TEL AVIV UNIVERSITY
The Iby and Aladar Fleischman Faculty of Engineering
The Zandman-Slaner School of Graduate Studies

REINFORCEMENT LEARNING IN MULTI-ROBOT SWARMS

A thesis submitted toward the degree of
Master of Science in Mechanical Engineering

by

Yinon Douchan

October 2018

TEL AVIV UNIVERSITY
The Iby and Aladar Fleischman Faculty of Engineering
The Zandman-Slaner School of Graduate Studies

REINFORCEMENT LEARNING IN MULTI-ROBOT SWARMS

A thesis submitted toward the degree of
Master of Science in Mechanical Engineering

by

Yinon Douchan

This research was carried out at Tel Aviv University
in the School of Mechanical Engineering
Faculty of Engineering
under the supervision of Prof. Avi Seifert
and Prof. Gal Kaminka

October 2018

In multi-robot systems, robots cannot act without interactions and conflicts must be resolved. Such conflict is a spatial conflict; robots cannot share the same spot at the same time and must avoid collision. While there are many approaches to collision avoidance and resolution, every approach has its advantages and disadvantages. Recent promising work tries to address multi robot spatial coordination by adaptive selection of reactive coordination methods using an intrinsic reward function named the *Effectiveness Index (EI)* which is based on the resource spending rate of the robots. While it has many desirable characteristics in terms of multi-robot coordination and shows some empirical success, its success is only limited and there are no theoretical guarantees of optimality or convergence, as we indeed show. The contributions of this work are in several areas: First, we start by listing gaps between existing theory and practice that rise in the context of reactive arbitration of coordination methods. Then, we give theoretical modelling and practical solutions in order to bridge those gaps. The theoretical modelling starts by representing a task run as an extensive form game. It then goes to creating a connection between the system-wide performance for a task run and the choices of each robot in each collision. Finally, it deals with the issue of how robots should act in order to achieve optimal system-wide performance. The practical solutions further bridge gaps that rise from running multi-robot systems in real-world applications. The last part of this work puts the theoretical modelling and practical solutions to the test by experimenting with different multi-robot domains both with real robots and in a simulation.

Table of Contents

Nomenclature

Symbol	**Meaning**	**Equivalent in [27]**
$N = \{1, 2, \ldots, n\}$	The set of players	$A = \{a_1, \ldots, a_n\}$
S_i	The set of actions available for player i	M
s_i	A specific action taken by player i	α_i or α
$S = S_1 \times ... \times S_n$	The set of all possible action profiles	
$s = (s_1, ..., s_n) \in S$	An action profile: the joint action composed of the action s_i of player i	
$\{s^c\}_{c=1}^{c=C} = \{(s_1^c, ..., s_n^c)\}_{c=1}^{c=C}$	A sequence of action profiles	
$h^j = (s^1, s^2, ..., s^j) \in S^j$	The history of joint actions played up until the j'th collision	
$g_i : S^j \mapsto \mathbb{R}$	Gain of player i as a function of the joint actions played up until collision j	$gain$
$u_i \in \mathbb{U}^j : S^j \mapsto \mathbb{R}$	The utility of player i given the joint actions played up until collision j	$u_i(\alpha_i)$
$\mathbb{U} : S^j \mapsto \mathbb{R}^n$	The set of utilities of each player as a function of the joint actions played up until collision j	
$A_i : S^j \mapsto \mathbb{R}$	Active time of player i as a function of the joint actions played up until collision j	I_i^a
$P_i : S^j \mapsto \mathbb{R}$	Passive time of player i as a function of the joint actions played up until collision j	I_i^p
$T \in \mathbb{R}$	Game time	T

Table 1: List of symbols and notations used in this thesis. Where appropriate, we also list equivalent notation used in [27], which is closely related.

List of Figures

List of Tables

Introduction

Multi-robot systems are comprised of multiple autonomous robots, each with its own control. Typically, the robots are designed and deployed to work towards a common goal, carrying on a shared task. Examples of such multi-robot systems include multirobot coverage [1, 51, 20, 34, 7], patrolling [16, 2, 50, 37], foraging [27, 35, 14, 22, 5], formation-maintenance [26, 13, 28, 29, 6], and more.

Necessarily, the robots share resources (at the very least, the space of their work area), and thus a fundemental challenges is the challenge of *multi-robot coordination*. As robots cannot act completely independently of others, they must coordinate their actions with other robots in order to avoid and resolve conflicts over resource use. Such coordination necessarily introduces some overhead into the workings of the robots, either by design or ad-hoc. Multi-robot coordination therefore both supports and competes with achievement of the goals of the robots.

One of the most basic types of conflicts of resource usage in multi-robot systems is the *spatial conflict*. Robots cannot share the same spot at the same time and must avoid and resolve collisions. They must coordinate *spatially*, acting so as to not collide and continue their task normally, if a collision occurs.

One example domain where spatial coordination is key to the success of doing the task is *order picking*. Order picking is the task of collecting items, usually in a logistic warehouse, in order to compose orders which are composed by a collection of items ordered by customers. A very known real-world implementation of such system is Amazon Robotics' order picking system [49, 23]. This system was acquired by Amazon in its takeover of Kiva System (for 775 million dollars; Amazon's second-largest acquisition). This system was built to replace most human labor in a logistics warehouses[1]. In such setting, robots must engage in spatial coordination, e.g., while moving in the passageways along shelves,

[1] Human workers used to walk 20 kilometers a day to pick ordered items in shelves in such warehouses, and labor was in short supply.

or when arriving the packing stations with the collected items. While robots receive their pick orders tasks (where to go, what to pick, where to bring it) from a centralized server, they each carry out their own collision avoidance and resolution: Planning non-colliding paths by a centralized server is computationally intractable [52], and is not tolerant of human motions and mechanical unpredictability [49].

This thesis will focus on the challenge of designing distributed multi-robot coordination algorithms. First, in Chapter 2 we provide background and review related work on multi robot coordination describing the challenges in multi-robot coordination: what characteristics of coordination algorithms are desirable and current existing approaches to multi-robot coordination. We will highlight a recent promising approach to coordination—*coordination method arbitration*—whereby various coordination algorithms are used interchangeably by different robots at different times, as conflicts occur. While empirically showing promising, the technique has no theoretical underpinning and its results are unexplained. Chapter 3 provides a theoretical model of the online coordination method arbitration, grounding it in game theory. It connects between optimal performance of the system and choices of the robots in any time of the task run and show how robots can arbitrate in order to achieve optimal system-wide performance. It also shows the extent to which this model is relevant to real-world robotics. Chapter 4 discusses practical ways to bridge the gaps between the theoretical model and the reality of multi-robot systems, also showing how earlier work on coordination method arbitration is a special case. The theoretical modeling and practical solutions will then be put to the test in Chapter 5 by experimenting in two multi-robot domains: physical robots carrying out a variant foraging task, and simulated robots carrying out order-picking, using the Alphabet Soup simulator [23] developed by the Kiva Systems team. Chapter 6 provides conclusions.

Multi-Robot Coordination: Background

Approaches to Multi-Robot Coordination: Brief Overview

There are several approaches to multi robot coordination algorithms, each with its own advantages and disadvantages. Desirable characteristics of coordination algorithms include the following:

- Computational efficiency: Multi-robot coordination often requires decision making in real time. Even when this is not critical, the computational efficiency of the algorithms have to be feasible in practice. For instance, the task allocation algorithms used in Amazon Robotics to decide which package or item will be picked and delivered are carried out by a central server, and obviously represent a feasible solution for this component of the coordination between the robots. On the ther hand, the computation of collision-free paths for the hundreds of robots is not centrally carried out, because of the computational load involved.

- Population changes: Addition or removal of robots to the system (robot birth, robot death) should be allowed by the coordination algorithm. New robots should assist, removed robots should not hinder performance.

- Scalability: The methods should continue to perform well regardless of the group size.

- Robustness to failure: An unexpected failure of one or more robots should not render the system inoperable or affect group performance significantly.

- Little or no communicational requirements between robots: Communication between robots is an additional layer of complexity in the system and it consumes energy. Therefore, one should minimize the need for communication.

- Domain independence: We desire one or few algorithms for many different multi-robot tasks, avoiding the need to specifically tailor each coordination algorithm to a specific domain and configuration.

- Analytical guarantees, empirical evidence of optimality in terms of group performance: This characteristic is of paramount importance. It is the final seal of approval that the algorithm is both conceptually right (has performance guarantees) and works in practice (empirical evidence).

We cannot do justice to a full survey of multi-robot coordination, especially since many of the relevant work is reported in manner where coordination and task algorithms are intimately coupled. Some surveys of interest may be found elsewhere [17, 33]. We will provide instead an overview (with some examples) of the key approaches, alongside brief discussions of advantages and disadvantages vis-a-vis the desired characteristics described above.

Pre-Deployment (Planning-Time) Coordination. In this approach, the work area itself and the behavior of the robots are designed such that the need for coordination is eliminated or at least minimized. For example, Fontan and Mataric [18] report on an algorithm that pre-allocates robots to different territories. Each robot operates in its territory but has the ability to pass objects to another, thus creating a bucket brigade like structure. They also discussed re-allocating the territories once a robot fails. The same general approach is taken by Elmaliach et al. [16, 15] and Agmon et al. [3] in allocating robots to patrolling areas and perimeters, such that robot spatio-temporal trajectories never intersect. Likewise, Locker-Room Agreements follow the same general approach, here for robot soccer [39].

In general, pre-deployment coordination can be very effective, when robots can be assumed to maintain their pre-computed restrictions. The planning and design is most commonly done by hand, rather than automatically. The computational *in*tractability of composing conflict-free "traffic laws" (motion constraints, such as "travel on right side of the road") for robots has been shown by Shoham and Tennenholtz [38] even in simple grid-worlds. Indeed, planning optimal collision-free trajectories for multiple robots is computationally intractable [52].

Post-Deployment (Execution-Time) Coordination. A different approach for handling coordination focuses on handling conflicts as they arise, or at least during execution, in an effort to address the computational cost of planning resource usage and coordination ahead

of time. This approach also addresses the assumption of planning methods, that robot will be able to follow through on their planned motion constraints, and that no changes to the scale or components of the system will be made.

Within this approach, some algorithms consider the overall goal of the robot, and respond within the context of this goal. For example, in navigation, the *Dynamic Window* algorithm [19] is a coordination method that uses limited planning in the space of admissible velocities. This method is capable of making decisions based not only on external constraints like obstacles and other robots, but also on internal constraints like maximal velocity and acceleration. We are using a dynamic window variant as one of the algorithms in our work.

A popular class of navigation methods which do some planning is the *Reciprocal Velocity Obstacles* (RVO) class of methods. This class plans ahead based on the space of admissible relative velocities to nearby obstacles and robots. A very popular RVO method is the *Optimal Reciprocal Collision Avoidance* (ORCA) [43], which guarantees collision-free paths, as long as (1) all robots use ORCA, and (2) all robots know other robots' shape and velocities. It does not guarantee any optimality of either a system-wide or individual goal (e.g., the makespan or cumulative distance travelled). Other algorithms focus on safety and provide better guarantees. For example, the *Passively Safe Partial Motion Planning* (PassPMP) algorithm[8].

More abstractly, Stone et al. [40] model the ad-hoc coordination problem from a game-theory persective. They show a method by which a robot can cause its teammates, without communicating with them, towards a globally-optimal coordinating solution. However, the model relies on modeling the payoffs (rewards) associated with all actions of all robots, and is intended as a theoretical exploration alone. It also assumes a single robot is driving the change, while others only respond.

At the extreme of the post-deployment approach lie *reactive coordination algorithms*. These are algorithms that respond to a collision, with no or very little planning with respect to the task goal of the robot, or the group, i.e., these are necessarily *myopic* algorithms. On the other hand, such algorithms are extremely simple to implement and use (both in practice and from a computational perspective), and are generally task-independent (because they do not use information about the goals of the task).

We use several such reactive algorithms in our work. One reactive algorithm is the *noise* algorithm by Balch and Arkin [6]. Given a collision, a robot repels itself backwards with some directional noise. In [35] the *repel* method is described. As the name suggests, once a robot collides with another robot it repels itself backward for an arbitrary time

interval or distance.

More sophisticated algorithms introduce stochasticity into the decision-making. A reactive algorithm named *aggression* was described by Vaughan et al [44]. When robots use this coordination method, the robot with the highest "aggression factor" gets the right of way while the other backs off. They describe three possible ways to determine the aggression factor of the robot - randomly, fixed or based on the robot's free personal space behind it. A related approach, by Danassis and Faltings [12] is called CA^3NONY, and is intended for domains where an optimal behavior will be to anti-coordinate[1], i.e. that each agent must choose an action which differs from other agents' actons in order for the outcome to be optimal. Here, agents are being *courteous*: If an agent collides with another agent, i.e. chooses the same resource at the same context, it backs off from this choice with a constant probability. In addition to this social convention the agents maintain a distributed bookkeeping scheme which prevents them from monopolizing resources, causing each agent to choose only one resource for one context. While this algorithm does not require any communication between agents and guarantees optimal behavior, it holds several assumptions on the context space and the reward structure. The algorithm assumes that the context space is discrete, repeats itself periodically and is shared between all agents. It also assumes that the marginal reward is decreasing.

It is now understood that while each method is effective in some settings, no method is always effective [35, 36]. The results in foraging show that the system-wide utility of a specific coordination method depends on the density of the system. For all methods, the system-wide utility decline once some density is reached. But the density in which this occurs differs from one method to the next. Certainly, some methods do better than others—but none are superior to others in all densities.

The behavior of the group in the investigations above mimics the *Law of Marginal Returns* in economics: Adding more robots does not necessarily increase productivity. Goldberg and Mataric [22] had attempted to capture the cause for this, by defining *interference*, a global signal which varies in the working space of the system denoting how much robots interfere with each other, e.g., due to lack of coordination. They suggested that by picking this global signal the robots may act accordingly. The problem is that in practice, this signal cannot be easily computed (as it involves internal measurements from each robot) or made public without communications. Furthermore, no theoretical connection has been made between interference and task performance.

[1] The definition of coordination in [12] differs from the definition of coordination in our work. We define coordination as the need to take an action due to an interaction between agents. They define coordination as a *consensus*: where agents need to choose the same action in order to achieve optimal results.

Learning and Adaptation for Improving Multi-Robot Coordination

Measures of coordination (such as *interference*, above), can be used to guide actions to improve coordination, e.g., through learning. For example, Rosenfeld et al. [35] show that for a fixed group size, areas of high density of robots correlate negatively with group performance, in a multi-robot foraging task. In addition, the higher the cost robots invest on coordination methods the less the group performance will be. They define the *likelihood of collision* around a robot as the ratio between the area of a circle of fixed radius around it and the total area robots take inside this circle. They represent cost of coordination by the *combined coordination cost* (CCC), a weighted average of all coordination costs of a robot like time and fuel. They show a strong negative correlation between the CCC and group performance for a fixed group size.

Rosenfeld et al. [35] also proposed an offline adaptive algorithm for the problem of multi-robot coordination, based on their CCC measure. This algorithm arbitrates between a set of coordination methods by using methods with larger CCC when the likelihood of collision is high and methods with lower CCC when the likelihood of collision is low. It does so by sorting the set of coordination methods from the one with lowest to the one with highest CCC and sets thresholds based on likelihood of collision to determine what method to choose. The adaptation was done by tuning the aforementioned threshold. They used two separate variants for this adaptation: Hill climbing and gradient learning, each one of them tunes the thresholds differently based on the group performance. The CCC measure was not explored theoretically, despite the empirical success of using it as the basis for learning (offline).

More generally, there is much work on utilizing learning to improve multi-robot (and multi-agent) coordination, mostly focusing on *reinforcement learning*, which is often used in the context of planning and decision-making. Indeed, this is the approach we take in this thesis: to improve coordination by using learning to adjust which reactive coordination method is to be used in each conflict. We only briefly describe it here, and refer the reader to [45, 41, 30, 25] for a deeper explanation. There are several investigations that are closely related to this approach, which we describe below in detail.

Reinforcement Learning (RL) is an area of Machine Learning, inspired by biology and behavioral psychology, where an agent, or more than one agent, takes actions sequentially; it receives a scalar signal (the *reward*). This scalar signal is then used by the agent as a part of a feedback mechanism to direct it towards better actions, where a better action is an action with a larger rewards. Most commonly, RL algorithms seek to maximize the accumulated rewards of the agents. Many such algorithms have proven optimality and

convergence guarantees.

Claus and Boutilier [11] show different variations of RL techniques in multi agent domains and the difficulties that rise using them. They divide learners to two different types: Independent learners (IL) and joint-action learners (JAL). ILs learn actions with no knowledge of actions of other agents while JALs learn with knowledge of the actions of all other agents. To ground RL use in multi-agent systems, Claus and Boutilier discuss learning in the context of game theory models. The show that even simple RL algorithms (such as a stateless version of Q-Learning, perhaps the most well-known RL algorithm), lead to non-intuitive results, depending on the settings of the game. In particular, they examine both IL and JAL agents in several identical-interest matrix games (where in every action profile every agent gets the same utility). For both IL and JAL they show that the agents converge to a Nash equilibrium, which is sub-optimal in terms of welfare. They also show that different learning parameters such as the learning rate or exploration rate can make the system converge to different equilibrium points. As we are interested in maximizing the global utility (the group goal), this is a serious challenge, which has been undertaken in many investigations.

Godoy et al [21] show that using simplistic reinforcement learning techniques with ORCA can make a multi-robot system achieve a better system-wide goal than with either using only ORCA or only the simplistic reinforcement learning techniques. They present the ALAN framework which uses a reinforcement signal composed of two factors: A goal-oriented and a politeness factor. The goal oriented factor is based on the direction cosine of the velocity vector of the robot and the displacement vector of the goal from the robot. The politeness factor is based on the vector cosine between the preferred velocity vector and the vector ORCA will output in the current robot's situation. The final reinforcement signal for the ALAN framework is a weighted sum of the goal-oriented factor and the politeness factor. This work has both similarities and dissimilarities to our work. In a similar manner to our work, this work uses reinforcement learning in order to choose the best action in any given time. Unlike our work, this work does not focus on reactive algorithms but rather builds upon a planning algorithm (ORCA). Furthermore, ALAN does not provide guarantees on task performance.

Wolpert and Tumer [48] described the Collective intelligence (COIN) framework. COIN framework models a system of multiple agents where there is little or no communication between agents and the agents are working towards a common global utility. Each agent in a COIN framework is a learning agent and emphasis is put on reinforcement learning. By taking into account the states of all agents at all times COIN framework defines several characteristics that a system should have for it to be a COIN. Those char-

acteristics are related to how well an agent learns (intelligence), the relation between how an agent's action affects his utility vs how other agents' actions affect the agent's utility (learnability), how the global utility of the system changes due to a change in an agent's personal utility (factoredness) and how a global utility is affected by an agent not existing at all (wonderful life utility). They show success in many multi agent domains [42, 46]. However, their framework relies on knowing the global utility, and/or the value (payoff) of others' actions. In practice, this is not often possible.

Most closely related to our work is the work by Erusalimchik et a. [27], which advocates the use of a reward called Effectiveness Index as the basis for coordination adaptation (rather than learning). As we build and extend this work significantly, we begin with a detailed overview of its workings.

A reactive robot takes action only when the need to avoid a conflict is immediate. If no conflict is imminent the robot focuses only on its task as if it was independent of other robots. Therefore, a task run for a reactive robot should have a specific structure. First, when it starts the run, it focuses on the task. Once it recognizes an imminent collision, it focuses on conflict avoidance for a specific time interval. When the robot decides the conflict has been averted, it returns to normal task execution up until next collision is imminent. This cycle continues up until task execution has ended. A graphical depiction of the timeline from this perspective is shown in Figure 2.1.

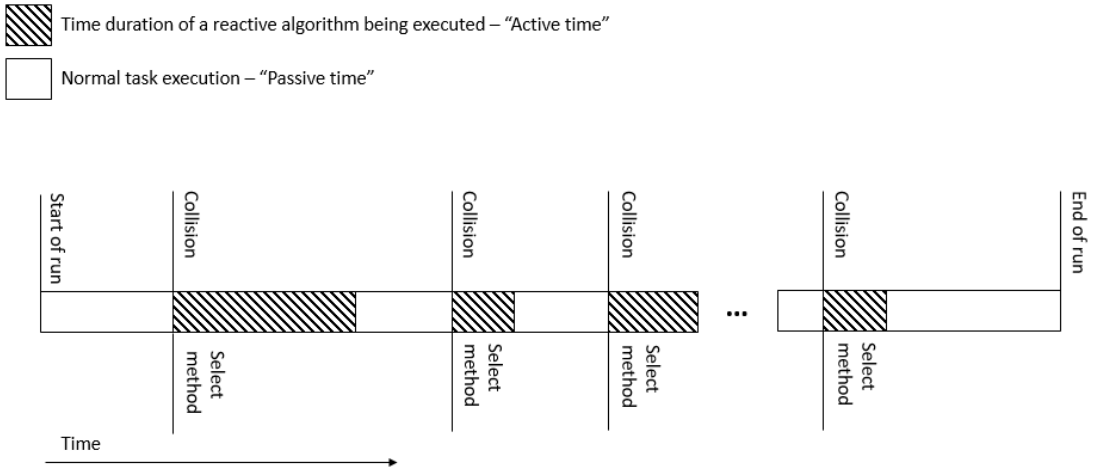

Figure 2.1: Task life cycle from the perspective of a reactive coordination method.

The time interval between a collision and the next one is split into two intervals. Erusalimchik et al. [27] coined terms for these: For the collision avoidance interval they coined

the term *active time*, as the reactive coordination method is being activated. The task execution interval is termed *passive time*, as from the coordination perspective, nothing is being done (there is no need to coordinate). The active time is the time spent by the robot executing a coordination method. The *passive time* is a time interval where the robot focuses on the task to be done.

The approach taken in [27] relied on the stateless version of Q-Learning [11], and learns which reactive coordination method to use. To do this, it uses—as the reward signal—the *effectiveness index* of the methods: the ratio between the coordination costs and the total time interval between the current collision and the next. The coordination costs are composed from the active time and other costs such as fuel and the total time between collisions is simply the sum of the active and passive times (Figure 2.2).

The structure of the task run with arbitration now looks as follows. When a robot detects that it is about to collide, it then carries out the following procedure:

1. Chooses a coordination method α based on its EI value, using the RL algorithm.

2. Performs coordination method immediately (and keep track of the active time duration)

3. Return to task execution until another collision is imminent (keeping track of the passive time)

4. Compute the EI of α and store it.

5. When a new collision in imminent, go to step 1.

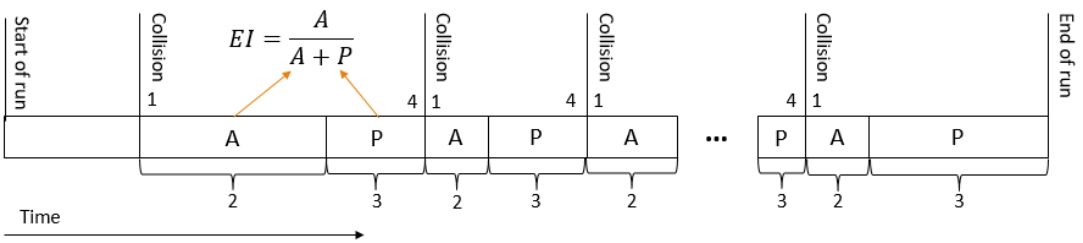

Figure 2.2: Task life cycle from the perspective of an EI learning coordinating robot.

Eruslimchik et al. [27] show that this procedure resulted in significantly improved performance in several domains. Furthermore, the effectiveness index combined with stateless Q-Learning has many of the desirable characteristics described therein:

- Being computationally light. Only a small calculation of a simple formula once in a conflict.
- Requires no communication. It is an intrinsic reward
- Domain independent. Different multi-robot domains have different measurements of performance and a performance measurement in one domain will probably be impossible to be used in another domain due to its domain-specific nature. Since Effectiveness index relies instead on properties common to all domains such as time investments and costs. It can be used everywhere.

Despite the empirical success of the EI measurement using reinforcement learning, it comes with no guarantees. Indeed, our research work began by applying the framework to the pick ordering domain, which turned out to be not at all trivial or necessarily successful [14]. We therefore sought to ground EI in theory, and along the way developed a more general model and family of rewards, which provide guarantees up to explicit assumptions, as well as a thorough discussion of approximation methods which can be used in practice, and are motivated by the theory.

A game-theoretic view of multi-robot swarms

We begin in Section 3.1 by introducing an abstract game-theoretic model of multi-robot tasks carried out by a swarm of robots. We then make incremental modifications to this abstract model, to bring it closer to the reality of physical interacting robots, when the robots cannot communicate (Sections 3.2–3.3). Finally, in section 3.4, we address the challenge of learning optimal actions according to the game-theoretic model we introduced.

Extensive form game representation of a task run with reactive method arbitration

When considering the task multiple robots (each engaging in its own coordination method arbitration), we follow Erusalimchik et al. [27] in representing the task as an extensive form game between n robots. The extensive form game represents every possible outcome as a function of the sequence of parallel coordination actions taken by all robots in every collision during the run. In this context, the outcome is the utility of each of the robots in the system gained in the alloted game time.

The root node of the game tree represents the first collision. Given that there are n robots, the first n layers of the game tree will each represent a robot and its possible actions in the first collision. This is because we focus on non-communicating coordination methods, and thus we will treat each collision as having no information on the actions and utilities selected and gained by other robots.

The actions independently taken by players are coordination methods: The gains (payoffs) from taking them and the costs which they entail differ between robots and between collisions, but are theoretically accounted for.

The next n layers will represent the second collision in the same manner. This sequence continues until a terminal node is reached—when the time for the task is done: A terminal node represents the end of the game (task) and holds the utility of each player. Since different actions can yield different time intervals between collisions, terminal nodes can each be of different depth depending on the sequence of collisions (and associated joint

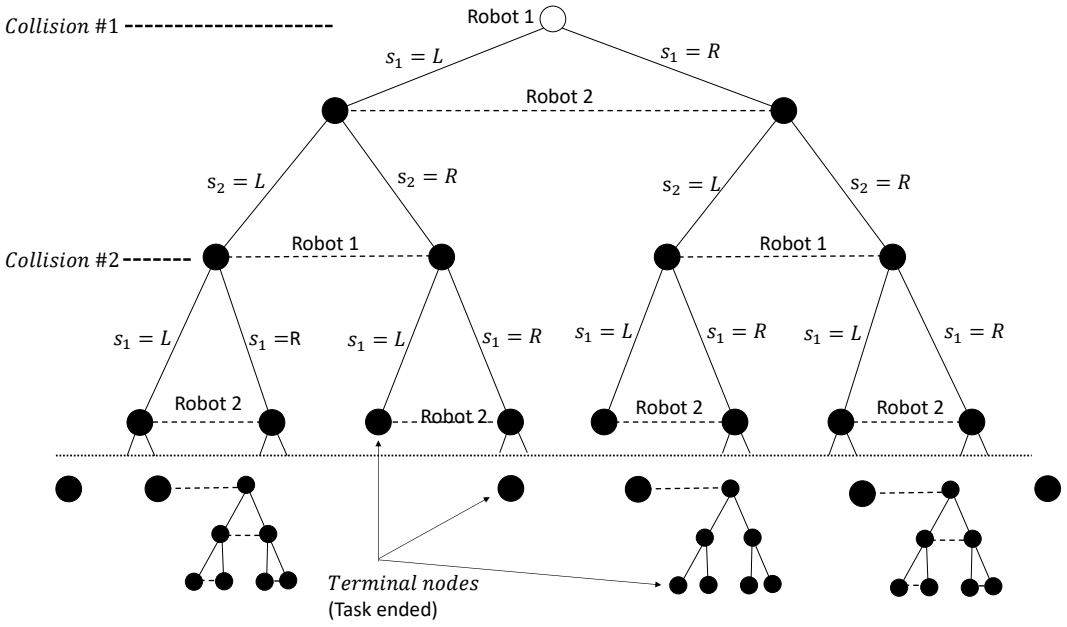

Figure 3.1: A two-player two-action task run represented as an extensive form game for the action sets $S_1 = S_2 = \{L, R\}$. Not all terminal nodes have the same depth, as different joint actions taken by the players can lead to more or less collisions.

actions chosen) during the game. Each such sequence is represented as a path in the game tree. Each terminal node will hold a vector of numerical values representing the utilities of each robot in the system.

A two-player two-action example of such an extensive form game is shown in Figure 3.1. It shows several paths from the root node to the terminal nodes.

Folding an extensive-form game to a sequence of normal-form games

The extensive form model of a task run represents every possible outcome of the task run. This is obviously of theoretical value only, as no robot—nor their designers—can predict the outcome of future collisions, nor their timing, nor their impact on global payoffs. In reality, robots only know their history of previous collisions, and the immediately imminent collision. Indeed, in many settings robots cannot know of the other robots' choices (which theoretically affects their own) and thus even this information is hidden from them.

In order for robots to make decisions based only the history and current collision, we must draw a connection between the global final utility (payoff) theoretically reached

using the extensive-form game, and the sequence of collisions in which the robots make collision-resolution choices. Robots may then rely on signals that are obtained during a joint collision.

To do this, we take an intermediate step and show how the extensive form game can be expressed as a sequence of normal form games, each representing a single joint collision. We first start with a few definitions:

- Robot i's action at the j'th collision will be denoted as s_i^j. The joint action of the robots at the j'th collision will be denoted as s^j.
- Robot i's history of actions until the j'th collision inclusive will be denoted as $h_i^j = (s_i^1, s_i^2, ..., s_i^j)$. The history of actions of the robots up until the j'th collision inclusive will be denoted as $h^j = (s^1, s^2, ..., s^j)$.
- The *cost* of robot i at the j'th collision will be denoted as c_i^j.
- The *gain* of robot i at the j'th collision will be denoted as g_i^j.
- The *utility* of robot i at the j'th collision will be denoted as u_i^j and is the difference between this robot's gains and costs at the j'th collision: $u_i^j = g_i^j - c_i^j$.
- The *global utility* of all the robots during the whole task run is denoted as U

We start with the most general case where outcomes of a robot at the j'th collision may depend on the entire history of play of all the robots up until collision j inclusive. This means that u_i^j, g_i^j, c_i^j are all functions of h^j. U will now depend on the entire history of play. Given that the number of collisions for the whole task run is c, U will be a function of h^c and will be defined as the sum of utilities of every robot and every collision during the task run (Eq. 3.1).

$$U(h^c) = \sum_{i \in N} \sum_{j=1}^{c} u_i(h^j) = \sum_{i \in N} \sum_{j=1}^{c} (g_i(h^j) - c_i(h^j)) \tag{3.1}$$

We can look at each joint collision as a normal-form (matrix) game representing the outcomes of this collision only, rather than the whole task run. For the j'th collision, the player set of this matrix is the set of robots performing the task and the action set of each

robot is its set of available coordination methods for this collision. Given the history of joint actions played up until collision j (inclusive), h^j, the payoffs of this matrix will be the sum of the utilities of the robots obtained only for the j'th collision $\sum_{i \in N} u_i(h^j)$ as a function of the history of play. We call this matrix the *Folded Game Matrix* (see Figure 3.2 for an illustration).

We define the $(\cdot)$ operator between a play history and a new joint action to be the concatenation of the new joint action to the history. For $h^{j-1} = (s^1, ..., s^{j-1})$ and s^j, $h^j = h^{j-1} \cdot s^j = (s^1, ..., s^{j-1}, s^j)$.

	$s_2 = L$	$s_2 = R$
$s_1 = L$	$u_1(h^{j-1} \cdot (L,L)) + u_2(h^{j-1} \cdot (L,L))$	$u_1(h^{j-1} \cdot (L,R)) + u_2(h^{j-1} \cdot (L,R))$
$s_1 = R$	$u_1(h^{j-1} \cdot (R,L)) + u_2(h^{j-1} \cdot (R,L))$	$u_1(h^{j-1} \cdot (R,R)) + u_2(h^{j-1} \cdot (R,R))$

Figure 3.2: An example of a two-player two-action folded game matrix for the action set $S_1 = S_2 = \{L, R\}$.

Global Utility and the Folded Game Matrices

Since robots in a system have limited sensing and communication capabilities, they are unable to know the utilities of other robots, even in the same joint action. Indeed, each robot does not even know how its own action affects its own immediate utility. The only information available to it is data from its own sensors and internal state information.

Previous work by Erusalimchik et al. [27] has examined using the *Effectiveness Index* (EI)—the ratio of active time to total cycle duration (since the last conflict) as a substitute for the robot's estimate of its utility. In particular, minimizing EI was proposed to be an alternative to maximizing the robot's utility. However, this conjectured connection was not satisfactorily proven.

We now take steps to formally tie the active and passive times of the collision to the utility of the robot resulting from the collision. We make the following observations and assumptions:

- Gains in active time are zero. When a robot is in active time it focuses on handling conflicts and not on the task and therefore, cannot contribute to it. For example, in foraging a robot can not retrieve a puck when focusing on avoiding collisions.

Therefore, we assume that gains occur only in passive time. This assumption can be expressed as $g_i(h^j) = g_i(P_i(h^j))$.

- We will further assume that the gains are linear to the passive time given history of play h^j. This can take several forms:

  1. $g_i(h^j) = \alpha(h^j)P_i(h^j)$. Given a play history up until time j, the gains will be linear in passive times. We stated that gains occur only in passive time. Therefore, it should at least be a function of the passive time.
  2. $g_i(h^j) = \alpha P_i(h^j)$ where $\alpha = const$. This is an even stronger assumption than the above that states that the rate of gain will be the same for every joint action. This claims that a change in action will not yield a higher rate of gain during the passive time but instead will cause different passive time length.

- We will also assume that costs are constant throughout the task run meaning that $c_i(h^j) = \beta(A_i(h^j) + P_i(h^j))$ where $\beta = const$.

Previous work by Rosenfeld et al [35], shows that there is a strong correlation between coordination costs (only the active time in our case) and group performance. The more a robot, or a group of robots invest on the task (passive time), the lower are their costs (active time) and the higher is their performance. Therefore, the gains of the system, and each robot individually, are proportional to the total passive time of the system. For robot i, this takes the form of $g_i(h^j) = \alpha P_i(h^j)$ (assumption option 2 on the gain).

Global Utility and Coordination Overhead

Definition 3.3.1. The *Coordination Overhead (CO)* is the total amount of time the system was in active time divided by the total time invested in the task run: $CO(h^c) = \frac{1}{T}\sum_{i\in N}\sum_{j=1}^{c} A_i(h^j)$.

Since T is the sum of all cycle length of any of the robots' task run, we can write $T = \sum_{j=1}^{c}(A_i(h^j) + P_i(h^j))$ for any robot i. Therefore, CO can also be written as $CO(h^c) = \sum_{i\in N}\frac{\sum_{j=1} A_i(h^j)}{\sum_{j=1}^{c}(A_i(h^j)+P_i(h^j))}$.

We will now show, given the above assumptions, that the global utility U is now a linear decreasing function of CO:

Theorem 3.3.1. *Given the assumptions on the cost and gain, U is a linear decreasing function of CO.*

Proof.

$$
\begin{aligned}
U &= \sum_{i\in N}\sum_{j=1}^{c}[u_i(h^j)] = \sum_{i\in N}\sum_{j=1}^{c}[g_i(h^j)-c_i(h^j)] \\
&= \sum_{i\in N}\sum_{j=1}^{c}[\alpha P_i(h^j)-\beta(A_i(h^j)+P_i(h^j))] \\
&= \sum_{i\in N}\sum_{j=1}^{c}(\alpha P_i(h^j)) - \sum_{i\in N}\sum_{j=1}^{c}\beta(A_i(h^j)+P_i(h^j))] \\
&= T\frac{\alpha\sum_{i\in N}\sum_{j=1}^{c}P_i(h^j)}{T} - nT\beta \\
&= T\alpha(1-CO(h^c)) - nT\beta \\
&= -T\alpha\cdot CO(h^c) + T(\alpha - n\beta)
\end{aligned}
$$

□

As a result of this connection, now it is possible to look at our problem as minimizing CO rather than maximizing U. Although there is now a connection between U and CO, it does not give information about how robots should choose their actions in a way that CO is minimized.

Connecting Coordination Overhead to the Folded Game Matrices

We further assume that for every collision, the outcomes of the robots' method selection depend only on the current joint action performed and not on the history of all joint actions performed. This also means that no matter what the collision index is, as long as the joint action stays the same, the outcomes of this collision stay the same. Therefore, variables that depend on the history of joint actions played until collision j, $h^j \in S^j$, depend only on the joint action that was played in time j, $s^j \in S$. We can now denote the active time as $A_i(s^j)$ and since it does not vary in time, we can denote it as $A_i(s)$. The same applies for P_i, g_i, c_i, u_i and U.

One consequence of this assumption is that instead of the task run having a big set of different folded game martices depending on the history of play, it has only one folded game matrix which is the same for every collision in the task run.

	$s_2 = L$	$s_2 = R$
$s_1 = L$	$u_1((L,L)) + u_2((L,L))$	$u_1((L,R)) + u_2((L,R))$
$s_1 = R$	$u_1((R,L)) + u_2((R,L))$	$u_1((R,R)) + u_2((R,R))$

Table 3.1: An example of a two-player two-action conflict matrix with a generic utility $u(s)$.

In the previous section we saw that maximizing *CO* maximizes the global utility. Using the above assumptions we can write:

$$CO(h^c) = \sum_{i \in N} \frac{\sum_{j=1}^{c} A_i(s^j)}{\sum_{j=1}^{c} [A_i(s^j) + P_i(s^j)]}$$

Given a joint action s and a robot i, we will define $EI_{tot}(s)$ to be the sum of the effectiveness indices of all robots: $EI_{tot}(s) = \sum_{i \in N} EI_i(s) = \sum_{i \in N} \frac{A_i(s)}{A_i(s)+P_i(s)}$. Let s^* be the joint action that minimizes EI_{tot}: $s^* = argmin_s(EI_{tot}(s))$. If the system always plays joint action s^* its *CO* will be: $CO(h^*) = \sum_{i \in N} \frac{c \cdot A_i(s^*)}{c \cdot [A_i(s^*)+P_i(s^*)]} = \sum_{i \in N} \frac{A_i(s^*)}{A_i(s^*)+P_i(s^*)} = EI_{tot}(s^*)$ where $h^* = (s^*, s^*, ..., s^*)$. We will now show that for every sequence of joint actions *CO* will be greater or equal to $EI_{tot}(s^*)$. This means that in order to minimize *CO* the system always needs to select s^* as the joint action.

Theorem 3.3.2. *for any number of collisions c and histories of play h^c, $CO(h^c) \geq EI_{tot}(s^*)$.*

Proof.

$$\begin{aligned}
CO(h^c) &= \sum_{i \in N} \frac{\sum_{j=1}^{c} A_i(s^j)}{\sum_{j=1}^{c} (A_i(s^j) + P_i(s^j))} & &= \sum_{i \in N} \frac{\sum_{j=1}^{c} A_i(s^j)}{T} \\
&= \frac{1}{T} \sum_{i \in N} \sum_{j=1}^{c} A_i(s^j) & &= \frac{1}{T} \sum_{j=1}^{c} \sum_{i \in N} A_i(s^j) \\
&= \frac{1}{T} \sum_{j=1}^{c} l(s^j) \sum_{i \in N} \frac{A_i(s^j)}{l(s^j)} & &= \frac{1}{T} \sum_{j=1}^{c} l(s^j) EI_{tot}(s^j) \\
&\geq \frac{1}{T} \sum_{j=1}^{c} l(s^j) EI_{tot}(s^*) & &= EI_{tot}(s^*) \frac{1}{T} \sum_{j=1}^{c} l(s^j) \\
&= EI_{tot}(s^*) \frac{1}{T} T & &= EI_{tot}(s^*)
\end{aligned}$$

□

Now we know what the system needs to do in order to perform best. However, robots can only know internal properties and therefore cannot know s^* since it requires to know the actions of other robots. Therefore, we need to find a way to make the robots converge to s^* by using internal measurements, without requiring knowledge of coordination methods selected and utilities obtained by other robots.

Learning Optimal Actions

Potential games

As we have seen in the previous section, achieving optimal group performance is done by converging to s^*. We therefore need a learning mechanism for each robot that will guarantee system-wide convergence to s^*. This is not a trivial task due to the robots' inability to know the actions and rewards of other robots. Therefore, for each robot this learning mechanism must be based only on intrinsic data such as its individual actions and rewards.

We show that the use of Potential Games [32] can solve the challenge of converging to s^* while requiring only intrinsic data from each robot. A potential game is a normal form game where for every player i, the difference in the payoff of every unilateral deviation of player i's action s_i is related to the difference of a single potential function $\psi(s)$ mapping joint actions to a scalar. The potential function can be seen as a global signal (not necessarily visible to the players) which depends on the joint action. There are several types of potential games with differing strength. We will present three types of those games in descending strength.

Definition 3.4.1. Exact potential game. A game with player set N, action set S and utility function $\mathbb{U}$ is an exact potential game if there exists a potential function $\psi : S \mapsto \mathbb{R}$ such that for every player i and actions s_i, s_i': $u_i(s_i', s_{-i}) - u_i(s_i, s_{-i}) = \psi(s_i', s_{-i}) - \psi(s_i, s_{-i})$.

Definition 3.4.2. Weighted potential game. A game with player set N, action set S and utility function $\mathbb{U}$ is an exact potential game if there exists a potential function $\psi : S \mapsto \mathbb{R}$ and a weight function $w \in R_n$ such that for every player i and actions s_i, s_i': $w_i(u_i(s_i', s_{-i}) - u_i(s_i, s_{-i})) = \psi(s_i', s_{-i}) - \psi(s_i, s_{-i})$.

Definition 3.4.3. Ordinal potential game. A game with player set N, action set S and utility function $\mathbb{U}$ is an exact potential game if there exists a potential function $\psi : S \mapsto \mathbb{R}$ such that for every player i and actions s_i, s_i': $u_i(s_i', s\_i) - u_i(s_i, s\_i) > 0 \iff \psi(s_i', s\_i) - \psi(s_i, s\_i) > 0$.

It is straightforward to see that any exact potential game is also a weighted and an ordinal potential game and that any weighted potential game is an ordinal potential game. It is also straightforward to see that the opposite is not always true.

Potential games hold several characteristics that normal form games do not necessarily hold:

- Potential games always have at least one pure-strategy Nash equilibrium.
- When players use pure strategies a change in one player's individual payoff due to changing its individual action will be aligned with the potential function. This means that in any potential game if one player chooses to change its action to a better action in terms of his payoff, the potential function will always benefit, vice versa.

If the players take turns and choose one by one the best action in terms of their individual payoffs, the system will converge to a pure-strategy Nash equilibrium and that Nash equilibrium is at least a local optimum of the potential function. This means that there exist simple learning techniques based on each player's payoffs such that if each player uses them, the whole system converges to a Nash equilibrium.

Therefore, in the context of multi-robot coordination, we need to find a reward function for each robot based on its intrinsic measurements in a way that the robots play a potential game with potential function EI_{tot}. Doing so will make the robots converge to an optimal joint action in terms of EI_{tot}.

Total EI as a Potential Function

In section 3.3 we've seen that the problem of optimizing the global utility narrows down to minimizing EI_{tot} by converging to a single joint action s^* while still using only internal measurements.

In order to do so we use the *Wonderful Life Utility (WLU)* first discussed in the work done by Wolpert and Tumer as a part of the *Collective Intelligence (COIN)* framework [47]. Given a global utility U, the WLU for robot i is a measurement of the difference between the resulting U and the global utility when robot i is absent. In terms of game theory, the absence of robot i is equivalent to the robot choosing a "null" action denoted by ϕ_i.

Definition 3.4.4. Wonderful Life Utility. Given a global utility U and a joint action $s = (s_i, s\_i)$, the wonderful life utility of robot i is:

$$WLU_i^U(s_i, s_{-i}) = U(s_i, s_{-i}) - U(\phi_i, s_{-i})$$

The WLU is, in fact, a measurement of robot i's marginal contribution to U. It is known that agents that learn with WLU as a utility function play a potential game with the global utility as the potential function [4, 31].

Theorem 3.4.1. *If players play with WLU as a payoff over a global utility U they play an exact potential game with $\psi = U$ as a potential function.*

Proof. We will look at a unilateral change in robot i from action s_i to action s_i' given the action profile of others s_{-i}. The change in WLU_i will be:

$$\begin{aligned} WLU_i^U(s_i', s_{-i}) - WLU_i^U(s_i, s_{-i}) &= U(s_i', s_{-i}) - U(\phi_i, s_{-i}) - U(s_i, s_{-i}) + U(\phi_i, s_{-i}) \\ &= U(s_i', s_{-i}) - U(s_i, s_{-i}) = \psi(s_i', s_{-i}) - \psi(s_i, s_{-i}) \end{aligned}$$

□

Since our goal is to optimize EI_{tot}, we can now make the robots choose actions according the WLU of EI_{tot}. If robots do so, not only they will converge to a joint action, the potential function will be $\psi = EI_{tot}$. Therefore, this joint action will at least be a local minimum of EI_{tot} due to the properties of potential games. We will now show a closed expression of $WLU_i^{EI_{tot}}$ in order to see according to what utilities robots should learn in order to converge to this minimum.

Theorem 3.4.2. *let $l_i(s)$ be the cycle length of robot i given a joint action s: $l_i(s) = A_i(s) + P_i(s)$. The WLU of EI_{tot} for robot i takes the form $\frac{A_i(s)+A_i^{\phi}(s)}{A_i(s)+P_i(s)} - \frac{A_i(\phi_i, s_{-i})}{l_i(\phi_i, s_{-i})}$ where $A_i^{\phi}(s) = \sum_{j \in N \setminus \{i\}} (A_j(s) \frac{l_i(s)}{l_j(s)} - A_j(\phi_i, s_{-i}) \frac{l_i(s)}{l_j(\phi_i, s_{-i})})$.*

Proof.

$$
\begin{aligned}
WLU_i^{EI_{tot}}(s) &= EI_{tot}(s) - EI_{tot}(\phi_i, s_{-i}) \\
&= \sum_{j\in N} \frac{A_j(s)}{A_j(s)+P_j(s)} - \sum_{j\in N} \frac{A_j(\phi_i,s_{-i})}{A_j(\phi_i,s_{-i})+P_j(\phi_i,s_{-i})} \\
&= \sum_{j\in N} \frac{A_j(s)}{l_j(s)} - \sum_{j\in N} \frac{A_j(\phi_i,s_{-i})}{l_j(\phi_i,s_{-i})} \\
&= \frac{A_i(s)}{l_i(s)} + \sum_{j\in N\setminus\{i\}} \frac{A_j(s)}{l_j(s)} - \sum_{j\in N\setminus\{i\}} \frac{A_j(\phi_i,s_{-i})}{l_j(\phi_i,s_{-i})} - \frac{A_i(\phi_i,s_{-i})}{l_i(\phi_i,s_{-i})} \\
&= \frac{A_i(s)}{l_i(s)} + \sum_{j\in N\setminus\{i\}} \frac{A_j(s)}{l_j(s)} - \sum_{j\in N\setminus\{i\}} \frac{A_j(\phi_i,s_{-i})}{l_j(\phi_i,s_{-i})} - \frac{A_i(\phi_i,s\_i)}{l_i(\phi_i,s\_i)} \\
&= \frac{A_i(s)}{l_i(s)} + \sum_{j\in N\setminus\{i\}} \frac{A_j(s)\frac{l_i(s)}{l_j(s)}}{l_i(s)} - \sum_{j\in N\setminus\{i\}} \frac{A_j(\phi_i,s_{-i})\frac{l_i(s)}{l_j(\phi_i,s_{-i})}}{l_i(s)} - \frac{A_i(\phi_i,s\_i)}{l_i(\phi_i,s\_i)} \\
&= \frac{A_i(s)+\sum_{j\in N\setminus\{i\}}(A_j(s)\frac{l_i(s)}{l_j(s)} - A_j(\phi_i,s_{-i})\frac{l_i(s)}{l_j(\phi_i,s_{-i})})}{l_i(s)} - \frac{A_i(\phi_i,s\_i)}{l_i(\phi_i,s\_i)} \\
&= \frac{A_i(s)+A_i^{\phi}(s)}{A_i(s)+P_i(s)} - \frac{A_i(\phi_i,s\_i)}{l_i(\phi_i,s\_i)}
\end{aligned}
$$

□

This is the most general form of the WLU of EI_{tot}. Different assumptions result in specific special versions that can be useful in various settings:

1. When a robot is absent from the system it cannot contribute to the system and its costs are zero. The expression $\frac{A_i(\phi_i,s\_i)}{l_i(\phi_i,s\_i)}$ is the effectiveness index of robot i when it is absent. Since we assumed that gains in active time are zero we can express the absence of the robot i as if it was always in active time during the cycle length $l_i(\phi_i,s\_i)$. This assumption can be expressed as $A_i(\phi_i,s\_i) = l_i(\phi_i,s\_i)$. Therefore we can see that $\frac{A_i(\phi_i,s\_i)}{l_i(\phi_i,s\_i)} = 1$. The WLU will now take the form $WLU_i^{EI_{tot}}(s) = \frac{A_i(s)+A_i^{\phi}(s)}{A_i(s)+P_i(s)} - 1$.

2. Collisions are synchronous in our model. This means that cycle length depends only on the joint action selected and not on the robot. For all pairs of robots $i, j \in N$ and all joint actions s: $l_i(s) = l_j(s)$. Therefore, we can remove the subscript and write $l(s)$. The effect of this assumption is that now $A_i^{\phi}(s)$ takes a simpler form: $A_i^{\phi}(s) = \sum_{j\in N\setminus\{i\}}(A_j(s) - A_j(\phi_i,s_{-i})\frac{l(s)}{l(\phi_i,s_{-i})})$

3. The absence of robot i does not affect cycle length: $l(s) = l(\phi_i, s_{-}i)$. $A_i^{\phi}(s)$ can now be expressed as $A_i^{\phi}(s) = \sum_{j \in N \setminus \{i\}} (A_j(s) - A_j(\phi_i, s_{-i}))$. Now A_i^{ϕ} has a meaning - it is the change in the total active time of the system due to the absence of robot i.

Now, WLU takes the form

$$WLU_i^{EI_{tot}}(s) = \frac{A_i(s) + \sum_{j \in N \setminus \{i\}} (A_j(s) - A_j(\phi_i, s_{-i}))}{A_i(s) + P_i(s)} - 1$$

Omitting the -1 element will not change the optimization problem. From the above formula of WLU we can see that either minimizing $A_i(s)$ or $A_i^{\phi}(s)$ minimizes WLU. Comparing WLU to EI we can see that EI is the same as WLU but with $A_i^{\phi} = 0$. This means that WLU, unlike EI, in addition to minimizing the robot's active time, also gives weight to minimizing the effects of the robot on other robots' active time.

Learning According to WLU

Robots initially do not know what action is best for every situation and therefore need to learn it. Now that we have an individual utility function (WLU), we know according to what utility they need to learn, but we still do not know how they will learn it.

There are many learning algorithms suitable for this task, and indeed we use a few in the experiments (see Chapter 5). As an example, we demonstrate learning using the Q-Learning algorithm [45], due to its simplicity. It is perhaps the most popular learning algorithm in the field of Reinforcement Learning [41]. We will show how we use this algorithm in the context of approximating WLU.

Q-Learning is based on iteratively learning according to a reward r in order to evaluate which action s is best to choose given that a robot was in state x and transitioned to state x'. This evaluation of the state and action pair is given by the *Q-Function* $Q(x,a)$. In the context of our model of reactive arbitration the reward will be the WLU approximation, the actions will be the coordination methods and for now, we focus on a single state and we will discuss multiple states later. This learning algorithm is built upon reactive arbitration. Given that a robot is about to collide, the robot will execute the following procedure (Figure 3.3):

1. Given state x it chooses $argmax_s(Q(x,s))$ with some exploration rate ε.

2. Performs the coordination method and obtains the active time A.

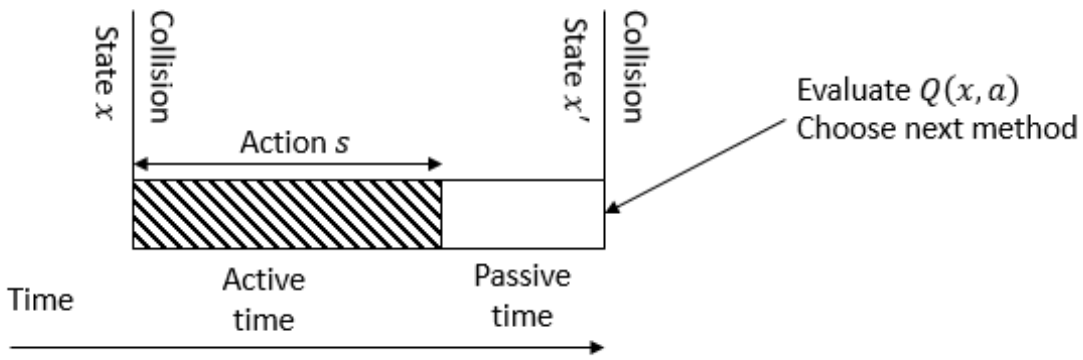

Figure 3.3: Reactive arbitration using the Q-Learning algorithm.

3. Returns to task execution up until next collision.

4. When the robot is about to collide once again the robot:

   (a) Obtains the passive time P

   (b) Calculates the WLU approximation $\widehat{WLU}$ according to A, P and other statistics.

   (c) measures the new state x'

   (d) Updates $Q(x,s)$ by the following rule: $Q(x,s) = (1-\alpha)Q(x,s) + \alpha(\widehat{WLU} + \gamma \max'_s(Q(x',s')))$ where $\alpha \in [0,1]$ is called the *learning rate* and $\gamma \in [0,1]$ is called the discount factor.

It is shown in [45] that in single agent domains in a *Markov Decision Process* (MDP), if an agent learns a policy using Q-Learning with any reward function $R(x',a,x)$, it will converge to the optimal policy in terms of the long-term reward.

In our context there are two main differences. The first difference is that learning is stateless. In terms of Q-Learning it simply means there will be only one state and in this case Q-Learning will become an exponential moving average. The second difference is that Q-Learning is designed for a single agent. There are extensions of Q-Learning for multi-agent systems [9, 24] but they either rely heavily on agent's knowledge of other agents or they hold no guarantees of convergence.

The use of stateless Q-Learning by multiple agents based on intrinsic measurements mimics the dynamics which lead potential games to converge to a potential function optimizers (each player unilaterally changes to a better action in terms of its individual payoff).

Therefore, we use stateless Q-Learning with $\widehat{WLU}$ in order for the robots to converge to an optimal joint action in terms of EI_{tot}.

Even though we have shown a method to learn optimal actions according to a more general model, there are still some gaps between this model and multi-robot coordination in practice. Those gaps will be thoroughly addressed in the next chapter.

Learning in Practice

In this chapter we first list all the gaps between our model and what happens in practice during multi-robot coordination. We then address each gap by suggesting a practical solution for it.

Gaps between our theoretical model and practice

- The WLU of EI_{tot} for a robot needs knowledge of other robots' active times.
- Collisions are asynchronous while this model is synchronous.
- Collisions are not mutual: When robots are about to collide, either synchronously or asynchronously, there is no guarantee that either one of them will actually recognize a collision and act to avoid it.
- Active and passive times may vary from the viewpoint of one robot, even for the same method.
- Different clusters of robots can collide at the same time in different places. This model does not take that into account.

Approximating WLU of total EI

Even though we now have a closed form of WLU, it is still required from the robot to know the active times of other robots. However, effects of a robot on other robots tend to be local - it is highly unlikely that a robot will affect the active time of another robot in the opposite edge of a map. Although we still do not know the active times of robots, even if they are close, each robot has sensing capabilities of its local environment. Using those sensing capabilities it can measure an approximation $\widehat{WLU}_i$ of how it affects on the active

time of those nearby robots. From now on, we will almost exclusively discuss the WLU of EI_{tot}. Therefore, unless stated otherwise, we will remove the superscript and denote $WLU_i(s)$ as the WLU of EI_{tot}.

WLU_i is composed of three elements - A_i, P_i and A_i^{ϕ}. Since a robot knows A_i and P_i, it only needs to approximate A_i^{ϕ}. Therefore, approximating WLU_i will take the form $\widehat{WLU}_i(s) = \frac{A_i(s)+\widehat{A_i^{\phi}}(s)}{A_i(s)+P_i(s)}$. Below are examples of how we can use this form of approximation:

- $\widehat{WLU}_i(s) = \frac{A_i(s)+0}{A_i(s)+P_i(s)} = EI_i(s)$. EI_i is also an approximation of WLU_i yet it is probably not a good approximation since it does not take into account its effect on other robots.

- $\widehat{WLU}_i(s) = \frac{A_i(s)+n\cdot A_0}{A_i(s)+P_i(s)}$ where n is the number of robots in the system and A_0 is an approximation of how much active time was added to each robot due to the presence of robot i. While this approximation does take into account effects on other robots, it is probably not feasible because of two reasons: The first being the fact that a robot does not necessarily know the number of robots in the system n and the second is the fact that effects are probably local and therefore do not necessarily involve all n robots in the system.

- $\widehat{WLU}_i(s) = \frac{A_i(s)+n_a\cdot A_0}{A_i(s)+P_i(s)}$ where n_a is the number of robots affected by this robot and A_0 is the same as before. This is a much more feasible structure than the former due to overcoming the two reasons why the former was infeasible: The robot does not need to know n - it only needs to know n_a. Due to locality of effects n_a can be measured by sensors. For example, one way of measuring n_a is by the number of robots in the vicinity. When using n_a instead of n, it no longer assumes that all robots are affected and therefore, it is probably a better approximation.

Due to both being simple and the most feasible out of the above suggested forms for $\widehat{WLU}_i$, we focus on the form $\widehat{WLU}_i(s) = \frac{A_i(s)+n_a\cdot A_0}{A_i(s)+P_i(s)}$. There are now two questions we need to answer: The first is how n_a is measured and the second is how A_0 is measured.

Measuring the number of affected robots

We suggest two possibilities for counting the number of affected robots:

1. By density - Before coordinating, a robot measures its density and assigns it to n_a. The density is a measurement of how many robots were in the vicinity of this robot.

2. By collisions - During the coordination and also in the resulting passive time the robot counts how many robots entered collision avoidance because of this robot. It then assigns this number to n_a.

Measuring by density has the advantage of being simple and possible to be done using sensing. However, there are two main disadvantages to this method: The first is the fact that density can be measured in many ways. For example, one way to measure it is to measure the number of robots in a given radius - different radii may yield different results and therefore different approximations. The second disadvantage is that the measurement is done only in the beginning of the collision and not over all the cycle. This can cause inaccuracies. For example, if in the beginning the density was low and during the rest of the cycle it was high we will get an under-approximation.

Unlike, measuring by density, measuring by collisions has both the advantage of being measured over all the cycle rather than only in the beginning and the advantage of having only one way to do so. However, its main disadvantage is that it is more difficult (yet not impossible) to use this measuring method.

Measuring active time added to other robots

In order for a robot to measure A_0, it needs to know both the active times of other robots with and without the absence of this robot. Both cannot be measured by the robot. A robot cannot measure anything when it is absent and when it is not absent, it cannot measure the active time of affected robots. However, we know that A_0 is a real value and it exists. Therefore, we suggest approximating it. Given a history of play h^c and a joint action $s \in S$, we will define $C(s) \subseteq \{1,...,c\}$ as the subset of collision indices where joint action s was played. In the same manner we will define $C(s_i)$ as the subset of collision indices where robot i chose individual action $s_i \in S_i$, regardless of the actions chosen by other robots. Below is a list of possible approximations:

- *EI*, $A_0 = 0$ - Assume active times of other robots remain unaffected. Equivalent to calculating EI_i.

- *Same EI for all*, $A_0 = A_i(s)$ - Assume each robot's active time is added this robot's active time.

- *Average over time*, $A_0 = \frac{1}{c}\sum_{j=1}^{c} A_i(s^j)$ - The addition in active time to other robots is this robot's average active time measured in its history of play.

- *Average over actions*, $A_0 = \frac{1}{|S_i|}\sum_{s' \in S_i} \frac{1}{|C(s')|}\sum_{j \in C(s')} A_i(s^j)$ - The addition in active time to other robots is by measuring this robot's average active time for each type of method it selected $s' \in S_i$ and then averaging over those averages.

- *Minimum over actions*, $A_0 = \min_{s' \in S_i}(\frac{1}{|C(s')|}\sum_{j \in C(s')} A_i(s^j))$ - The addition in active time to other robots is by finding the individual action $s' \in S_i$ that has the lowest average active time.

- *Maximum over actions*,$A_0 = \max_{s' \in S_i}(\frac{1}{|C(s')|}\sum_{j \in C(s')} A_i(s^j))$ - The addition in active time to other robots is by finding the individual action $s' \in S_i$ that has the highest average active time.

Dealing with Asynchronous Collisions

In section 4.1 we made a list of the gaps between existing models of reactive arbitration and what happens in practice. One of the gaps is the fact that collisions can be asynchronous. There is absolutely no guarantee that all robots will collide together. Figure 4.1 shows the difference between theory and reality by showing two diagrams, each representing a task run.

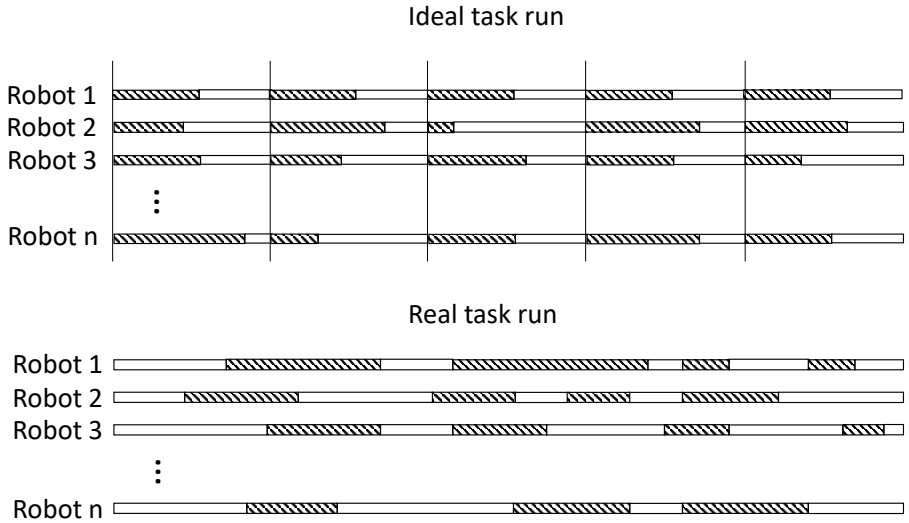

Figure 4.1: A complete task run according to theory vs. reality. Bars with diagonal lines denote the active times and bars with no lines denote the passive times. Perpendicular lines denote the times where robots jointly collide and each chooses a coordination method. An ideal run is expressed as a sequence of synchronous collisions while in practice it is not so.

Therefore, in order to represent a task run with asynchronous and non-mutual collisions, we treat every time where at least one robot detected a collision as a synchronous joint collision, even if not all robots detected a collision. In terms of the collision matrix, if a robot did not actually detect an imminent collision, we treat it as if it stays with its

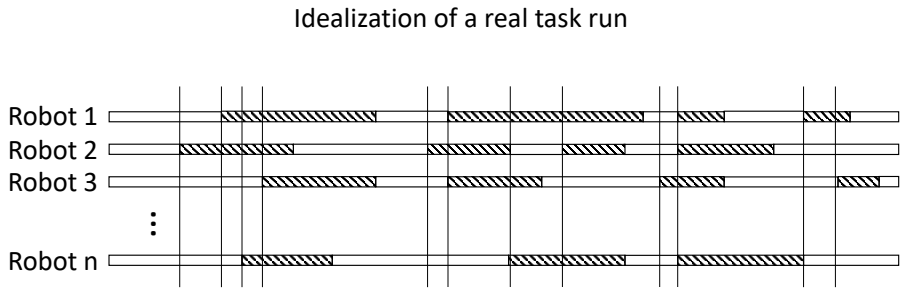

Figure 4.2: Treating a task run as a run with synchronous collisions. If a robot does not detect a collision it effectively choses a 'nop' action. Bars with diagonal lines denote the active times and bars with no lines denote the passive times. Perpendicular lines denote a synchronous joint collision.

Dealing with Non-Mutual Collisions

In addition to collisions being asynchronous they can also be non-mutual. This means that even if robots jointly collide at the same time, there is no guarantee that all of them will recognize they collided, whether they are close or far from each other. This is mainly caused by limited sensing capabilities. For example, if a robot can only sense robots that are in front of it, it will not recognize another robot colliding it from the back.

When a joint collision occurs and a robot cannot recognize it - there is nothing this robot can do but keep doing what he did. On the other hand, whenever it knows it collided it must do something. If it collided in passive time, it should coordinate. The main question is what it should do when it is in active time and it collides once again with a robot. One option is to keep coordinating as if no collision occurred. Another option is to preempt the current coordination method and choose either the same or a new coordination method. According to our modeling the rightest way to treat collisions during active time is to preempt the current coordination method and choose a new coordination method.

Even though preempting is the best way in theory to deal with collisions in active time, we chose not to do so. This is due to the fact that the criteria for detecting a collision can cause thrashing - repeated preemption of the coordination mechanism due to the collision criteria being true for consecutive samples. Therefore, not responding to collisions during active time acts as an anti-thrashing mechanism. For example, let there be a multi robot system where each robot's collision criteria is that there is at least one robot within a distance less than d from it. Each robot, in its controller's main loop checks this condition repeatedly several times per second. If another robot indeed was in a distance less than d from this robot for a respectable amount of time, this robot will enter coordination and then for each cycle in its controller's main loop it will repeatedly preempt the coordination method.

Dealing with varying active and passive times

In Section 3 we have assumed that outcomes of a collision rely only on the joint action selected. In terms of a WLU approximation it includes all of its elements - A_i, P_i and A_i^{ϕ}. It can be directly inferred that the cycle length $l = A_i + P_i$ will stay the same given a joint collision. However, a robot does not know the joint action played and from its viewpoint the active and passive times it will obtain will vary, even if it chooses the same individual action repeatedly because the active and passive times also depend on other robots. In addition, in practice the cycle length may vary even for the same joint action. Therefore, we would like to do some averaging on A_i, P_i and A_i^{ϕ} and then calculate a WLU approximation which is an averaging over the last collisions $\widehat{WLU} = \frac{\overline{A_i}+\overline{A_i}^{\phi}}{\overline{A_i}+\overline{P_i}}$.

This can cause inaccuracies in learning WLU approximations. Stateless Q-Learning, for example, is effectively an exponential moving average depending on its learning rate—the closer the learning rate the stronger the averaging is. Given that an individual action $s_i \in S_i$ was performed $C(s_i)$ times, averaging the WLU approximation of this action will give the expression $\frac{1}{|C(s_i)|}\sum_{j \in C(s_i)} \frac{A_i(s^j)+A_i^{\phi}(s^j)}{A_i(s^j)+P_i(s^j)}$. Such averaging sums the WLU approximations and divides them by the number of collisions. What we actually want is to first average A_i, P_i and A_i^{ϕ} and then assign it in the WLU approximation formula. This may cause different results.

We now show an example: Consider the case where a robot i collided 20 times. In all collisions but the last one $A_i = 500$ and $P_i = 10$ and in the last collision $A_i = 500$ and $P_i = 1000000$. The total time the robot spent in active time will be $500 * 20 = 10000$ and in passive time: $10 * 19 + 1000000 = 1000190$. Therefore EI^i_{tot} will be $\frac{10000}{10000+1000190} \approx 0.01$.

This means that the robot invested very little time on coordinating. On the other hand, if the robot uses a non moving average based on sampling EI for each collision it will get $\frac{1}{20}(19 \cdot \frac{500}{500+10} + 1 \cdot \frac{500}{500+1000000}) \approx 0.93$. This will cause the robot to estimate that it does poorly in terms of time fraction it invested on the task (93% of the time spent on collision avoidance) while the truth is that it does well (1% of the time spent on collision avoidance).

The above difference between the two averaging methods is caused by the fact that the cycle length $A_i + P_i$ is real-valued signal in continuous time while sampling of A_i, P_i and A_i^{ϕ} is disctete. *Semi Markov Decision Processes (SMDP)* [10] are models that represent discrete sampling of a continuous-time reward. They also introduce a Q-Learning variant for SMDPs. We therefore suggest a heuristic based on SMDP which is a modification of Q-learning that will overcome this inaccuracy without changing the informational demands of the robots. We call this heuristic the *Continuous Time Q-Learning*. The formula is similar to Q-Learning but with some: First is the learning rate α is now a function of cycle length - The bigger the cycle length the closer it will be to 1, thus giving more weight to collisions with longer cycle length. A_i, P_i and A_i^{ϕ} are now also scaled according to cycle length.

1: $\alpha \leftarrow 1 - e^{-\frac{\Delta t}{\tau}}$
2: $A_i' \leftarrow (1 - e^{-\frac{A_i}{\tau}})$
3: $P_i' \leftarrow e^{-\frac{A_i}{\tau}}(1 - e^{-\frac{P_i}{\tau}}) \cdot P_i$
4: $A_i^{\phi'} \leftarrow (1 - e^{(} - \frac{A_i^{\phi'}}{\tau})) \cdot A_o$
5: $q(x_i, s_i) \leftarrow (1-\alpha)q(x_i, s_i) + \alpha(-\frac{A_i' + A_i^{\phi'}}{A_i' + P_i'} + \gamma \cdot max_{s'}(Q(x_i', s')))$

Algorithm 1: q_learning_continuous_time($A_i, P_i, A_o, \tau, \gamma, x_i, x_i', s_i$)

Different clusters of robots collide at the same time in different places

In a real-world system the actions selected by the robots are not the only factor in the outcomes of the system. An additional factor is the clustering of the robots during a collision - robots are found in different areas of the map at the same time. Therefore, while a stateless representation can drive the system towards better performance, it is limited by the fact that it is unable to distinguish between different clustering of a joint collision.

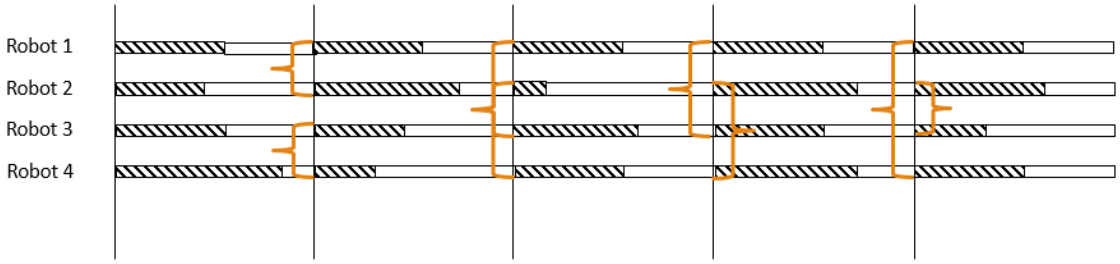

Figure 4.3: Different clusters of robots jointly collide at different times in an idealized system with synchronous collisions.

Figure 4.3 shows an example of an ideal system where all collisions are synchronous. In this example: In the second collison robot 1 collides with robot 2 and robot 3 with robot 4. In the third collision all robots collide with each other. In the fourth collision robot 1 collides with robot 3 and robot 2 collides with robot 4. In the fifth collision robot 1 collides with robot 4 and robot 2 collides with robot 3.

Therefore, we need to find a way for the robot to recognize what robots are in the same cluster with it. In many multi-robot systems it is very feasible to assume that robots are homogeneous. This, in general, means that they have the same body and behavior. Therefore, when robots are homogeneous there is no need for a robot to know which robots collide with it, but only how many collided with it. This is a significantly easier task. Therefore, we address this gap by defining the *Density* as a state space. The density measured by a robot is the number of robots which are within a given radius around it. The density will approximate the cluster size and will make the robot able to choose different actions for different densities. Learning the actions can be done with any learning algorithm that takes states into account, including the Q-Learning and continuous time Q-Learning algorithms.

Experiments

Experimentation environments

Before describing the experiments that were conducted, we will introduce the environments we conducted experiments on. We initially started with the *Alphabet Soup* simulator and then moved to experimenting with real robots - the Krembot swarm robots.

Alphabet Soup

The Alphabet Soup simulator simulates the multi-robot task of order picking. Order picking is the task of collecting items, usually in a structure like a logistic warehouse, in order to compose orders made by customers. In the Alphabet Soup simulator the items are portrayed as letters and the orders are portrayed as words. It is comprised of several *word stations* where each word station has a list of words to be composed, *buckets* which contain letters, *letter stations* which are used to re-fill buckets with letters and the robots which do all the work. The robots have three main tasks: The first is to take a bucket to a word station in order to put one letter in this station. The second is to return a bucket to its original position and the third is to take a bucket to a letter station.

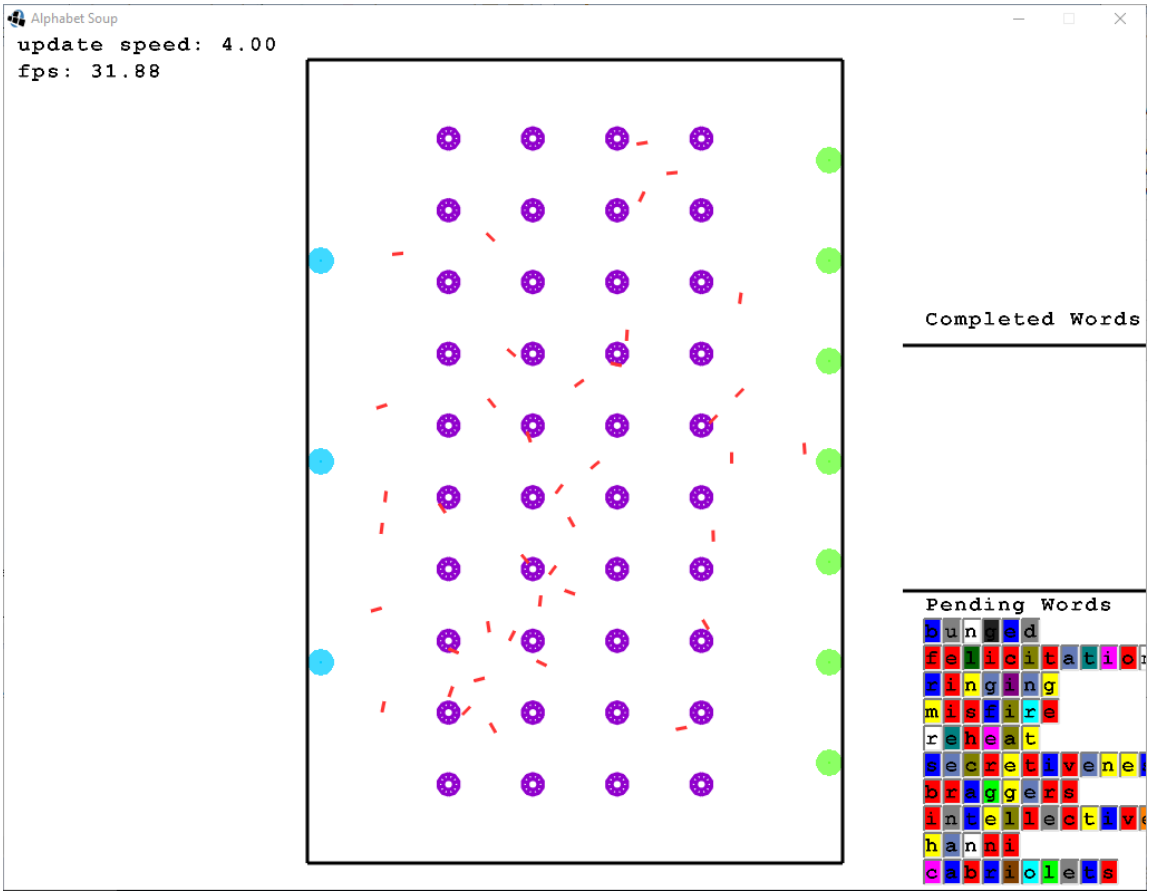

Figure 5.1: The Alphabet Soup simulator. Red lines are the robots, Purple circles are the buckets, Green circles are the word stations and cyan circles are the letter stations.

In this simulator, the task allocation for the robots is centralized and the collision avoidance mechanism is a reactive heuristic which is a combination of dynamic window (moving towards most vacant direction) and waiting in place for a random amount of time. Robots apply this collision avoidance mechanism when they sense that they are too close to other robots.

Modifications

The task allocation mechanism remains unmodified by us. So is the mechanism which detects and decides when a robot should coordinate. We only modify the coordination mechanism itself. We do it by replacing the original collision avoidance mechanism with a mechanism that arbitrates between reactive coordination methods. It should be noted

that since the original collision avoidance mechanism can be treated as a reactive method, it can be included inside the mechanism which arbitrates between reactive methods.

Experimentation settings

The main measurement of performance for this simulator is the amount of letters placed in word stations in a given amount of time. Unless stated otherwise, Each simulation is 10 minutes long with the last 30 seconds used for measuring performance and other statistics. In addition, unless stated otherwise, measuring n_a is done by collisions and not by density.

Krembot swarm robots

After testing reactive arbitration in the Alphabet Soup simulator we moved to testing the adaptation in a real world environment. We used Robotican's Krembot swarm robots in a domain which is a variant of multi-robot foraging.

Krembot robots are swarm robots with relatively limited sensing and processing capabilities. They are cylindrical-shaped robots with a height of 10.5 cm and a diameter of 6.5 cm. Below is a list of specifications that made our implementation possible:

- Bumpers: 4 bumpers evenly spread in 90 degree intervals around their circumference.
- RGB and ambient sensors (RGBA): 8 sensors evenly spread in 45 degree intervals around their circumference.
- Proximity sensors: 8 sensors evenly spread in 45 degree intervals around their circumference, together with the RGB and ambient sensors. Each sensor can sense proximity up to a range of 25.5 cm.
- RGB LEDs: Between the RGB, ambient and proximity sensors. 8 RGB leds evenly spread in 45 degree intervals around their circumference.

For a more detailed list of specifications, visit `https://www.robotican.net/kremebot`. It should be noted that despite they have a 9 degrees-of-freedom IMU, the Krembots we used are prototypes which do not utilize this IMU. Therefore, the Krembots do not have sensing capabilities of the linear acceleration, angular velocity and the absolute orientation of themselves.

The domain we tested reactive arbitration on, is a variant of multi-robot foraging. In Foraging, robots search an area in order to find pucks and retrieve them to their homebase. In general, gathering pucks requires robots to have grippers. Krembots, however, do not have grippers. Therefore, when we say they take and retrieve pucks they act as if they found and retrieved pucks.

In this domain there are a few stations fixed in position where robots can acquire pucks from. Once a robot reaches one of those stations, it takes the puck and retrieves it to a small area which we call the homebase. Once a robot retrieves this puck it returns for searching to acquire a puck from a station. We say it is a variant of foraging and not foraging because in foraging there are only pucks spread over the field and no fixed stations which provide those pucks. It should be noted that since the Krembot robots have no localization capabilities they are unable to either remember or plan a path to one of the stations. Therefore, they do it by randomly searching for a station.

We implemented this domain using a 150x80 cm table where we evenly spread 11 stations, each fixed to a position. In order to make the robots be able to go to the homebase, we put a light source in the bottom right corner that the robot could home to using its RGB and ambient sensors. The behavior of the robot can be described by three states and a few transitions from one state to another caused by various triggers. The three states are:

- Wander: Search for a station by randomly wandering over the field. Whenever the robot is in state wander its LED light will be magenta (both red and blue simultaneously).

- Go to homebase: Go to the homebase to retrieve the puck after a station was found. When the robot is in this state its LED light will be blue.

- Resolve conflict: The robot enters this state when it detects an imminent collision with another robot (not a static obstacle). In this state the robot learns based on EI and chooses a reactive coordination method. When the robot is in this state its LED light will be red.

If a robot detects an imminent collision with a static obstacle it executes a fixed behavior, unlike with a robot where it executes a coordination method by reactive method arbitration. For each of the three states there are several transitions from it to other states:

- Wander -> Go to homebase: The robot found a station.

- Go to homebase -> Wander: The robot retrieved a puck to the homebase.

- Wander/Go to homebase -> Resolve conflict: The robot detected an imminent collision with another robot.

- Resolve conflict -> Wander/Go to homebase: The robot finished executing the reactive coordination method and goes back to its previous state.

This domain requires the robot to have an ability to distinguish between a few objects. The objects are: A static obstacle, a station, a robot and the homebase. Each of those objects have a different way to be recognized by the robots:

- Robots - Each robots' LEDs emit either blue light, red light or both. Therefore, each robot senses another robot by measuring the intensity of the red and blue light sensed by the RGB sensors and see if at least one of the two passes a fixed threshold.

- Static obstacles - If the proximity sensors of a robot have a reading lower than a specific threshold and the red and blue lights sensed by the RGBA sensors do not cross the threshold for detecting a robot, the robot considers it as detecting a static obstacle.

- Homebase - When in state "Go to homebase", a robot goes towards the direction which has the highest intensity of green light. In order for a robot not to confuse between the homebase and other robots (which also emit light), we wrapped the lights of the homebase with a green cellophane paper in order for it to emit only green light and not red or blue lights. We also added another light source above the green light in order to avoid occlusion of the green light by the robots.

- Station - We used small wooden cylinders for the stations. Each cylinder is of enough height to be touched by the bumpers and short enough to not be recognized by the RGBA and proximity sensors. Therefore, when in state "Wander", if one of the bumpers of the robot is pressed and the robot did not detect another robot or a static obstacle as described above, it treats this sensation as detecting a station. If this happens when the robot is in state "Go to Homebase" it backs off with a random arc in order to avoid this obstacle and go towards the homebase.

Since the above implementation heavily relies on sensing light, we dimmed the surrounding lights as much as possible in order to prevent light pollution which may cause the Krembots to falsely sense one of the objects.

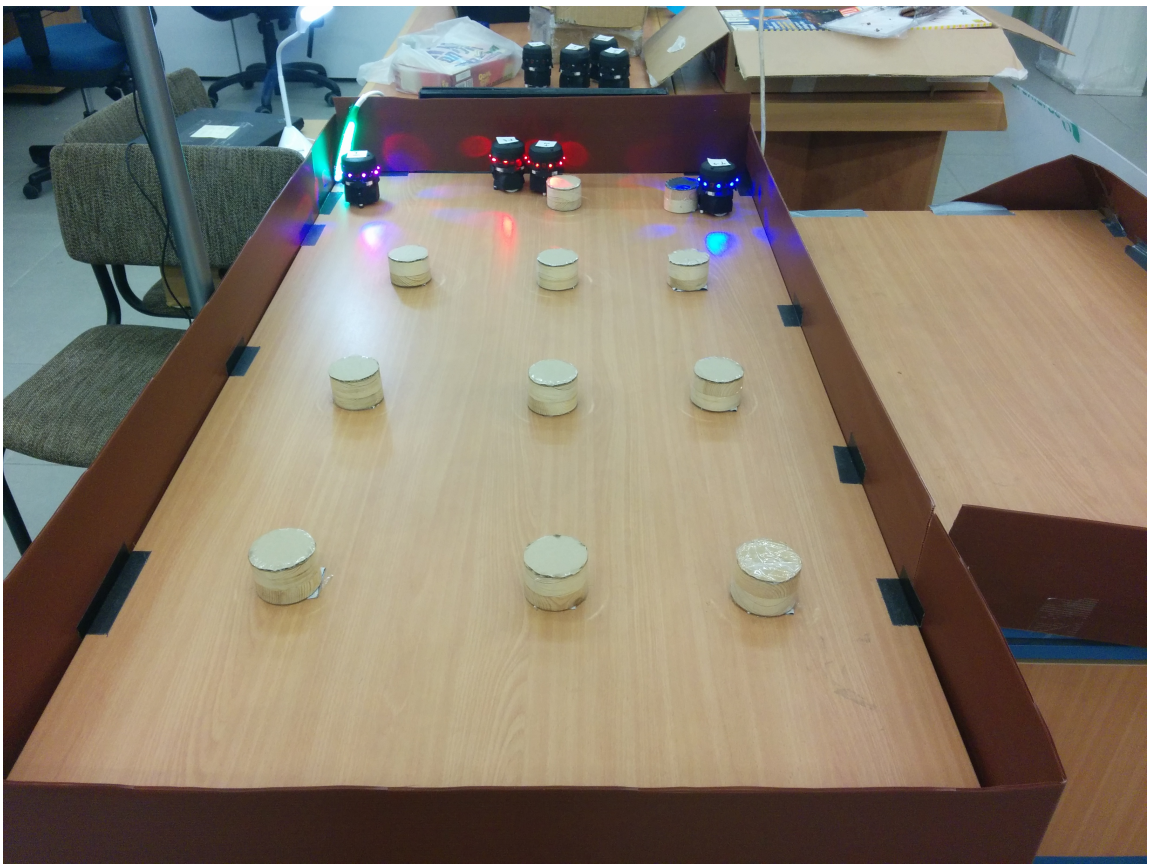

Figure 5.2: Krembot experimentation environment. An example with 4 robots.

Figure 5.2 shows the environment where experiments with the Krembots were ran. On the table, the wooden cylinders are the stations where robots gather pucks from. The green light in the upper-left corner is the homebase. There may be a situation where robots obstruct the light emitted from the homebase, causing other robots not to be able to orient themselves towards the homebase. Therefore, we added another light source above the green light for the robots to be able to orient towards the homebase from afar. There are four robots in this figure: The leftmost robot retrieved a puck to the homebase and is in state "Wander" (magenta LED light) searching for a station to collect a puck from. The two robots in the center have detected that they are about to collide with each other and therefore are in state "Resolve conflict" (red LED light) and the robot to the right is in state "Go to homebase" (blue LED light) since it found a station to collect a puck from.

Experimentation settings

In a similar manner to Alphabet Soup, the main objective of the robots is to gather as many pucks to the homebase in a given time. We would like to experiment with reactive arbitration that will maximize this objective.

We experimented with two Best evade coordination methods, one with a time parameter of 500ms and the other with a time parameter of 10000ms. The duration of each run is 1 hour long. For each hour-long we logged each event such as a collision or a puck that was retrieved. From this log we extracted statistics on the number of pucks retrieved and the coordination method choices of the robots. We extracted statistics based only on the last 15 minutes of the run since we want the learning to stabilize.

Using the above configuration, we tested the performance of 4 robots and 8 robots for several configurations. We measure the group performance of each configuration and the time fraction the robots spent on Best evade 500. This time fraction includes the active time and the resulting passive time of choosing Best evade 500. Since there are only two methods, it is easy to derive the time fraction of Best evade 10000.

Experiments: Learning vs. Adaptation

We distinguish between *learning* and *adaptation*. Learning focuses on *converging to a policy* which consistently chooses the best action for each state. On the other hand, adaptation focuses on *rapidly changing between policies* according to what is best now.

Previous work by Erusalimchik et al [27] where online reactive arbitration was used, focused on adaptation. They used stateless Q-Learning in order for the robots to learn the *EI*. However, to make it adaptive in their specific implementation, they used a very high learning rate (as high as 0.8). This can cause the robots to rapidly switch between policies.

Adaptive methods do not always work

Despite its empirical success in [27], adaptation is not a magic bullet. As a first step with the Alphabet Soup simulator, in [14] we initially tried EI-based adaptation over a set of five coordination methods:

- Repel - Go backwards for a given amount of time.
- Noise - Go towards a random direction for a given amount of time.

- Aggression - Choose between backing of like in Repel ("Meek" behavior) or staying put until robot has moved ("Aggressive" behavior)

- Original - The original reactive method of Alphabet Soup

- Best Evade - Always go to most vacant direction for a given amount of time.

We measured performance as a function of the number of robots in the system. We first tested the performance of each of the five reactive methods alone. We then tested the performance of random selection between those methods. Finally, we tested the performance of the EI reward with stateless Q-Learning. The parameters of the stateless Q-Learning formula were selected for it to be adaptive: Learning rate $\alpha = 0.5$ and exploration rate $\varepsilon = 0.1$. Simulations were run for one minute with the last 30 seconds for measuring results. Each coordination method except Original has a time parameter - the amount of time (in this case in ms) that the robot spends in executing this method. For each of the methods we selected 20 milliseconds.

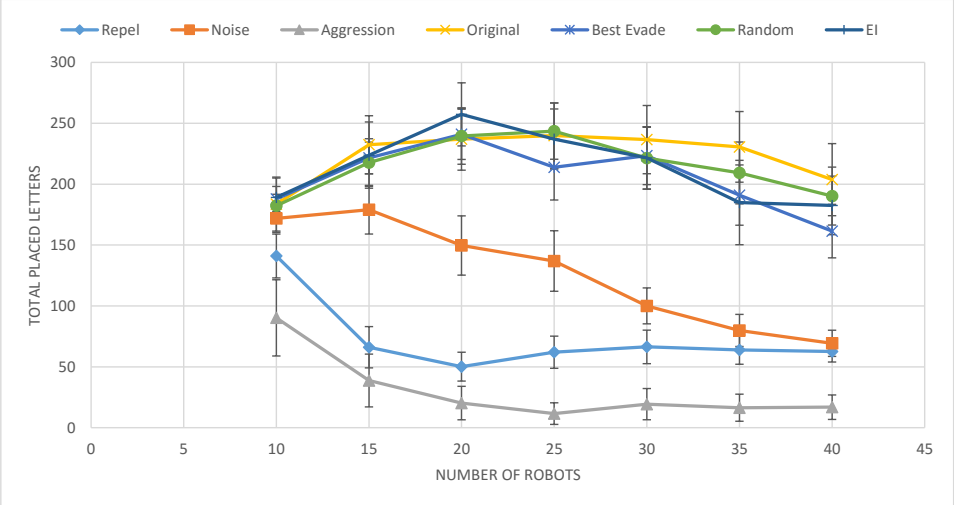

Figure 5.3: Initial results obtained in [14]. x-axis is the group size and the y-axis is the group performance in terms of total placed letters.

Figure 5.3 shows that EI-based adaptation does not perform best. Both random selection and the Original coordination methods slightly outperform it.

If in the previous experiment we tested a heterogeneous action set we now test homogeneous action sets. Those action sets are composed of the same type of action but with different time parameters. We now show that with real robots EI-based adaptation works very poorly. Using 4 Krembots, we test two coordination methods of the same type but with different time parameters - Best Evade 500 and Best Evade 10000. We first test each method separately, then test random selection between the two methods and finally test EI-based adaptation with the two methods. The Q-Learning algorithm with a learning rate of 0.5 and exploration rate of 0.1 was used (same as in previous experiment in Alphabet Soup. Unlike previous experiment, we measure performance (total retrieved pucks) as a function of the time fraction of Best Evade 500. The time fraction of a method is the total percent of time the system invested in this method - both the active time and the passive time resulting from using this method.

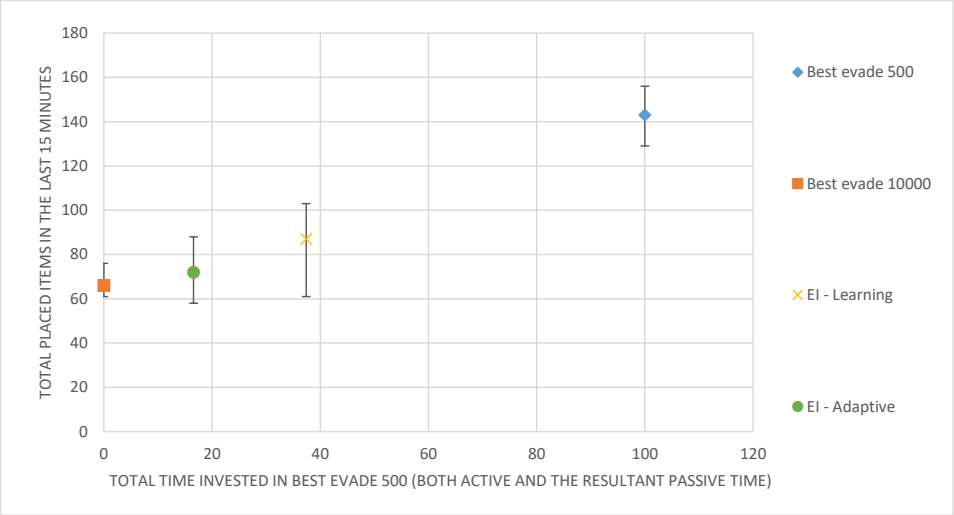

Figure 5.4: Initial results for Best Evade 500 and Best Evade 10000 with 4 Krembot robots. x-axis is the time fraction of Best Evade 500 and the y-axis is the group performance in terms of total retrieved pucks.

Figure 5.4 shows that EI-based adaptation performs poorly - significantly lower than Best Evade 500 and even slightly lower than random selection. If EI-based adaptation works best we would expect it to always select Best Evade 500 or adapt to something better.

We return to Alphabet Soup and experiment with various homogeneous action sets

composed of only Best Evade methods. The number of robots was set to 40. We started with the method set Best Evade 20, Best Evade 2000 and moved on to larger sets with more parameters in between 20 and 2000. Below is a list of all action set sizes and the parameters used. All of the simulation runs below used the stateless Q-Learning algorithm with the same parameters - learning rate 0.5 and exploration rate 0.1.

Number of methods	Parameters
2	20, 2000
6	20, 100, 200, 500, 1000, 2000
12	20, 100, 200, 500, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000
24	20, 50, 75, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000

Table 5.1: The number of coordination method set and the parameters used in each of the sets.

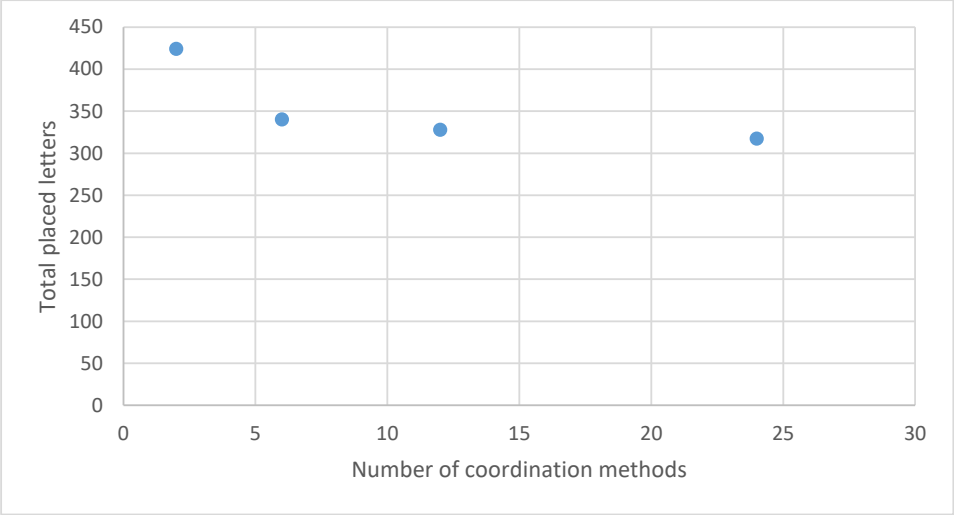

Figure 5.5: Group performance as a function of the number of coordination methods.

Figure 5.5 shows that the larger the set of methods the robots can arbitrate between, the lower their group performance is. We would expect from an optimal algorithm to

either improve performance or stay the same the more choice it has arbitrating between algorithms, especially given that in the above action sets larger action sets contain all the actions of smaller action sets.

Adaptive methods can sometimes work better than learning

We now show that adaptive methods can sometimes work better than learning. In order to do so, we test adaptation with EI and compare it to learning with EI. We compare Best Evade 20 and Best Evade 2000 in the Alphabet Soup simulator. Adaptation with EI is done by using stateless Q-Learning with a learning rate of 0.5 and exploration rate of 0.1. Learning EI is done by using stateless Q-Learning with a learning rate of 0.05 and exploration rate of 0.02 in order for the robots to not rapidly change between actions.

We show the performance of the two configurations as a function of the time fraction of Best Evade 20 in Alphabet Soup. In addition, we show two curves which we call the *Individual Mix* and the *Population mix*. The individual mix is a configuration of the robots where during the whole run each robot, given a collision, chooses Best Evade 20 with probability p and Best Evade 2000 with probability $1-p$. The population mix is a configuration of the robots where during p percent of the robots always choose Best Evade 20 and the rest always choose Best Evade 2000.

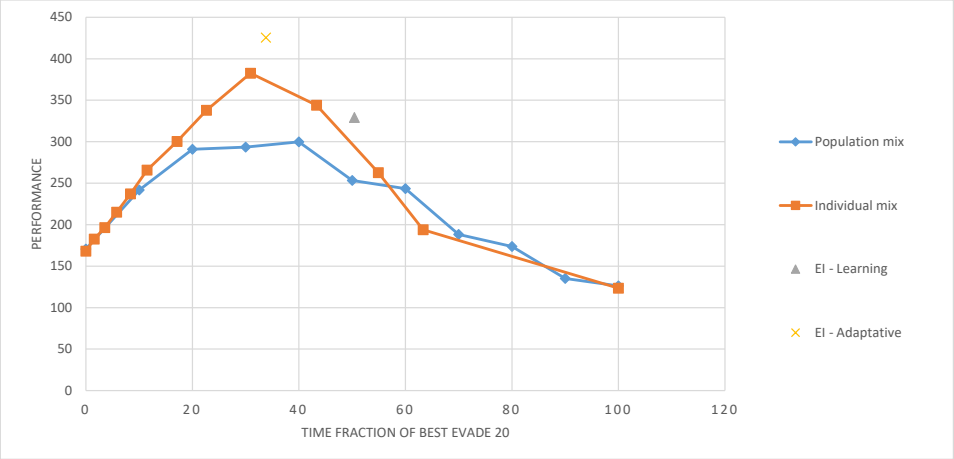

Figure 5.6: Group performance as a function of the number of coordination methods.

Figure 5.6 shows that adaptation performs significantly better than learning. In addition, the individual mix, for most of the time fractions, performs significantly better than the population mix. When robots put an emphasis on learning they eventually converge to a population mix. Therefore, we can conclude that it is not always best for each robot to converge to one action.

Learning can perform better than adaptation

In a similar manner, we now show that unlike Alphabet Soup, learning performs better than adaptation using the Krembots. We tested Best Evade 500 and Best Evade 10000 with EI using the same Q-Learning parameters for learning and adaptation. Figure 5.7 indeed shows that learning performs significantly better than adaptation.

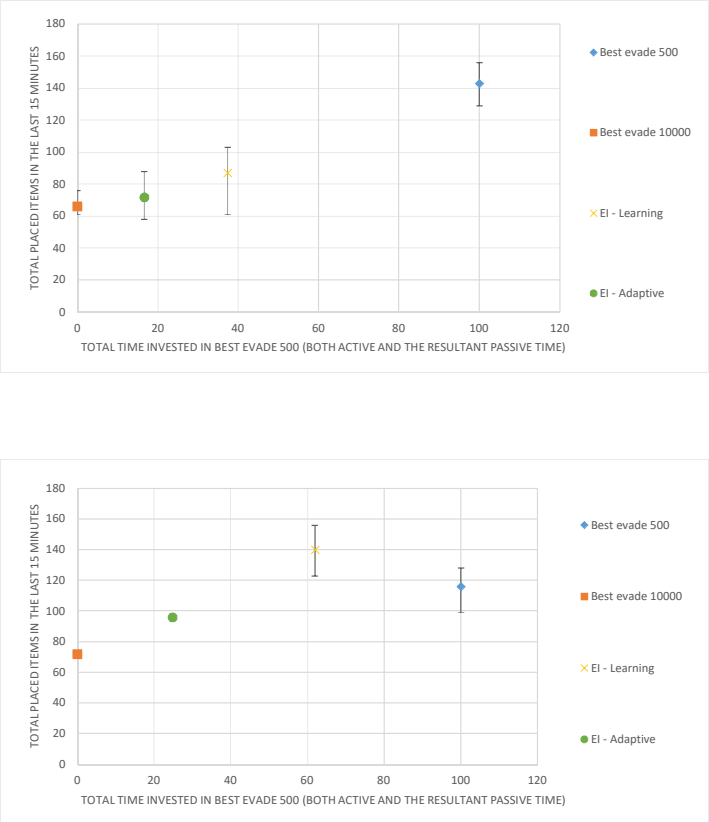

Figure 5.7: Learning vs. adaptation in Krembots for 4 robots (upper chart) and 8 robots (bottom chart).

Learning can improve performance

We show that even if learning is not always better than adaptation, it can improve performance in comparison to other algorithms. We test two heterogeneous action sets in Alphabet Soup. We list the action sets and their timing parameters:

1. Repel (700), Noise(540), Aggression(500), Original, Best evade(600).

2. Repel (200), Noise(500), Aggression(2000), Original, Best evade(200).

For each of the action sets we compare the Original coordination method to random choice and then to the Minimum Over Actions WLU approximation. For the WLU approximation we used the continuous time Q-Learning algorithm with $\tau = 10^{10}$ nanoseconds and an exploration rate of 0.02.

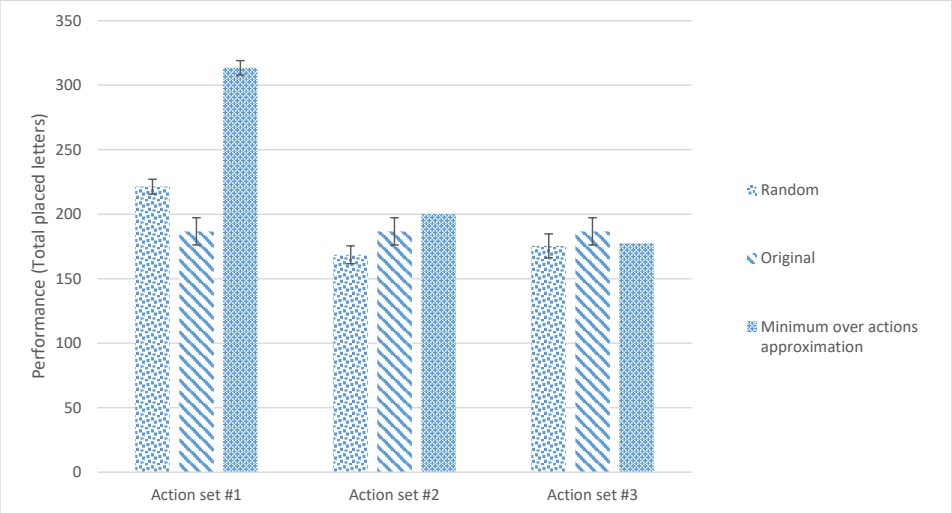

Figure 5.8: Performance of random choice vs. the original coordination method and the Minimum Over Actions WLU approximation.

Figure 5.8 shows that learning using the WLU approximation significantly improves performance in action set 1 and performs slightly better in action set 2.

We also show with the Krembots that not only learning EI with regular Q-Learning improves performance, but also with more learning variants: EI and Minimum Over Actions approximations, both using continuous-time Q-Learning with $\tau = 10^{10}$ nanoseconds and

an exploration rate of 0.02 both improve performance significantly over EI-based adaptation.

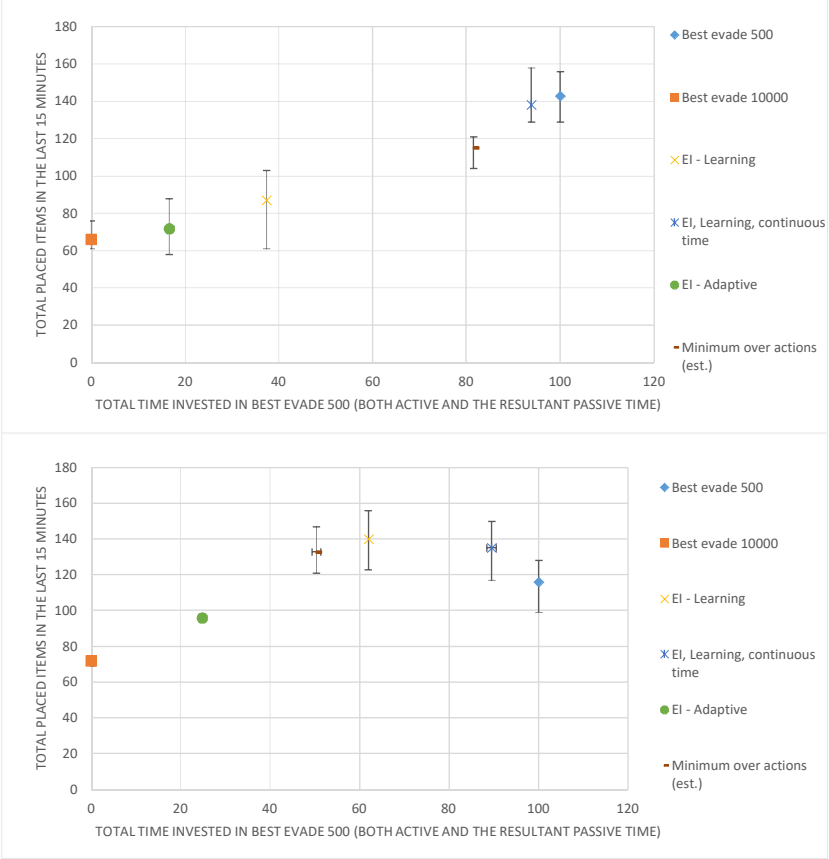

Figure 5.9: Different variants of learning vs EI-based adaptation in Krembots for group sizes of 4 (upper chart) and 8 (lower chart).

Figure 5.9 indeed shows that all variants of learning perform significantly better than adaptation.

Experiments: Regular Q-Learning vs. Continuous time Q-Learning

We will now compare regular Q-Learning to continuous time Q-Learning. We do so by measuring the performance of different WLU approximations each with regular Q-Learning and continuous time Q-Learning. The WLU approximations are the ones described in section 4.2. The parameters of regular Q-Learning are: learning rate was 0.05

and the exploration rate was 0.02. The parameters of continuous time Q-Learning are: $\tau = 10^{10}$ nanoseconds and the exploration rate was 0.02. The experiments were run in the Alphabet Soup simulator with the action set containing two actions: Best Evade 20 and Best Evade 2000.

Figure 5.10: Performance of different WLU approximations with regular Q-Learning (upper chart) vs with continuous time Q-Learning (lower chart) and where they are relative to the population mix.

Figure 5.10 shows a consistent pattern in the results in terms of the time fractions. The approximations show up in a specific order in terms of time fractions. The Maximum Over Actions and Average Over Actions tend to be at the leftmost side of the graph, preferring Best Evade 2000. The Average Over Actions approximation also perfers mostly Best Evade 2000, though in continuous time Q-Learning it has more bias towards selecting Best Evade 20. Following it is the Minimum Over Actions approximation which is somewhere in the middle. After this approximation is the EI approximation which is one time in the middle and one time towards 80% Best Evade 20. Finally, there is the Same EI For All approximation which always sticks to about 80% Best Evade 20. This pattern is consistent

with the characteristics of the approximations - approximations that give a heavier weight on effects of the robot on other robots, such as the Maximum over actions and Average over actions, tend to choose Best Evade 2000 while approximations that give lower weight such as EI and the Minimum over actions tend more towards Best Evade 20. In both of the configurations this order is kept but it can be seen that in continuous-time Q-Learning the results are shifted towards Best Evade 20.

In terms of performance it can be seen that EI performs best with Regular Q-Learning and poorly with continuous time Q-Learning while the Minimum Over Actions approximation performs best with Continuous Time Q-Learning while it does not do so with regular Q-Learning though it does not perform as poorly as EI did with continuous time Q-Learning. As we have stated before, EI is also a WLU approximation. We know it is a poor WLU approximation since it assumes it does not have any negative effects on other robots. It indeed can be seen that in continuous time Q-Learning EI is an under-approximator.

Experiments: Estimating number of affected robots by different methods

In section 4.2 we presented two ways of measuring n_a: By density and by collisions. We want to see how the two measurement methods differ in both behavior and performance. Therefore, we take two steps: The first is, given a configuration composed of an action set, a coordination method and a learning algorithm, to obtain the histogram of n_a for each coordination method for both ways of measuring it. The second step will be to see the performance of each way of measuring n_a.

We start with the Alphabet Soup simulator using the following configuration: The action set is composed of two actions, Best Evade 20 and Best Evade 2000. The reward function is the Minimum Over Actions approximation. The learning algorithm is the continuous time Q-Learning algorithm with a time constant of 10^{10} nanoseconds and an exploration rate of 0.02. The number of robots is 40.

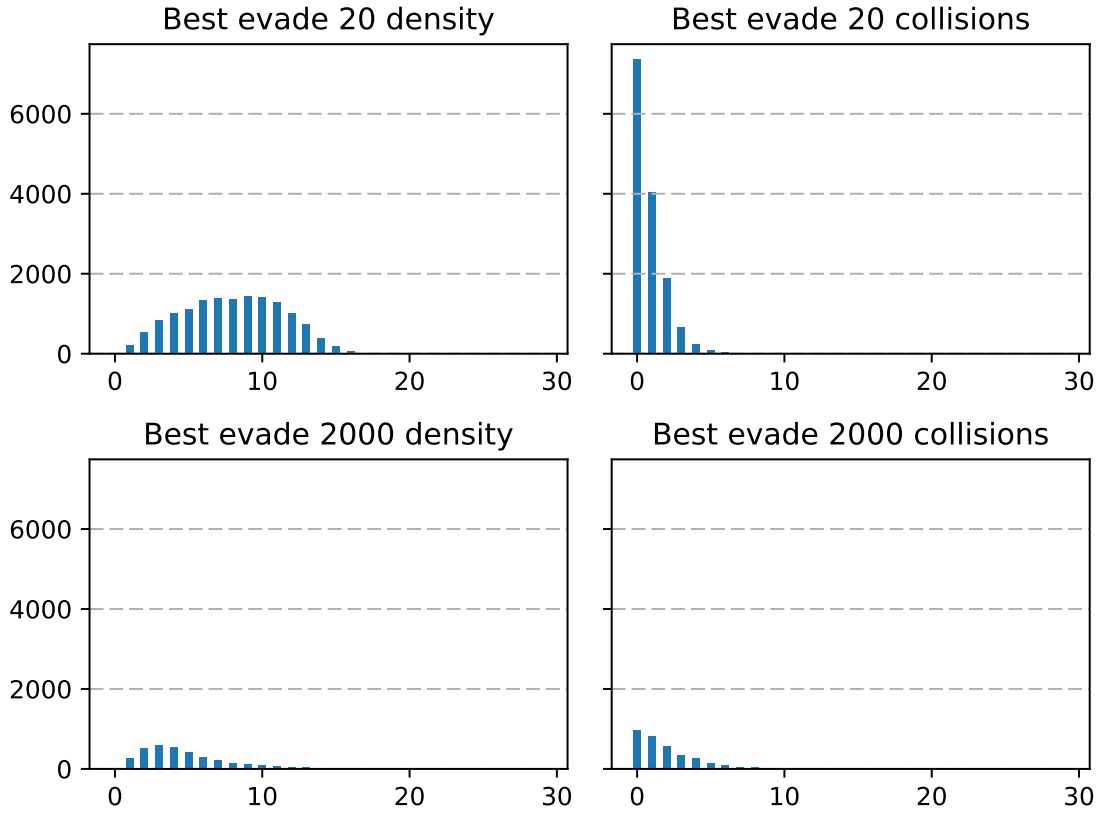

Figure 5.11: Histograms of the measurements of n_a - density versus collisions.

Figure 5.11 shows that for both Best Evade 20 and Best Evade 2000, measuring n_a by density tends to yield higher n_a values than by collision. A closer look at the histograms shows that the amount of bias towards lower n_a for measuring by collisions is different for the two coordination methods.

Best evade 20 by density	Best evade 2000 by density	Best evade 20 by collisions	Best evade 2000 by collisions
7.99	4.69	0.82	2.1

Table 5.2: Average n_a for each measurement method and coordination method.

Table 5.2 shows the average n_a for each measurement method and coordination method. Indeed, measuring by density yields higher n_a, but it can also be seen that the ratio of change from by density to by collisions in Best Evade 20 is $\frac{7.99}{0.82} = 9.74$ which is much higher than for Best Evade 2000: $\frac{4.69}{2.1} = 2.23$. We denote this ratio as r. Since WLU

approximations have the structure $\frac{A+n_a \cdot A_0}{A+P}$ it means that measuring by collisions will "punish" Best Evade 20 less. Therefore, we will expect that measuring n_a by collisions will make the system prefer Best Evade 20 more than with measuring n_a by density.

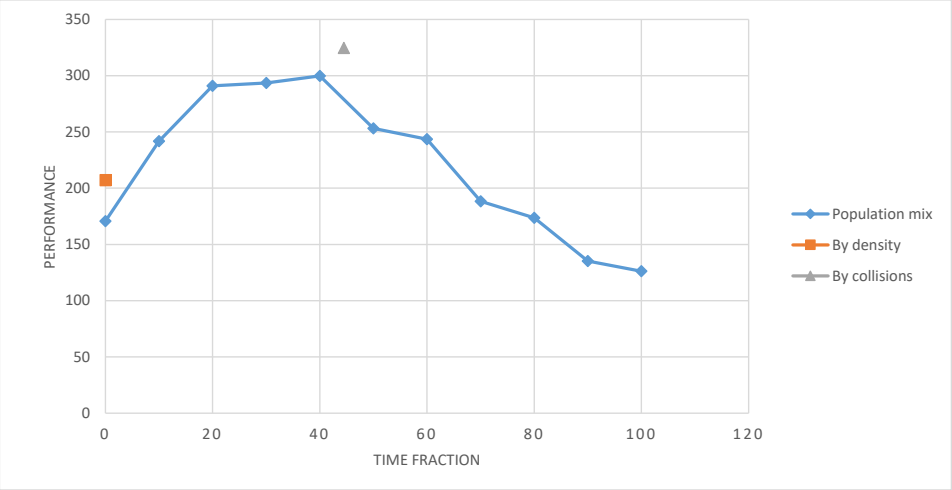

Figure 5.12: Group performance for each of the configurations (y-axis) and the average time fraction of the run spent on Best Evade 20 (x-axis) for 40 robots.

Figure 5.12 shows the difference between measuring n_a by collisions and by density. Indeed when measuring by collisions, robots have significantly more bias towards choosing Best Evade 20.

We now repeat with the Krembots what we did with the Alphabet Soup. With the Krembots, we used the following configuration: The action set is composed of two actions, Best Evade 500 and Best Evade 10000. The reward function is the Minimum Over Actions approximation. The learning algorithm is the continuous time Q-Learning algorithm with a time constant of 10^4 milliseconds and an exploration rate of 0.02. We tested over two group sizes: 4 robots and 8 robots.

In order to measure density we need to estimate it using the robot's sensors. Therefore, we used the 8 built in RGBD (RGB and distance) sensors in each Krembot. We estimated density by measuring the number of RGB and distance sensors which sense a robot. Since we programmed the robots to emit either a red or blue light, an RGBD sensor senses a robot if the red, blue and distance elements pass pre-set red blue and distance thresholds calibrated accordingly. In order to measure the number of collisions per cycle we looked at a portion of the videos of each run and measured it manually.

For each collision we measured the number of collisions per cycle, we also gathered the density obtained by the robot for the same collision. Using this data we constructed histograms of the density and number of collisions for each coordination method.

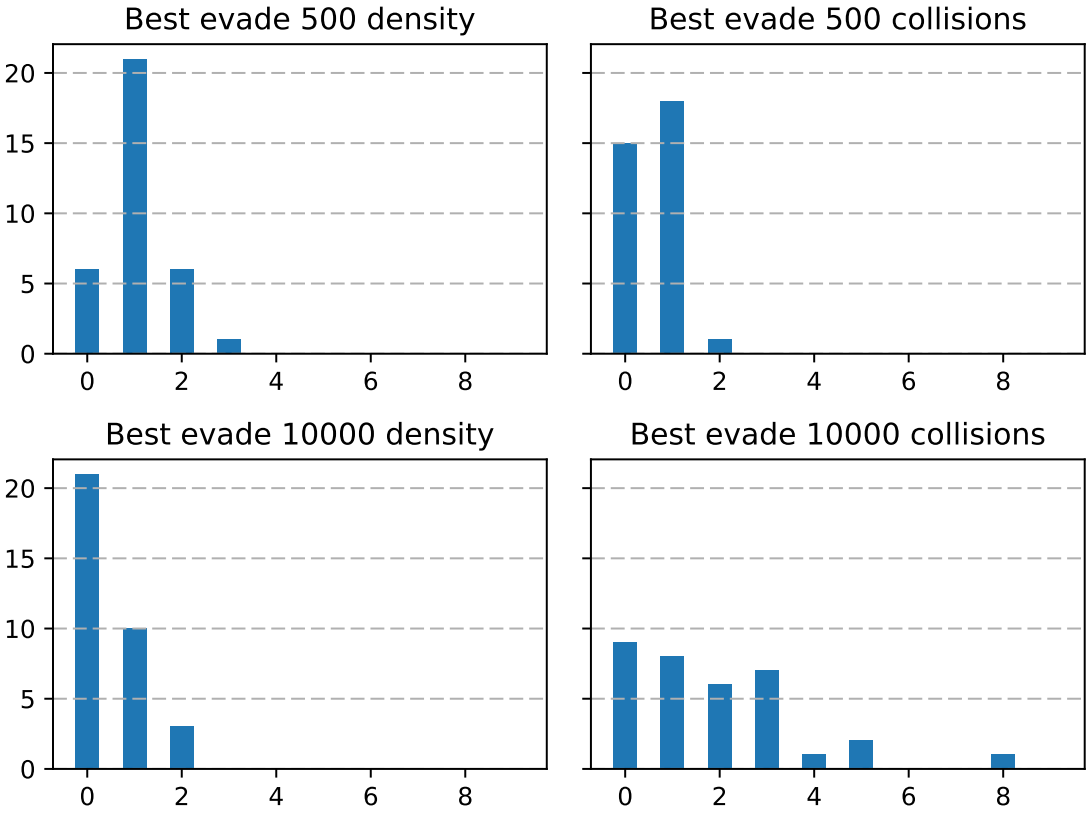

Figure 5.13: Histograms for n_a for best evade 500 and best evade 10000 when calculated by measuring density versus when calculated by number of colliding robots with 4 Krembots.

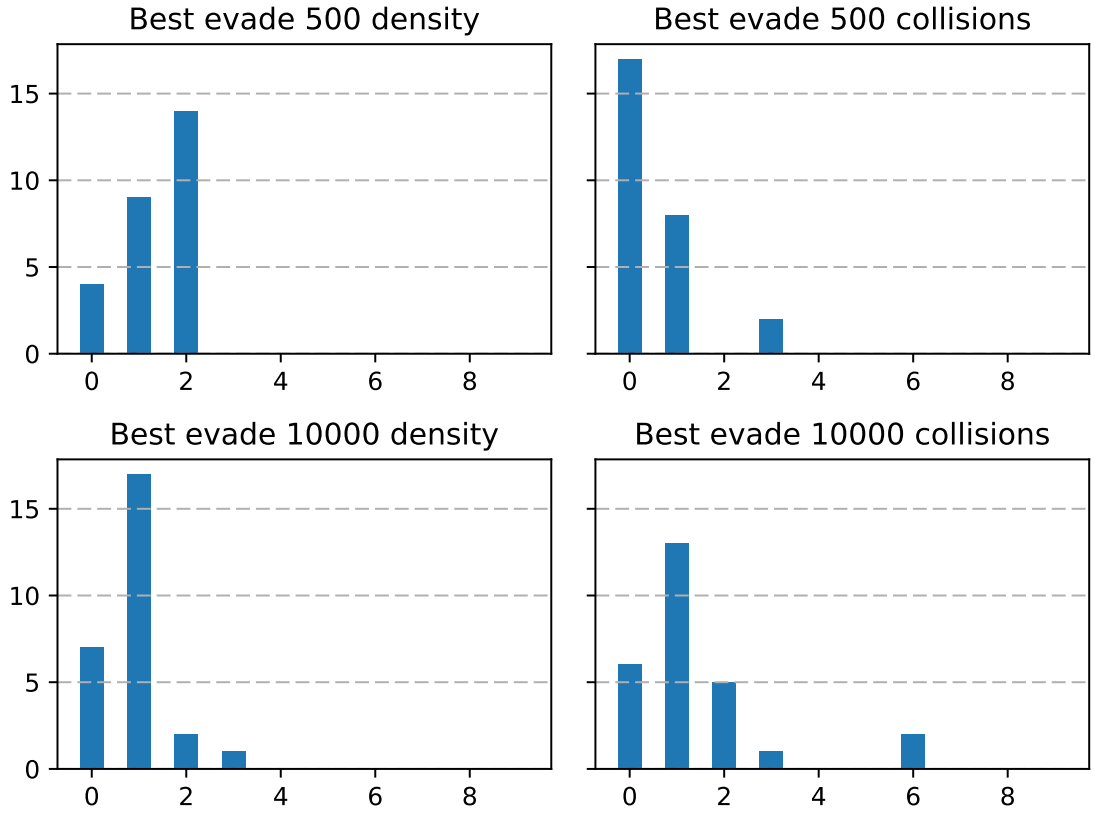

Figure 5.14: Histograms for n_a for best evade 500 and best evade 10000 when calculated by measuring density versus when calculated by number of colliding robots with 8 Krembots.

Figures 5.13 and 5.14 show the obtained histograms.

	Best evade 500 by density	Best evade 10000 by density	Best evade 500 by collisions	Best evade 10000 by collisions
4 Krembots	1.06	0.47	0.59	1.94
8 Krembots	1.37	0.89	0.52	1.41

Table 5.3: Average densities vs. average collisions for Alphabet Soup and the two Krembot configurations.

Unlike Alphabet Soup, n_a is not larger when measuring by density. However, the density-collision ratio of Best Evade 500 is larger than that of Best Evade 10000. For 4 robots, the ratio of Best Evade 500 is $r = \frac{1.06}{0.59} = 1.8$ and of Best Evade 10000 is $r = \frac{0.47}{1.94} = 0.24$. For 8 robots, the ratio of Best Evade 500 is $r = \frac{1.37}{0.52} = 2.63$ and of Best Evade 10000

is $r = \frac{0.89}{1.41} = 0.63$. This means that in both 4 robots and 8 robots we will expect measuring n_a by collisions to "punish" Best Evade 10000 more and thus to make the system to prefer Best Evade 500 more.

After we have obtained the histograms we now measure the performance of each method to measure n_a for 4 robots and 8 robots. For measuring n_a by density, we use the regular formula of a WLU approximation: $\frac{A+n_a \cdot A_o}{A+P}$. Since we measured collisions manually, we measure density and add a compensation factor to the WLU formula according to r in order to scale from density measurement to collision measurement according to the obtained histograms. The approximation will take the form $\frac{A+\frac{n_a}{r}A_o}{A+P}$.

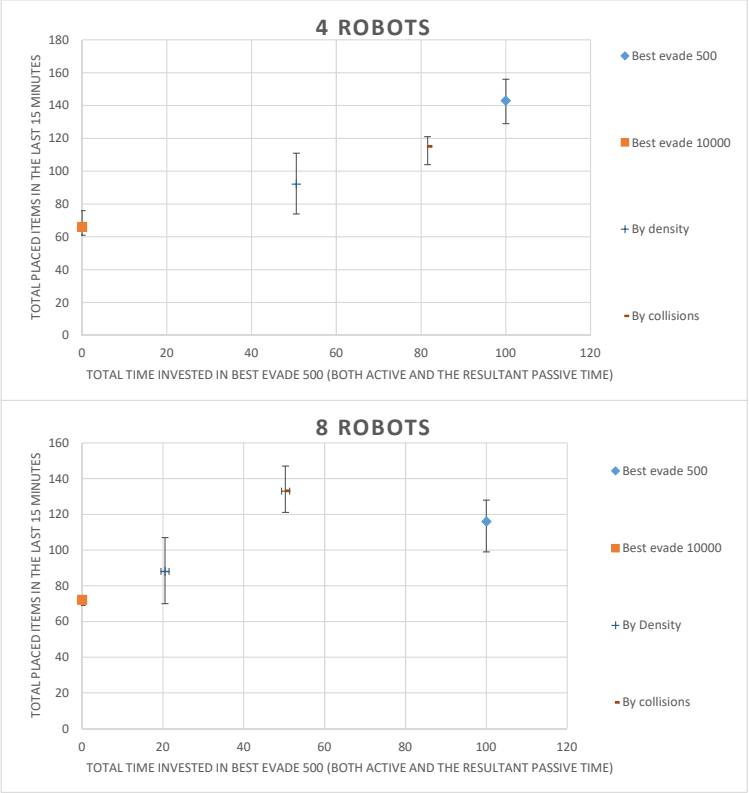

Figure 5.15: Group performance for each of the configurations (y-axis) and the average time fraction of the run spent on Best evade 500 (x-axis) for 4 robots and 8 robots.

Figure 5.15 shows that indeed the compensation factor (By collisions) gave significantly more bias towards Best evade 500 and improved the performance of learning with the best EI approximation.

Experiments: Rewards based on global utility

Up until now we only tested reward functions that are approximations to the WLU of EI_{tot}. We know that EI_{tot} is indirectly connected to U. Despite the fact that EI_{tot} is domain independent and U is domain dependent, we would like to test whether this indirect connection may impair performance. Therefore, we compare $WLU(EI_{tot})$ approximations to rewards directly based on U.

In the COIN framework presented in [48], they use reward functions that are based on U. We experiment with those reward functions in Alphabet Soup. In the context of Alphabet Soup, the performance U is the total placed letters. Given a time interval Δt, we denote Δl as the amount of letters the system placed in this time. Therefore $\frac{\Delta l}{\Delta t}$ will simply be the (average) rate of placed letters by the system in Δt. In the same manner we define Δl_i for the amount of letters placed by robot i. Each robot learns rewards based on those variables in the same manner as in reactive arbitration. This implies that Δ_t will represent a robot's interval between collisions. The reward functions used in [48] are:

- Team Game (TG) - The rate of placed letters of all the robots in a given time interval. For robot i, $TG_i = \frac{\Delta l}{\Delta t}$.

- Selfish Utility (SU) - The rate of placed letters of robot i in a given time interval. $SU_i = \frac{\Delta l_i}{\Delta t}$.

- Wonderful Life utility (WLU) - The wonderful life utility of U (not EI_{tot}). We approximate this WLU by the formula $WLU_i = SU_i - \frac{\sum_{j \in col_i} SU_j \cdot A_j}{\Delta t}$ where col_i is the set of robots which collided with robot i in its collision cycle and A_j is the active time of robot j.

- Aristocrat Utility (AU) - This reward is similar to WLU of U. If the main concept of WLU is the difference between the group performance with the robot and without the robot, the AU is the difference between the group performance with the robot performing the current action and the group performance with the robot performing an "average" action. The approximation of the AU is $AU_i = SU_i - \frac{\sum_{s \in S} P_i(s) Q_i(s)}{\Delta t}$ where $P_i(s)$ is the percentage of time robot i chose action s and $Q_i(s)$ is the Q-value of robot i for action s.

Name	Reward formula	Learning method	Parameters
TG	$\frac{\Delta l}{\Delta t}$	Q-Learning	Learning rate 0.1, exploration rate 0.1
SU	$\frac{\Delta l_i}{\Delta t}$	Q-Learning	Learning rate 0.1, exploration rate 0.1
WLU	$SU_i - \frac{\sum_{j \in col_i} SU_j \cdot A_j}{\Delta t}$	Q-Learning	Learning rate 0.1, exploration rate 0.1
AU	$SU_i - \frac{\sum_{s \in S} P_i(s) Q_i(s)}{\Delta t}$	Q-Learning	Learning rate 0.1, exploration rate 0.1

Table 5.4: Reward functions used for arbitrating Repel, Noise, Aggression, Original and Best Evade

Table 5.4 shows all the approximations done alongside with what learning method and parameters were used. We ran simulations with those approximations for 8 minutes and measured the rate of placed letters during the whole run. For EI 0.5, we averaged results over 110 runs, for EI 0.1 104 runs, for TG 103 runs, for SU 106 runs, for WLU 109 runs and for AU 112 runs.

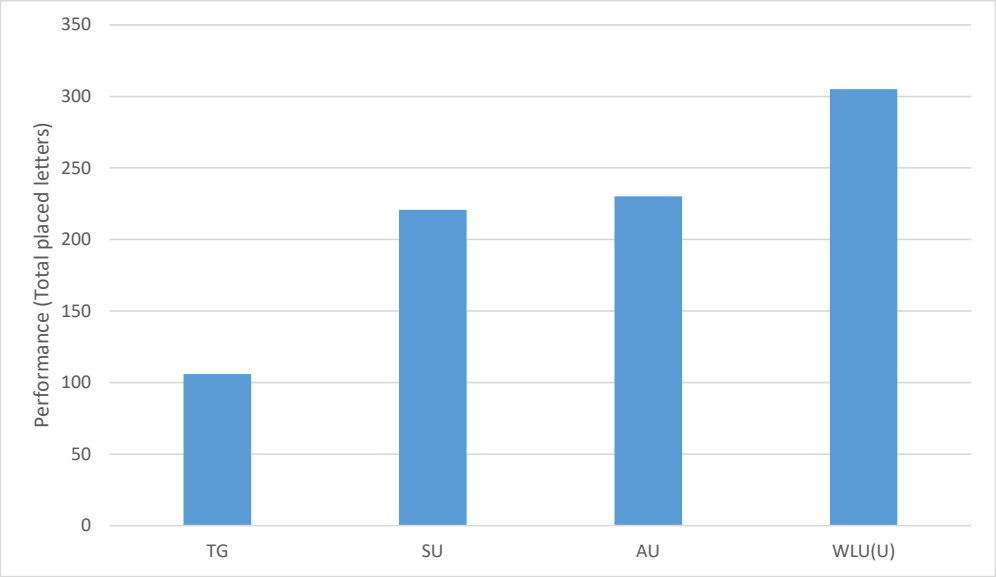

Figure 5.16: Performance of rewards based on U.

Figure 5.16 shows that the WLU of U outperforms all other rewards based on U. This indeed comes in concordance with the results in [48].

Given that the WLU of U performs best we would like to compare it to different approximations of the WLU of EI_{tot}. We compare $WLU(U)$ to three approximations of $WLU(EI_{tot})$:

- Minimum Over Actions, continuous time Q-Learning with $\tau = 10^{10}$ nanoseconds and exploration rate of 0.02.
- Regular Q-Learning, learning rate 0.05, Exploration rate 0.02.
- WoLF-PHC [9], Exploration rate 0.02, α=0.05, δ_w=0.0005,δ_l=0.005.

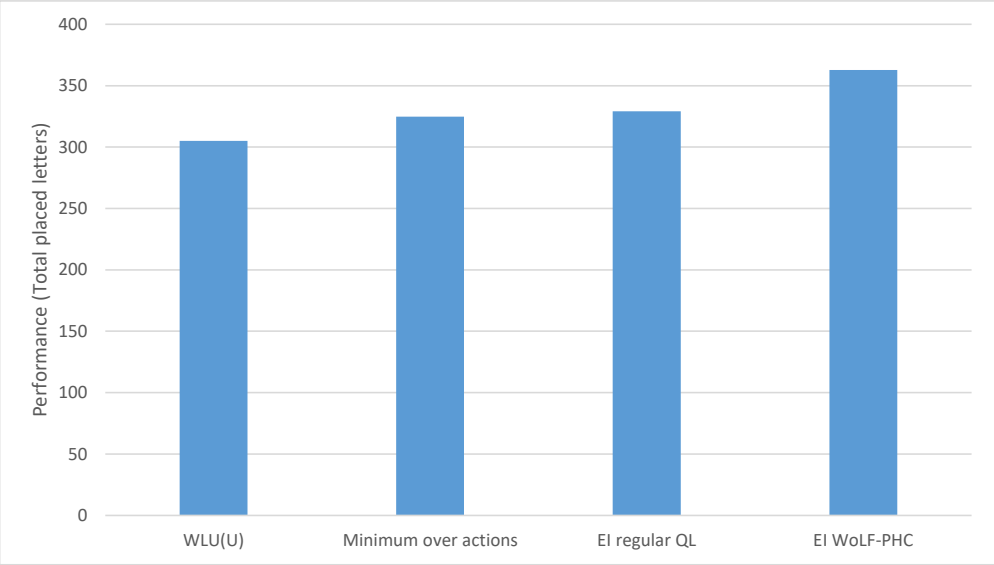

Figure 5.17: Performance of $WLU(U)$ vs. different approximations of $WLU(EI_{tot})$ with different learning algorithms.

Figure 5.17 shows that $WLU(U)$ has the lowest performance in comparison to all $WLU(EI_{tot})$ approximations. This shows that consisting directly on U and its derivatives does not improve performance, even though rewards based on EI_{tot} are connected to U only indirectly.

Experiments: Different learning algorithms

In this section we test how different learning algorithms change performance given one reward function. We choose the Minimum Over Actions WLU approximation and test 3

different learning algorithms over 8 action sets in the Alphabet Soup simulator. Table 5.5 lists the learning algorithms used and their parameters.

Learning algorithm name	Parameters
Regular Q-Learning	$\alpha = 0.05$, Exploration rate 0.02
Continuous time Q-Learning	$\tau = 10^{10}$ nanoseconds, Exploration rate 0.02
WoLF-PHC	Exploration rate 0.02, α=0.05, δ_w=0.0005,δ_l=0.005.

Table 5.5: The learning algorithms tested and their parameters.

Given the learning algorithms we now specify the action sets that the robots will use with the algorithms. We divide the 8 action sets (a)-(h) into 3 sub-groups: The first group (a),(b) and (c) is composed of action sets containing two Best Evade methods, one with parameter 20 and the other parameter is significantly higher than 20 and it varies. We would like to see how performance is affected as a result of this variance. The second group, (d) and (e) is composed of action sets each composed of 24 Best Evade methods with different time parameters. The difference between (d) and (e) is that in (e) the timing parameters are 10 times bigger than in (d). We would therefore want to test how a multiplier on the time parameters will affect performance. The third group (f), (g) and (h) are action sets that were initially tested in [14]. Each action set is composed of 5 heterogeneous methods and action sets differ only in the timing parameters of their methods. It should be noted that the Original coordination method has no timing parameter since it is the built-in method in Alphabet Soup. Each method in action set (h) has a time parameter arbitrarily chosen to be 20 (except Original). In action set (g), unlike (h), actions were not chosen arbitrarily. For each action we varied the time parameters and chosen best parameter according to the highest area-under-curve of performance when testing performance as a function of number of robots. Action set (f) was obtained when it was found that EI-based arbitration performed significantly better than each method or a random choice between methods. Table 5.6 lists the action sets.

Action set	Methods
(a)	Best evade 20 and 2000
(b)	Best evade 20 and 5000
(c)	Best evade 20 and 10000
(d)	Best evade 20, 50, 75, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900 and 2000
(e)	Best evade 200, 500, 750, 900, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 11000, 12000, 13000, 14000, 15000, 16000, 17000, 18000, 19000, 20000
(f)	Repel (700), Noise (540), Aggression (500), Original and Best evade (600)
(g)	Repel (200), Noise (500), Aggression (2000), Original and Best evade (200)
(h)	Repel (20), Noise (20), Aggression (20), Original and Best evade (20)

Table 5.6: The learning algorithms tested and their parameters.

We now test the performance of each of the learning algorithms over all the action sets for two group sizes: 10 robots and 40 robots.

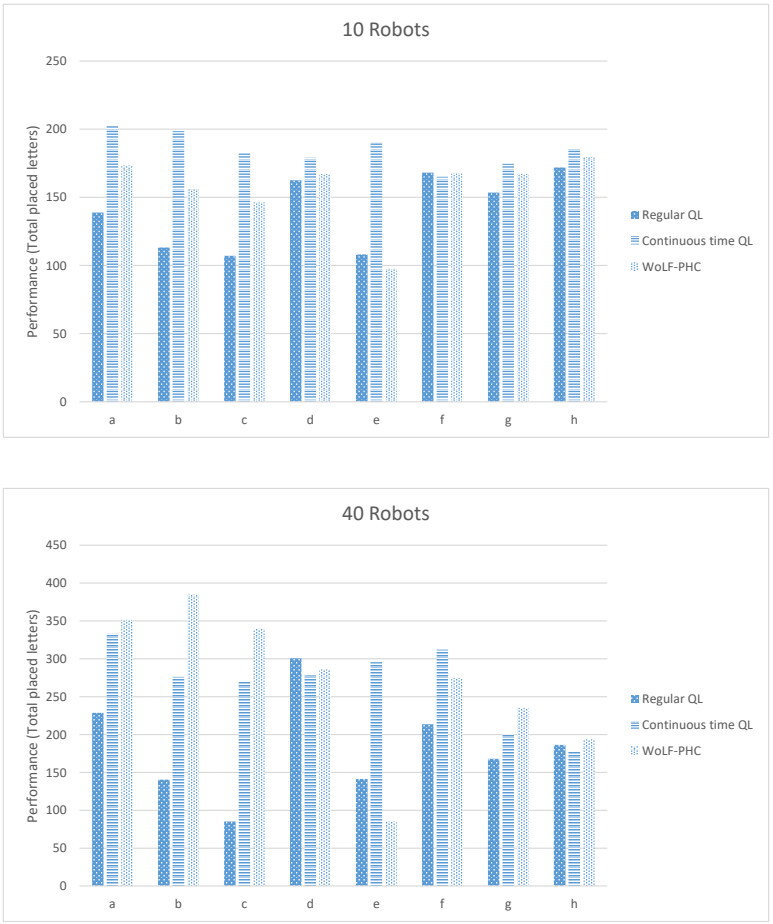

Figure 5.18: Performance of the Minimum Over Actions approximation using action sets (a) to (h) with regular Q-Learning, continuous time Q-Learning and WoLF-PHC.

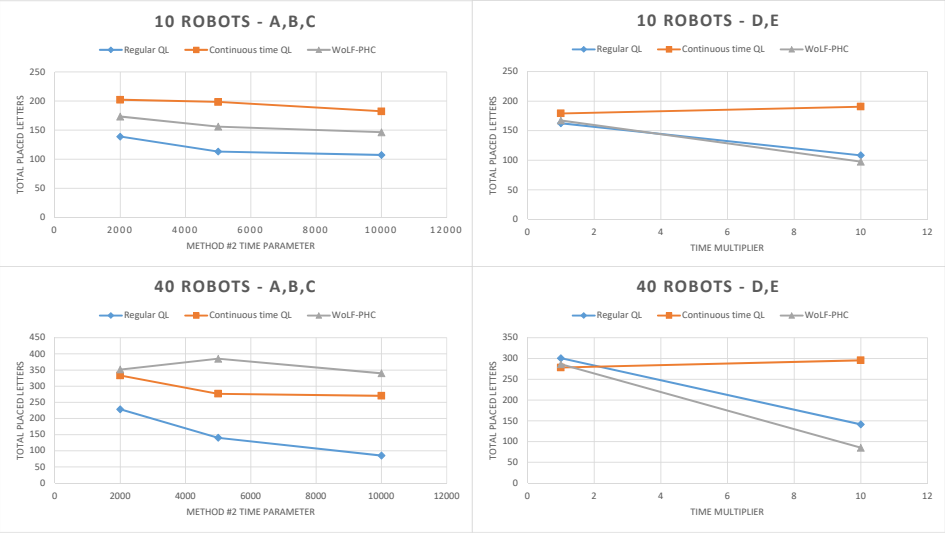

Figure 5.19: (a)-(c): Performance as a function of the second method's time parameter. (d),(e): Performance as a function of the timing parameter multiplier.

The simulation in figure 5.18 that not one learning algorithm works best in all action sets: In both 10 robots and 40 robots regular Q-Learning almost always falls behind the two other algorithms. Continuous time Q-Learning mostly performs best with 10 robots, especially in action sets (a),(b),(c) and (e). WoLF-PHC performs best with 40 robots in (a),(b) and (c) yet significantly falls behind in (e).

Figure 5.19 shows for the first group of action sets how performance varies as a function of the timing parameter of the second action. It also shows for the second group how performance varies as a function of multiplying the timing parameters. In the second group we can see that the performance of continuous time Q-Learning slightly improves while the performance of the two other algorithms significantly decline. We give a hypothesis to why continuous time Q-Learning performs best with 10 robots and why WoLF-PHC mostly performs best with 40 robots.

Why continuous time Q-Learning is best with 10 robots

The more robots there are in a system, the more frequent interactions between robots will be. We hypothesize that when there are 10 robots in the system, the number of interactions is fairly low and therefore outcomes of a robot i mostly depend on the robot's actions

rather than the joint action. Therefore, the multi-robot system can now be treated as many single robots. In such a configuration, robots need to find the action which minimizes their own WLU approximation. In section 4.5 we've shown that regular Q-Learning can cause inaccuracies in measuring the WLU approximation of a method. Since part of the WoLF-PHC algorithm uses regular Q-Learning, it may also become inaccurate. Therefore, this may cause regular Q-Learning and WoLF-PHC to select methods that are not optimal in terms of the WLU approximation.

Why WoLF-PHC performs best with 40 robots on (a), (b) and (c)

WoLF-PHC, unlike Q-Learning, learns a stochastic policy rather than a deterministic one. A stochastic (and stateless) policy corresponds to the individual mix while a deterministic policy corresponds to the population mix. As we have seen in figure 5.6, a stochastic stateless policy mostly performs better than a deterministic stateless policy. Therefore, WoLF-PHC may perform better in such settings.

Different time constants in continuous time Q-Learning

We now show how performance varies with different time constants when using the continuous time Q-Learning algorithm. Using Alphabet Soup we simulate continuous time Q-Learning with the Minimum Over Actions approximation. The action set is the same action set as in table 5.6. For each action set we test five time constants: $\tau = 10^{10}, \tau = 5 \cdot 10^9, \tau = 10^9, \tau = 10^8$ and $\tau = 10^7$.

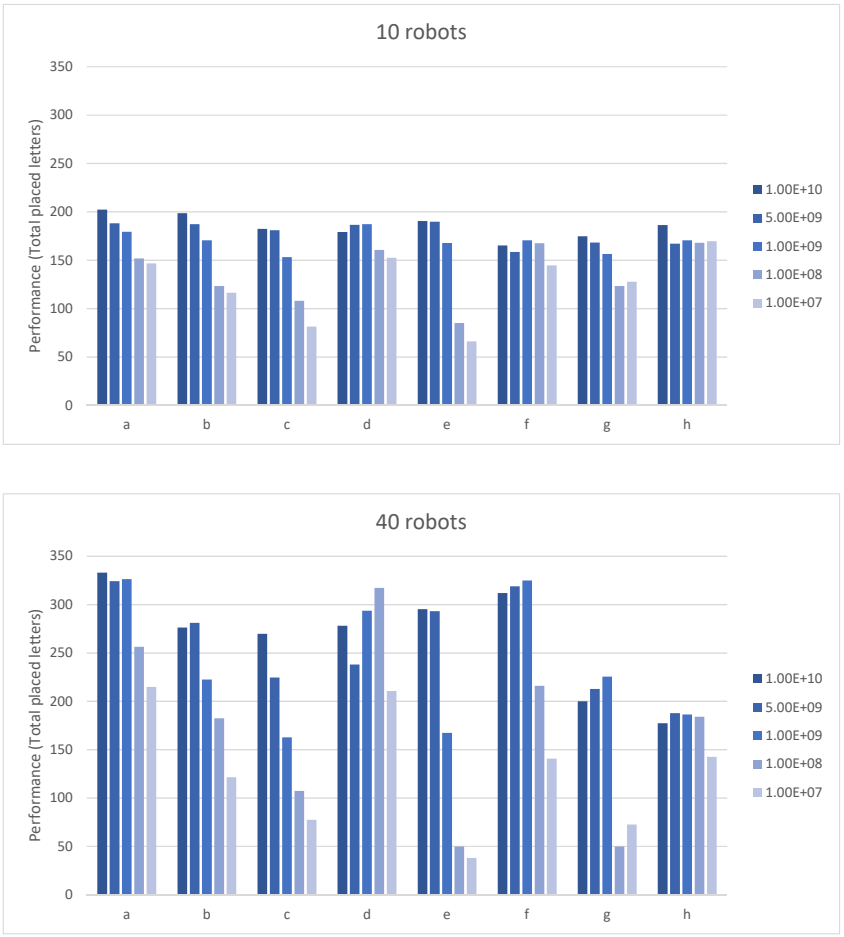

Figure 5.20: Performance of the Minimum Over Actions approximation with the continuous time Q-Learning algorithm with different time constants (τ).

Figure 5.20 shows that in most of the action sets, the lower the time constant is, the lower the performance gets. Since higher time constants give higher averaging it indeed shows that converging gives a better result.

Experiments: Stateless Q-Learning vs. stateful Q-Learning

Up until now we only experimented with stateless learning methods. We would now want to see how learning over a state space affects the performance of learning. For continuous time Q-Learning we would like to choose one time constant based on best performance.

Since we have seen that higher time constants give higher performance, we choose $\tau = 10^{10}$ as the time constant for continuous time Q-Learning. The state space we test will be based on the density around the robot when it collided. We test 3 variants of this state space:

1. All densities - Each state corresponds to only one density
2. Median - Two states. One for densities up to the median density and one for densities higher than the median density.
3. Quartiles - Four states. Each state corresponds to a set of densities which are in the same quartile of the density histogram.

We measure the median density and the density quartiles by looking at the density histogram of the stateless runs for the Minimum Over Actions approximation with continuous time stateless Q-Learning having $\tau = 10^{10}$ and exploration rate 0.02. Table 5.7 shows the median and quartile densities obtained for each of the action sets.

	Median	Quartiles (boundaries)
(a), 10 robots	3	2, 3, 4
(a), 40 robots	6	4, 6, 8
(b), 10 robots	3	2, 3, 5
(b), 40 robots	9	6, 9, 11
(c), 10 robots	3	2, 3, 4
(c), 40 robots	6	5, 6, 8
(d), 10 robots	4	3, 4, 5
(d), 40 robots	12	9, 12, 15
(e), 10 robots	2	2, 2, 3
(e), 40 robots	5	3, 5, 6
(f), 10 robots	2	2, 2, 3
(f), 40 robots	8	5, 8, 11
(g), 10 robots	4	2, 4, 5
(g), 40 robots	12	8, 12, 14
(h), 10 robots	4	2, 4, 5
(h), 40 robots	20	16, 20, 24

Table 5.7: Median and quartile densities for each of the action sets.

After we measured the median and quartile densities we would like to compare stateless learning to stateful learning. We use the Minimum Over Action approximation with

the continuous time Q-Learning algorithm. The algorithm's parameters are: $\tau = 10^{10}$, exploration rate 0.02 and we set the learning to be myopic by setting the discount factor to be $\gamma = 0$. We test over the same 8 action sets as in 5.6 and on 10 and 40 robots.

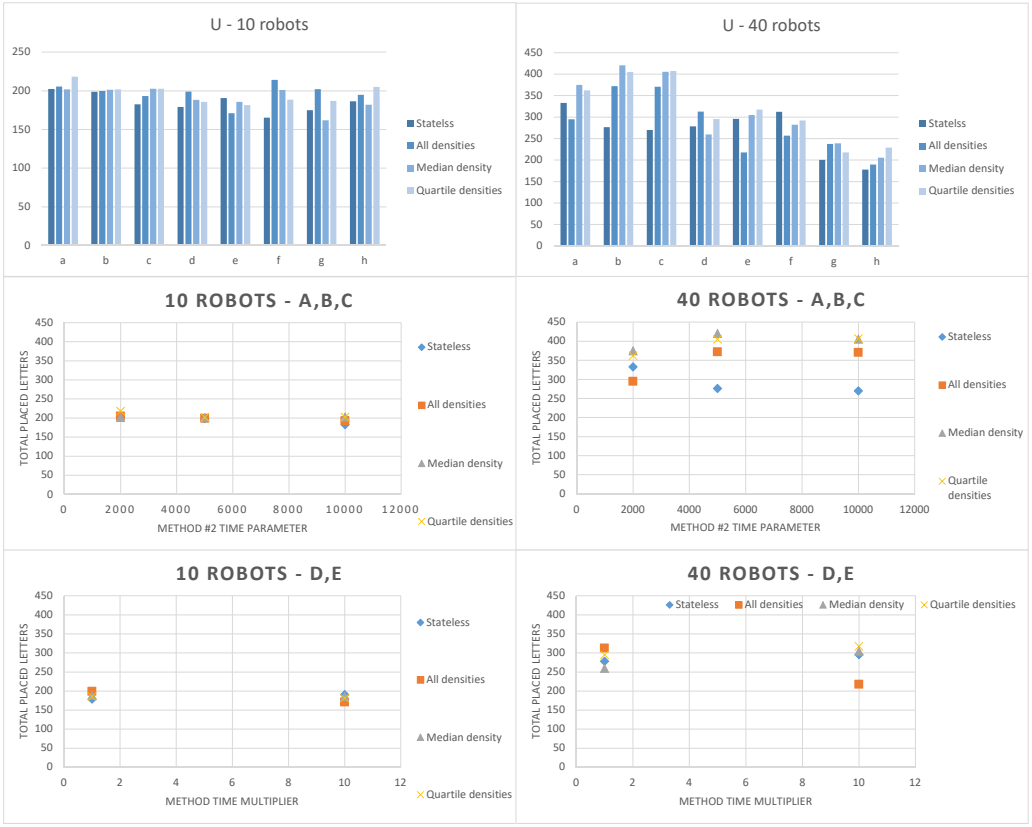

Figure 5.21: Performance of the Minimum Over Actions approximation for stateless learning vs. stateful learning with different state spaces.

Figure 5.21 shows that with 10 robots there is no significant difference, but with 40 robots there is a significant improvement in performance, especially in action sets (a), (b) and (c). Even though using all densities as a state space can improve performance, it can sometimes cause the performance to go below stateless learning, for example in action sets (a) and (e) with 40 robots. One possible explanation is the fact that this is the biggest state space and robots learn more slowly. Indeed the median and quartile state spaces that are sized 2 and 4 respectively, do not show this drop and many times perform better than stateless learning.

We now show how performance varies with different stateful learning algorithms. On average, using quartile densities as a state space yields the highest performance over the

four previously tested configurations. Therefore, quartile densities will be our benchmark state space for testing the performance of those stateful learning algorithms. We test three learning algorithms for the same 8 action sets as in table 5.6. The learning algorithms we test are described in table 5.8:

Learning algorithm name	Parameters
Regular Q-Learning	$\alpha = 0.05$, Exploration rate 0.02, $\gamma = 0$
Continuous time Q-Learning	$\tau = 10^{10}$ nanoseconds, Exploration rate 0.02, $\gamma = 0$
WoLF-PHC	Exploration rate 0.02, α=0.05, δ_w=0.0005,δ_l=0.005, $\gamma = 0$

Table 5.8: The learning algorithms tested for multiple states and their parameters.

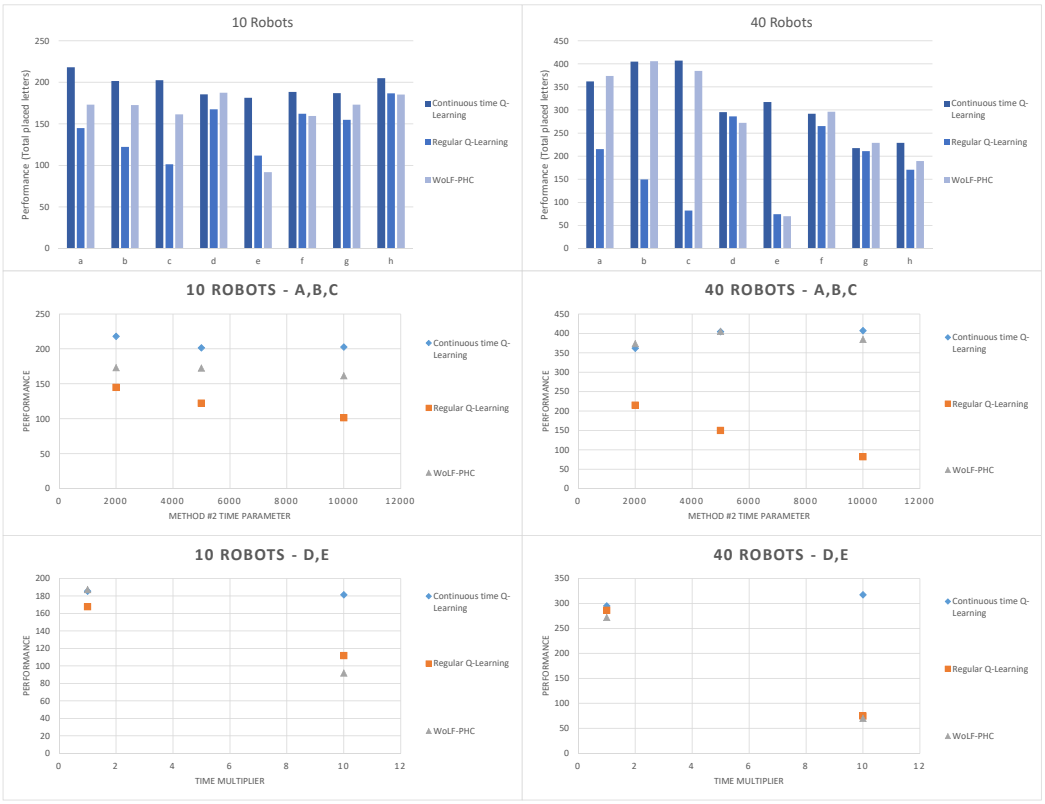

Figure 5.22: Performance of the Minimum Over Actions approximation for regular Q-Learning, continuous time Q-Learning and WoLF-PHC, each with density quartiles as a state space.

Figure 5.22 shows that continuous time Q-Learning either outperforms other algorithms or is on par with them. Special attention should be put to the first two groups of action sets: In both the first group (a)-(c) and the second group (d)-(e) we see that the performance of continuous time Q-Learning stays steady as parameters vary while the performance of the other two algorithms tends to decline.

Stateful learning vs. adaptation

In the previous section we have seen that the Minimum Over Actions approximation with multiple states performs well and stays steady while other algorithms sometimes decline in their performance. Therefore, we would like to compare this configuration to what was done in previous work and shown empirical success [27] - adaptation with EI. We test the two configurations on the same 8 action sets as in 5.6 for 10 robots and 40 robots. Table 5.9 shows the configurations for stateful learning and adaptation.

Configuration	Reward	Learning algorithm	Parameters
Stateful learning	Minimum Over Actions	Continuous time Q-Learning (stateful)	$\tau = 10^{10}$ nanoseconds, Exploration rate 0.02, $\gamma = 0$
Adaptation	EI	Regular Q-Learning (stateless)	Exploration rate 0.1, α=0.5

Table 5.9: Stateful learning vs. adaptation - configurations.

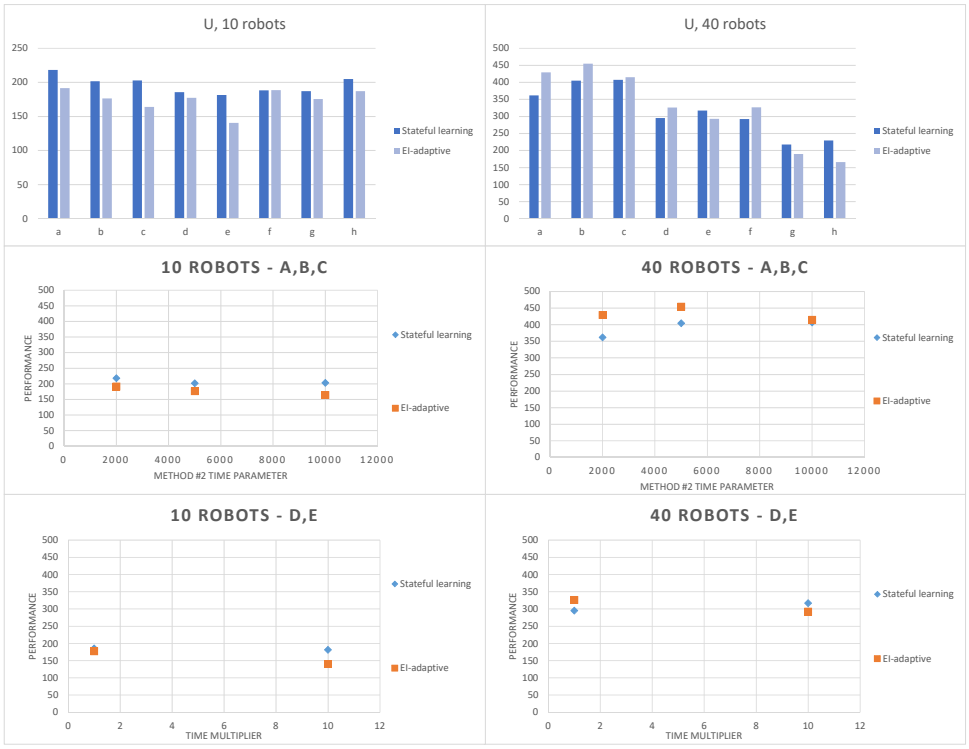

Figure 5.23: Performance of stateful learning in comparison to EI-based adaptation.

Figure 5.23 show that the performance of stateful learning and adaptation are close to each other. With 10 robots stateful learning almost always performs best while with 40 robots adaptation sometimes performs better than stateful learning. It can also be seen that for the second group of action sets (d) and (e) the performance of adaptation declines in the same manner as for stateful regular Q-Learning and stateful WoLF-PHC in figure 5.22

Conclusions

The starting point of this study was the promising approach of EI-based adaptation to the challenge of multi robot coordination. While EI-based adaptation demonstrates empirical success, this work also showed empirical evidence that its performance is sub-optimal. As a result, this thesis moved to more elaborate and extensive theoretical modeling and suggested practical solutions for reactive method arbitration.

The first part of the thesis focused on theoretical modeling, under a few assumptions, to show a mathematical connection between the robots' *Coordination Overhead* (*CO*) in the group task, to the global utility of the system. We then connected between the *CO* of the whole run to the EI of the robots per single collision. Using the two connections we now have a complete mathematical derivation between the EI of robots for a single collision and the global utility of the system. Using potential games we have shown a way for the robots to learn a reward for each collision that will yield optimal global utility.

The second part of the thesis, considered several practical solutions to challenges that may rise in practice when applying the theoretical model. First, we developed a continuous time variant of Q-Learning in order to address possible inaccuracies of regular Q-Learning that may rise in such domains. Second, we have suggested that the density of a robot can be used as a state space in order for it to know how many robots it may collide with.

After these gaps in theory and practice were addressed, the theoretical modeling and practical solutions were put to the test by experimenting in both simulation and real world. Experiments show that the algorithms used in order to overcome the gaps in previous work do improve performance while holding guarantees. Even though there is still room for improvement, both theoretically and practically, this work answered key questions in multi-robot coordination in general and reactive arbitration in particular.

References

[1] Noa Agmon, Noam Hazon, and Gal A. Kaminka. The giving tree: Constructing trees for efficient offline and online multi-robot coverage. *Annals of Math and Artificial Intelligence*, 52(2–4):143–168, 2008.

[2] Noa Agmon, Sarit Kraus, and Gal A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-08)*, 2008.

[3] Noa Agmon, Sarit Kraus, and Gal A. Kaminka. Multi-robot adversarial patrolling: Facing a full-knowledge opponent. *Journal of Artificial Intelligence Research*, 42: 887–916, December 2011.

[4] Gürdal Arslan, Jason R. Marden, and Jeff S. Shamma. Autonomous vehicle-target assignment: a game theoretical formulation. *ASME JOURNAL OF DYNAMIC SYSTEMS, MEASUREMENT AND CONTROL*, page 2007, 2007.

[5] Tucker Balch. The impact of diversity on performance in multi-robot foraging. In *Proceedings of the third annual conference on Autonomous Agents*, pages 92–99. ACM, 1999.

[6] Tucker Balch and Ron C. Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, December 1998.

[7] Maxim A Batalin and Gaurav S Sukhatme. Spreading out: A local approach to multi-robot coverage. In *Distributed Autonomous Robotic Systems 5*, pages 373–382. Springer, 2002.

[8] Sara Bouraine, Thierry Fraichard, Ouahiba Azouaoui, and Hassen Salhi. Passively safe partial motion planning for mobile robots with limited field-of-views in unknown dynamic environments. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2014.

[9] Michael Bowling and Manuela Veloso. Rational and convergent learning in stochastic games. In *International Joint Conference on Artificial Intelligence*, pages 1021–1026. LAWRENCE ERLBAUM ASSOCIATES LTD, 2001.

[10] Steven J. Bradtke and Michael O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In *Advances in Neural Information Processing Systems*, pages 393–400. MIT Press, 1994.

[11] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998:746–752, 1998.

[12] Panayiotis Danassis and Boi Faltings. A courteous learning rule for ad-hoc anticoordination. *arXiv preprint arXiv:1801.07140*, 01 2018. URL `https://arxiv.org/pdf/1801.07140.pdf`.

[13] J. P. Desai, J. Ostrowski, and V. Kumar. Controlling formations of multiple mobile robots. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 4, pages 2864–2869 vol.4, May 1998. doi: 10.1109/ROBOT.1998.680621.

[14] Yinon Douchan and Gal A. Kaminka. The effectiveness index intrinsic reward for coordinating service robots. In Spring Berman, Melvin Gauci, Emilio Frazzoli, Andreas Kolling, Roderich Gross, Alcherio Martinoli, and Fumitoshi Matsuno, editors, *13th International Symposium on Distributed Autonomous Robotic Systems (DARS-2016)*. Springer, November 2016.

[15] Yehuda Elmaliach, Asaf Shiloni, and Gal A. Kaminka. A realistic model of frequency-based multi-robot fence patrolling. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08)*, volume 1, pages 63–70, 2008.

[16] Yehuda Elmaliach, Noa Agmon, and Gal A. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Math and Artificial Intelligence*, 57(3–4):293–320, 2010.

[17] A. Farinelli, L. Iocchi, and D. Nardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2015–2028, Oct 2004. ISSN 1083-4419. doi: 10.1109/TSMCB.2004.832155.

[18] M. Fontan and M. Matarić. Territorial multi-robot task division. *IEEE Transactions of Robotics and Automation*, 14(5):815–822, 1998.

[19] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, Mar 1997.

[20] Luca Giuggioli, Idan Arye, Alexandro Heiblum Robles, and Gal A. Kaminka. From ants to birds: A novel bio-inspired approach to online area coverage. In Spring Berman, Melvin Gauci, Emilio Frazzoli, Andreas Kolling, Roderich Gross, Alcherio Martinoli, and Fumitoshi Matsuno, editors, *13th International Symposium on Distributed Autonomous Robotic Systems (DARS-2016)*. Springer, November 2016.

[21] Julio Godoy, Ioannis Karamouzas, Stephen J. Guy, and Maria Gini. Adaptive learning for multi-agent navigation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1577–1585. International Foundation for Autonomous Agents and Multi-Agent Systems, 2015.

[22] D. Goldberg and M. Matarić. Interference as a tool for designing and evaluating multi-robot controllers. In *AAAI/IAAI*, pages 637–642, 1997. URL `citeseer.nj.nec.com/goldberg97interference.html`.

[23] Christopher J. Hazard and Peter R. Wurman. Alphabet soup: A testbed for studying resource allocation in multi-vehicle systems. In *In Proceedings of the 2006 AAAI Workshop on Auction Mechanisms for Robot Coordination*, pages 23–30, 2006.

[24] Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, 4(Nov):1039–1069, 2003.

[25] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996. URL `http://arxiv.org/abs/cs.AI/9605103`.

[26] Gal A. Kaminka and Ruti Glick. Towards robust multi-robot formations. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-06)*, 2006.

[27] Gal A. Kaminka, Dan Erusalimchik, and Sarit Kraus. Adaptive multi-robot coordination: A game-theoretic perspective. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-10)*, 2010.

[28] Gal A. Kaminka, Meytal Traub, Yehuda Elmaliach, Dan Erusalimchik, and Alex Fridman. On the use of teamwork software for multi-robot formation control (an extended abstract). In *Proceedings of the Twelfth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-13)*, 2013.

[29] Gal A. Kaminka, Ilan Lupu, and Noa Agmon. Construction of optimal control graphs in multi-robot systems. In Spring Berman, Melvin Gauci, Emilio Frazzoli, Andreas Kolling, Roderich Gross, Alcherio Martinoli, and Fumitoshi Matsuno, editors, *13th International Symposium on Distributed Autonomous Robotic Systems (DARS-2016)*. Springer, November 2016.

[30] J. Kober, J. Andrew (Drew) Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, July 2013.

[31] Jason R. Marden and Adam Wierman. Overcoming limitations of game-theoretic distributed control. *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6466–6471, 2009.

[32] D Monderer and LS Shapley. Potential games. *Games and Economic Behavior*, 14 (1):124–143, 1996.

[33] Lynne E Parker. Multiple mobile robot systems. In *Springer Handbook of Robotics*, pages 921–941. Springer, 2008.

[34] Ioannis Rekleitis, Ai Peng New, Edward Samuel Rankin, and Howie Choset. Efficient boustrophedon multi-robot coverage: an algorithmic approach. *Annals of Mathematics and Artificial Intelligence*, 52(2):109–142, Apr 2008. ISSN 1573-7470. doi: 10.1007/s10472-009-9120-2. URL `https://doi.org/10.1007/s10472-009-9120-2`.

[35] Avi Rosenfeld, Gal A. Kaminka, Sarit Kraus, and Onn Shehory. A study of mechanisms for improving robotic group performance. *Artificial Intelligence*, 172(6–7): 633–655, 2008.

[36] Paul Rybski, A. Larson, M. Lindahl, and Maria Gini. Performance evaluation of multiple robots in a search and retrieval task. In *Proceedings of the Workshop on Artificial Intelligence and Manufacturing*, pages 153–160, Albuquerque, NM, August 1998.

[37] F. Sempe and A. Drogoul. Adaptive patrol for a group of robots. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2865–2869 vol.3, Oct 2003. doi: 10.1109/IROS.2003.1249305.

[38] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: Off-line design. *Artif. Intell.*, 73:231–252, 1995.

[39] Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.

[40] Peter Stone, Gal A. Kaminka, Sarit Kraus, Jeff Rosenschein, and Noa Agmon. Teaching and leading an ad hoc teammate: Collaboration without pre-coordination. *Artificial Intelligence*, 203:35–65, 2013. doi: 10.1016/j.artint.2013.07.003.

[41] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. URL `http://www.cs.ualberta.ca/~sutton/book/the-book.html`.

[42] Kagan Tumer, Adrian K. Agogino, and David H. Wolpert. Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, AAMAS '02, pages 378–385, New York, NY, USA, 2002. ACM. ISBN 1-58113-480-0. doi: 10.1145/544741.544832. URL `http://doi.acm.org/10.1145/544741.544832`.

[43] J. van den Berg, S. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. *Robotics Research*, pages 3–19, 2011.

[44] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Matarić. Go ahead, make my day: robot conflict resolution by aggressive competition. In *Proceedings of the 6th International Conference on the Simulation of Adaptive Behavior*, 2000.

[45] Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Machine Learning*, 8:279–292, 1992. URL `http://dblp.uni-trier.de/db/journals/ml/ml8.html#WatkinsD92`.

[46] David Wolpert, Kevin R. Wheeler, and Kagan Tumer. Collective intelligence for control of distributed dynamical systems. *CoRR*, cs.LG/9908013, 1999. URL `http://arxiv.org/abs/cs.LG/9908013`.

[47] David H. Wolpert and Kagan Tumer. An introduction to collective intelligence. Technical report, Handbook of Agent technology. AAAI, 1999.

[48] David H Wolpert and Kagan Tumer. An introduction to collective intelligence. *arXiv preprint cs/9908014*, 1999.

[49] Peter R. Wurman, Raffaello DÁndrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, Spring, 2008.

[50] Chuanbo Yan and Tao Zhang. Multi-robot patrol: A distributed algorithm based on expected idleness. *International Journal of Advanced Robotic Systems*, 13 (6):1729881416663666, December 2016. ISSN 1729-8814. doi: 10.1177/1729881416663666. URL `http://dx.doi.org/10.1177/1729881416663666`.

[51] Roi Yehoshua, Noa Agmon, and Gal A. Kaminka. Robotic adversarial coverage of known environments. *International Journal of Robotics Research*, 2016. doi: 10.1177/0278364915625785.

[52] Jingjin Yu and Steven M LaValle. Optimal multi-robot path planning on graphs: Structure and computational complexity. *arXiv preprint arXiv:1507.03289*, 2015.