented in the context of foraging, where a decision between searching and resting had to be made. Basically, the Variable Delta Rule proposes to react to specific events, like success and failure, to modify the probability to engage the corresponding task accordingly. In the simple case of two possible tasks, e.g. foraging or resting, one probability value is sufficient and can be adapted directly, without bothering that the probabilities have to sum up to 1.

In the case of more than two possible tasks, this thesis proposes to calculate the probability distribution by weighting artificial motivation levels. These values may be either fixed or dependent on some parameters. In the latter case, this thesis presents a mechanism for probabilistic control that uses adaptive threshold functions as a basis for motivation.

Up to this point, all adaptation is controlled manually: the designer defines some fixed rules modifying concrete values by using common sense. These rules rely on prior knowledge about the tasks. The next section presents an alternative mechanism for Task Allocation that renounces such knowledge.

3.3.5 Decentralized Reinforced Control

In decentralized reinforced control, Task Allocation is driven by the policy of a reinforcement learning method (cf. section 2.3). Initially, the robots do not know which tasks are better in which state. Each robot has to autonomously learn a policy that maximizes accumulated reward in the long run.

In fact, decentralized reinforced control is a special case of probabilistic control, because a policy is nothing else than a discrete probabilistic distribution which is adapted by strict rules derived from the used reinforcement learning method. Sarsa, for example, formulates a rule saying: “each time a reward is observed, update the probability of re-choosing the task that led to the reward.”

Balch [Bal99] uses Q -learning in a multi-foraging mission. The robots have to learn a policy for selecting from a set of behaviors dependent on their perceptual state. Balch compares three different methods for rewarding agents:

Local performance-based reinforcement: Each robot is rewarded individually if it delivers a food item.

Global performance-based reinforcement: All robots are rewarded if any robot delivers an item.

Local shaped reinforcement: Each robot is rewarded progressively for accomplishing portions of the foraging task.

Both performance-based techniques are tied to the performance metric, which is the delivery of food items. Such delayed rewards can slow down learning. By rewarding sub-tasks of foraging more directly, the learning time can be decreased significantly. On the downside, local shaped reinforcement is formed by the designer’s opinion of how the mission should be achieved. Thus, the robots are not fully autonomous in finding an efficient solution for the mission. Bad design of rewards may even distract the robots from the optimal solution.

Balch uses Q -learning essentially for training. In his scenario, robots even have to learn basic abilities. Initially, a robot does not know that it should drop a food item in the delivery zone. Having to learn such elementary abilities slows down adaptation unnecessarily, especially in live missions. For instance: assume that one robot has specialized in foraging green items and another one in foraging red items. After this training, the robots are put in a live mission, where only red items are present. As a result, the green-specialized robot may have to learn foraging red items from scratch. If the robot uses a greedy policy, it may even be unable to adapt, because it will not explore this possibility anymore.

Dahl et al. [DMS03, DMS09] examine a simpler multi-foraging mission: each robot can choose between two spatially separated foraging cycles. The swarm as a whole has to manage an appropriate proportion of robots in one cycle and the other. This ratio is influenced by the food density in each cycle and by the group dynamics. Too many robots in one cycle will increase collision probability and decrease efficiency. Dahl et al. use Q -learning and ϵ -greedy action selection to adapt to different circumstances.

Martinson and Arkin [MA03] use Q -learning for adapting the selection of different roles in a foraging scenario. In their work, robots assume either the role of a forager, soldier or mechanic. Foragers gather prey, soldiers defend against enemies and mechanics repair robots that are damaged by enemies or otherwise stuck in the environment. The selection of a role activates a predefined complex behavior, which is implemented by a Finite State Automata.

For on-line learning, this thesis proposes to use Sarsa(λ) instead of Q -learning, because the on-line performance of the off-policy control approach of Q -learning may be worse than the performance of the on-policy control approach of Sarsa(λ) (cf. section 2.3.3.1).

Decentralized reinforced control is a very elegant mechanism for Task Allocation. Reinforcement learning allows robots to adapt to dynamic environments inclusive group dynamics. In order to limit the space of possible actions, complex behaviors should be used. Of course, these behaviors can also be controlled by separate reinforcement learning.

The most critical and difficult design aspect in reinforced control is the definition of a proper reward function. Especially when the essential rewards result long time after the corresponding action, additional rewards may be needed to speed up learning and maintain adaptivity in practice.

3.3.6 Relevance for Swarm Robotics

Autonomous Task Allocation focuses on the control of individuals. By proper definition of local rules, the swarm may emerge complex behavior that efficiently achieves the global mission.

Since most mechanisms are inspired by social insects and rely on local and limited sensing and communication, Autonomous Task Allocation is “naturally” related to Swarm Robotics. In fact, most literature speaking of mechanisms for Task Allocation in Swarm Robotics refers to Autonomous Task Allocation.

Unfortunately, the bottom-up approach of Autonomous Task Allocation involves some disadvantages. Local rules, thresholds and rewards are hard to design in order to emerge a desired coordinated behavior. As a result, solution quality for missions with highly coordinated tasks is likely to suffer in comparison to a top-down approach.

On the other hand, Autonomous Task Allocation has a big strength: inherent robustness. Each individual tries to perform as best as possible solely relying on its own view of the world. As a result, failures of other robots are perceived as a part of the dynamic environment to which all swarm members will naturally adapt.

Kalra et al. [KM06] compare threshold-based Task Allocation and market-based approaches. They come to the conclusion that threshold-based Task Allocation is nearly as efficient as market-based Task Allocation but on a fraction of the expense. If communication is error prone, the threshold-based approach can even win against the market-based approach.

3.4 Hybrid Task Allocation

Hybrid Task Allocation covers all allocation strategies that use a combination of Autonomous and Heteronomous Task Allocation.

This thesis proposes two different categories of combination: *interlaced control* and *side-by-side control*. Interlaced control mixes mechanisms from both Autonomous and Heteronomous Task Allocation to create a completely new methodology, whereas side-by-side control switches between fully developed mechanisms from both categories.

3.4.1 Interlaced Control

Interlaced control may be the most powerful type of combination because it is able to invent new methods that benefit from both Autonomous and Heteronomous Task Allocation. This section presents some mechanisms and frameworks in this category.

3.4.1.1 Approximation of Neighbor Actions

If robots are aware of each other and can approximate the next actions of their neighbors, each robot is able to exploit this knowledge and adapt its own plan. This approach is based on local decisions that are influenced by other near robots. As a result, the process is a mixture of Autonomous and Heteronomous Task Allocation. On the one hand, each robot is autonomous in decision making. On the other hand, the robot is influenced by its neighbors making its decisions partly heteronomous.

An example for an architecture using neighbor approximation is MVERT, which was developed by Stroupe [Str03]. MVERT stands for “Move Value Estimation for Robot Teams”. This method is successfully applied in simulation and on physical robots for mapping, dynamic target tracking and exploration. By approximating the next-step contributions of its neighbors, each robot is able to adapt its next action. MVERT performs better than completely autonomous action selection. At the same time it limits computation time compared to a one-step optimal omniscient planner.

Another example for a similar approach is presented by Atay and Bayazit [AB07]. In their work, approximation is not based on sensory input but on communication. Robots actively exchange plans with the k-next neighbors to maintain a self-deployment mission. Additionally to exploration and observation, the swarm has to manage communicational connectivity. Atay and Bayazit propose the exchange of the following information:

Intentions: Each robot informs about the position that maximizes its utility.

Directives: Each robot computes optimal positions for its neighbors and informs each of them accordingly.

Target Assignment: Each robot informs about its target coverage when located at the intended location.

By the exchange of intentions, directives and target assignment, each robot is able to adapt its strategy to the desires of its neighbors. This local plan refinement enables the swarm to emerge optimized allocations.

Furthermore, Parker [Par02] presents a similar approach based on the ALLIANCE framework. In her multi target observation scenario, each robot locally broadcasts messages about its observation status. Received messages from other robots result in repulsive forces whereas sensed targets result in attractive forces. The robot combines these forces to find an accumulated movement direction.

3.4.1.2 Hoplites

Hoplites [KFS05], presented by Kalra et al., is a framework that combines market-based approaches with the approximation of neighbor actions. Accordingly, two different modes are featured: active and passive coordination.

In passive coordination, which is the default behavior and conforms an adapted version of Stroupe's MVERT [Str03], each robot broadcasts its locally best plan. This plan is frequently adapted as the robot receives the plans of its neighbors. If a robot discovers that it would be more profitable if another robot changed its plan, it switches to active coordination.

In active coordination, robots use a market-based approach for negotiation about plans. In order to increase their personal profit, robots offer virtual money that other robots get if they change their plans in return. As soon as the negotiation process is concluded, each participating robot switches back to passive coordination.

3.4.1.3 AFFECT

Gage [Gag04] presents an emotion-based recruitment approach called AFFECT. In his study, market-based task announcement is mixed with a threshold-based mechanism for autonomously deciding when to reply to the announcement.

AFFECT uses the emotion *shame* to control the willingness to respond to a received help message from another robot. After this task announcement, the robot remains calm and delays its respond to the point of time where the shame variable exceeds a given private threshold. Each robot increases and decreases shame at a different rate that may depend on the robot's utility for the task.

This technique ensures that robots stay calm as long as possible. Initially, a task announcement is probably ignored completely or answered by a single robot only. Likely, this robot is the best-suited one, because it seemed to increase its shame value at the highest rate. As a result, the amount of needed communication messages is reduced drastically. AFFECT is very well suited for stealth missions because robots will not reveal their position immediately if any robot needs help.

Interlaced control mixes different mechanisms from both Autonomous and Heteronomous Task Allocation. This enables the creation of complex algorithms that hopefully balance each other.

There are not many frameworks that intentionally use interlaced control. This may be a result of the effort that is needed to develop and test new mechanisms. Fortunately, the definition of approaches using both Autonomous and Heteronomous Task Allocation can be based on existing mechanisms, too. This is discussed in side-by-side control.

3.4.2 Side-by-Side Control

In side-by-side control, the swarm is controlled by fully developed mechanisms from both Autonomous and Heteronomous Task Allocation. From local (robot) view, these methods can either be used alternatively or concurrently.

3.4.2.1 Alternative Control

If individual robots use different mechanisms for Task Allocation, the swarm can profit from the side-by-side execution of these strategies.

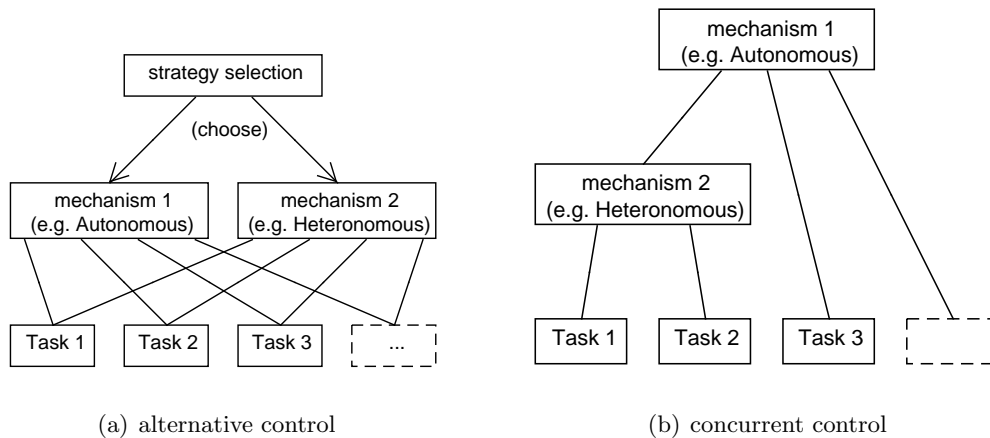


Figure 3.9: (a) Example for alternative side-by-side control: each robot selects one of two alternative approaches for Task Allocation: mechanism 1 or mechanism 2. Both mechanisms are able to allocate all available tasks and exclude each other. (b) Example for concurrent side-by-side control: each robot uses two different approaches in order to solve Task Allocation. If mechanism 1 selects mechanism 2, both mechanisms are active concurrently.

Bees, for example, use the waggle dance to communicate where food can be found. An individual that takes this advice is under heteronomous control, whereas an individual ignoring the signal is autonomous. Assuming that every individual is controlled by one of two alternative strategies, the swarm is divided into two castes: searchers and retrievers. Searchers are self-directed whereas retrievers are directed by others. By a priori choosing an appropriate ratio of the castes, the swarm profits from both strategies. Unfortunately, predefined ratios have to conform the environmental conditions. If, for instance, food is not clustered but dispersed, retrievers are worthless. [DC04]

By allowing each individual to decide whether to follow an order or to ignore it, the swarm may be enabled to manage a proper ratio of both strategies on its own. Such autonomous decisions can also be found in bee colonies: if bees already have knowledge about food sites, they ignore the information from the waggle dance with high probability (93 %) [GBF08].

Figure 3.9(a) shows an example for alternative control: each robot selects one of two alternative mechanisms to solve its complete Task Allocation problem. From global view, different mechanisms are used side-by-side. The selection of mechanisms can either be done off-line or on-line. By live adaptation, the swarm is able to react to changes in the environment. To manage a proper ratio of mechanisms in the swarm, any form of learning or adaptation can be used, e.g. reinforcement learning or the Variable Delta Rule (cf. section 3.3.4.1). For example: Kaminka et al. [KEK09] use Q -learning to select the best coordination strategy in case of robot collision. The reward function, defined by an effectiveness index, maximizes the time between conflicts and reduces both time and resources spent for coordination.

A typical example for alternative control is the incorporation of a fall-back solution. Heteronomous Task Allocation mechanisms, for example, are known to result in better allocations than Autonomous Task Allocations but do not perform well if communication is disturbed. In this case, robots could fall back to another mechanism that is better suited for the present situation.

3.4.2.2 Concurrent Control

In contrast to alternative control, concurrent control allows each robot to use different mechanisms at the same time. Often, those mechanisms are organized in a hierarchical structure and incorporate some additional rules that define precedences.

Figure 3.9(b) shows an example for concurrent control: mechanism 1 is used to decide whether to use mechanism 2 to control task 1 and 2 or to execute some other tasks. This special case demonstrates that the complete Task Allocation problem (for tasks 1, 2, 3, ...) can be broken into parts that can each be approached by different mechanisms.

Concurrent control allows to use mechanisms for a smaller set of tasks. Because mechanisms in Heteronomous Task Allocation often do not scale very well, they seem to be irrelevant for Task Allocation in Swarm Robotics at first sight. However, Heteronomous Task Allocation is very efficient in smaller groups of robots. Thus, it can be used as a sub-routine for parts of the swarm. In foraging, for example, the auction mechanism is better used to locally coordinate a cooperative transportation task than to decide on comparably independent actions. Note that if local communication and leading is possible, there is no reason to constrain design to simple rules that solely rely on emergence to solve complex problems.

Side-by-side control allows to combine fully developed mechanisms for Task Allocation. Instead of reinventing the wheel by the development of complex new strategies, existing approaches can be organized in an alternative or concurrent way. In many cases, the global mission is decomposable into sub-missions that can be targeted by well-engineered allocation strategies. This leads to hierarchical structures that combine various mechanisms for Task Allocation.

3.5 Summary

This chapter has presented different mechanisms that can be used to solve Multi Robot Task Allocation (MRTA) in Swarm Robotics. A taxonomy was proposed that enables the categorization of techniques with respect to the control approach. Heteronomous Task Allocation utilizes a top-down approach by allowing some leader robots to command worker robots. Autonomous Task Allocation permits each individual robot to behave self-directed, which hopefully emerges a desired behavior of the swarm as a whole. Finally, Hybrid Task Allocation combines both autonomous and heteronomous mechanisms.

Heteronomous Task Allocation was further divided into Centralized and Distributed Task Allocation. Centralized approaches feature omniscient control, blackboard control and centralized reinforced control, which all rely on a single leader giving orders. In contrast, distributed approaches allow to dynamically grant the leader role to avoid a single point of failure. The best-known method in Distributed Task Allocation is the market-based approach which basically uses some form of auction to negotiate about who will execute a task.

Although mechanisms in this category generally do not scale very well, heteronomous control remains important for Swarm Robotics. At least when used as a sub-routine for smaller groups of robots, explicit cooperation can be achieved efficiently.

Autonomous Task Allocation was further divided into rule-based, threshold-based, probabilistic and decentralized reinforced control. Rule-based control is the most general category for the implementation of autonomous control. It is very well suited for the definition of complex behavior. This abstraction from basic behaviors enables a robot to use some higher level forms of control.

Threshold-based approaches utilize a special parameter, called threshold, to control the activation of single tasks. The threshold can either be interpreted as a straight line, which is considered by the activation threshold model, or as a parameter that controls the probability to activate a task, which is defined by a, potentially adaptive, threshold function.

In probabilistic control, a robot is controlled by a discrete probability distribution that maps all available tasks to corresponding probability values. Although these chances can be adapted by hand, it is more elegant to adapt abstract motivations that serve as a basis for the calculation of the final probabilities. This kind of probabilistic control was discussed under the designation *Motivation-based Task Allocation*, which features an approach utilizing both thresholds and motivations.

Decentralized reinforced control gives a robot the ability to learn a policy that drives Task Allocation. Although literature focuses on Q -learning for the adaptation to dynamic environments, this thesis proposes to use Sarsa(λ) because it is better suited for continuous on-line learning.

Hybrid Task Allocation was further divided into interlaced and side-by-side control. Interlaced control allows the creation of new methods by merging different approaches whereas side-by-side control utilizes fully developed mechanisms.

This chapter has given a broad overview of different mechanisms for solving Multi Robot Task Allocation (MRTA) in the context of Swarm Robotics. Designers that want to create a new swarm robotic system should first investigate if they want to use a heteronomous, autonomous or hybrid approach.

Heteronomous mechanisms aim at commanding the swarm as a whole and have the potential to produce optimal solutions. On the downside, top-down coordination involves effort in both communication and computation.

Autonomous mechanisms aim at individual self-directed control and have the potential to emerge optimal solutions with a fraction of the expense. On the downside, bottom-up coordination is hard to handle and simple rules that emerge a desired behavior are difficult to find.

Hybrid mechanisms aim at balancing autonomous and heteronomous control and have the potential to tailor strategies that avoid the disadvantages of both approaches while profiting from the advantages. On the downside, this potential may be fictitious because combinations could as well cancel the components' advantages and strengthen their disadvantages. Nevertheless, this thesis encourages to experiment with new mixed approaches.

Figure 3.10 gives an overview of the categories and mechanisms covered by this thesis. The boxes with dotted border indicate that each category can be extended by new mechanisms.

Note that every mechanism is situational. As a result, this thesis does not favor one technique over another. At least by considering hybrid approaches, all presented mechanisms are relevant for Task Allocation in Swarm Robotics.

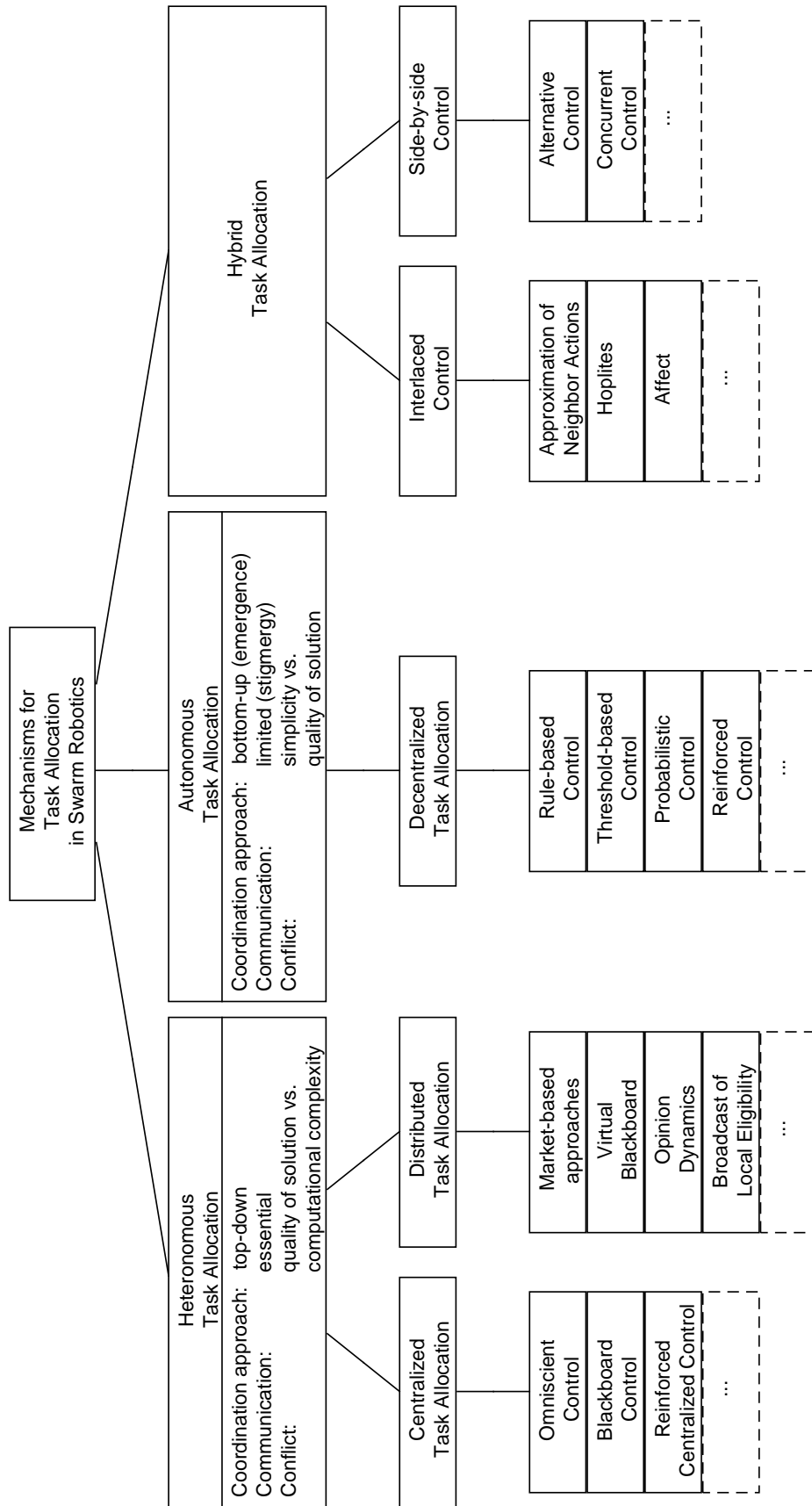


Figure 3.10: Overview of mechanisms for Task Allocation in Swarm Robotics.

Chapter 4

Swarmulator - A Simulator for Swarm Robotics

Simulation is a very important tool in the development of robotic swarms because experiments with physical robots are very time-consuming. By emulating robotic actions in a simulated environment, designers can efficiently test different control approaches and forecast the outcome of experiments with real robots. Simulation does not only save time but also money: physical robots are very valuable, and damage to them should be avoided at all costs.

Within the scope of this thesis, the *Swarmulator*, a simulator for Swarm Robotics, was developed. Basically, the Swarmulator is a platform for the control of multiple simulation runs that carry out experiments in a virtual world. If desired, simulation runs fill csv-files with statistical data. Additionally, the Swarmulator lets the user take a look into running experiments to watch their advancement.

Although the Swarmulator is designed to support this thesis, namely for testing mechanisms for Task Allocation in Swarm Robotics, the underlying framework is suitable for the implementation and execution of any experiment that is based on stepwise computation of a virtual world.

This chapter gives a brief overview of the Swarmulator. To begin with, the basic simulation platform is described. After that, the Swarmulator's applicability for Task Allocation in Swarm Robotics is explained.

4.1 Simulation Platform

This section points out that the Swarmulator is a simulation platform that can be used to run arbitrary experiments. It is based upon the well-known model-view-controller architecture and features a modular design that enables users to import new types of experiments at runtime. Additionally, it supports batch processing by allowing scripted creation and automated execution of experiments.

4.1.1 Architecture

The Swarmulator uses a model-view-controller (MVC) architecture to manage the execution of each experiment. The world serves as a model that can be modified by computing its next simulation step. A simulation thread serves as a controller that executes this stepwise transition of the world. The user is able to follow the process by opening viewports that reflect the world's current state.

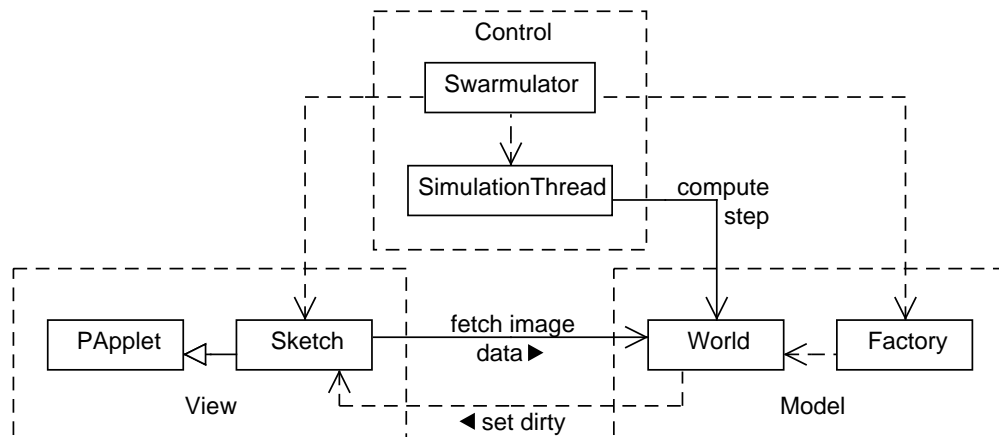


Figure 4.1: Simplified representation of the model-view-controller pattern used in the Swarmulator.

Figure 4.1 shows a simplified version of the MVC pattern in use. Basically, the following steps are processed: The Swarmulator uses a factory to create a world. This world is associated to a thread that computes steps for the world. At user's request, the Swarmulator opens a new view that observes a specific world and automatically fetches image data if the world has changed. For each view, a Processing [PC12] applet is used to draw the attached world.

Figure 4.2 depicts a more accurate UML class diagram of the Swarmulator. The Swarmulator's main frame controls an arbitrary number of simulation runs (although not stated in the diagram, the class `SimulationRun` implements the interface `Runnable`). After creating a simulation run for a simulatable world, which has been created by a world factory, the Swarmulator can start a thread that finally computes steps via the execution of the run method. The processing speed of the experiment is steered by a parameter of the simulation run that can be changed from the main frame or from any simulation sketch (a Processing applet) that visualizes the world. Besides controlling speed, the main frame is additionally able to terminate a simulation run. On the other side, the simulation sketch additionally manages a view that is used to draw a cutting of the included world. This viewport, which can be altered by the user, defines which image data needs to be fetched from the world. For the sake of thread safety, both step computation and image fetching should be implemented via synchronized methods.

Figure 4.3 shows the main frame of the Swarmulator in action. Basically, the frame lists all created simulations and presents their status. In the picture, the first two simulations are running whereas the third one has not been started yet. As indicated by the numbers, the first experiment should be automatically stopped after 20 minutes of the world's internal time. New simulations can be created via the plus-button.

The interfaces `ISimulatedWorld` and `IWorldFactory` already indicate that the Swarmulator features a modular design. This feature is described in the next section.

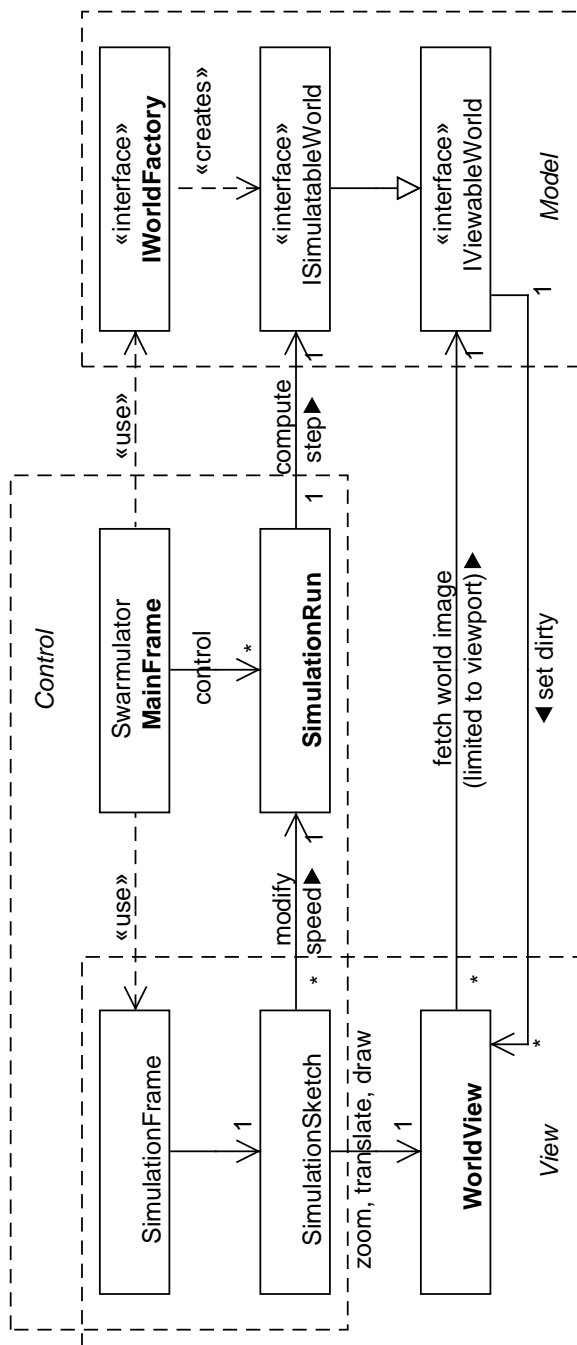


Figure 4.2: UML class diagram of the Swarmulator.

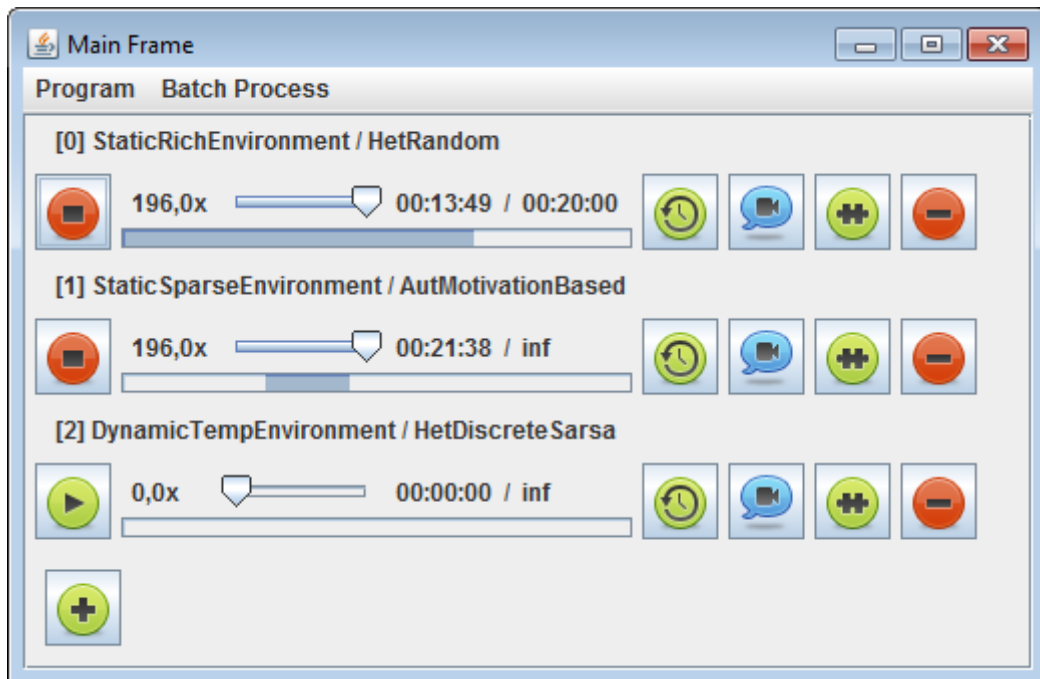


Figure 4.3: Main frame of the Swarmulator in action.

4.1.2 Modular Design

Creating new types of experiments that should be simulated by the Swarmulator is easy. The user only needs to implement two interfaces:

- `ISimulatedWorld` and
- `IWorldFactory`.

Figure 4.4 shows these interfaces and their relation to other classes. Note that experiments are basically defined by the construction of a world model (cf. figure 4.2). By creating a world that implements the interface `ISimulatedWorld` (and hence `IViewableWorld`), the user can specify both the world’s dynamics (via the implementation of the method `computeStep`) and the world’s representation (via some form of image that updates a `WorldView`).

To be able to create a user defined world, the Swarmulator needs a corresponding factory. By implementing the interface `IWorldFactory`, the user can specify which type of world should be created dependent on some given settings. These settings can be derived from a defining textual phrase (cf. figure 4.4). Batch processing, which is further described in section 4.1.3, uses them to create different kinds of worlds in a row.

The Swarmulator features the possibility to import world factories from jar-files. By this, users can distribute their experiment’s definition without having to include the simulation platform itself. As long as the jar-file contains a top-level file “factories.cfg” that names the included world factories, the Swarmulator is capable of locating the corresponding classes. After importing such a jar-file, the user is able to access the given world factories in order to create and execute simulations.

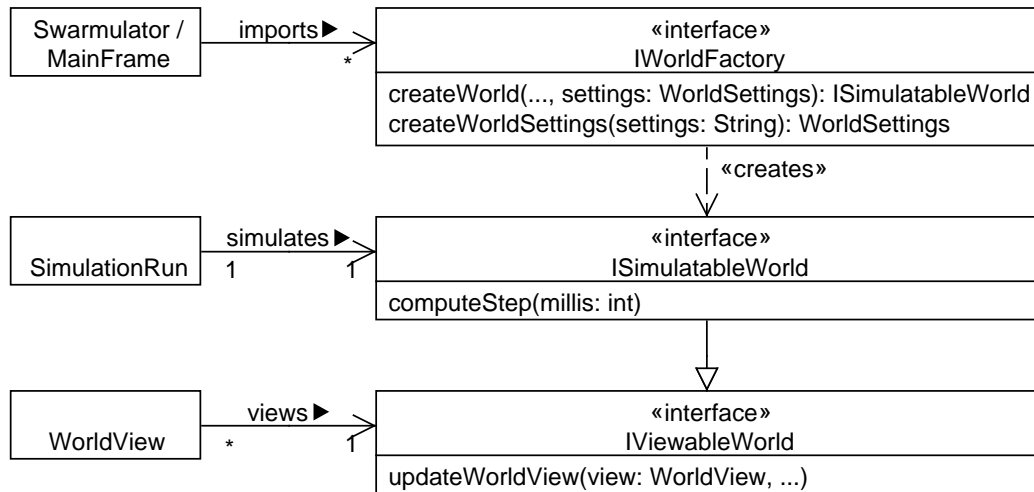


Figure 4.4: Interfaces for the creation of new experiments.

4.1.3 Batch Processing

To ease the creation of simulations, the Swarmulator allows the user to run scripts. These scripts are composed of statements that initialize simulation runs by setting some parameters and by using world factories with specified world settings.

Code 4.1 shows an example for a script's content. First, some settings are defined: the initial random seed value to use, the step length in milliseconds, the stop time, a logging directory and the world settings. The star stands for the default settings of the attribute. For example, a random seed of * means that the initial value is randomly chosen for each simulation created in the following.

As already stated in the last section, worlds are created via a factory that accepts additional settings. In scripts, these world settings can either be set for all following create-statements or defined locally, separated by a ":" from the world factory's class name. In code 4.1, three simulated worlds are created by a factory named "MyWorld-Factory": the first one with setting "variantA", the second one with "variantB" and the third one with "variantA" again. By the way: the Swarmulator will ask the user to import a jar-file if the given factory name is not known.

Additionally to running scripts, the Swarmulator is able to automatically process the created list of simulations. The user can define how many of these experiments should be executed at the same time. If desired, finished simulations can automatically be terminated and removed from the list. Alternatively, the user could keep the simulations for further execution.

The Swarmulator is well suited for the execution of arbitrary experiments. By implementing a simple interface for simulatable worlds, the user is able to define any form of stepwise dynamics that can be processed by the simulation platform. Additionally, image data can be included that represents the world's state and that can be viewed in one or more viewports. Thanks to their modular design, world factories can be imported at runtime. Additionally, batch processing enables the Swarmulator to automatically carry out simulations that are created by a script.

```

1 // settings for world creation
2 randomSeed: *
3 stepLength: 200
4 stopTime: *
5 logDir: *
6 worldSettings: variantA
7
8 // create some simulations
9 create: MyWorldFactory
10 create: MyWorldFactory:variantB
11 create: MyWorldFactory

```

Code 4.1: Example of a script used for batch processing.

Although the Swarmulator is able to carry out any form of experiment, it is primarily designed for the simulation of mechanisms for Task Allocation in Swarm robotics. The next section explains which features were implemented to credit this applicability.

4.2 Applicability for Task Allocation in Swarm Robotics

The Swarmulator does not only provide interfaces for the creation of arbitrary simulatable worlds but also implements a default world that is applicable for the simulation of swarms. Robots are modeled as active components of this world and may be equipped with a task to execute. By defining a task that chooses between other tasks, robots can make decisions regarding Task Allocation. This section gives further information about these concepts.

4.2.1 World of Components

The default world implemented in the Swarmulator is called `UserWorld`. As shown in figure 4.5, it implements the interface needed for simulations and extends a class named `ComponentWorld`. As the name suggests, a `ComponentWorld` contains components. Each time the world is initiated to compute its next step, it delegates the computation to all of its components. By this, the components proceed to the next step, too.

Components are basically defined by a position and an orientation in their home world. Each component is able to update its position and orientation but it has only a limited view to its world (cf. the interface `IComponentWorld` in figure 4.5). In order to sense the positions of other components, some kind of sensor has to be used.

Because a swarm may be composed of a massive amount of individuals, the frequent use of sensors that search for components in a specified area can get computationally expensive. In order to limit the amount of components that have to be checked in a spatial query, the world manages a *quadtree* [Sam84] for each type of sensor, which is a tree data structure that uses nodes with exactly four children (an example is shown in figure 4.6). In this case, quadtrees store components with respect to their two-dimensional location, and allow sensors to find their targets quickly.

In order to ease the logging of experimental data, the world also administrates the access to csv-files. Components can query a desired file writer by name and add their data without having to worry about the concrete logging directory.

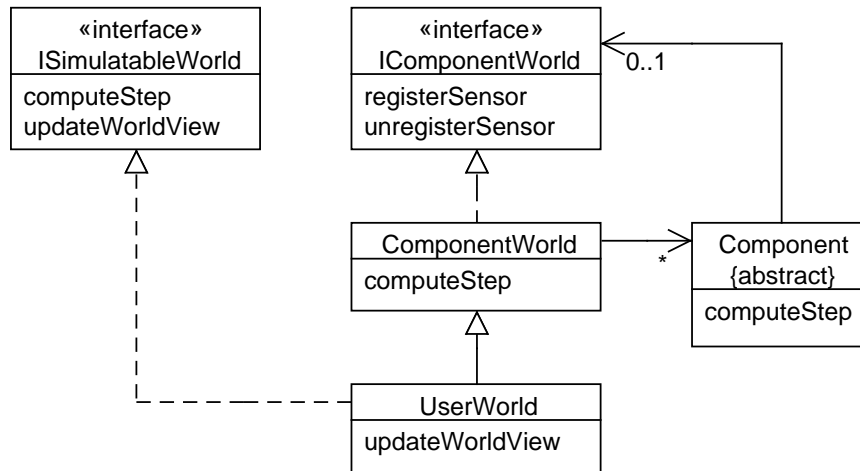


Figure 4.5: UML class diagram for the class *ComponentWorld* and its neighbors (for the sake of simplicity this version omits some methods).

4.2.2 Tasks for Active Components

Active components are components that can be equipped with a task to execute. Each active component, generally a robot, delegates the computation of its next step to the task. In the terms of this thesis, a task represents a set of rules that activates basic behaviors of its owner.

Figure 4.7 shows the relation between an active component and its ACT (active component task). The component delegates the computation of its steps to its ACT whereas the ACT controls the active component. Primarily, when not associated to a component, an ACT is a “blueprint”. As soon as the ACT is set for an active component, a private copy is created and bound to the component.

By default, an ACT cannot manipulate the component, because the accessible interface `IActiveComponent` does not provide corresponding functionality. To enable subclasses of the abstract class `ACT` to efficiently affect the component, new interfaces have to be specified. Copying a blueprint for a component that does not support the desired interface should throw an exception.

For example: the Swarmulator features a task class named `ACTMoveToLocation`. As long as an instance of this ACT is not associated to an active component, it is just a blueprint defining movement to a specific location. After giving the blueprint to a component, which effectively associates a copy of the task, the component moves to the given target. Because `ACTMoveToLocation` needs to control movement, the associated component must implement a predefined interface called `IMobileComponent`. Otherwise, an exception is thrown at assignment.

The Swarmulator features some more ACTs that ease the implementation of new tasks. Figure 4.8 shows two of them: `ACTEndless` takes any task and executes it over and over again, `ACTSequence` takes a list of at least one task and executes one after another. Note that `ACTSequence` terminates when the contained list has been processed.

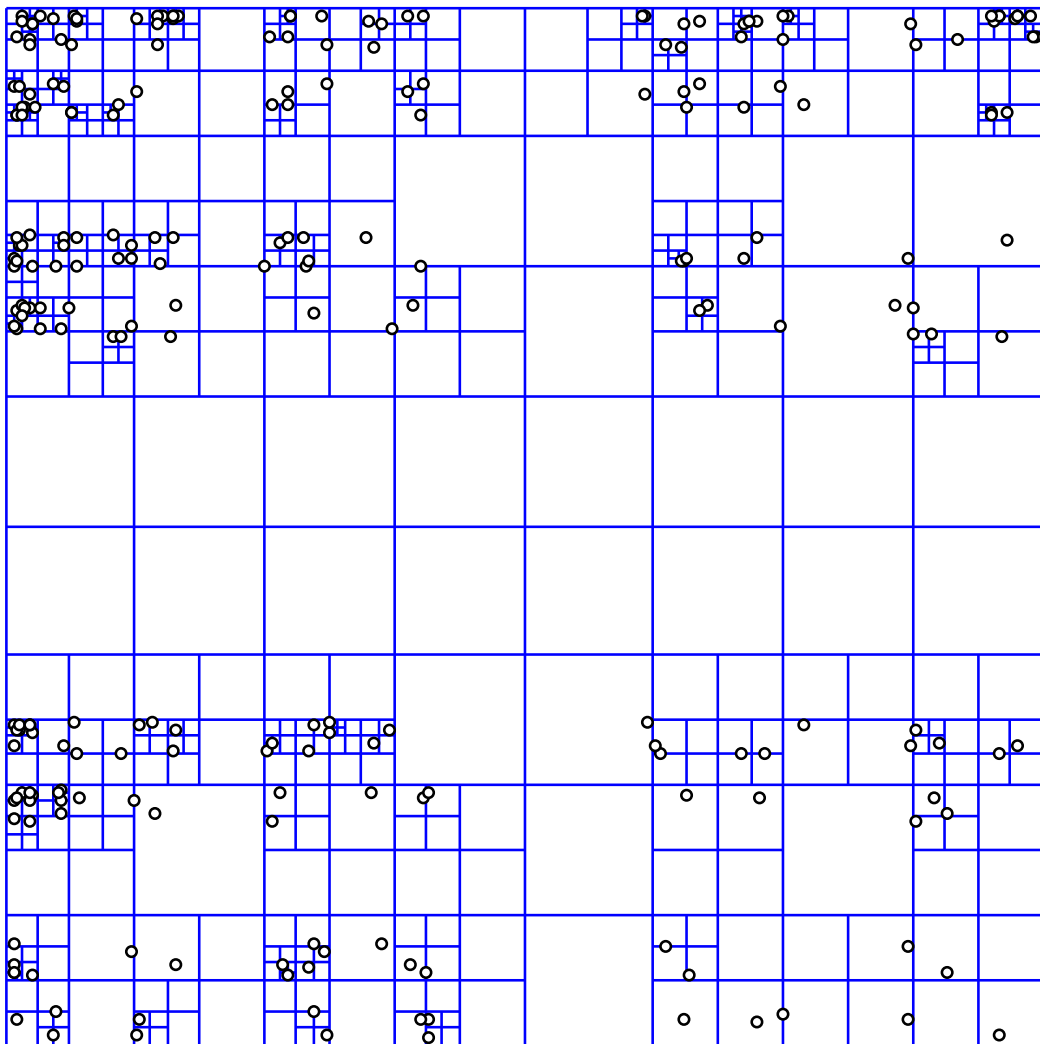


Figure 4.6: Example quadtree for position management: each cell is split into four smaller cells if the number of contained objects exceeds 1. Image is taken from Wikimedia Commons*.

* http://commons.wikimedia.org/wiki/File:Point_quadtrees.svg

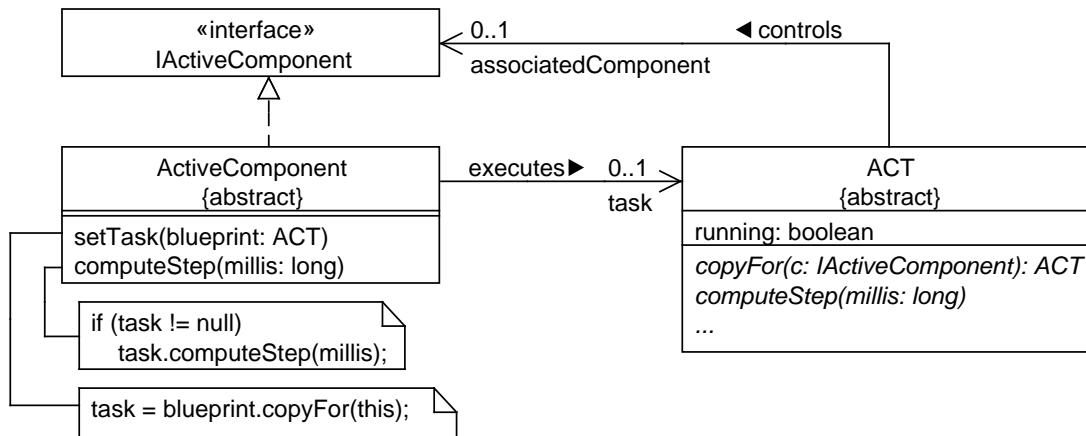


Figure 4.7: UML class diagram for active components and tasks.

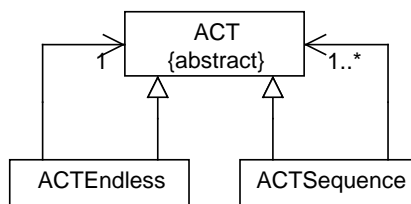


Figure 4.8: UML class diagram for the common active component tasks ACTEndless and ACTSequence.

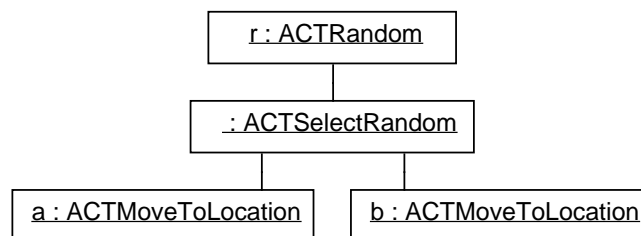


Figure 4.9: Example UML object diagram for an instance of ACTRandom.

This concept of tasks for active components should not be misinterpreted: a component should **neither** be able to change its own task **nor** be able to change the task of another component. The assignment of an ACT, which can be interpreted as the “brain” of the robot, is subject of the world factory. To enable robots to choose from a set of tasks, an ACT for Task Allocation has to be designed.

4.2.3 Tasks for Task Allocation

The Swarmulator uses ACTs to manage Task Allocation. Basically, such an allocation task maintains a list of ACTs that are candidates for delegation. By defining circumstances under which the selected task changes, like after receiving a command message or when a certain sensory input is available, the allocation task can effectively solve the problem of Task Allocation in a reasonable way.

Innately, the Swarmulator offers three different ACTs for Task Allocation: `ACTRandom`, `ACTLinkedMotivation` and `ACTSarsa`. These approaches are described in the following subsections.

4.2.3.1 Random Task Allocation

The task `ACTRandom` is an endless version of the – also implemented – task `ACTSelectRandom`, which randomly chooses from a given list of ACTs. Because `ACTSelectRandom` terminates when the selected task is finished, it needs to be wrapped by the task `ACTEndless` to achieve continuous selection of the given tasks. Figure 4.9 shows an example for an instance of `ACTRandom`: the allocation task `r` delegates to a selection task that randomly chooses to delegate either to the movement tasks `a` or to `b`.

4.2.3.2 Motivation-based Task Allocation

The task `ACTLinkedMotivation` delegates the computation of steps to an ACT that is selected by a chance dependent on the motivation for the task. Motivation values are adapted by a motivation changer that applies changes each time a selected task succeeds or fails.

Figure 4.10 demonstrates the construction of this ACT in a UML class diagram: the task `ACTMotivated` contains a motivation value that is used by `ACTSelectMotivated` to select it with a probability according to the motivation’s weight in comparison to all other available motivated tasks. `ACTLinkedMotivation` uses a `MotivationChanger` to create its selection task and to delegate the adaptation of motivations when a selected task terminates.

4.2.3.3 Reinforced Task Allocation

The task `ACTSarsa` implements $Sarsa(\lambda)$, a temporal-difference learning method that incorporates eligibility traces (cf. section 2.3 about reinforcement learning). The performance of `ACTSarsa` is very much dependent on the specification of its constructor parameters. Most of these parameters are instances of reinforcement learning interfaces provided by the Swarmulator.

Figure 4.11 shows a stylized UML class diagram that gives an overview of `ACTSarsa` and the interfaces that need to be implemented. As demonstrated in pseudo-code 4.2, the reward function is used to derive a reward value from the environment. Subsequently, the current state is observed and an appropriate next action is chosen by using the Q -values as a basis for the given policy. After that, the eligibility for the recent

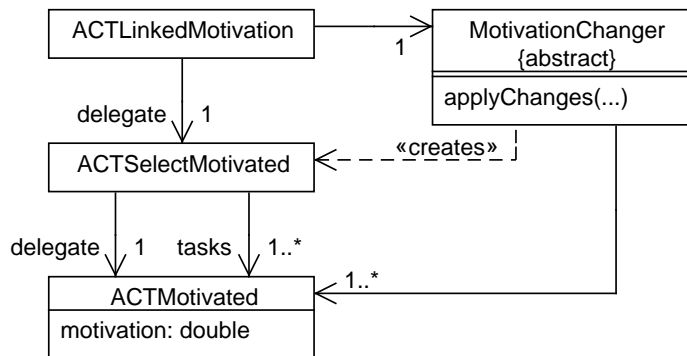


Figure 4.10: UML class diagram for *ACTLinkedMotivation*.

state-action pair is set to 1. Each Q -value then targets the novel expected accumulated reward, which is calculated as the sum of the observed reward and the discounted expected future rewards. The degree of approximation is limited by the step-size parameter α and the current eligibility values. Finally, the eligibilities fade out at a rate dependent on the decay-rate λ and on the discount-rate γ .

Note that this implementation of Sarsa(λ) is very generic. Although the Swarmulator provides default implementations for an ϵ -greedy policy as well as for an action value function and an eligibility function based on hash maps, the users are free to specialize all parameters to their needs. This includes the implementation of function approximation in order to allow states with continuous variables.

In the Swarmulator, Task Allocation is realized by a task itself. By this, different mechanisms can be mixed in a natural way. For instance: assume that a robot has to choose between resting and two foraging approaches. This problem can be split into two independent problems: choose between resting and foraging, and – in foraging – choose between foraging approach 1 and 2. Both allocations can be addressed by a different approach, e.g. by motivation-based and reinforced Task Allocation.

4.3 Summary

The Swarmulator is a graphical simulation platform that can be used to perform any kind of experiment that is based on stepwise execution. Experiments are defined by the implementation of a world factory that constructs virtual environments that are processed by simulating threads. At runtime, these factories can be imported from jar-files and be used for the creation of simulation runs. The Swarmulator is also equipped with simple batch processing capabilities that allow both automated generation and finishing of simulations.

In order to support this thesis, the Swarmulator was developed as a tool for testing mechanisms for Task Allocation in Swarm Robotics. It features a pre-implemented world that consists of a potentially massive amount of components that may use sensors to percept each other. To avoid inefficiency in simulation, the world uses quadtrees to reduce the computational effort when using sensors.

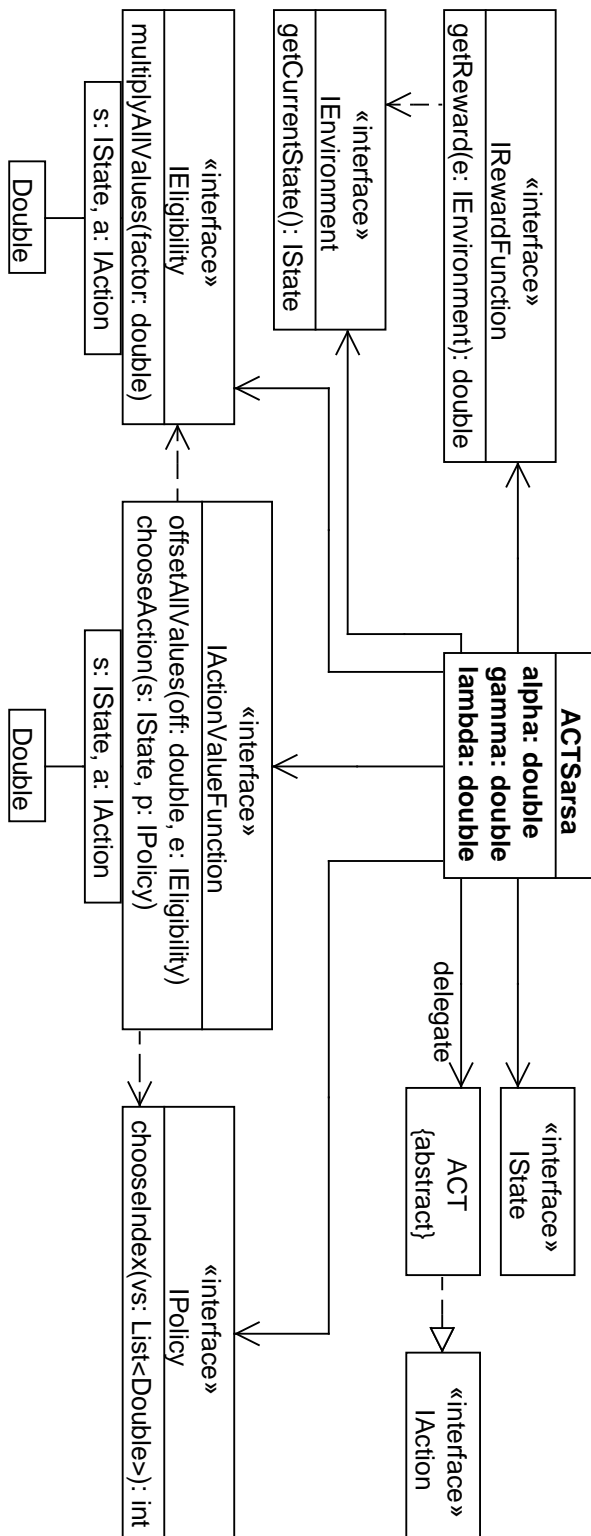


Figure 4.11: Stylized UML diagram for ACTSarsa.

```

1 // Execute when task a, which was started in state s, finished
2 {
3     S lastS = s;
4     ACT lastA = a;
5
6     // observe reward r (that followed action a)
7     double r = rewardFunction.getReward(environment);
8
9     // observe current state s'
10    currentS = environment.getCurrentState();
11    // choose next action a' using policy derived from Q
12    nextA = actionValueFunction.chooseAction(currentS, policy);
13
14    // calculate  $\Delta = r + \gamma * Q(s', a') - Q(s, a)$ 
15    double lastQ = actionValueFunction.getValue(lastS, lastA);
16    double currentQ = actionValueFunction.getValue(currentS,
17        nextA);
18    double  $\Delta = r + \gamma * currentQ - lastQ$ ;
19
20    // set eligibility  $e(s, a) = 1$ 
21    eligibilities.setValue(lastS, lastA, 1);
22
23    // update action values:  $Q(s, a) = Q(s, a) + \alpha * \Delta * e(s, a)$ 
24    actionValueFunction.offsetAllValues( $\alpha * \Delta$ , eligibilities);
25
26    // fade out eligibility values
27    eligibilities.multiplyAllValues( $\gamma * \lambda$ );
28
29    // reset (s, a)
30    s = currentS;
31    a = nextA;
32 }

```

Code 4.2: Stylized implementation of Sarsa(λ) in ACTSarsa. All variables highlighted in red are specified when constructing an instance of ACTSarsa (cf. figure 4.11). This fragment of code is inspired by pseudo-code from the book “Reinforcement Learning: An Introduction” by Sutton and Barto [SB98, p. 181].

Active components – generally robots – can be equipped with a task. This task controls the component, e.g. by the activation of basic behaviors, and it can thus be interpreted as the component’s “brain”. Because the assignment of this “brain” is reserved to the world factory that constructs the component, a component’s task cannot be changed in simulation. Nevertheless, by the assignment of a task that contains other tasks to delegate control to, an active component is able to switch between tasks on its own. Note that this does not mean that Autonomous Task Allocation is used: the selection of tasks can still be controlled by signals received from other components, which would make the underlying mechanism a heteronomous one.

This chapter presented the Swarmulator, a simulation platform that was implemented in the context of this thesis. The next chapter uses this platform to implement and compare different approaches for solving Task Allocation in a concrete foraging scenario.

Chapter 5

Experiments

This chapter describes how the Swarmulator is used to experiment with different approaches to solve Task Allocation in a concrete scenario. Basically, the scenery is inspired by nature: like a swarm of bees or ants, the robotic swarm has to collect food items while concurrently managing an adequate nest temperature. This additional temperature aspect makes the scenario more complex than other foraging missions in literature.

First, the scenery is described in detail. This includes the definition of the experimental world and its components, the description of the global mission and the derivation of tasks that can be allocated. Second, some approaches are presented that are assumed to solve the Task Allocation problem. Finally, some of these approaches are tested and compared in static and dynamic environments.

5.1 Scenery

Before Task Allocation approaches can be defined, the scenery has to be set. The experimental world contains at least one nest that is the home of a robotic swarm and target location for foraging. As long as its temperature is hold at an adequate level, the nest is able to process food to energy. Food items emerge from the environment at different rates and at different positions.

A swarm consists of five robots that have to choose from four different kinds of tasks: *forage*, *heat up*, *cool down* and *rest*. All actions – except *rest* – consume energy. Because food items are the only source of energy in the environment, *forage* is the only task that may have a positive energy balance – but only if it is executed successfully *and* if the nest’s temperature is at an adequate level.

First of all, the world and its components are described. This includes the definition of properties for both the environment and the robots. After that, the global mission is specified, which gives detailed information about when the swarm is successful, and when it is not. Finally, a concrete set of complex tasks is defined that serves as a basis for the Task Allocation problem.

5.1.1 World and Components

This section describes the properties of the world and its components. The experimental world features one type of robot, which is realized by an active component. All other components and the world’s properties are defined as the swarm’s environment.

5.1.1.1 Environment

The environment consists of two basic component types: nests and food items. The position of the nest is fixed but the position and frequency of food appearance may change over time. Additionally, the environment features physical temperatures that influence each other.

The world's temperature is defined by an aerial temperature that influences the temperature of all covered components. Components that are cooler will receive thermal energy and heat up, whereas components that are hotter will lose thermal energy and cool down.

Note that the aerial temperature cannot be influenced by the world's components. The air just serves as a constant source or sink for thermal energy that drives components to the world's temperature. Nevertheless, the aerial temperature may change according to the world's dynamics. Like in reality, the temperature may, for example, follow a day/night cycle.

The nest is the most important passive component of the environment. It has a temperature that is basically influenced by the world's temperature but may be influenced by other components' temperatures, too.

The optimal temperature of the nest amounts 20°C . In a range of $\pm 2^{\circ}\text{C}$, the nest is able to assimilate food items to increase its internal energy. The speed of assimilation depends on the offset from the optimal temperature: the lower the offset, the higher the speed. In the world's view, the process of food absorption is illustrated by attraction to the center of the nest. When the food item reaches the center, it fades away and counts as assimilated.

Each time a food item is assimilated, the nest's energy is increased by 100 J. Below 18°C and above 22°C , the nest is unable to process food into energy. Food that is dropped in the nest but not attracted to the center will dissolve after some time.

Food items appear in special areas that deploy prey at a fixed rate. The position of appearance is random but limited to the covered space. Food items that are not collected by robots dissolve after some time.

This thesis works with two types of food areas: a large belt-like area around the nest and a relatively small circle that is equally sized as the nest. The belt represents a rich environment where food can be found easily. In contrast, the circle is used in sparse environments that limit the appearance of food to precise positions.

Like the world temperature, the density of food may be influenced by the world's dynamics. This can be done by rearrangement of the food sources. The environment might, for example, switch between rich and sparse food distribution by replacing a present food belt with a small circle.

After the definition of the environment, the swarm's robots and their abilities can be described.

5.1.1.2 Robots of the Swarm

For the sake of simplicity, this thesis works with a homogeneous swarm of robots. Although all robots have the same capabilities, concrete Task Allocation approaches may limit themselves to a subset of abilities. To ease the implementation, collisions are

not considered. As a result, collision avoidance behavior can be omitted. Furthermore, the communication between robots is limited to light signals that can be sensed in a short range.

To be able to forage, each robot needs some form of movement and food retrieval behavior. Additionally, it must be able to manipulate its own temperature in order to influence the nest's temperature. Furthermore, because the tasks *forage*, *heat up* and *cool down* need energy, robots have to be equipped with a personal battery. As a result, each robot is capable of the following basic behaviors:

- **Head to location:** The robot autonomously moves to the given location in the world. If no target is given, the robot stops. Movement costs some energy and is internally implemented by temporal limited speed, acceleration, braking and rotation.
- **Take food item:** This behavior removes a food item from the world and puts it in the robot's inventory. Taking a food item fails if there is no item in range or the robot already carries one. Prey in the inventory cannot dissolve.
- **Drop food item:** If the robot carries prey, it drops the food item at its current position. Dropped food dissolves after some time.
- **Recharge:** Recharging the robot's battery drains energy from the energy balance of the nest. If the robot is outside the nest, recharge will fail.
- **Heat up / cool down:** By draining energy from the battery, the robot is able to increase or decrease its thermal energy at a limited rate. If the battery is empty, thermal energy, and thus temperature, cannot be influenced anymore.
- **Set light color:** The robot is able to set the color of its light. This enables the robots to communicate via simple light signals.

A robot is always able to head back to the nest because it knows the nest's location. In a real life experiment, this could be implemented by heading to a specific light source. Additionally to the nest's location, each robot is equipped with the following sensors:

- **Food sensor:** Senses food items in a fixed radius around the robot. Returns a list of positions.
- **Light sensor:** Senses light signals of near other robots. Returns a list of sensed colors.
- **Nest temperature sensor:** Retrieves the nest's temperature if the robot is situated in the nest.
- **Nest energy sensor:** Retrieves the current energy level of the nest if the robot is situated accordingly.

In principle, these abilities allow the robot to search food, retrieve it to the nest and manage the nest's temperature. The next section concretizes in which circumstances the swarm's behavior can be called successful.

5.1.2 Global Mission

The global mission of the swarm is to gather food items from the environment, retrieve them to the nest and maintain the nest's temperature to ensure that prey can be processed to energy.

In fact, this formulation is not precise enough to create optimal Task Allocation approaches because it does not specify if nest temperature has to be maintained all the time or only if the environment contains enough food to justify the effort. This issue is clarified by the following specification:

The global mission of the swarm is to maximize the energy level of the nest.

This formulation includes that it is counterproductive to heat up or cool down if there is not enough food available. Because the swarm does not know if the environment is rich or sparse, it always has to spend some energy for exploration.

A swarm is said to perform better than another one if it is able to achieve a higher energy level in the long run. Note that simulation is not able to prove if one swarm beats the other in the limit. Often, it is hard to decide if the algorithm is unable to emerge a better allocation scheme or if emergence just takes more time.

Swarms that perform bad over a long period of time can be accounted impractical. Especially when the nest's energy balance drops so much that the deficit could never be compensated by a higher starting energy, the swarm's approach can be considered as useless, at least in the tested scenario.

5.1.3 Tasks

In order to ease the definition of comparable Task Allocation approaches, the following subsections define tasks that abstract from the basic behavior set of the robots. For the sake of simplicity, all tasks begin in the nest, end in the nest and are finished by a complete recharge of the robot's battery.

5.1.3.1 Basic Tasks

The basic task set defines behaviors that are essential for the accomplishment of the global mission. None of these tasks can be removed without risking swarm failure in typical environments.

- **Forage** is a complex behavior that performs a random walk for finding food items in the environment. The task succeeds if a food item can successfully be retrieved to the nest. If the robot moves too far away from the nest, it aborts its search for food and returns to the nest. In this case, foraging fails. In a rich environment where food is near to the nest, foraging takes about 10-20s of simulation time and consumes about 2 J of energy.
- **HeatUp** uses the corresponding basic behavior for a fixed amount of time (10s). In comparison to foraging, heating costs more energy (5 J).
- **CoolDown** is implemented analogous to **HeatUp**.
- **Rest** is a very simple task that does not activate any behavior for a short period of time (5 s).

Table 5.1 gives an overview of execution times and energy consumption. The amount of energy used in foraging is bounded by the robot's battery capacity, which is 15 J. Remember that a successfully assimilated food item increases the nest's energy balance by 100 J.

Basic task	Execution time	Consumed energy
Forage	> 10 s	1-15 J
HeatUp	10 s	5 J
CoolDown	10 s	5 J
Rest	5 s	-

Table 5.1: Overview of execution times and consumed energy of basic tasks.

5.1.3.2 Additional Tasks

To improve Task Allocation, some mechanisms might need additional tasks to execute. These tasks are optional and their use is limited to appropriate approaches.

- **SetColor:** `SetColor` wraps the corresponding basic behavior. This task needs only one step of simulation to succeed. It is used for simple communications via the robot’s light signal.
- **OrientN/E/S/W:** Each of these four tasks orients the robot towards a different compass point. The inclusion of rotation tasks could improve the swarm’s efficiency, especially in scenarios where food can only be found in specific directions.

These tasks serve as examples for tasks that are not necessary for mission accomplishment but contribute to the design of more efficient solutions. In fact, the basic task `Forage` performs poorly and could be implemented in a much better way. Because this thesis sets a focus on solving Task Allocation and not on optimizing foraging, `Forage` remains untouched.

5.2 Approaches

This section presents different approaches to solve the Task Allocation problem in the given foraging scenario. According to the taxonomy presented in chapter 3, the approaches are classified by the underlying mechanism, which is either heteronomous, autonomous or hybrid.

Within the scope of this thesis it is not possible to test all approaches that seem to be promising. Instead, a focus is set on simple adaptive approaches and on reinforcement learning. Nevertheless, this section sketches other approaches, too.

5.2.1 Heteronomous Task Allocation

Heteronomous Task Allocation uses communication in order to negotiate about the allocation of tasks. Although it is possible to implement arbitrarily complex communication protocols that are solely based on light signals, approaches that need such protocols should be excluded from consideration. In the context of this thesis, only very simple forms of communication are discussed further.

According to section 3.2, Heteronomous Task Allocation relies on decision making by some form of leader and splits into Centralized and Distributed Task Allocation. These categories are investigated in the following subsections.

5.2.1.1 Centralized Task Allocation

In Centralized Task Allocation, only one single leader exists that controls the whole swarm. Like a queen in a bee colony, this controller plays a superior role for the swarm's success. Because all tasks are finished inside the nest, the leader can reside there. All other robots are directed by light signals of the queen-like entity.

Unfortunately, the leading robot is not equipped with all-embracing sensors. Therefore, *omniscient control* is not possible. Although the leader could manage some form of blackboard that contains information gathered by the working robots, *blackboard control* is excluded from consideration because the needed signalling from worker to leader complicates communication.

Communication is limited to one direction: from the central leader to all near workers. Basically, four light signals are needed, each allocating one of the basic tasks *Forage*, *HeatUp*, *CoolDown* and *Rest*.

Centralized Reinforced Control. This thesis proposes to use on-line reinforcement learning to control the central leader. The queen-like entity learns to command the swarm by trial and error in light switching whereas the workers are hard-wired and simply activate the task that corresponds to the sensed light signal.

As presented in section 4.2.3.3, the Swarmulator features the task *ACTSarsa*, which is an implementation of $Sarsa(\lambda)$, a temporal difference learning control algorithm using eligibility traces (cf. section 2.3). $Sarsa(\lambda)$ is an on-policy control method that is expected to have a better on-line performance than off-policy methods like Q -learning, which is the most common method used in literature.

In order to effectively use *ACTSarsa*, the underlying elements of reinforcement learning and some parameters – the step-size parameter α , the discount-rate γ and the decay-rate λ – have to be specified. This includes the definition of actions and states, the specification of an action-value function and an eligibility function, the construction of a reward function and the selection of a policy.

Because the states are dependent on the nest's temperature, which is a continuous value, this thesis implements two different variants of *ACTSarsa*:

1. *ACTDiscreteSarsa*: This approach uses a discrete set of five states for learning: in a range of 0.4°C from the nest's optimal temperature, the state is considered *OK*; from there up to an offset of 2°C , the state is named *LOW* and *HIGH* respectively; higher offsets from the optimal temperature deactivate the assimilation of food and are therefore called *TOO-LOW* and *TOO-HIGH* respectively. As a result, the action-value function and the eligibility function can simply be implemented in a tabular form, storing a value for each possible state-action pair.
2. *ACTKernelSarsa*: This approach uses function approximation to face the continuous temperature. Although the action-value function is implemented in a tabular form (defined by a grid size of 0.4°C), the interpretation of these values is different than in the discrete approach. To get the actual Q -value $Q(s, a)$ for action a in state s (with temperature t), the values of all table cells for action a and temperatures in range $[t - 1.2, t + 1.2]$ are accumulated with respect to weights that are defined by the area under an Epanechnikov kernel that virtually spans the temperature range (cf. figure 5.1). In order to adapt the Q -values correctly, the eligibility function is defined in a way that spreads an eligibility value of 1 to an according raster of temperature-states in range $[t - 1.2, t + 1.2]$.

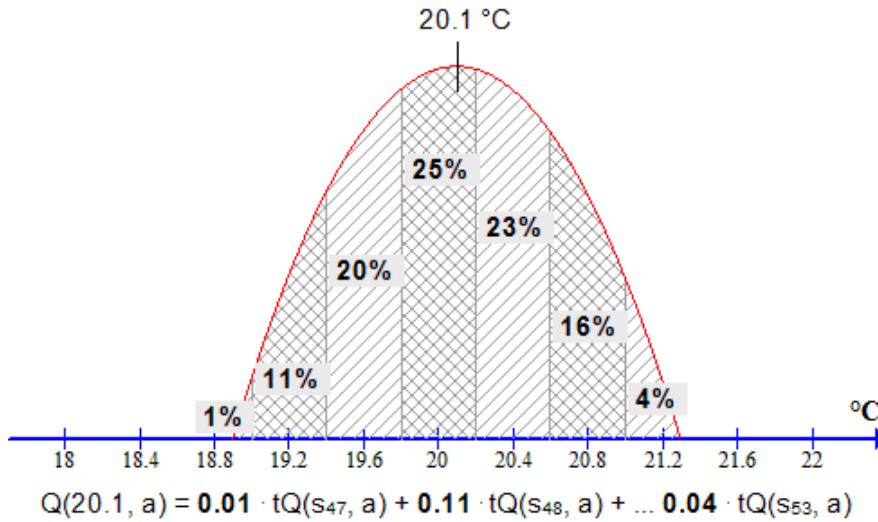


Figure 5.1: Exemplary utilization of the Epanechnikov kernel in *ACTKernelSarsa*: the actual Q -value for selecting action a at 20.1°C is calculated from the values tQ that are saved in a raster of 0.4°C . The state s_{47} corresponds to a temperature in range $[18.6, 19.0)$, s_{48} to $[19.0, 19.4)$, and so on.

Table 5.2 summarizes all reinforcement parameters that are commonly used in both the discrete and the kernel approach. In both cases, the action set is discrete and contains four commanding tasks, each setting a task-specific color for 10 seconds. After finishing one of the actions, the observed reward is constructed from the perceived sensory data. Because the global mission solely claims to optimize the nest’s energy balance, the reward could simply be the increase of the nest’s energy during the execution of the last task. Unfortunately, this will most likely inhibit learning at very high or very low nest temperature: as long as the nest is unable to consume food, there would be no positive reward at all. To prevent this dead end, where it is impossible to say whether heating or cooling is better, the reward incorporates the change towards the optimum nest temperature. In other words: the decision maker is rewarded by both energy accumulation and temperature improvement. For action selection, both approaches use an ϵ -greedy policy. All parameters are set to values that seem to be promising for reasonable learning.

5.2.1.2 Distributed Task Allocation

Distributed Task Allocation, as presented in section 3.2.3, features approaches where the leader role is not fixed. The most common mechanisms in this category use market-based techniques, especially auctions.

Market-based approaches are very well suited for scenarios where robots should negotiate about who should execute a task. In the given scenario, where the same tasks have to be accomplished over and over again and where the fitness of each robot does not play a superior role, market-based approaches do not seem to be very promising. Anyway, the communication protocols needed for the implementation of auctions or similar negotiation procedures are difficult to implement via simple broadcasting of light signals.

Element	Implementation	Description
Actions	action name	execution time (color)
	<code>CommandForage</code>	10 s (green)
	<code>CommandHeatUp</code>	10 s (red)
	<code>CommandCoolDown</code>	10 s (blue)
	<code>CommandRest</code>	10 s (gray)
Reward function	$r = (\text{increase of energy}) +$ $(\text{decrease of failure from } 20^\circ\text{C})$ $r = (e_{now} - e_{last}) +$ $(t_{last} - 20.0 - t_{now} - 20.0)$	e_{now} = current nest energy e_{last} = last nest energy t_{now} = current nest temperature t_{last} = last nest temperature
Policy	ϵ -greedy with $\epsilon = 0.1$	selects the greedy action at 90% and a random other action at 10% of the time
α	$\alpha = 0.1$	step-size parameter, limits adaptation to target Q -value (supported by recently observed reward)
γ	$\gamma = 0.9$	discount-rate, limits the amount of future reward incorporated in Q -values
λ	$\lambda = 0.5$	decay-rate, limits the length of eligibility traces

Table 5.2: Overview of used elements and parameters in both the discrete Sarsa and the kernel Sarsa approach.

Virtual blackboards do not necessarily need a sophisticated communication protocol. Robots could simply indicate their state by emitting a light signal. By the accumulation of state signals, robots get an idea of what their neighbors are doing which enables them to make better decisions. If, for instance, a robot senses that some other robots recently started to cool down the nest, the robot might refrain from doing the same.

Although virtual blackboards are promising for information accumulation, the action selection rules that are based on the blackboard information have to be designed carefully.

This thesis does not follow up Distributed Task Allocation in the given foraging scenario. Autonomous mechanisms that do not need any form of communication are much more interesting for further investigation.

5.2.2 Autonomous Task Allocation

Autonomous Task Allocation relies on the emergence of complex swarm behavior from individuals' actions. Each robot decides on its own what to do next, solely based on its sensory data. Explicit communication is not used.

In the context of the given foraging mission, complex behavior of the swarm could include intelligent division of labor. If one robot is sufficient to maintain the nest's temperature, then it may be the best solution to let one single robot do this job while all other robots try to forage.

5.2.2.1 Rule-based Control

As already discussed, rule-based control is a very vague approach that can be seen as a superclass for all possible mechanisms. As a result, this category name should only be used if rules are defined from scratch without using any approach that automatically defines a set of rules.

In fact, rule-based control is already in use: the task **Forage** coordinates movement and food grabbing / dropping with simple rules that use the food sensor and the location of the home nest.

5.2.2.2 Threshold-based Control

Threshold-based control uses some sort of stimulus that is raised when a task demands execution. As soon as the stimulus passes a given threshold, the corresponding task is activated. A threshold-based approach for solving the given Task Allocation problem could be defined as follows.

Both tasks **HeatUp** and **CoolDown** call for activation at a specified temperature. Heating should occur when the nest's temperature gets critically low whereas cooling should be activated when temperature gets too high. By dispersing the actual activation thresholds, the robots can be prevented from activating the same action all at the same time. Alternatively, a sigmoid curve can be used to define the activation probability dependent on the stimulus. As described in section 3.3.3.2, the threshold then determines which stimulus is needed to get a chance of 50 % or higher.

The tasks **Forage** and **Rest** are less intuitive to be activated by a threshold. One way could be the definition of a stimulus that is incremented each time foraging is not achieved. A corresponding threshold would then represent the amount of time that needs to elapse before foraging is activated the next time. This threshold should, of course, be adapted dependent on the success in foraging. If foraging succeeds, the threshold can be decremented, if foraging fails, the threshold should be incremented. This amount of decrement / increment could be dependent on the number of successes / fails. This is already very much the same as the Variable Delta Rule proposed by Labella et al. [LDD04a, LDD04b, Lab07], except that they adapt a probability value instead of a waiting time.

Up to this point, three different stimuli have been defined: one activating **HeatUp**, one activating **CoolDown** and one activating **Forage**. **Rest** is set as the default task and thus activated if no other task demands execution. Because more than one task could demand execution, a decision rule has to be defined, for example by prioritizing temperature maintenance over foraging. Alternatively, the task to execute could be selected randomly.

Although this threshold-based approach seems to be promising, it is not included into the experiments in this thesis because a similar but simpler approach based on motivations should be tested instead.

5.2.2.3 Probabilistic Control

Probabilistic control selects tasks based on a given probability distribution. The simplest way of achieving such a distribution is by defining a motivation value for each task that represents the weight of the task. Each task's probability is then defined as the fraction of overall motivation that demands the execution of the task.

Finished Task	Result	Adaptation of Motivation	
Forage	success	Forage	$m = m + 0.4$
		Rest	$m = m - 0.4$
	failure	Forage	$m = m - 0.05$
		Rest	$m = m + 0.05$
any	any	HeatUp	$m = -(nestTemp - 20)/2$
		CoolDown	$m = (nestTemp - 20)/2$

Table 5.3: Adaptation in motivation-based approach.

This thesis follows the motivation-based approach and defines simple adaptation rules that should be sufficient to control the swarm in different environments. Each motivation is initially set to 0.5 and may vary in the range $[0, 1]$. The only exception from this rule is the task **Forage** which at least has got a motivation value of 0.1. Motivation values are updated according to table 5.3. Note that success in foraging is rewarded with much more motivation than fails in foraging are punished. This results from the fact that one single food item returns much more energy than a single foraging run costs.

Opposite to the threshold-based approach described in the last section, the motivation-based approach does not need to define an additional selection rule if multiple task are demanded equally. The motivation values are sufficient to describe the chance of each task. At least in this context, the motivation-based approach can be called simpler than the threshold-based one.

Note that both the threshold-based and the motivation-based approach have a common disadvantage: they always maintain the nest’s temperature even if there is no food at all. This problem could be addressed by using the food sensor to sense if there is food that demands the nest temperature to be maintained. Due to the increase in complexity, this modification should only be considered if it is really needed in the actual experiments.

5.2.2.4 Decentralized Reinforced Control

Decentralized Reinforced control is very similar to its centralized counterpart (cf. section 5.2.1.1). In centralized reinforced control, a central leader learned the allocation of commanding tasks and the workers were hard-wired. In this section, no central leaders exists and each worker has to learn itself what task to execute under which circumstances.

In order to learn the selection of tasks, each worker is equipped with an implementation of the task **ACTSarsa**. Like in the centralized counterpart, two possible variants are available:

1. **ACTDiscreteSarsa**: A discrete set of five states is used for learning: *TOO-LOW*, *LOW*, *OK*, *HIGH* and *TOO-HIGH*.
2. **ACTKernelSarsa**: Each temperature value has a unique state. Nevertheless, Q -values are stored in a tabular form, using a grid size of 0.4°C . To get the actual Q -value for an arbitrary state, all values on the raster in a range of three cells are combined using fractions of an Epanechnikov kernel that is centered at the given state’s temperature.

The autonomous version copies most of the values used in the centralized learning approach (cf. table 5.2). Of course, the action set of commanding tasks is replaced by the tasks **Forage**, **HeatUp**, **CoolDown** and **Rest**. Because these tasks do not have equal execution times, the reward function has to be normalized. This is done by dividing the reward value by the duration of the last executed task.

Autonomous learning may be better than heteronomous learning but it doesn't have to. On the one hand, the autonomous version has the potential to learn more efficient solutions by utilizing each worker individually. On the other hand, workers are frequently rewarded for the actions of other robots, which slows down the learning process due to misinterpreted rewards.

5.2.3 Hybrid Task Allocation

Hybrid Task Allocation combines mechanisms from both Autonomous and Heteronomous Task Allocation. This section presents exemplary approaches that build upon the approaches presented in the last sections.

5.2.3.1 Interlaced Control

Interlaced control mixes autonomous and heteronomous methods to create novel approaches. An example for such an approach would be the deployment of workers that command foraging if they retrieved a food item successfully. Other workers are forced to follow such a command but decide on their own what to do next if no commanding signal is observed.

5.2.3.2 Side-by-side Control

Side-by-side control combines fully developed mechanisms to form hybrid approaches. This category can be split into alternative and concurrent control.

Alternative Control. In alternative control, individuals choose from a set of approaches and follow one of them. This selection can be done in advance by deploying a heterogeneous swarm. Alternatively, the robots may use a learning method to find the best mixture of approaches on their own.

This thesis follows the simpler approach, deploying a heterogeneous swarm that consists of two robots using the motivation-based approach presented in section 5.2.2.3 and three robots commanded by a queen-like entity that learns commanding by following kernel Sarsa, as described in section 5.2.1.1.

Concurrent Control. In concurrent control, individuals follow multiple fully developed mechanisms at the same time. An example for such an approach is the use of centralized reinforced control where a leader learns to command **HeatUp**, **CoolDown** and **AnythingElse**. If the workers observe the latter signal, they decide on their own whether to execute **Forage** or **Rest**. This autonomous decision could be made by a motivation-based approach.

This section presented different approaches to face the Task Allocation problem in the given foraging scenario. The next sections simulate and compare a selection of them in static and dynamic scenarios.

Approach Name		Category	Deployed Robots	Description
hR	Heteronomous Random	-	1 leader and 5 workers	leader randomly commands; workers obey
hDS	Heteronomous Discrete Sarsa	centralized reinforced control	1 leader and 5 workers	leader learns commanding based on five discrete nest states; workers obey
hKS	Heteronomous Kernel Sarsa	centralized reinforced control	1 leader and 5 workers	leader learns commanding based on continuous nest states; workers obey
aR	Autonomous Random	-	5 workers	each worker randomly selects tasks
aM	Autonomous Motivation	probabilistic control	5 workers	each worker follows its motivations for task selection, motivations are adapted
aDS	Autonomous Discrete Sarsa	decentralized reinforced control	5 workers	each worker learns the selection of tasks based on five discrete nest states
aKS	Autonomous Kernel Sarsa	decentralized reinforced control	5 workers	each worker learns the selection of tasks based on continuous nest states
hyb	Hybrid (hKS + aM)	alternative control	1 leader and 5 workers	the leader and 3 workers follow Heteronomous Kernel Sarsa; 2 workers follow Autonomous Motivation

Table 5.4: Summary of approaches used in the experiments.

5.3 Simulation and Comparison

This section deploys swarms in static and dynamic scenarios. Both scenario types are needed to test whether the used approaches to the Task Allocation problem can handle different environmental settings or not.

For simulation, the Swarmulator is used (cf. chapter 4). A world factory is implemented that is able to create a virtual world for each experimental configuration. Both scenario and Task Allocation approach are defined by the world's settings. Through the course of each simulation, data is written to csv-files. For evaluation of this data, the statistical software R [R D11] is used.

Because testing all approaches that are sketched in section 5.2 would go beyond the scope of this thesis, a focus is set on the heteronomous and autonomous reinforcement learning approaches and on the autonomous motivation-based approach. For proof of concept, a hybrid version composed of centralized reinforced control (using kernel Sarsa) and motivation-based control is also considered. For comparison with some kind of worst case strategy, random approaches are added, too. As shown in table 5.4, this results in a total of eight different approaches to simulate and compare.

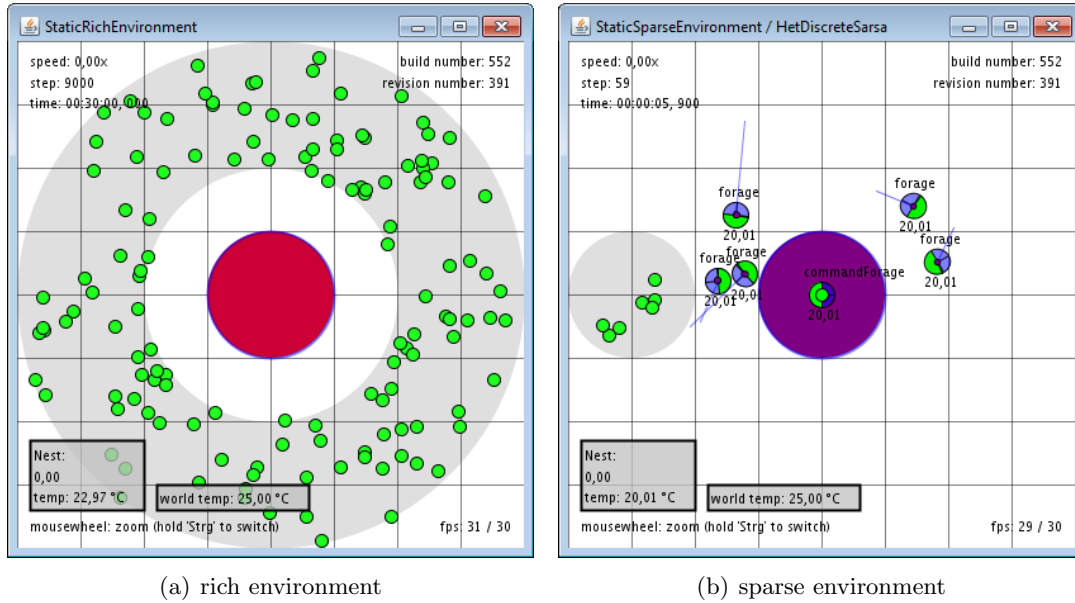


Figure 5.2: *Static scenarios. In (b) a heteronomous swarm is deployed.*

5.3.1 Static Scenarios

In static scenarios, the world's temperature and the density of food is defined by constant parameters. Two different environmental settings are investigated: a rich environment, which offers enough food that all robots can forage successfully, and a sparse environment, which emphasizes the need to **Rest** and which barely provides enough food to justify maintenance of the nest's temperature.

Figure 5.2 shows both static scenarios. In each scenario, the world's temperature is set to constant 25 °C. Initially, the nest's temperature is set to 20 °C. After 30 minutes of simulated time, the nest's temperature has already exceeded the critical value of 22 °C (cf. figure 5.2(a)). As a result, a deployed swarm has to adapt quickly in order to sustain the assimilation of food.

5.3.1.1 Rich Environment

The rich environment provides food in a belt around the nest. Basically, the deployed swarm should learn to forage most of the time and to cool down the nest if needed.

For each approach, 10 experiments are carried out for 24 hours of simulated time. The boxplots in figure 5.3 give an overview of the nest's energy balances that result from the simulations. This serves as a first impression of how good each approach performs. The following conclusions can already be made:

1. As expected, the random approaches perform worst.
2. The motivation-based approach has the best performance, closely followed by the hybrid approach.
3. All reinforcement learning approaches have a very high fluctuation in performance. Some experiments perform very well whereas others perform very bad. As indicated by the thick line, which represents the median, the use of Autonomous Kernel Sarsa even resulted in a negative energy balance in at least half of its simulations.

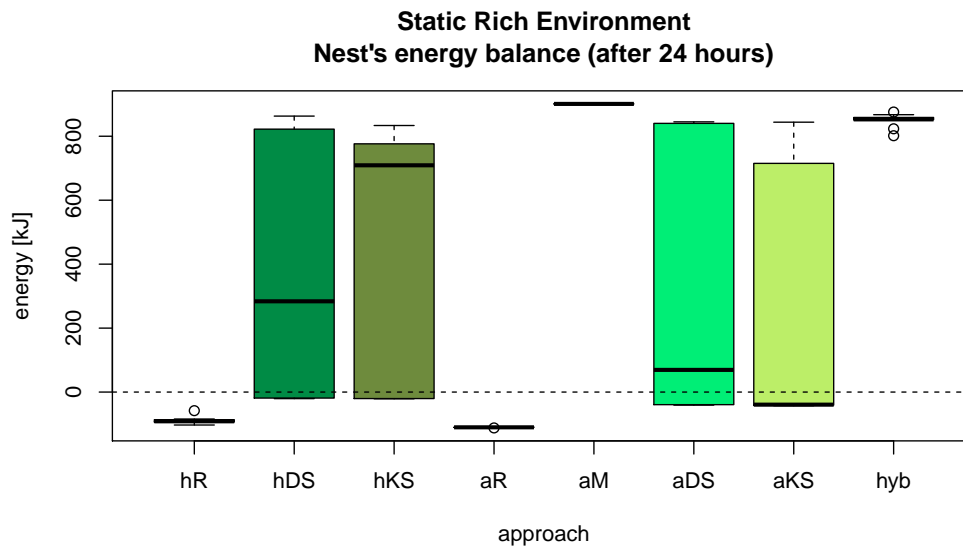


Figure 5.3: Boxplot of the nest's final energy balance for each approach in the rich environment.

The most important questions that arises from these observations is why the reinforcement learning approaches fail so often. To investigate this problem, the course of the nest's temperature is analyzed. Figure 5.4 shows such courses using the example of Heteronomous Discrete Sarsa. Apparently, the approach runs into a trap when exceeding the critical temperature of 22°C too much. Although the leading entity is rewarded for getting closer to 20°C , it does not seem to learn it.

This inability can be related to the used policy: the ϵ -greedy policy with $\epsilon = 0.1$ selects the greedy action in 90 % of the time and randomly chooses from the non-greedy actions in 10 % of the time. Although `CommandCoolDown` promises a higher reward than `CommandHeatUp`, both tasks are selected with equal probability if they are both non-greedy. In fact, `CommandRest` is quickly learned to be the greedy action when the critical temperature is exceeded and no more food can be assimilated.

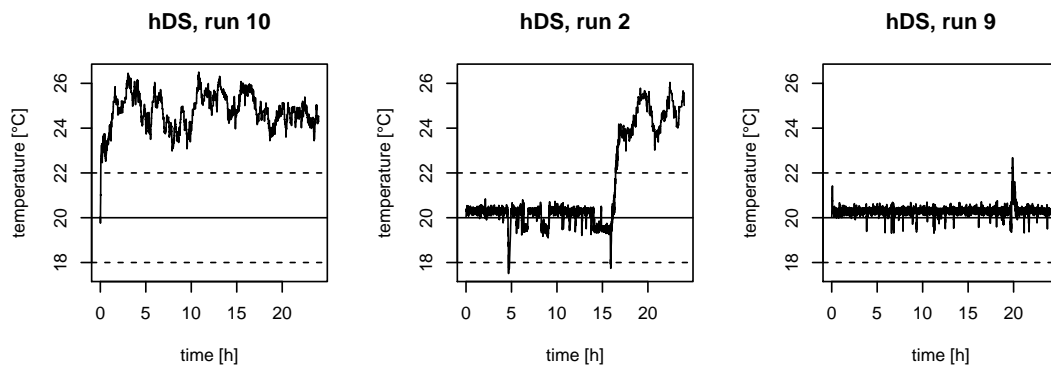


Figure 5.4: Examples for course of nest's temperature: all three runs are simulated in the rich environment with a swarm using Heteronomous Discrete Sarsa.

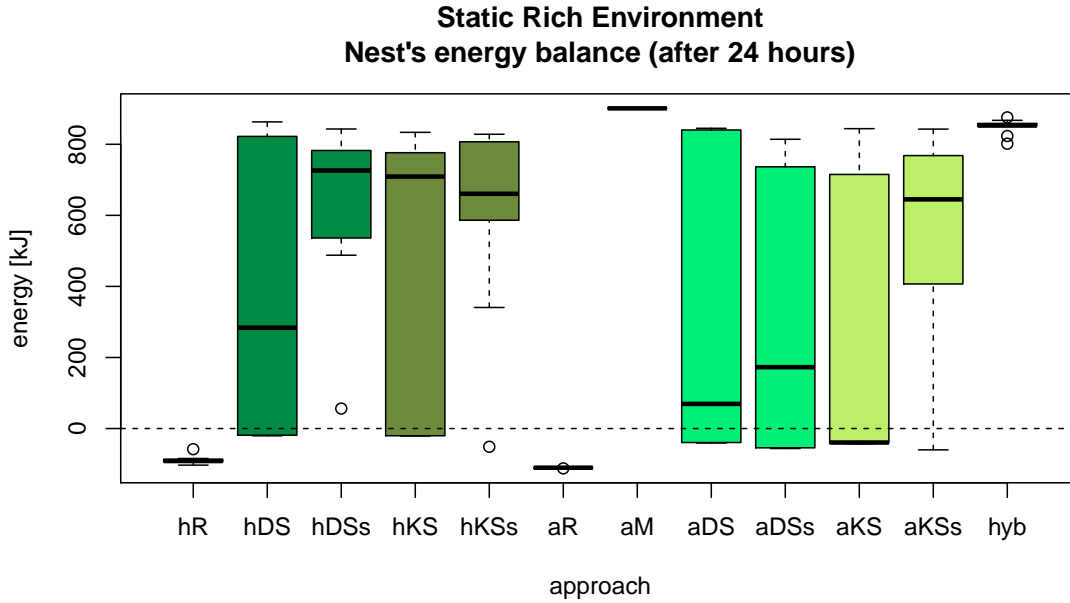


Figure 5.5: Boxplot of the nest’s final energy balance for each approach in the rich environment (extended version including reinforcement learning approaches using a softmax policy, as indicated by the “s”, instead of an ϵ -greedy one).

In order to solve this problem, all reinforcement learning approaches are alternatively equipped with a softmax action selection policy. This policy weights all actions dependent on their Q -values and thus grades the non-greedy actions, too. To ensure exploration, the action with the lowest Q -value has at least a probability of 2.5%. In the concrete problematic case, the policy is expected to select the task `CommandCoolDown` with higher probability than the heating task.

Figure 5.5 shows the final nest’s balances for all approaches again but extended by the alternative reinforcement learning approaches using the softmax policy. Apparently, nearly all new approaches perform better than their ϵ -greedy counterparts. As exemplarily shown in figure 5.6, the softmax policy helps to get out of the critical temperature range. Only Autonomous Discrete Sarsa does not seem to profit in the given time. This may result from the fact that each robot selects its tasks autonomously which makes it hard to relate rewards to own actions: if, for instance, one robot executes `CoolDown` and another one executes `HeatUp` concurrently, both robots will have difficulties with learning from the observed – unknowingly accumulated – reward.

In order to test whether the approaches have adapted to the given scenario, the selection of tasks in the final phase of simulation is worth a look. Figure 5.7 illustrates the tasks’ distribution during the 24th hour. In all simulations of Heteronomous Discrete Sarsa (hDS), for example, the fraction of `HeatUp` in the final hour was below 25%, a level that is indicated by the horizontal dotted line. Except in one simulation run, the fraction was even clearly below 20%. Large boxes, like the one for `Rest` in hDS, indicate that the individual runs performed very differently. Note that the data points for each task are dependent to each other because the fractions for each run sum up to 100%. Although this makes the depiction relatively vague in describing the distribution of tasks, some interesting observations can still be made:

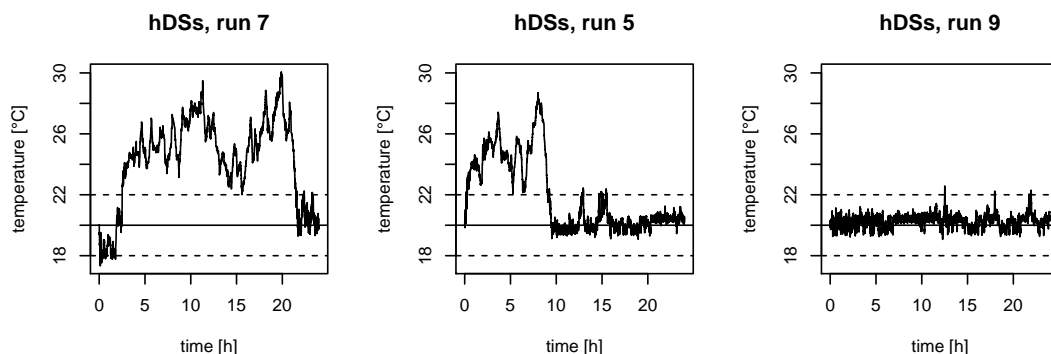


Figure 5.6: Examples for course of nest’s temperature: all three runs are simulated in the rich environment with a swarm using Heteronomous Discrete Sarsa (equipped with softmax policy).

1. Apparently, only the Autonomous Motivation approach is able to disable the selection of the – in this scenario – unnecessary task `HeatUp`. This comes from the fact that all other depicted approaches use some form of reinforcement learning that makes the robots continuously select all tasks in order to explore their possibilities.
2. Beside the hybrid approach, Heteronomous Discrete Sarsa (with softmax policy), Heteronomous Kernel Sarsa (with ϵ -greedy policy), and Autonomous Kernel Sarsa (with softmax policy) appear to be the only reinforcement learning approaches that clearly discriminate between the selection of `HeatUp` and `CoolDown`.

In summary, all approaches – except the random ones – adapt to the hostile world temperature, at least by increasing the chance to `Rest`.

Figure 5.8 gives an alternative view of the approaches’ performances. In contrary to the boxplots in figure 5.5, this type of chart contains the *mean* value of the nest’s energy balance (remember that the thick line in a boxplot marks the *median* but not the mean). On average, the centralized versions of reinforcement learning performed better than their autonomous counterparts. Note that the difference of the pairs hDS/aDS and hKSs/aKSs is much smaller than the difference of the pairs hDSs/aDSs and hKS/aKS. Also note the massive variation between (nearly) negative and very high levels of energy in all reinforcement learning approaches.

Conclusion. Within 24 hours of simulation in the static rich environment, the Autonomous Motivation approach is, without a doubt, the best suited approach. In fact, this approach even shows that the rich environment does not contain enough food to make each `Forage` successful. Otherwise, the motivation for resting would reach zero and render the selection of task `Rest` impossible (cf. figure 5.7). Within the scope of their possibilities, the reinforcement learning approaches also perform well but they tend to fail if the ϵ -greedy policy is used and tend to need much time for learning if the softmax policy is used. Note that this conclusion only applies for the 24 hours of simulation. Longer runs will probably strengthen all reinforcement learning approaches.

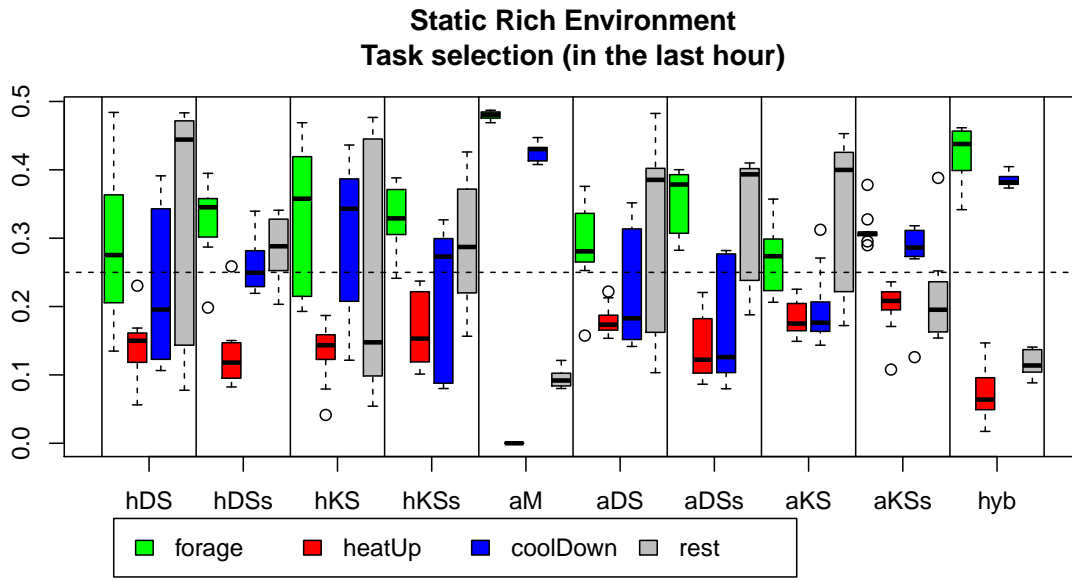


Figure 5.7: Boxplots for the distribution of tasks in the final hour of simulation in the rich environment. Random approaches are omitted because their depiction just shows the expected equal distribution.

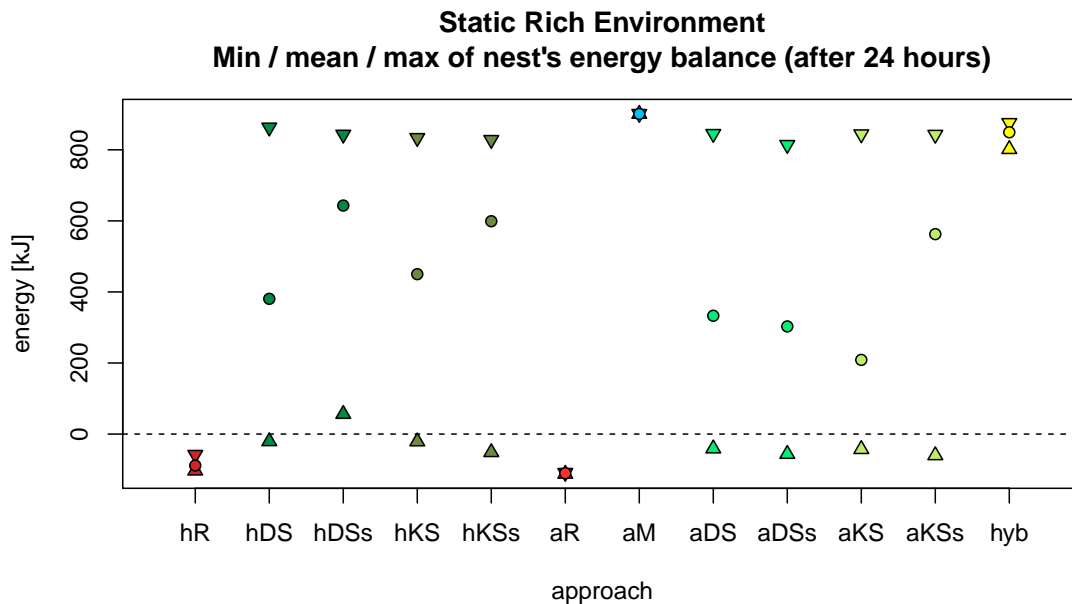


Figure 5.8: Minimum, mean and maximum of the nest's final energy balance for each approach in the rich environment.

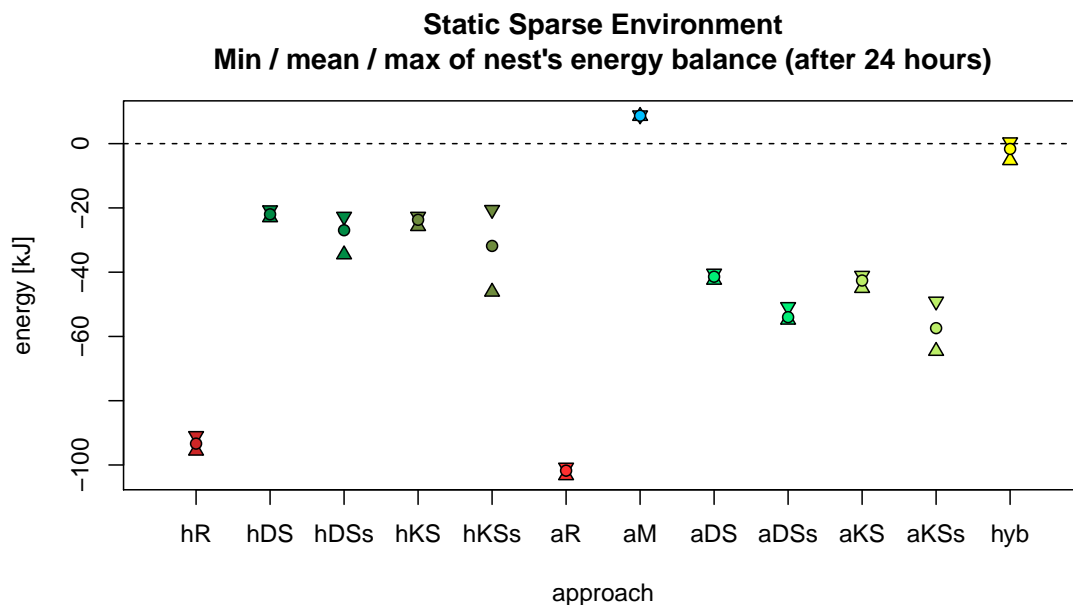


Figure 5.9: Minimum, mean and maximum of the nest's final energy balance for each approach in the sparse environment.

5.3.1.2 Sparse Environment

The sparse environment provides very little food in a circular area at the left side of the nest. The amount is still fair enough to justify maintenance of the nest's temperature.

For each approach, 10 experiments are carried out for 24 hours of simulated time. The summary in figure 5.9 shows the minima, means and maxima of the nest's final energy balance, grouped by the used approach. In comparison to the corresponding summary of the rich environment (cf. figure 5.8), this illustration suggests that the approaches tend to a more definite solution because the difference between minimum and maximum is low.

In fact, all approaches finally tend to **Rest**, as shown in figure 5.10. This is an understandable decision because the environment does not contain much food. Unfortunately, only the approaches **Autonomous Motivation** and **Hybrid** seem to be able to profit from the collection of food items. All other approaches waste more energy in exploration than they could possibly gather. As a result, most reinforcement learning approaches even don't bother about maintaining the nest's temperature. Their execution rates of **HeatUp** and **CoolDown** are very similar.

Looking at figure 5.10 reveals that both **Heteronomous Discrete Sarsa** and **Heteronomous Kernel Sarsa** do care about the nest's temperature if the softmax policy is used: **CoolDown** is selected much more often than **HeatUp**. Additionally, **Forage** is selected quite often, which lets assume that both approaches learned to spend energy in order to get food. This assumption is supported by the amount of assimilated food, as depicted in figure 5.11.

Conclusion. Within 24 hours of simulation in the static sparse environment, the **Autonomous Motivation** approach is, without a doubt, the best suited approach. This

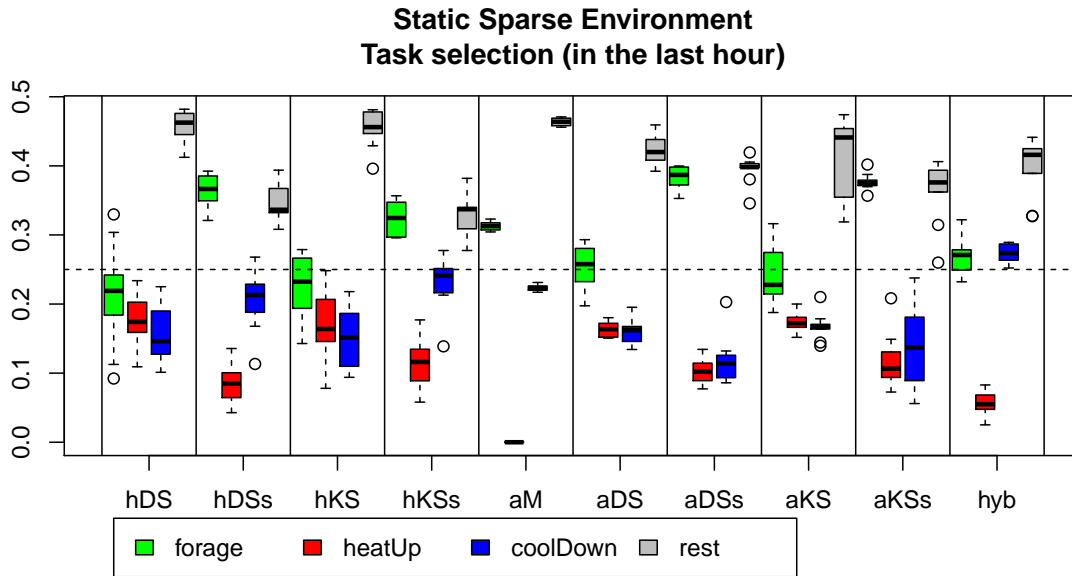


Figure 5.10: Boxplots for the distribution of tasks in the final hour of simulation in the sparse environment.

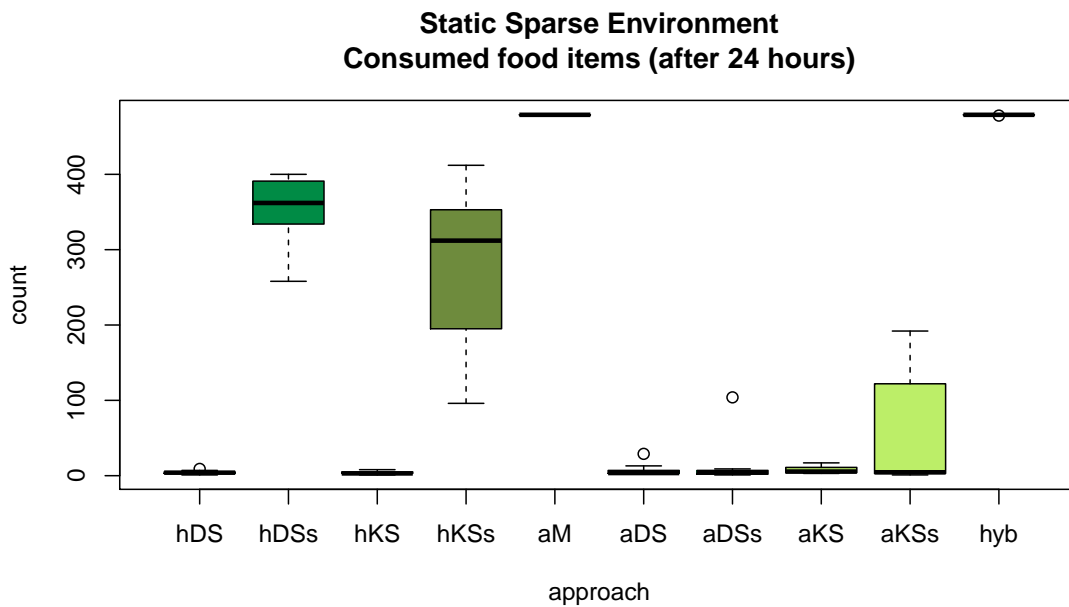


Figure 5.11: Boxplots for assimilated food at the end of the simulation in the sparse environment.

result is not surprising because there is still a fair amount of food available. Interestingly, both heteronomous Sarsa approaches that are equipped with the softmax policy (hDSs and hKSs), and even Autonomous Kernel Sarsa with softmax policy (aKSs) a little, tend to maintain the nest’s temperature in order to assimilate food. Although the energy needed for exploration inhibits to reach a positive balance, successful foraging at least slows down the decrease of energy. Even the best performing swarm using reinforcement learning couldn’t avoid a negative trend in energy.

5.3.2 Dynamic Scenarios

In dynamic scenarios, the swarm’s environment changes during a single simulation run. As a result, the robots have to adapt continuously in order to perform well. In this thesis, two different types of dynamics are tested: the first environment features a slow and continuous change of the world’s temperature whereas the second environment abruptly changes between a rich and a sparse food density.

5.3.2.1 Dynamic Temperature

Regarding food density, the dynamic temperature environment is equal to the rich environment presented in section 5.3.1.1: a very sufficient amount of food continuously appears in a belt around the nest.

Other than in the rich environment, the world’s temperature is not constant but follows a sine wave. Starting with a value of 20 °C, the aerial temperature changes in the interval of 15 to 25 °C. The period is set to 12 hours of simulation.

For each approach, 10 experiments are carried out for 36 hours of simulated time. This duration corresponds to three periods of the temperature’s dynamics. It can be expected that the robots learn from failures in the first period and perform better in the following periods.

Figure 5.12 illustrates the performance of all approaches. None of them performs badly, even the random approaches are able to maintain a positive balance, at least on average. This comes from the fact that – every six hours – the robots do not need to maintain the nest’s temperature because the world’s temperature already changes to the range where food assimilation is possible.

Although the motivation-based approach still performs best, the reinforcement learning methods seem to catch up. It is noticeable that the ϵ -greedy policy tends to work a little better than the softmax policy. The biggest difference between the two policies can be found in Autonomous Discrete Sarsa. This observation reminds of the rich environment where the softmax policy improved all reinforcement learning approaches except Autonomous Discrete Sarsa (cf. figure 5.8).

In contrast to the rich environment with its constant temperature, the dynamic temperature environment supports the reinforcement learning approaches. On the one hand, the world’s temperature changes so slow that most swarms can adapt without any difficulties. On the other hand, even if a swarm fails to hold the optimal temperature, the robots get a second chance. At least, none of the reinforcement learning methods seems to make the same mistake twice. Roughly speaking, the world’s dynamics guide the robots to the critical temperature areas where they can learn the most important thing: `CoolDown` at ≤ 18 °C and `HeatUp` at ≥ 22 °C.

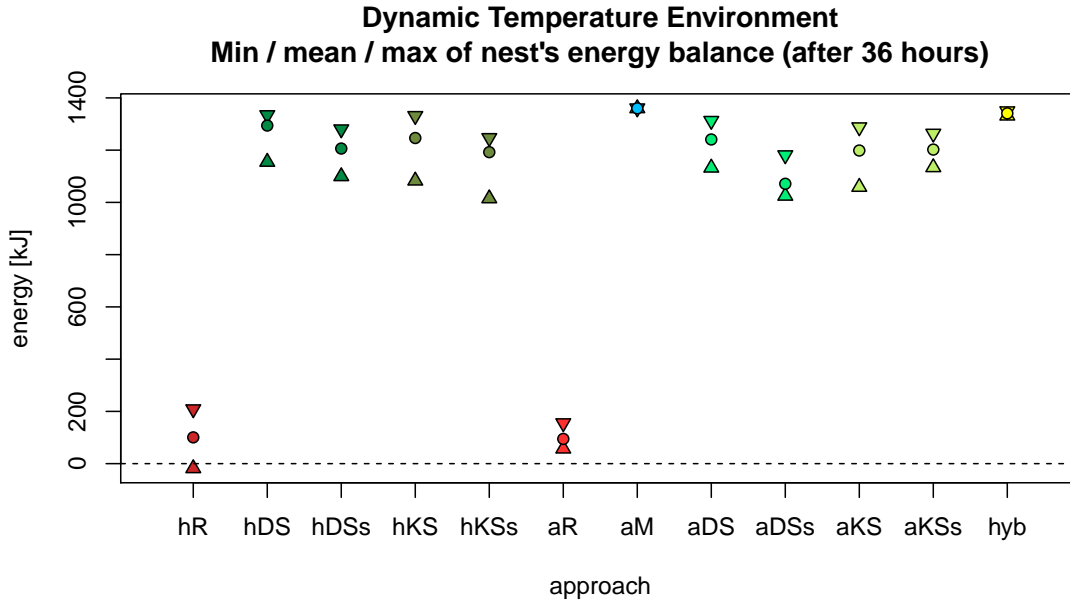


Figure 5.12: Minimum, mean and maximum of the nest's final energy balance for each approach in the dynamic temperature environment.

Conclusion. Within 36 hours of simulation in the dynamic temperature environment, all approaches, except the random ones, perform very well. Although the reinforcement learning approaches perform much better than in previous experiments, they are not able to reach the performance of the Autonomous Motivation approach.

5.3.2.2 Dynamic Food Density

In the environment with dynamic food density, the world's temperature is set to constant 25 °C again. Starting with a rich environment, the food density abruptly switches every six hours, either from rich to sparse or reverse. In fact, this is a dynamic combination of the static environments presented in section 5.3.1.

For each approach, 10 experiments are carried out for 36 hours of simulated time. This duration features both the rich and the sparse environment three times. From the results in the other scenarios, it can be expected that the motivation-based approach performs best again.

Figure 5.13 illustrates the performance of each approach. At first glance, it may surprise that the boxplots of all reinforcement learning methods using an ϵ -greedy policy are smaller than in the static rich environment (cf. figure 5.5). This seems to be caused by the short duration of the first, rich phase, which basically determines the final outcome. As a result the outcomes cannot differ very much.

Nearly all ϵ -greedy swarms failed at the second phase and did not get out of the following temperature trap. From those swarms, only some that followed Autonomous Kernel Sarsa (aKS) could profit from later rich phases because they managed to hold the temperature throughout the second phase. Nevertheless, as soon as the temperature gets too high, *all* ϵ -greedy swarms fail to get back into the well tempered area.

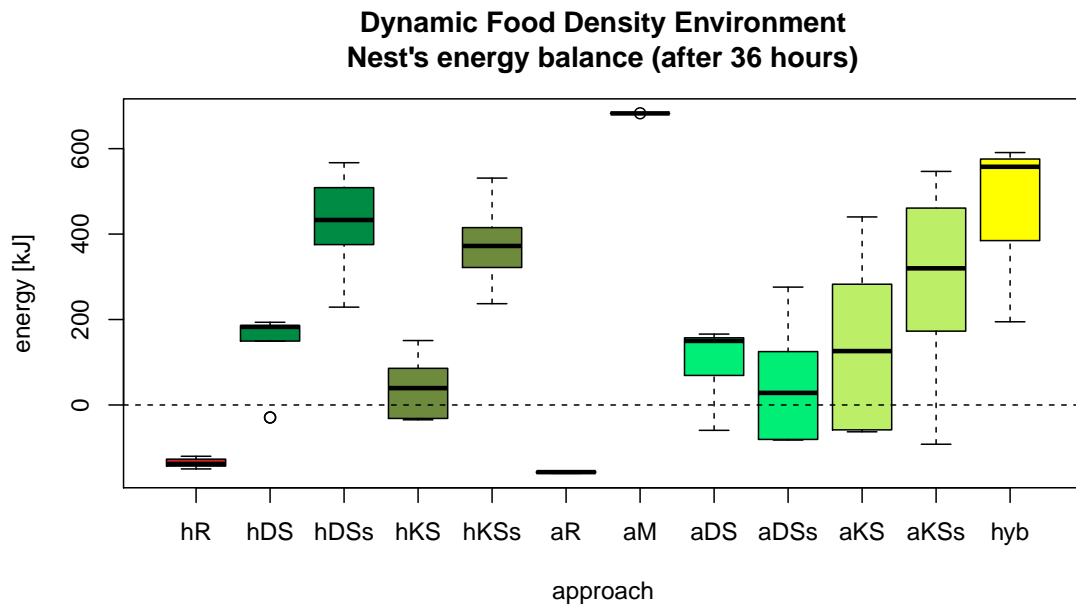


Figure 5.13: Boxplot of the nest's final energy balance for each approach in the dynamic food density environment.

As in the static rich environment, the reinforcement learning approaches generally perform better when the softmax action selection policy is used. The only exception from this rule seems to be softmax-variant of Autonomous Discrete Sarsa (aDSs) but it is possible that this approach will get better if it has more time to adapt.

Exemplarily for a reinforced method, figure 5.14 shows the progress of the nest's energy balance and the nest's temperature for each swarm that used Heteronomous Kernel Sarsa with the softmax policy (hKSs). Although this approach seems to be relatively successful because it is able to profit from multiple rich phases, failures can always occur. For example: run number (1) successfully maintains the nest's temperature for four phases. Although phase five, the next to last one, is a rich phase, the swarm abruptly fails. This may be caused by an abrupt change of rewards at phase entry. Easily, high rewards that result from a former `CommandForage` can misleadingly be related to a recent `CommandHeatUp`, which – in return – will be selected more often and quickly push the swarm out of the well tempered area.

Conclusion. Within 36 hours of simulation in the dynamic food density environment, the Autonomous Motivation approach is, as expected, the best suited approach. Interestingly, the hybrid approach, which – on average – takes the second rank, seems to be less robust than in the static scenarios: its performance fluctuates much more. The reinforcement learning approaches have the same difficulties as in the static scenarios: swarms using the ϵ -greedy policy easily get stuck in temperature states where no food can be assimilated. As in the static rich scenario, the softmax policy helps to leave these trap states. Only Autonomous Discrete Sarsa does not seem to profit in the given time.

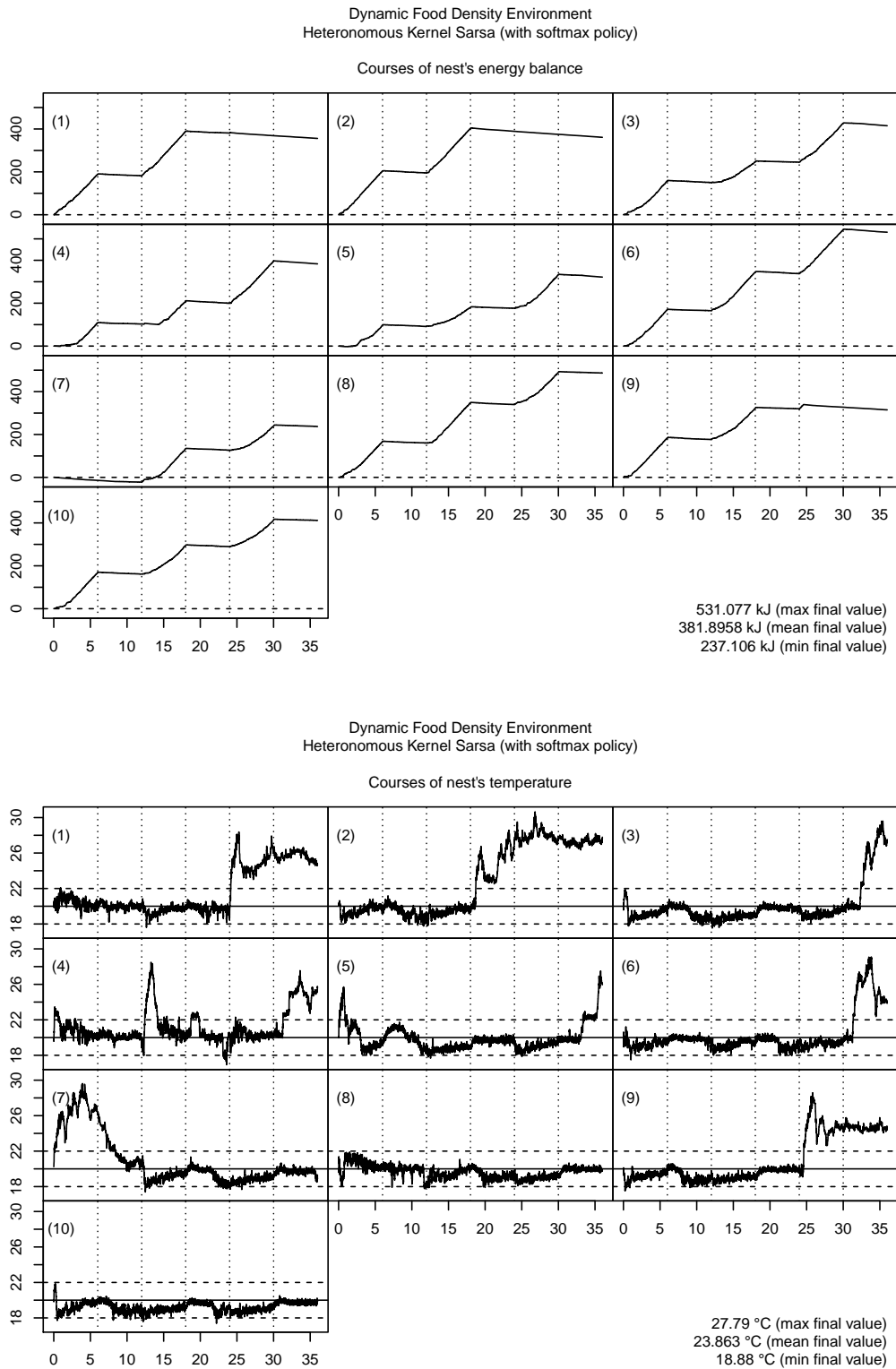


Figure 5.14: Balance and temperature courses for each swarm that uses Heteronomous Kernel Sarsa with softmax policy (hKSs) in the dynamic food density environment. The x-axis is measured in hours of simulated time. The vertical dotted lines mark the points of time where the food density changes.













		Means of the nest's final energy balance (in kJ)			
Approach		Static Rich	Static Sparse	Dynamic Temperature	Dynamic Food Density
	hR	-88.47 (11)	-93.35 (11)	100.64 (11)	-136.00 (11)
	hDS	380.69 (7)	-21.98 (3)	1294.15 (3)	138.53 (6)
	hDSs	643.21 (3)	-26.94 (5)	1206.20 (6)	434.26 (3)
	hKS	450.03 (6)	-23.70 (4)	1246.43 (4)	36.14 (10)
	hKSs	599.11 (4)	-31.82 (6)	1192.31 (9)	381.90 (4)
	aR	-110.19 (12)	-101.76 (12)	94.73 (12)	-157.18 (12)
	aM	901.09 (1)	8.69 (1)	1360.15 (1)	682.52 (1)
	aDS	333.06 (8)	-41.41 (7)	1240.92 (5)	106.97 (8)
	aDSs	303.03 (9)	-53.99 (9)	1071.27 (10)	39.09 (9)
	aKS	208.84 (10)	-42.59 (8)	1198.59 (8)	118.35 (7)
	aKSs	562.72 (5)	-57.42 (10)	1202.22 (7)	275.30 (5)
	hyb	849.38 (2)	-1.64 (2)	1340.87 (2)	477.84 (2)

Table 5.5: Ranking of all approaches with respect to the scenario. The ranking is based upon the means of the nest's final energy balance.

5.3.3 Discussion

Table 5.5 gives an overview of the performance of all approaches with respect to the given scenario, measured by the mean of the nest's final energy balance. The number in brackets shows the rank of the approach within each environment.

In all scenarios, the random approaches performed worse. Noticeably, the heteronomous one performed a little better. This observation can be explained: if a leading entity commands a task, potentially all five robots are inside the nest and execute the given task concurrently. As a result, the effect of `CoolDown` and `HeatUp` is much higher which raises the probability to maintain the nest's temperature by accident.

The other end of the spectrum can also be identified easily: Autonomous Motivation gains the gold medal and the hybrid approach gains silver. It can be said that the simple technique of motivation-based control is perfectly suited for the given scenarios although the underlying mechanism is very simple. On the downside, this approach will always maintain the nest's temperature, even if the swarm has no chance to reach the well tempered area or if there is no food at all that would justify the effort. Nevertheless, it is the best choice if such scenarios do not occur, as in the experiments.

Of course, the motivation-based approach is not optimal because the nest's temperature is maintained more than it needs to be, especially in the sparse environment. Optimization can be achieved by lowering the motivations for `HeatUp` and `CoolDown`. Also note that the performance of the approach is expected to depend on the swarm's size. For instance: if 1000 robots start cooling with a probability of 1%, an average of 10 robots will start cooling at the same time. This effect could, of course, be countered by a cooling task that automatically stops at the desired temperature state.

In summary, motivation-based control is a simple and effective mechanism to solve Task Allocation in the given foraging mission. Nevertheless, the approach has to be adapted properly to the expected environmental conditions and to the swarm's size that may be deployed.

All remaining approaches are based on the reinforcement learning method of Sarsa(λ). Each one is defined by three features:

1. Control type: either heteronomous (h) or autonomous (a)
2. State representation: either discrete (DS = discrete Sarsa) or continuous (KS = kernel Sarsa)
3. Policy: either ϵ -greedy (original version) or softmax action selection (s)

Based on the outcome of experiments, the heteronomous control type seems to work better than its autonomous counterpart. This may result from the fact that the leader potentially commands five robots at the same time which raises the reward of each action. On the downside, the chance of leaving the well tempered state is much higher because the impact of `CommandHeatUp` and `CommandCoolDown` is greater than if autonomous decisions are taken, at least on average. Note that this result is only applicable for the given time of experimentation. The autonomous variants seem to learn slower but may potentially outperform their heteronomous counterparts in longer simulations.

The discrimination of discrete and kernel Sarsa is already harder. Generally, the discrete approach performs better. One exceptional case is aKSs, which clearly outperforms aDSs in all scenarios except the static sparse one. Again, this observation may result from the given time of experimentation. Longer simulations with a greater number of experiments would be needed to draw satisfying conclusions.

As discussed in the static rich environment, the selection of an appropriate policy is very important. In the given scenarios, the ϵ -greedy policy tends to get stuck in unpleasant temperature states where no food can be assimilated. As a result, these approaches learn that resting is the only reasonable action. In contrast, the softmax action selection policy is able to leave the trap states because it allows to discriminate the usefulness of non-greedy actions with respect to the learned Q -values.

In summary, the Heteronomous Discrete Sarsa approach with softmax policy (hDSs) seems to be the best choice from the reinforcement learning approaches. Noticeably, the Autonomous Kernel Sarsa method with softmax policy (aKSs) does also perform very well in the static rich and the dynamic food density environment. In general, autonomous approaches perform worse but they may not have enfolded their full potential in the given time.

The hybrid approach was included for proof of concept. It seems to profit very much from the robots that are driven by motivations. On the other side, the inclusion of robots controlled by Heteronomous Kernel Sarsa seems to handicap the approach, especially in the dynamic food density scenario. This may just result from the fact that reinforcement learning forces to choose non-greedy actions once in a time.

In summary, the experiments that were carried out advise to simply use the pure motivation-based approach. It is easy to implement, performs very well and is robust in both static and dynamic scenarios, at least as long as the amount of available food justifies the effort of maintaining the nest's temperature.

5.4 Summary

This chapter presented experiments in a foraging scenario that is complicated by the need to maintain the nest's temperature at a specific level. Robots have to choose between the tasks *Forage*, *HeatUp*, *CoolDown* and *Rest*.

First, the scenery was described. This included the definition of the world's components: a nest, which converts food to energy as long as it is well tempered, food areas, which produce food at a specified rate, and robots, which consume energy in order to collect food and in order to maintain the nest's temperature. The aerial temperature of the world influences the temperatures of both the nest and the robots.

Second, approaches to solve the given Task Allocation problem were sketched, following the taxonomy presented in chapter 3. Because the robots' communication is limited to the local broadcast of a light signal, all mechanisms that rely on complex communication protocols were excluded from further consideration. Additionally, a focus was put on reinforcement learning methods and simple adaptive approaches.

Finally, discrete and continuous variants of both centralized and decentralized reinforced control were tested in static and dynamic scenarios. Additionally, a simple motivation-based approach and a hybrid approach using side-by-side control have been simulated.

All methods proved to be better than random allocation but only the motivation-based approach could manage a positive energy balance in both rich and sparse scenarios. This motivation-based method turned out to be the overall winner in the tested scenarios: it is simple, robust and performs best.

Reinforced control was also able to perform well. Unfortunately, the performance of the tested methods varied very much. It is expected that all variants could be improved by redefinition of some parameters and elements, e.g. the reward function. In many cases, the learning methods even had the problem of running into a kind of "dead end" where exploration did not suffice to learn better action selection.

This thesis advises to carefully analyze the mission and its environmental conditions before choosing a concrete approach to solve Task Allocation. The experiments showed that reinforcement learning has great potential but needs to be adapted well. In clear missions like the present one, it is easier to implement adaptation rules for a motivation-based approach than to construct a reward function that successfully drives the learning process.

Chapter 6

Conclusion and Outlook

This thesis engaged mechanisms for Task Allocation in Swarm Robotics in theory and practice. In its theoretical part, the background of the corresponding research field has been enlightened and an overview of approaches to Task Allocation has been given. In its practical part, the thesis used the Swarmulator – a simulation platform that was developed within the scope of this work – to experiment with different mechanisms in a concrete foraging scenario.

The presented overview should help designers of swarm robotic systems to find mechanisms that are appropriate for their needs. In order to clearly arrange different approaches, a new taxonomy was proposed which discriminates mechanisms into Heteronomous, Autonomous and Hybrid Task Allocation.

In the experimental part of this work, a focus was set on centralized and decentralized reinforced control. Additionally, a simpler adaptive probabilistic approach based on motivations was simulated. In the tested static and dynamic environments, this motivation-based method outperformed all reinforcement learning approaches because it was much easier to define adequate adaptation rules for motivations than to construct a reward function that harmonizes with the learning method and its parameters. Nevertheless, the motivation-based approach just performed better because it was tailored to the concrete scenarios. Under other environmental conditions the motivation-based approach would break down and be outperformed by reinforcement learning.

Swarm Robotics is still a young research field and in continuous development. Task Allocation is a very important problem that has to be faced by designers of robotic swarms. Recent research shows that there is still much work to do and the investigation of Task Allocation in Swarm Robotics is far from finished. Shiroma and Campos [SC09], for example, investigate the potential of robots that follow multiple tasks concurrently instead of being restricted to one task at a time. Another direction of research examines self-organized task partitioning [PBF⁺11]. These are first steps towards autonomous robotic swarms that do not only efficiently tackle Task Allocation but also decompose the global mission into tasks and subtasks on their own.

It would be interesting to simulate these new approaches in the Swarmulator. Due to its modular design, this tool is applicable for a wide range of scenarios and can easily be adapted to new ones, as demonstrated by Gutschale [Gut12] who investigates Multi Robot Task Allocation with respect to triggered events, like robot failure. Further extension, for example by the feature to load and save ongoing experiments, will make the Swarmulator an even more powerful tool that supports research in the field of Swarm Robotics and beyond.

List of Figures

1.1	Artificial bee	2
2.1	Affiliation and categories of Swarm Robotics	8
2.2	MRS taxonomy	13
2.3	Robots of the Centibots Project in action	17
2.4	The iRobot Swarm	18
2.5	Jasmine robots in action	19
2.6	Swarm-bots in simulation	19
2.7	The Swarmanoid robots: foot-bot, hand-bot and eye-bot	20
3.1	Proposed taxonomy for Task Allocation in Swarm Robotics	33
3.2	Communication in Heteronomous Task Allocation	34
3.3	Blackboard control	36
3.4	Langton's ant after 11000 steps	43
3.5	Foraging via chain forming	46
3.6	Light-based control	47
3.7	Logistic probability functions	50
3.8	Snapshot of an experiment using the Variable Delta Rule	52
3.9	Side-by-side control (alternative and concurrent)	59
3.10	Overview of mechanisms for Task Allocation in Swarm Robotics	62
4.1	Simplified representation of the MVC pattern used in the Swarmulator	64
4.2	UML class diagram of the Swarmulator	65
4.3	Main frame of the Swarmulator in action	66
4.4	Interfaces for the creation of new experiments	67
4.5	UML class diagram for ComponentWorld	69
4.6	Example quadtree for position management	70
4.7	UML class diagram for active components and tasks	71
4.8	UML class diagram for common active component tasks	71
4.9	Example UML object diagram for ACTRandom	71
4.10	UML class diagram for ACTLinkedMotivation	73
4.11	Stylized UML diagram for ACTSarsa	74
5.1	Exemplary utilization of the Epanechnikov kernel in ACTKernelSarsa	83
5.2	Static scenarios	89
5.3	Boxplots of final energy balances in rich environment	90
5.4	Examples for course of the nest's temperature (using hDS)	90
5.5	Boxplots of final energy balances in rich environment (extended)	91
5.6	Examples for course of the nest's temperature (using hDSs)	92
5.7	Boxplots for the final hour's task distribution in rich environment	93

5.8	Min/mean/max of final energy balance in rich environment	93
5.9	Min/mean/max of final energy balance in sparse environment	94
5.10	Boxplots for the final hour's task distribution in sparse environment . .	95
5.11	Boxplots of assimilated food counts in sparse environment	95
5.12	Min/mean/max of final energy balance in dynamic temp. environment .	97
5.13	Boxplots of final energy balances in dynamic food density environment .	98
5.14	Balance and temperature courses of hKSs in dynamic food density env.	99

List of Tables

2.1	Properties of a robotic swarm, as defined by Dudek et al.	12
5.1	Overview of execution times and consumed energy of basic tasks	81
5.2	Overview of used elements and parameters in centralized reinforced control	84
5.3	Adaptation in motivation-based approach	86
5.4	Summary of approaches used in the experiments	88
5.5	Ranking of approaches dependent on scenario	100

Content of the enclosed CD

The content of the enclosed CD is organized as follows:

/	
├── experiments	Swarmulator version and importable world factory as jar-files that were used to carry out the experiments described in this thesis.
│ ├── data	Compressed experimental data and corresponding batch-files that were used by the Swarmulator in order to create the data.
│ └── evaluation	Evaluation graphs and corresponding R-scripts that were used for creation.
├── references	Copies of the related work that is referenced in this thesis.
├── Swarmulation	Implementation of the world factory that is able to create the experimental scenarios. Java project including the source code and a jar-file that can be imported into the Swarmulator.
├── Swarmulator	Implementation of the Swarmulator version 1.0.1. Java project including the source code and a runnable jar-file.
└── thesis	This thesis in pdf format.

Bibliography

- [AB07] Nuzhet Atay and Burchan Bayazit. Emergent Task Allocation for Mobile Robots. In *Proceedings of the Robotics: Science and Systems (RSS III)*, June 2007.
- [AL06] Sherief Abdallah and Victor Lesser. Learning the Task Allocation Game. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 850–857. ACM Press, May 2006.
- [BA02] Khashayar R. Baghaei and Arvin Agah. Task Allocation Methodologies for Multi-Robot Systems. Technical Report ITTC-FY2003-TR-20272-01, Information Telecommunication and Technology Center, University of Kansas, Lawrence, KS, November 2002.
- [Bal99] Tucker Balch. Reward and Diversity in Multirobot Foraging. In *IJCAI '99 Workshop on Agents Learning About, From and With other Agents*, pages 92–99, August 1999.
- [BC07] B. B. Biswal and B. B. Choudhury. Development and Simulation of a Task Assignment Model for Multirobot Systems. *International Journal of Engineering*, 1(2):12–23, August 2007.
- [Bru09] Arne Brutschy. Task Allocation in Swarm Robotics: Towards a method for self-organized allocation to complex tasks. Rapport d'avancement de recherche, Université Libre de Bruxelles, 2009.
- [BŞ07] Levent Bayindir and Erol Şahin. A Review of Studies in Swarm Robotics. *Turkish Journal of Electrical Engineering & Computer Sciences*, 15(2):115–147, 2007.
- [BTD96] Eric Bonabeau, Guy Theraulaz, and Jean-Louis Deneubourg. Quantitative Study of the Fixed Threshold Model for the Regulation of Division of Labour in Insect Societies. In *Proceedings: Biological Sciences*, volume 263, pages 1565–1569. The Royal Society, November 1996.
- [CCL⁺90] Philippe Caloud, Wonyun Choi, Jean-Claude Latombe, Claude Le Pape, and Mark Yim. Indoor Automation with Many Mobile Robots. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems (IROS '90): 'Towards a New Frontier of Applications'*, volume 1, pages 67–72. IEEE, July 1990.
- [CD07] Alexandre Campo and Marco Dorigo. Efficient Multi-foraging in Swarm Robotics. In Fernando Almeida e Costa, Luis Rocha, Ernesto Costa,

- Inman Harvey, and António Coutinho, editors, *Advances in Artificial Life*, volume 4648 of *Lecture Notes in Computer Science (LNCS)*, pages 696–705. Springer Berlin / Heidelberg, 2007.
- [CFK97] Y. Uny Cao, Alex S. Fukunaga, and Andrew B. Kahng. Cooperative Mobile Robotics: Antecedents and Directions. *Autonomous Robots*, 4:226–234, 1997.
- [CLIA10] Hung Cao, Simon Lacroix, Félix Ingrand, and Rachid Alami. Complex Tasks Allocation for Multi Robot Teams under Communication Constraints. In *Proceedings of the 5th National Conference on Control Architecture of Robots (CAR 2010)*, May 2010.
- [Cor91] Daniel D. Corkill. Blackboard Systems. *AI Expert*, 6(9):40–47, September 1991.
- [Das09] Prithviraj Dasgupta. A Dynamic-bid Auction Algorithm for Cooperative, Distributed Multi-Robot Task Allocation. Technical report, University of Nebraska, August 2009.
- [DC04] Anna Dornhaus and Lars Chittka. Why do honey bees dance? *Behavioral Ecology and Sociobiology*, 55(4):395–401, February 2004.
- [DFDCG09a] Frederick Ducatelle, Alexander Förster, Gianni A. Di Caro, and Luca Maria Gambardella. New task allocation methods for robotic swarms. In *Proceedings of the 9th IEEE/RAS Conference on Autonomous Robot Systems and Competitions*, May 2009.
- [DFDCG09b] Frederick Ducatelle, Alexander Förster, Gianni A. Di Caro, and Luca Maria Gambardella. Task allocation in robotic swarms: new methods and comparisons. Technical report, IDSIA - Dalle Molle Institute for Artificial Intelligence, January 2009.
- [DFG⁺11] Marco Dorigo, Dario Floreano, Luca Maria Gambardella, Francesco Mondada, Stefano Nolfi, Tarek Baaboura, Mauro Birattari, Michael Bonani, Manuele Brambilla, Arne Brutschy, Daniel Burnier, Alexandre Campo, Anders Lyhne Christensen, Antal Decugnière, Gianni A. Di Caro, Frederick Ducatelle, Eliseo Ferrante, Alexander Förster, Javier Martinez Gonzales, Jerome Guzzi, Valentin Longchamp, Stéphane Magnenat, Nithin Mathews, Marco Montes de Oca, Rehan O’Grady, Carlo Pinciroli, Giovanni Pini, Philippe Rétonnaz, James Roberts, Valerio Sperati, Timothy Stirling, Alessandro Stranieri, Thomas Stützle, Vito Trianni, Elio Tuci, Ali Emre Turgut, and Florian Vaussard. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. Technical report, Université Libre de Bruxelles, July 2011.
- [Dia04] M. Bernardine Dias. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, Carnegie Mellon University, January 2004.
- [DJMW93] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. A Taxonomy for Swarm Robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS ’93)*, volume 1, pages 441–447. IEEE, July 1993.

- [DMS03] Torbjørn S. Dahl, Maja J. Matarić, and Gaurav S. Sukhatme. Distributed Multi-Robot Task Allocation through Vacancy Chains. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2003)*, pages 2293–2298. IEEE, September 2003.
- [DMS09] Torbjørn S. Dahl, Maja J. Matarić, and Gaurav S. Sukhatme. Multi-Robot Task Allocation through Vacancy Chain Scheduling. *Robotics and Autonomous Systems*, 57(6-7):674–687, June 2009.
- [DŞ04] Marco Dorigo and Erol Şahin. Guest editorial. Special issue: Swarm Robotics. *Autonomous Robots*, 17(2-3):111–113, 2004.
- [dWvdK06] Mathijs de Weerd and Roman van der Krogt. Inefficiencies in Task Allocation for Multiagent Planning with Bilateral Deals. In Rong Qu, editor, *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006)*, pages 33–38, December 2006.
- [Gag04] Aaron Gage. *Multi-Robot Task Allocation Using Affect*. PhD thesis, University of South Florida, August 2004.
- [GBF08] Christoph Grüter, M. Sol Balbuena, and Walter M. Farina. Informational Conflicts Created by the Waggle Dance. *Proceedings of the Royal Society B: Biological Sciences*, 275(1640):1321–1327, June 2008.
- [GM01] Brian P. Gerkey and Maja J. Matarić. Principled Communication for Dynamic Multi-Robot Task Allocation. In Daniela Rus and Sanjiv Singh, editors, *Experimental Robotics VII*, volume 271 of *Lecture Notes in Control and Information Sciences*, pages 353–362. Springer Berlin / Heidelberg, 2001.
- [GM03] Brian P. Gerkey and Maja J. Matarić. Multi-Robot Task Allocation: Analyzing the Complexity and Optimality of Key Architectures. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2003)*, volume 3, pages 3862–3868. IEEE, November 2003.
- [GM04] Brian P. Gerkey and Maja J. Matarić. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research*, 23(9):939–954, September 2004.
- [Gut12] Robert Gutschale. An Extended Simulator for Motivation-Based and Fault Tolerant Task Allocation in Multi-Robot Systems. Bachelor’s thesis, Ludwig Maximilians Universität München, March 2012.
- [INS01] Luca Iocchi, Daniele Nardi, and Massimiliano Salerno. Reactivity and Deliberation: a survey on Multi-Robot Systems. In *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, volume 2103 of *Lecture Notes in Computer Science (LNCS)*, pages 9–32. Springer Berlin / Heidelberg, 2001.
- [IRI06] IRIDIA (Artificial Intelligence research laboratory of the Université Libre de Bruxelles). Homepage of the Swarm-bots project. Website, 2006. <http://www.swarm-bots.org> [15.01.2012].

- [IRI11] IRIDIA (Artificial Intelligence research laboratory of the Université Libre de Bruxelles). Homepage of the Swarmanoid project. Website, 2011. <http://www.swarmanoid.org> [15.01.2012].
- [K⁺08] University of Karlsruhe et al. Homepage of the I-SWARM project. Website, 2008. <http://www.i-swarm.org> [15.01.2012].
- [KB00] Michael J. B. Krieger and Jean-Bernard Billeter. The call of duty: Self-organised task allocation in a population of up to twelve mobile robots. *Robotics and Autonomous Systems*, 30(1-2):65–84, January 2000.
- [KEK09] Gal A. Kaminka, Dan Erusalimchik, and Sarit Kraus. Adaptive Multi-Robot Coordination: A Game-Theoretic Perspective. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, May 2009.
- [KFS05] Nidhi Kalra, Dave Ferguson, and Anthony Stentz. Hoplites: A Market-Based Framework for Planned Tight Coordination in Multirobot Teams. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2005)*, pages 1170–1177. IEEE, April 2005.
- [KKC⁺05] Sergey Kornienko, Olga Kornienko, C. Constantinescu, M. Pradier, and Paul Levi. Cognitive micro-Agents: individual and collective perception in microrobotic swarm. In *Proceedings of the IJCAI-05 Workshop on Agents in Real-Time and Dynamic Environment*, pages 33–42, July 2005.
- [KM06] Nidhi Kalra and Alcherio Martinoli. A Comparative Study of Market-Based and Threshold-Based Task Allocation. *Distributed Autonomous Robotic Systems 7*, pages 91–101, 2006.
- [KOV⁺03] Kurt Konolige, Charles Ortiz, Regis Vincent, Andrew Agno, Michael Eriksen, Benson Limetkai, Mark Lewis, Linda Briesemeister, Enrique Ruspini, Dieter Fox, Jonathan Ko, Benjamin Stewart, and Leonidas Guibas. CentiBOTS: Large-Scale Robot Teams. In Alan Schultz, Lynne E. Parker, and Frank Schneider, editors, *Multi-Robot Systems: From Swarms to Intelligent Automata*, volume 2. Kluwer, 2003.
- [KZDS05] Nidhi Kalra, Robert Zlot, M. Bernardine Dias, and Anthony Stentz. Market-Based Multirobot Coordination: A Comprehensive Survey and Analysis. Technical report, Robotics Institute, Carnegie Mellon University, December 2005.
- [Lab07] Thomas H. Labella. *Division of Labour in Groups of Robots*. PhD thesis, Université Libre de Bruxelles, 2007.
- [Lan86] Christopher G. Langton. Studying Artificial Life with Cellular Automata. *Physica D: Nonlinear Phenomena*, 22(1-3):120–149, 1986.
- [LDD04a] Thomas H. Labella, Marco Dorigo, and Jean-Louis Deneubourg. Efficiency and Task Allocation in Prey Retrieval. In Auke Jan Ijspeert, Masayuki Murata, and Naoki Wakamiya, editors, *Proceedings of the First International Workshop on Biologically Inspired Approaches to Advanced Information Technology (BioADIT 2004)*, volume 3141 of *Lecture*

- Notes in Computer Science (LNCS)*, pages 274–289. Springer Berlin / Heidelberg, January 2004.
- [LDD04b] Thomas H. Labella, Marco Dorigo, and Jean-Louis Deneubourg. Self-Organised Task Allocation in a Group of Robots. In *Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems (DARS 2004)*, June 2004.
- [LJGM06] Kristina Lerman, Chris Jones, Aram Galstyan, and Maja J. Matarić. Analysis of Dynamic Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research*, 25(3):225–242, March 2006.
- [LL94] Tim C. Lueth and Thomas Laengle. Task Description, Decomposition, and Allocation in a Distributed Autonomous Multi-Agent Robot System. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS '94): 'Advanced Robotic Systems and the Real World'*, volume 3, pages 1516–1523. IEEE, September 1994.
- [LWS⁺07a] Wenguo Liu, Alan Winfield, Jin Sa, Jie Chen, and Lihua Dou. Strategies for Energy Optimisation in a Swarm of Foraging Robots. In Erol Şahin, William Spears, and Alan Winfield, editors, *Swarm Robotics*, volume 4433 of *Lecture Notes in Computer Science (LNCS)*, pages 14–26. Springer Berlin / Heidelberg, 2007.
- [LWS⁺07b] Wenguo Liu, Alan Winfield, Jin Sa, Jie Chen, and Lihua Dou. Towards Energy Optimization: Emergent Task Allocation in a Swarm of Foraging Robots. *Adaptive Behavior*, 15(3):289–305, September 2007.
- [MA03] Eric Martinson and Ronald C. Arkin. Learning to Role-Switch in Multi-Robot Systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2003)*, volume 2, pages 2727–2734. IEEE, September 2003.
- [MAA12] MAAREC (Mid Atlantic Apiculture & Extension Consortium). The Colony and Its Organization. Website, 2012. <https://agdev.anr.udel.edu/maarec/honey-bee-biology/the-colony-and-its-organization/> [12.02.2012].
- [Mat95] Maja J. Matarić. Issues and Approaches in the Design of Collective Autonomous Agents. *Robotics and Autonomous Systems*, 16(2-4):321–331, December 1995.
- [McL04] James D. McLurkin. Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots. Masters Thesis, Massachusetts Institute of Technology (MIT), May 2004.
- [MDJ07] David Miller, Prithviraj Dasgupta, and Timothy Judkins. Distributed Task Selection in Multi-agent based Swarms using Heuristic Strategies. In Erol Şahin, William Spears, and Alan Winfield, editors, *Swarm Robotics*, volume 4433 of *Lecture Notes in Computer Science (LNCS)*, pages 158–172. Springer Berlin / Heidelberg, 2007.

- [MdOFM⁺10] Marco Antonio Montes de Oca, Eliseo Ferrante, Nithin Mathews, Mauro Birattari, and Marco Dorigo. Opinion Dynamics for Decentralized Decision-Making in a Robot Swarm. In Marco Dorigo, Mauro Birattari, Gianni A. Di Caro, René Doursat, Andries Petrus Engelbrecht, Dario Floreano, Luca Maria Gambardella, Roderich Groß, Erol Şahin, Hiroki Sayama, and Thomas Stützle, editors, *Proceedings of the Swarm Intelligence 7th International Conference (ANTS 2010)*, volume 6234 of *Lecture Notes in Computer Science (LNCS)*, pages 251–262. Springer Berlin / Heidelberg, 2010.
- [MdOSBD10] Marco Antonio Montes de Oca, Thomas Stützle, Mauro Birattari, and Marco Dorigo. Incremental Social Learning Applied to a Decentralized Decision-Making Mechanism: Collective Learning Made Faster. In I. Gupta, S. Hassas, and J. Rolia, editors, *Proceedings of the 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2010)*, pages 243–252. IEEE Computer Society, September / October 2010.
- [MFG⁺02] Francesco Mondada, Dario Floreano, André Guignard, Jean-Louis Deneubourg, Luca Maria Gambardella, Stefano Nolfi, and Marco Dorigo. Search for Rescue: an Application for the SWARM-BOT Self-Assembling Robot Concept. Technical report, Université Libre de Bruxelles, September 2002.
- [MP10] Yogeswaran Mohan and S. G. Ponnambalam. Swarm Robotics: An Extensive Research Review. In Igor Fuerstner, editor, *Advanced Knowledge Application in Practice*, chapter 14, pages 259–278. InTech, November 2010.
- [MPG⁺04] Francesco Mondada, Giovanni C. Pettinaro, André Guignard, Ivo W. Kwee, Dario Floreano, Jean-Louis Deneubourg, Stefano Nolfi, Luca Maria Gambardella, and Marco Dorigo. Swarm-Bot: A New Distributed Robotic Concept. *Autonomous Robots*, 17(2-3):193–221, September 2004.
- [MS09] Sifat Momen and Amanda J. C. Sharkey. An Ant-like Task Allocation Model for a Swarm of Heterogeneous Robots. In *Proceedings of the 2nd Swarm Intelligence Algorithms and Applications Symposium (SIAAS 2009), AISB 2009 convention*, pages 31–38, April 2009.
- [MY05] James D. McLurkin and Daniel Yamins. Dynamic Task Assignment in Robot Swarms. In *Proceedings of the Robotics: Science and Systems (RSS I)*, June 2005.
- [NGB⁺09] Shervin Nouyan, Roderich Groß, Michael Bonani, Francesco Mondada, and Marco Dorigo. Teamwork in Self-Organized Robot Colonies. *IEEE Transactions on Evolutionary Computation*, 13(4):695–711, August 2009.
- [NGTR08] Amir M. Naghsh, Jeremi Gancet, Andry Tanoto, and Chris Roast. Analysis and Design of Human-Robot Swarm Interaction in Firefighting. In *Proceedings of the 17th IEEE International Symposium on Robot and*

- Human Interactive Communication (RO-MAN 2008)*, pages 255–260. IEEE, August 2008.
- [ØMS01] Esben H. Østergaard, Maja J. Matarić, and Gaurav S. Sukhatme. Distributed Multi-Robot Task Allocation for Emergency Handling. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, volume 2, pages 821–826. IEEE, November 2001.
- [OVM05] Charles L. Ortiz, Régis Vincent, and Benoit Morisset. Task Inference and Distributed Task Management in the Centibots Robotic System. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 860–867. ACM Press, July 2005.
- [P⁺12] Carlo Pinciroli et al. The ARGoS Website. Website, 2012. <http://iridia.ulb.ac.be/argos/> [07.02.2012].
- [Par97] Lynne E. Parker. L-ALLIANCE: Task-Oriented Multi-Robot Learning in Behavior-Based Systems. *Advanced Robotics, Special Issue on Selected Papers from IROS '96*, 11(4):305–322, 1997.
- [Par98] Lynne E. Parker. ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.
- [Par02] Lynne E. Parker. Distributed Algorithms for Multi-Robot Observation of Multiple Moving Targets. *Autonomous Robots*, 12(3):231–255, May 2002.
- [PBF⁺11] Giovanni Pini, Arne Brutschy, Marco Frison, Andrea Roli, Marco Dorigo, and Mauro Birattari. Task partitioning in swarms of robots: An adaptive method for strategy selection. Technical report, Université Libre de Bruxelles, May 2011.
- [PC12] The Processing Community. Processing Homepage. Website, 2012. <http://processing.org/> [12.03.2012].
- [PCdDB10] Sébastien Paquet, Brahim Chaib-draa, Patrick Dallaire, and Danny Bergeron. Task Allocation Learning in a Multiagent Environment: Application to the RoboCupRescue Simulation. *Multiagent and Grid Systems*, 6(4):293–314, December 2010.
- [PKG⁺02] Giovanni C. Pettinaro, Ivo W. Kwee, Luca Maria Gambardella, Francesco Mondada, Dario Floreano, Stefano Nolfi, Jean-Louis Deneubourg, and Marco Dorigo. Swarm Robotics: A Different Approach to Service Robotics. In *Proceedings of the 33rd International Symposium on Robotics (ISR 2002)*, pages 71–76, October 2002.
- [PTO⁺11] Carlo Pinciroli, Vito Trianni, Rehan O’Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni A. Di Caro, Frederick Ducatelle, Timothy Stirling, Álvaro Gutiérrez, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a

- Modular, Multi-Engine Simulator for Heterogeneous Swarm Robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 5027–5034. IEEE, September 2011.
- [R D11] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2011.
- [RF12] The Robocup Federation. RoboCup Homepage. Website, 2012. <http://www.robocup.org/> [03.02.2012].
- [RT12] Randal R. Rucker and Walter N. Thurman. Colony Collapse Disorder: The Market Response to Bee Disease. *PERC Policy Series*, No. 50, January 2012.
- [Sam84] Hanan Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.
- [Sar07] Sanem Sariel. *An Integrated Planning, Scheduling and Execution Framework for Multi-Robot Cooperation and Coordination*. PhD thesis, Istanbul Technical University, June 2007.
- [SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- [SB06] Sanem Sariel and Tucker Balch. Efficient Bids on Task Allocation for Multi-Robot Exploration. In Geoff Sutcliffe and Randy Goebel, editors, *Proceedings of the 19th International Florida Artificial Intelligence Research Society (FLAIRS) Conference*, pages 116–121. AAAI Press, May 2006.
- [SBS06] Sanem Sariel, Tucker Balch, and Jason Stack. Distributed Multi-AUV Coordination in Naval Mine Countermeasure Missions. Technical report, Georgia Institute of Technology, April 2006.
- [SC09] Pedro Mitsuo Shiroma and Mario F. M. Campos. CoMutaR: A framework for multi-robot coordination and task allocation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009)*, pages 4817–4824. IEEE, October 2009.
- [SD99] Anthony Stentz and M. Bernardine Dias. A Free Market Architecture for Coordinating Multiple Robots. Technical report, The Robotics Institute, Carnegie Mellon University, December 1999.
- [Sel09] Brennan Sellner. *Proactive Replanning for Multi-Robot Teams*. PhD thesis, Carnegie Mellon University, January 2009.
- [Sha07] Amanda J. C. Sharkey. Swarm Robotics and minimalism. *Connection Science*, 19(3):245–260, August 2007.
- [SK11] University of Stuttgart and University of Karlsruhe. Swarmrobot: Open-source micro-robotic project. Website, 2011. <http://www.swarmrobot.org> [07.02.2012].

- [SMC07] Thomas Schmickl, Christoph Möslinger, and Karl Crailsheim. Collective Perception in a Robot Swarm. In Erol Şahin, William Spears, and Alan Winfield, editors, *Swarm Robotics*, volume 4433 of *Lecture Notes in Computer Science (LNCS)*, page 144157. Springer Berlin / Heidelberg, 2007.
- [Smi80] Reid G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.
- [SRI02] SRI International. Centibots Project Homepage. Website, 2002. <http://www.ai.sri.com/centibots/index.html> [04.02.2012].
- [SSB⁺05] Jörg Seyfried, Marc Szymanski, Natalie Bender, Ramon Estaña, Michael Thiel, and Heinz Wörn. The I-SWARM Project: Intelligent Small World Autonomous Robots for Micro-manipulation. In Erol Şahin and William Spears, editors, *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science (LNCS)*, pages 70–83. Springer Berlin / Heidelberg, 2005.
- [STBE09] Sanem Sariel-Talay, Tucker Balch, and Nadia Erdogan. Multiple Traveling Robot Problem: A Solution Based on Dynamic Task Selection and Robust Execution. *IEEE/ASME Transactions on Mechatronics*, 14(2):198–206, April 2009.
- [STBE11] Sanem Sariel-Talay, Tucker Balch, and Nadia Erdogan. A Generic Framework for Distributed Multirobot Cooperation. *Journal of Intelligent Robotic Systems*, 63(2):323–358, August 2011.
- [STM⁺09] Thomas Schmickl, Ronald Thenius, Christoph Möslinger, Gerald Radspieler, Serge Kernbach, Marc Szymanski, and Karl Crailsheim. Get in touch: cooperative decision making based on robot-to-robot collisions. *Autonomous Agents and Multi-Agent Systems*, 18(1):133–155, February 2009.
- [Str03] Ashley W. Stroupe. *Collaborative Execution of Exploration and Tracking Using Move Value Estimation for Robot Teams (MVERT)*. PhD thesis, Robotics Institute, Carnegie Mellon University, September 2003.
- [SW05] Malcolm Strens and Neil Windelinckx. Combining Planning with Reinforcement Learning for Multi-Robot Task Allocation. In Daniel Kudenko, Dimitar Kazakov, and Eduardo Alonso, editors, *Adaptive Agents and Multi-Agent Systems II*, volume 3394 of *Lecture Notes in Computer Science (LNCS)*, pages 260–274. Springer Berlin / Heidelberg, 2005.
- [SWSW12] P. S. Sreetharan, J. P. Whitney, M. D. Strauss, and R. J. Wood. Monolithic Fabrication of Millimeter-scale Machines. *Journal of Micromechanics and Microengineering*, provisionally scheduled for March 2012.
- [WM00] Barry Brian Werger and Maja J. Matarić. Broadcast of Local Eligibility for Multi-Target Observation. In *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS 2000)*, pages 347–356. Springer, October 2000.

- [Woo09] Michael J. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons Ltd, second edition, May 2009.