ECOLE
POLYTECHNIQUE
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Intuitive mission specification for robot swarm by learning from demonstration

inverse reinforcement learning for robot swarms

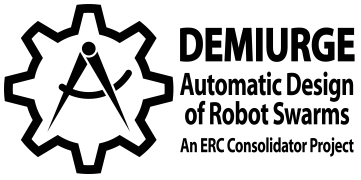
Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil en informatique

Ilyes Gharbi

Directeur
Professeur Mauro Birattari

Superviseur
Jonas Kuckling

Service
IRIDIA



Année académique
2021 - 2022

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme: DEMIURGE Project, grant agreement No 681872

Acknowledgement

First, I am extremely grateful to my thesis director, Mauro Birattari, for his enthusiasm and guidance toward my work and for encouraging me to always push the boundaries of my research.

I cannot begin to express my thanks to my supervisor, Jonas Kuckling, who is the best mentor I could have ever hope for. This thesis would not have been possible without the tremendous amount of time he spent supervising me.

I would also like to thank the researchers of IRIDIA, especially David Garzon Ramos, Miquel Kegeleirs and Guillermo Legarda Herranz, for the countless times they helped me during this last year and for making me comfortable at IRIDIA.

Finally, I would like to express my deepest thanks to my parents and Amélia for their unconditional love, support and encouragement during these difficult final years. Thank you.

Ilyes Gharbi

Résumé

Intuitive mission specification for robot swarm by learning from demonstration

Une alternative intéressante à la conception manuelle de programmes de contrôle pour les essaims de robots est la conception automatique par optimisation. Cependant, cette approche souffre de certaines contraintes. L'une d'elles est la recherche de la fonction objective relative à la mission de l'essaim de robots. En effet, cette fonction objective n'est pas toujours triviale, même pour un expert en robotique en essaim. Idéalement, une fonction objective adéquate devrait être calculable pour toute mission aussi complexe soit elle. De plus, même les non-experts en robotiques en essaim devraient être en mesure de concevoir de bons comportements en essaim pour des missions complexes, et cela, cela sans aucun savoir spécifique a priori.

Ainsi, ce mémoire présente Demonstration-`Cho`, une méthode pour produire des programmes de contrôle pour robots, sans avoir besoin d'expliciter de fonctions objectives de la mission de l'essaim de robots. Cette méthode permet d'apprendre une représentation d'une fonction objective depuis des démonstrations humaines de comportement d'essaim de robots. Grâce à la fonction objective ainsi apprise, Demonstration-`Cho` génère automatiquement le programme de contrôle menant au comportement d'essaim souhaité.

Durant ce mémoire, quatre missions ont été utilisées pour tester les performances de Demonstration-`Cho`. L'évaluation des performances de l'essaim est basée sur la répartition spatiale des robots à la fin de la mission. Les résultats ont montré que Demonstration-`Cho` est capable de produire des programmes de contrôle de performances comparables à ceux produits par `EvoStick` et AutoMoDe-`Chocolate` munient des fonctions objectives explicites de chaque missions.

Mots-clés : robotique en essaim, apprentissage par renforcement inverse, apprentissage par démonstration, AutoMoDe, Orchestra, E-Puck.

Mémoire présentée en vue de l'obtention du diplôme d'Ingénieur Civil en informatique
Ilyes Gharbi, 2021-2022

Abstract

Automatic design by optimization is one promising alternative to the manual design of control software for robot swarms. Although this approach offers many benefits, it also entails some challenges. One is to find the correct objective function fitting a specified robot swarm mission. Indeed, these objective functions are not always trivial, even for experts in swarm robotics. Ideally, suiting objective functions could be processed for missions as complex as wanted. Furthermore, even non-expert operators could be able to retrieve relevant swarm behaviour for such complex missions without any prior advanced knowledge.

Hence in this master thesis, I present Demonstration-`Cho`, a method to produce robot control software without providing an explicit objective function. This method allows learning a representation of an objective function from a set of human demonstrations. With the learned objective function, Demonstration-`Cho` automatically generates control software, using AutoMoDe-`Chocolate`, leading to desired collective behaviours.

In this master thesis, I assess Demonstration-`Cho` in four missions, providing only demonstrations of the spatial distribution of the swarm at the end of the mission's time. The results of the experiments show that Demonstration-`Cho` can design control software with comparable performances to those generated with `EvoStick` and AutoMoDe-`Chocolate` with the explicit objective function of the missions.

Keywords : swarm robotics, inverse reinforcement learning, apprenticeship learning, AutoMoDe, ARGoS, E-Puck.

Contents

Chapter 1

Introduction

Swarm robotics [18], inspired by the field of swarm intelligence [17], is the engineering field that aims to design large groups of autonomous robots collaborating to accomplish a specific mission. The swarm behaviour emerges from the many interactions of the robots with each other and the environment. A characteristic of a robot swarm is that every robot only has access to local information and local interaction, leading to decentralised self-organisation. From these, some desirable properties, such as flexibility or resilience, can emerge.

Although desirable properties might emerge thanks to decentralisation and self-organisation, these characteristics lead to a significant challenge in the field of swarm robotics; the micro-macro link problem [30]. This problem arises from the non-trivial relation between the swarm's desired collective behaviour and the robot's designed individual behaviour. There is currently no general methodology to design the robots' behaviour to access the desired emergent overall swarm behaviour.

Thus, several design approaches exist to design and implement the individual behaviours in swarm robotics. Two of them are manual and automatic designs. On the one hand, the manual design approach involves a human expert designer conceiving the control software by trial-and-error. This approach cannot guarantee to succeed in every situation in a reasonable amount of time, and human resources [10].

In automatic design, there is no human intervention beyond the specification of the mission. Several automatic design methods have been proposed [22] [12] [53]. Automatic design is mostly performed using evolutionary swarm robotics [62]. In this approach, an artificial neural network (ANN) controls the robot by taking its sensors' values as input and outputting the value of the robot's actuators. The weights of the ANN are updated by optimising a mission-specific performance measure via artificial evolution, typically in simulation [33]. The benefit of evolutionary swarm robotics, like the other optimisation-based automatic design, is to avoid the need to explicitly reduce the collective behaviour of the swarm to the individual behaviour of the robots in the swarm. Thus, the ANNs used in evolutionary swarm robotics are universal approximators [37]. An ANN can represent any behaviour constrained to the robot's sensors and actuators. Although evolutionary swarm robotics perform well in simulation, it is less efficient when applied to real robots due to the reality gap [38]. Indeed, the issue with the ANN of evolutionary swarm robotics is the high variance of its parameters to optimise. This high variance causes an ANN optimised to overfit too much in simulation. However, the simulation is inevitably different from reality, so the ANN may perform poorly compared to its performance in simulation when applied to real robots, even if it leads to optimal behaviour in simulation.

Automatic modular design is an automatic design of swarm robot methods that proposes addressing the reality gap problem. In machine learning, the bias-variance trade-off states that the variance of a model estimation can be reduced by increasing bias in the model estimation [6]. Hence, automatic modular design introduces bias in the representation capacity of behaviours of its method. AutoMoDe [25] is a family of automatic modular design methods. To produce the control software, the authors of AutoMoDe decided on a set of manually pre-defined behaviour modules to be automatically assembled in a finite state machine. By constraining the available behaviour options, this approach reduces the representation power of its model. This increase of bias in the behaviour selection lowers the variance in the parameter to be optimised by AutoMoDe. Hence, AutoMoDe is more efficient in crossing the reality gap than evolutionary swarm robotics [23].

To produce the control software, the design methods in the AutoMoDe family optimise a mission-specific objective function. Unfortunately, computing explicitly that task-specific objective function is none trivial, especially when the mission gets more complex even for experts in swarm robotics [30].

An interesting approach to overcoming the mission-specific objective function issue can be investigated in reinforcement learning (RL) [39]. In RL, an agent needs to learn an optimal set of actions to take in order to maximise its reward. Like in automatic design of control software for robot swarms, it is assumed in RL that the reward function is known a priori. In inverse reinforcement learning (IRL), the reward function is derived from observations of the agent's behaviour [59]. The Apprenticeship Learning algorithm is one approach to approximate the reward function by learning from an expert's demonstrations [1]. This technique can indeed permit to overlook the need to find a mission-fitting objective function as it will be learned during the design process and optimised to generate relevant behaviours.

1.1 Objective of the master thesis

The goal of this master thesis is the specification of swarm robotics missions without the need for an explicit objective function. To that end, I worked on three sub-goals:

1. Create a graphical mission-builder by expending the features of the software Orchestra (see Section 3.4 and Chapter 4).

2. Investigate how to learn swarm robotics behaviours from feedback with the algorithm APRIL (see Chapter 5).

3. Investigate how to learn robotics behaviours from demonstrations with the apprenticeship learning algorithm (see Chapter 6).

I considered two algorithms in this master thesis because the results with the first one, APRIL, were not satisfying, as I will explain in Chapter 5. However, working with the second one, the apprenticeship learning algorithm, produced positive results.

1.2 Contributions of the master thesis

During this master thesis, I delivered three contributions to meet the previously stated goals:

1. An arena builder, implemented in Orchestra to build an experimental arena for a robot swarm mission from the list provided in Section 6.4.

2. A demonstration-maker, implemented in Orchestra to make demonstrations of the final spatial distribution of the robots swarm (see section 3.4).

3. Demonstration-`Cho`, an automatic modular design method that combines apprenticeship learning for learning the task-specific objective function from expert demonstrations, and AutoMoDe-`Chocolate` to design control software according to the learned objective function.

In this master thesis, I also assess the quality of the control software generated by Demonstration-`Cho` in four missions. Indeed, I use four experiments relying on the spatial distribution of the robot swarm to assess the performance of Demonstration-`Cho` with two control software design process baselines I call Objective-`Evo` and Objective-`Cho` in this master thesis. The first one is `EvoStick` and the one second is AutoMoDe-`Chocolate`. Both baseline methods can assess their performance on the mission-specific reward function of each mission. The results of the experiments showed that Demonstration-`Cho` produced control software with comparable performances with the ones produced by Objective-`Evo` and Objective-`Cho` (see Chapter 8).

Chapter 2

Related Work

Powerful and cutting-edge evolutionary algorithms using artificial intelligence are widely used to produce accurate behaviour in simulation. Although evolutionary algorithms perform well in simulation, their performances drop when behaviours optimized in simulation transfer to real robots. Hence, the automatic modular design proposes to be more robust to the reality gap problem.

Still, the problem of specifying accurately what the robot swarm needs to accomplish is non-trivial. Indeed, the more complex the mission is, the more difficult it will be to specify it mathematically. Hence, this master thesis also investigates the field of inverse reinforcement learning that proposes to learn the objective function rather than computing it explicitly.
This chapter is structured as follows: Section 2.1 defines swarm robotics and its main properties; Section 2.2 presents evolutionary robotics as it serves as the baseline for the experiments of this master thesis; Section 2.3 presents automatic modular design and one particular implementation used in this master thesis called AutoMoDe-`Chocolate`; Section 2.4 presents reinforcement learning and its formalism briefly; Section 2.5 presents inverse reinforcement learning and its formalism; Section 2.6 presents Preference-base Policy learning and its formalism.

2.1 Swarm Robotics

Swarm intelligence [7] [18] is the discipline dealing with collective systems of many self-organized agents collaborating in a decentralized way and relying solely on local communication and interactions. Swarm robotics [18] [26] [30] [5] [19] [13] is the field of robotics that applies the principles of swarm intelligence. In swarm robotics, complex behaviour emerges from the many interactions among the robots composing the swarm with themselves and the environment.

Typically, a swarm of robots has the following properties:

— The swarm is typically composed of many homogeneous robots, although some examples of heterogeneous robot swarm exist [13].

— The robots interact locally and have only access to local information.

— The robots are simple and display simple behaviours.

— The overall behaviour of the swarm emerges from the interaction of the robots between themselves and the environment.

Thanks to the properties mentioned above, it is easier to ensure the following desirable properties for a robot swarm:

— Scalability: the robots in the swarm only rely on local information. Hence, they do not know the swarm's size, and the neighbourhood's size remains approximately constant. So the swarm can maintain its function while its population of robots increase or decrease.

— Flexibility: the control software designed in simulation allows the robots to adapt to different environments. For example, if the swarm mission is to aggregate in a marked location, the robot can adapt if this marked location changes place.

— Resilience: due to its inherent decentralization and self-organization, the swarm of robots isn't affected by the loss of some robots if they can be replaced. Indeed, all robots in the swarm are interchangeable, and none of them is responsible for the global coherence of the group.

2.2 Evolutionary Swarm robotics

Evolutionary robotics (ER) is a method to automatically produce robot controllers by using algorithms inspired by Darwin's principle of natural selection [70] [21] [63] [8] [62]. In ER, each robot controller candidate, often ANNs, is represented by a genotype. The genotype is a string formed by the set of parameters of the candidate robot controller. Then, an evolutionary algorithm is applied to genotypes to select the set of parameters with optimal performance regarding some evaluation metric. The particular case of evolutionary robotics with ANN is called neuro-evolutionary [36] [74] [64].

ER is also applied to swarm robotics [71] [72] both in simulation and real robots. For example, Groß and Dorigo use artificial evolution to successfully produce transport behaviour for simulated insect-like robots [29]. Duarte *et al.* even transfer the collective behaviour of aquatic swarm robot from simulation to reality [20]. Although they successfully transferred the collective behaviours to the real robots, some motion patterns did not transfer well due to the reality gap. Hasselmann *et al.* has empirically studied the effect of reality gap to neuro-evolution [33] and concluded that the robustness of neuro-evolution to the reality gap is the main issue to address in that field. Indeed, the ANNs used in neuro-evolution are universal function approximates with generally high variance in the parameters to be optimized. This high variance is the cause of the poor robustness of neuro-evolution to the reality gap.

Francesca *et al.* define EvoStick, a method based on evolutionary robotics and which implement automatic design for robot swarm [25].

2.3 Automatic Modular Design

An alternative to evolutionary swarm robotics is automatic modular design [68]. The difference between those two techniques is that in automatic modular design, the process does not optimize ANN but rather assembles manually pre-defined modules to create robot control software. This approach induces bias into the conception of the robot behaviour as it reduces the representation power of the control software in the behaviour space. However, the bias-variance trade-off in machine learning states that injecting bias permits to reduce the variance. Hence, automatic

modular design's main feature is to be more robust to the reality gap than neuro-evolutionary swarm robotics.

2.3.1 AutoMoDe

Francesca *et al.* propose AutoMoDe, a family of automatic modular design methods [25]. AutoMoDe aims to automatically design control software for robot swarms by optimizing a mission-specific objective function. Multiple instances, called flavours, compose the AutoMoDe family [9]:

— AutoMoDe-`Vanilla` [23] is the first flavour of AutoMoDe. It uses a set of manually predefined modules to produce control software for robot swarms. There are two types of modules, the behaviour and the transition modules. The behaviour modules represent simple actions, described in Tables 2.1. The robot performs a behaviour until meeting some specific conditions described by the transition modules shown in Table 2.2. These modules were meant for the reference model RM1.1 of the e-puck (see Section 3.1). The modules are assembled in the form of a probabilistic finite state machine (PFSM). The probabilistic finite state machine assembled by AutoMoDe-`Vanilla` is composed of behaviour modules linked by the transition modules (see Figure 2.1). Some behaviour and transition of modules have parameters for their operation. The transition between two behaviour modules through a transition module's condition is probabilistic. The optimisation algorithm of AutoMoDe-`Vanilla` is F-Race [11] (see Section 2.3.2). Francesca *et al.* evaluate the performance of AutoMoDe-`Vanilla` in regards to EvoStick and two types of human designer called U-Human and C-Human [23]. The U-Human designer is free to conceive the control software as he pleased. To conceive the control software, the C-Human is constrained to use the same modules as AutoMoDe-`Vanilla`. The evaluation of the design processes was done on five missions: SCA, LCN, CFA, SPC and AAC (see Section 6.4). The experiments of Francesca *et al.* show that AutoMoDe-`Vanilla` outperforms EvoStick and U-Human. However, C-Human outperformed AutoMoDe-`Vanilla`. As C-Human uses the same modules as AutoMoDe-`Vanilla`, the results of the experiments motivate the authors to investigate how to upgrade the optimizer of AutoMoDe-`Vanilla`.

— AutoMoDe-`Chocolate` [24] is an improved version of AutoMoDe-`Vanilla`. It uses the same modules as AutoMoDe-`Vanilla`. However, the results of the experiments of Francesca *et al.* [23] on the same five missions as for AutoMoDe-`Vanilla` evaluation, motivate the authors to replace F-Race by Iterated F-Race (irace) [11] [55] [54] as the optimizing algorithm. Irace is presented in Section 2.3.2. Francesca *et al.* show that AutoMoDe-`Chocolate` outperform C-Human [24] on the same missions as in the experiments with AutoMoDe-`Vanilla`.

— AutoMoDe-`Gianduja` [32] extends AutoMoDe-`Chocolate` by allowing robot direct communication. AutoMoDe-`Gianduja` modules are based on the reference module RM2 of the e-puck [32]. AutoMoDe-`Gianduja` addresses one limitation of AutoMoDe-`Vanilla` and AutoMoDe-`Chocolate` with is the non-exploitation of explicit communications among the robot swarm. Hasselmann *et al.* evaluate the performance of AutoMoDe-`Gianduja` in regards to AutoMoDe-`Chocolate` and EvoCom on three missions [32]: `Aggregation`, `Stop` and `Decision` (see Section 6.4). Hasselmann *et al.* define EvoCom as the extension of EvoStick that uses the reference model RM2 of the e-puck. The results of the experiment by Hasselmann *et al.* show that AutoMoDe-`Gianduja` outperforms the other two design processes on the three missions. The results show that AutoMoDe-`Gianduja` uses the communication in the swarm meaningfully and efficiently to accomplish the considered missions.

— AutoMoDe-`Waffle` [66] uses the same modules as AutoMoDe-`Chocolate`. However, the AutoMoDe-`Waffle` novelty is the automatic selection of the hardware configuration and the number of robots in the swarm. Salman *et al.* considered *economic constraints* experiments to study the performance of AutoMoDe-`Waffle` [66]. Salman *et al.* evaluate AutoMoDe-`Waffle` on three missions: `Anytime-Selection`, `End-Time-Aggregation` and `Foraging` (see 6.4). The evaluation of the three missions was done under the economic constraint of the total monetary budget available and the battery capacity of each robot in the swarm. The results of the experiments show that the performance of AutoMoDe-`Waffle` depends on the nature of the swarm mission. However, the experiments with real robots asses that AutoMoDe-`Waffle` is robust to the reality gap.

— AutoMoDe-`Maple` [46] uses the same modules as AutoMoDe-`Chocolate`. However, rather than assembling the modules in a probabilistic state machine, AutoMoDe-`Maple` assembles the modules in the form of a behaviour tree. Kuckling *et al.* compare the performance of AutoMoDe-`Maple` with AutoMoDe-`Chocolate` and EvoStick on two missions [46]: `Foraging` and `Agreggation` (see Section 6.4). The results of the experiments suggest that behaviour trees are a promising alternative architecture to the probabilistic finite state machine architecture.

— AutoMoDe-`Mate` [56], specialized in designing spatially-organized behaviour, extends AutoMoDe-`Chocolate`. The modules of AutoMoDe-`Mate` are based on the reference module RM3.1 of the e-puck [56]. The difference with the modules of AutoMoDe-`Chocolate` is that the modules of AutoMoDe-`Mate` consider the omnidirectional camera and the RGB LEDs of the e-puck. Mendiburu *et al.* evaluate the performance of AutoMoDe-`Mate` in comparison to AutoMoDe-`Chocolate` and EvoSpace. Mendiburu *et al.* define EvoSpace as the extension of EvoStick based on the reference model RM3.1 of the e-puck. Mendiburu *et al.* conducts experiments both in simulation and with real robots on three mission: `Any-Point Coverage`, `Networked Coverage` and `Conditional Coverage`.

— AutoMoDe-`TuttiFrutti` [28] uses modules based on the RGB LEDs and the omnidirectional camera of the e-puck to design control software. Garzón Ramos *et al.* evaluate the performance of AutoMoDe-`TuttiFrutti` alongside the performance of EvoColor, an extended version of EvoStick that produces control software for the e-puck that can display and perceive colours [28]. The experiments conducted by Garzón Ramos *et al.*, in both simulation and with real robots, evaluate the performance of the two design processes on three missions: `Aggregation`, `Stop` and `Foraging` (see 6.4). The results of the experiments show that AutoMoDe-`TuttiFrutti` outperforms EvoColor for the three classes of missions.

— AutoMoDe-`Coconut` [69] uses the same modules as AutoMoDe-`Chocolate`. The only difference with AutoMoDe-`Chocolate` is that AutoMoDe-`Coconut` introduces a parameter to control the type of exploration scheme of the modules. Spaey *et al.* compare the performance of AutoMoDe-`Coconut` and AutoMoDe-`Chocolate` on experiments in simulation and reality[69]. The experiments missions are: `Aggregation`, `Foraging` and `Grid Exploration` (see Section 6.4). The particularity of the experiments of Spaey *et al.* is that the two design processes are also evaluated on an unbounded alternative version of the three missions. In this unbounded alternative version, three walls have been removed from the arena. The results of the experiments did not show a difference in performance between the two design processes. Surprisingly, although AutoMoDe-`Chocolate` only use ballistic motion in the exploration modules, it yields a similar performance as AutoMoDe-`Coconut` on the unbound version of the missions.

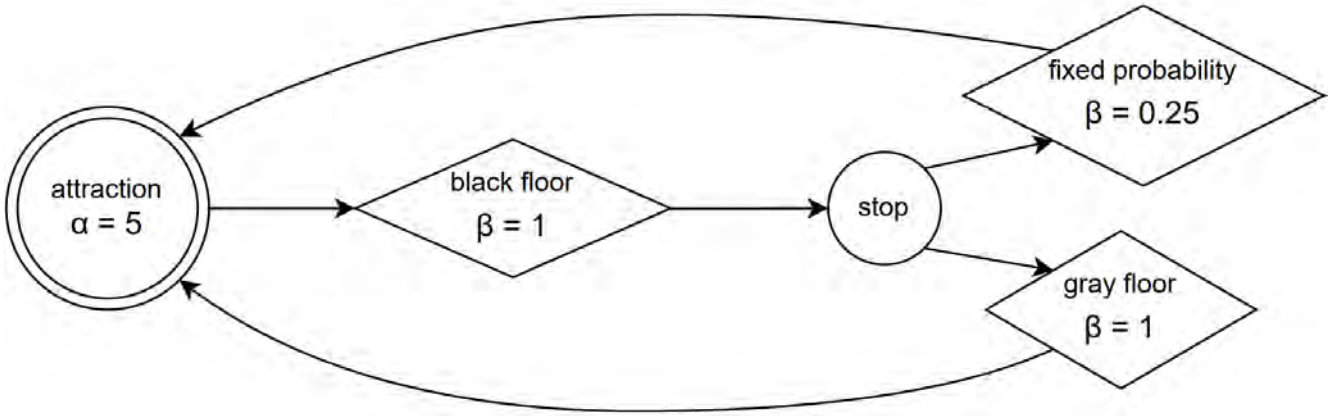


Figure 2.1 – From the paper of Mauro Birattari, Antoine Ligot and Gianpiero Francesca [9], this figure represents an example of control software produced by AutoMoDe in the form of a probabilistic finite state machine. The circles represent the modules, and the diamonds represent the transition conditions with probability β.

— AutoMoDe-`IcePop` [44] uses the same modules as AutoMoDe-`Chocolate`. The difference between the two flavours is that AutoMoDe-`IcePop` uses simulated annealing as an optimizing algorithm. Kuckling *et al.* compare the performance of AutoMoDe-`IcePop` in comparison to AutoMoDe-`Chocolate` in experiments conducted in simulation and pseudo-reality on two missions: `AAC` and `Foraging` (see Section 6.4). The experiment results suggest that simulated annealing is a viable optimizer for automatic modular design for robot swarm.

— AutoMoDe-`Cedrata` [42], like AutoMoDe-`Maple`, uses behaviour tree architecture. However, AutoMoDe-`Cedrata` has modules explicitly defined for behaviour trees. The reference model used for defining the modules of AutoMoDe-`Cedrata` is RM2.2. Kuckling *et al.* conduct experiments to evaluate the performance of AutoMoDe-`Cedrata` against human designer using the same modules as AutoMoDe-`Cedrata`. The experiments were conducted on two classes of missions: `Marked Aggregation` and `Stop` (see Section 6.4). The results show that the human designer could produce control software with satisfactory results with the modules of AutoMoDe-`Cedrata` even with no prior experience with behaviour trees. On the other hand, the results show that AutoMoDe-`Cedrata` cannot produce communication-based control software.

— AutoMoDe-`Arlequin` [50] is similar to AutoMoDe-`Chocolate` except for the behaviour modules. Indeed, the behaviour modules of AutoMoDe-`Arlequin` are ANNs automatically generated in a mission-agnostic way. ANNs reduce the amount of human intervention for conceiving the behaviour modules. Ligot *et al.* compare AutoMoDe-`Arlequin` to AutoMoDe-`Chocolate` and EvoStick in experiments on two missions: `Foraging` and `Agreggation-XOR` (see Section 6.4). The experiments' results show that AutoMoDe-`Arlequin` suffers from the reality gap, however, less than EvoStick.

Behaviour	Parameter(s)	Description
Exploration	$\tau \in \{1, 2, ..., 100\}$	The robot moves straight. If $prox_i \geq 0.1$ for $i \in \{1, 2, 7, 8\}$, the robot turns on itself for a random number of control cycles chosen in $\{0, 1, ..., \tau\}$.
Stop	None	The robot stays still.
Phototaxis	k fixed to 5	The robot moves straight to light source if perceived; otherwise, moves straight. Obstacle avoidance is embedded and depends on k.
Anti-phototaxis	k fixed to 5	The robot moves away from light source if perceived; otherwise, moves straight. Obstacle avoidance is embedded and depends on k.
Attraction	$\alpha \in [1, 5]$ and k fixed to 5	The robot moves straight to the neighbour robot thanks to the rang-and-bearing device; otherwise, move straight. Obstacle avoidance is embedded and depends on k and α.
Repulsion	k fixed to 5	The robot moves away from neighbour robots; otherwise, moves straight. Obstacle avoidance is embedded and depends on k.

Table 2.1 – This table represents the set of behaviour modules used in AutoMoDe-`Vanilla` and AutoMoDe-`Chocolate` [25]. These behaviour modules are the state in the probabilistic finite state machine representing the behaviour of each robot in the swarm. The behaviour modules were conceived for the E-puck's reference model RM1.1.

Condition	Parameter(s)	Description
Black-floor	$\beta \in [0, 1]$	If $gnd_i = 0$ for $i \in \{1, 2, 3\}$, the transition is enable with probability β.
Grey-floor	$\beta \in [0, 1]$	If $gnd_i = 0.5$ for $i \in \{1, 2, 3\}$, the transition is enable with probability β.
White-floor	$\beta \in [0, 1]$	If $gnd_i = 1$ for $i \in \{1, 2, 3\}$, the transition is enable with probability β.
Neighbor-count	$\eta \in [0, 20]$ and $\xi \in \{0, 1, ..., 10\}$	The transition is enable with probability $z(n) = \frac{1}{1+e^{\eta(\xi-\eta)}}$, where n is the number of neighbouring robots.
Inverted-Neighbor-count	$\eta \in [0, 20]$ and $\xi \in \{0, 1, ..., 10\}$	The transition is enable with probability $1 - z(n)$.
Fixed-probability	$\beta \in [0, 1]$	The transition is enable with a fixed probability of β.

Table 2.2 – This table represents the set of transition condition modules used in AutoMoDe-`Vanilla` and AutoMoDe-`Chocolate` [25]. These transition condition modules are the transition conditions in the finite state machine representing the condition of transition of the behaviour of each robot in the swarm. The transition condition modules were conceived for the e-puck's reference model RM1.1

2.3.2 F-Race and Iterated F-Race

F-Race [11] is an optimization algorithm. It is used to find the configuration of an algorithm that optimizes a certain evaluation metric. In the case of AutoMoDe, the configuration to be optimized is the topology of the PFSM and the parameters of its module, and the evaluation metric is the swarm mission objective function.

The racing approach consists of evaluating algorithm instances in parallel on several instances. After evaluating the configurations on each instance, the statistically significantly worse than other configurations are discarded. Hence at each step, fewer candidates remain until only a set of elite configurations is kept.

Iterated F-Race (Irace) [54] performs several iterations, each reminiscent of a race, similar to F-Race. The initial configurations of an iteration are then sampled around the surviving elites of the previous iteration. The computational budget is the number of simulations performs in irace to evaluate the configurations.

2.4 Reinforcement Learning

Reinforcement learning [39] is the branch of Machine Learning involving a learning agent in an environment that must learn behaviour through trials and errors. The agent learns from its action by trying to optimize a reward function. The learning agent can be powered by any model, from probabilistic finite state machines to deep neural networks.

Mathematically, a classical reinforcement learning problem is modelled by a Markovian Decision Process (MDP) which includes:

— a set of states S.

— a set of actions A.

— a probabilities of transition from a state $s \in S$ to a state $s' \in S$ under an action $a \in A$ noted $P(s'|s, a)$.

— a reward function granted after a transition from state $s \in S$ to state $s' \in S$ under an action $a \in A$ noted $R_a(s, s')$.

Hence, the goal of a reinforcement learning agent is to learn a behaviour (or policy) that would maximize the reward function of each action taken in each state.

To compute this optimal policy, many techniques can be deployed. The most known is called the Q-learning [73]. In this technique, actions taken by the agent for a certain state have a certain quality named the Q-value. Q-values are updated along the agent learning. They can be stored in a table with as many entries as there are combinations of action-state. When the number of action-state combinations is too large to store all the Q-values, the Q-values can be approximated by using neural networks. This techniques is called Deep reinforcement learning (DRL) or Deep Q-Learning (DQL) [57].

Reinforcement learning (RL) is widely used for single robot training and can produce impressive results. For example, Ng *et al.* achieve to train an autonomous helicopter to fly upside down by RL [60]. Heidrich-Meisne *et al.* even propose to apply neuro-evolution in RL [36]. They evaluate the performance of their method called CMA-ES with other RL techniques. Their experience shows that their method outperforms every RL technique considered in the paper's conducted experiments.

RL can also be applied for multi-agent systems of a few learning agents [14]. In the multi-agents case, the Q-values of an agent also depend on the joint actions it has with the other agents. This means that an agent no longer only ought to maximize its reward but also the overall reward of the group. This also holds even to the detriment of its reward function maximization. However, this approach supposes a centralized structure that distributes the reward to the agents. RL for decentralized multi-agent is not often addressed in the literature, even less for multi-robotics applications. Buşoniu *et al.* compare centralized and decentralized multi-agent RL [15]. They illustrate the difference between the two approaches with an example of a two-link rigid robotic manipulator. In this experiment, two robots are linked with two rigid sticks, forming a structure similar to a double pendulum. The goal is to balance the system near a certain angle. Both centralized and decentralized RL is applied to balance the system with the learning two robots. The results of the experiment by Buşoniu *et al.* show that the decentralized approach learns well-performing controllers while using less computational resources than the centralized approach.

2.5 Inverse Reinforcement Learning

Classical RL assumes that the reward function is known. When it is not the case, the problem is modelled as an MDP without a known reward function, noted $MDP\backslash R$. This particular case

of RL problem is called inverse reinforcement learning (IRL) [4] [75].

In IRL, an agent must learn a policy by observing another agent's behaviour demonstration. Therefore, IRL can be seen as the inverted problem of RL. Indeed, whereas in RL, the agent seeks to learn a policy to maximize a reward function, in IRL, the agent seeks an accurate reward function representation to explain the expert policy.

One technique for IRL is the apprenticeship learning [1]. This technique assumes the existence of a « true » reward function noted $R^* = w^* \cdot \phi(s)$, where $s \in S$ is the agent state, $\phi : S \rightarrow [0,1]^k$ is a vector of features over the set of all agent's states, and $w^* \in \mathbb{R}^k$ is the « true » weighting vector. A policy π is the mapping between the state and the probability distribution of the actions. The value of a policy can be computed as the expected discounted reward collected by π over time for all initial states:

$$E_{s_0 \sim D}[V^\pi(s_0)] = E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] \tag{2.1}$$

$$= E[\sum_{t=0}^{\infty} \gamma^t w.\phi(s_t)|\pi] \tag{2.2}$$

$$= w.E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi] \tag{2.3}$$

The apprenticeship learning algorithm defines $\mu(\pi) = E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi]$, the expected features of that policy π. The apprenticeship learning algorithm requires an estimation of the expert's expected feature $mu_E = \mu(\pi_E)$. Specifically, given the set of states generated by the m expert's demonstration $\{s_0^{(i)}, s_1^{(i)}, ...\}_{i=1}^m$, the empirical estimate of μ_E is defined as $\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^m \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)})$. The value of a policy π can then be written: $E_{s_0 \sim D}[V^\pi(s_0)] = w \cdot \mu(\pi)$.

In apprenticeship learning, the expert's demonstration is represented by the feature expectation vector μ_E. The goal of the algorithm is to find a policy with performance close to the expert regarding the unknown « true » reward function $R^* = w^*.\phi$. Hence, the algorithm needs to find a policy $\widetilde{\pi}$ such that $||\mu(\widetilde{\pi}) - \mu_E||^2 < \epsilon$. The weighting vector w to compute the learned reward function $R = w \cdot \phi$ is obtained by solving the quadratic constrained optimization problem:

$$\begin{aligned} \max_{t,w} \quad & t \\ \text{s.t.} \quad & w^T \mu_E \geq w^T \mu_j + t \qquad \forall 0 \leq j \leq i-1 \\ & ||w||_2 \leq 1 \end{aligned} \tag{2.4}$$

This problem is solved using a support vector machine (see Section 3.5). In Figure 3.4, we see the representations of the policies on the expected feature space. For each algorithm iteration, the vector w is updated so the algorithm can produce a policy optimizing the newly learned reward function. Eventually, the distance between the expected features vector of a produced policy and the expected features vectors of the expert demonstration will be small, meaning that the produced policy matches the expert policy.

Abbeel *et al.* applied the apprenticeship learning in simulation in two experiments [1]. In the first experiment, an agent is placed in a gridworld where some regions have positive rewards. The reward function is not accessible, but so are expert trajectories. The agent then must learn from the expert's trajectories in the gridworld to find the region with the positive reward. The second experiment consists of simulated self-driving cars circulating on a three-band road. On

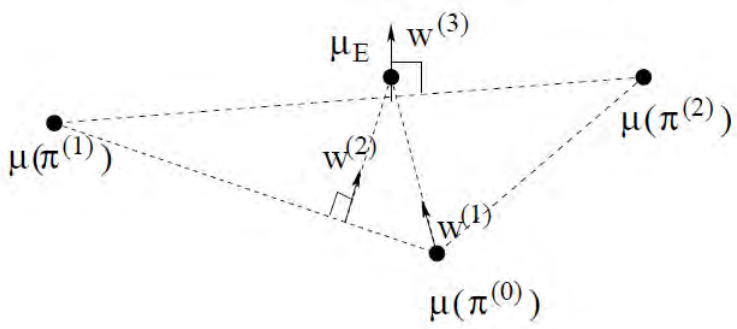


Figure 2.2 – This figure represents three iterations of the apprenticeship learning where the expected features of the produced policies approach the expected features of the expert demonstration μ_E (taken From the paper of Akrour and Ng [1]).

this road, other cars circulate. In this experiment, several « driving style » are shown to the learning agent. The driving style ranges from avoiding all collisions or colliding with all cars, or trying to stay on the left. The experiment's goal is to use apprenticeship learning to try learning a policy to mimic the demonstrated driving styles. In both experiments conducted by Abbeel *et al.*, the apprenticeship learning achieved to derive implicit reward function that led to policies that performed well on the « true » reward functions of each experiment.

2.6 Preference-based Policy Learning

Another way to avoid specifying an objective function is to use the preference-based policy learning (PPL) approach [3]. PPL consists of a three phases process:

1. The demonstration phase where the agent demonstrates to the expert a candidate policy.

2. The ranking phase where the expert will rank this policy in regards to previously demonstrated policy.

3. The updating phase where the model of the expert's preference is updated. Also, the learning agent learns a new candidate policy based on the updated expert's preference model.

The process is repeated until a policy close to the expert's expectation is produced.

PPL defines *sensori-motor states* (SMS) as combinations of the robot sensors' values with its motors values. The SMS describes the behaviour of a learning agent. Demonstrations of the policy in PPL are called *policy trajectory* and are defined as the unit vector over all SMS. The algorithm stores the ranking of the trajectories by the expert in an archive $U_t = \{u_0, ...u_t; (u_{i_1} \prec u_{i_2}), i = 1...t\}$, where u_i are the vectors representing the policy trajectories. In the first part of the archive, each entry is the expert's preferred trajectory at iteration i. In the second part, each entry shows that trajectory u_{i_2} is preferred to trajectory u_{i_i}, allowing us to rank every trajectory in the archive. A utility function for a trajectory is defined as $J_t = w_t.u$, where w_t is obtained by solving the quadratic constrained optimization problem:

$$\begin{aligned} \min_{w,\xi} \quad & \frac{1}{2}||w||_2^2 + C \sum_{1 \leq i \leq t} \xi_{i_1 i_2} \\ \text{s.t.} \quad & w^T u_{i_2} - w^T u_{i_1} \geq \xi_{i_1 i_2} \qquad \forall 1 \leq i \leq t \\ & \xi_{i_1 i_2} \geq 0 \end{aligned} \tag{2.5}$$

This problem is also solvable by a support vector machine (see Section 3.5).

To select which policy to demonstrate next, PPL computes a criterion called the Expected Utility of Selection (EUS):

$$EUS(x) = E_{\theta, x > x^*}[w^T x_i] + E_{\theta, x^* > x}[w^T x_i^*] \tag{2.6}$$

where x is the selected policy, x^* is the current best-selected policy, and w is computed by solving the optimization problem 2.5. $E_\theta[w^T x]$ is defined as the expected utility. The subscript θ, called the belief, comes from the Bayesian setting of the algorithm. Indeed, the belief θ represents the uncertainty over the utility function defined as a distribution over the space of utility functions. Hence, the left-hand side of Eq.2.6 measures the expected utility of selecting policy x when x is better than x^*. The right-hand side of Eq.2.6 is the complementary case, measuring the expected utility of selecting x when x^* remains the preferred policy.

A variant PPL algorithm, called Active Preference-learning based reinforcement learning (APRIL) [2], improves the phase where the agent updates its model of the expert's preference. Instead of using the EUS, APRIL computes an Approximated Expected Utility of Selection criterion (AEUS) to select the following policy to demonstrate to the expert. The issue with EUS resides in using the same w in both terms of Eq.2.6. Indeed, the trajectory x splits the search space of the w noted W into two versions: W^+ where the expert prefers x over x^* and the complementary case W^- where x^* is the preferred one. The EUS criterion is then rewritten as follows:

$$EUS(x) = E_{x \sim \pi_x}[E_{w \ in \ W_t^+}[w^T x_i] + E_{w \ in \ W_t^-}[w^T x_i^*]] \tag{2.7}$$

Taking the expectation over all w in both W^+ and W^- is intractable as w ranges in a high continuous space. In the original paper of APRIL, Akrour *et al.* consider two approximations leading to the mathematical expression of the AEUS criterion of a policy π [2]:

$$AEUS(\pi_x) = E_{x \sim \pi_x}[\frac{1}{F(w^+)} w^{+T} x + \frac{1}{F(w^-)} w^{-T} x^*] \tag{2.8}$$

where $F(w)$ is the objective function from the optimization problem 2.5. Both w^+ and w^- are obtained by solving the optimization problem 2.5 by considering whether the expert prefers the last policy x or not rather than the last expert's preferred trajectory x^*.

Akrour *et al.* validate APRIL in comparison with the apprenticeship learning algorithm on RL benchmarks problems [2]. The benchmark is the cancer treatment problem. A stochastic transition function emulates a patient's tumour size and toxicity evolution in this problem. The transition function even implements a stochastic death mechanism. The agent's role is to learn a drug dosage selection to stabilize the tumour by avoiding killing the patient. The performance of the policy is measured by the average tumour size and toxicity level. Apprenticeship learning quickly find the optimal policy within only two iterations. APRIL took fifteen iterations to produce the optimal policy. The second RL experiment is the mountain car. In the mountain car problem, the agent's goal is to drive a car on the top of a steep mountain. The mountain's steepness prevents the

car from reaching the top by only accelerating forward. Hence, the agent must learn to drive backwards to a behind hill to build enough potential energy to reach the top of the mountain. For this experiment, once again, apprenticeship learning finds the optimal policy in fewer iterations than APRIL.

Chapter 3

Technical Assets

Automatic design for robot swarm operates in simulation to produce control software deployed on real robots. My work during this master thesis employs technical assets for working in simulation and reality.

Demonstration-`Cho` is designed to produce control software for a specific model of robot, the e-puck presented in Section 3.1. Before deploying the produced control software to the real e-puck, I worked with a robot simulator called ARGoS presented in Section 3.3.

In this chapter, I will also present Orchestra. This software aims to interface between an operator and a robot swarm. As part of my method, I implemented additional functionalities that I present in Section 3.4.

Section 3.5 of this chapter will briefly present the support vector machines (SVM). This machine learning technique is crucial for learning the implicit objective function from demonstrations in Demonstration-`Cho`, the automatic modular design I propose in this master thesis.

3.1 E-puck

The e-puck [58] is a small mobile robot developed at the EPFL in Lausanne by Michael Bonani and Francesco Mondada for educational and research purposes. The model I used during this master thesis is a modified version from the IRIDIA lab (see Figure 3.1), augmented with a range-and-bearing device, an omnidirectional camera, and an overo gumstix board [27]. Each version of AutoMoDe design control software for the modified e-puck relative to a reference model [34]. The reference model formalises what the robot can sense and do. AutoMoDe-`Chocolate` uses the reference model RM1.1 given by Table 3.1.

The entries below the $\ll$ Input $\gg$ column in Table 3.1 correspond to the sensors of the e-puck. Eight infra-red sensors placed around the e-puck body correspond to the proximity and light sensor. One sensor is placed underneath the robot and is responsible for reading the ground colour. The camera at the top of the e-puck perceives the number of neighbouring robots. A range-and-bearing device provides the attraction vector defined as follows:

$$V = \begin{cases} \sum_{m=1}^{n} (\frac{1}{1+r_m}, \angle b_m), & \text{if robots are perceived} \\ (1, \angle 0), & \text{otherwise} \end{cases} \tag{3.1}$$

where range $r_m \in [0, 0.7]m$ and the bearing $\angle b \in [0, 2\pi]rad$ are to the respectively the distance angle to the robot neighbour m.

The entry below the « Output » column in Table 3.1 corresponds to the two wheels actuators in which the control software writes the target velocity v_l and v_r for the left and right wheels, respectively.

Input	Value	Description
$prox_{i\in1,...,8}$	[0,1]	reading of proximity sensor i
$light_{i\in1,...,8}$	[0,1]	reading of light sensor i
$ground_{i\in1,...,8}$	black, gray, white	reading of ground sensor i
n	[0,20]	number of neighboring robots perceived
V	([0.5, 20], [0, 2π])	attraction vector

Output	Value	Description
$v_{l,r}$	[-0.12,0.12]ms^{-1}	target linear wheel velocity

Table 3.1 – Reference model RM1.1 for the modified version of the e-puck [34]

Figure 3.1 – Upgraded e-puck used during the experiments at IRIDIA [27]

3.2 ROS

The Robot Operating System (ROS) is an open-source program that provides a communication structure layer above the operating system of a cluster of heterogeneous robots [65]. The communication with ROS is performed by processes called « node » exchanging « messages » to each other. To send a message, a node publishes it to a certain « topic » other nodes can subscribe to in order to receive the published message. For example, suppose a process is interested in the

odometry of the robots. In that case, it can subscribe to the « odometry » topic so that it will receive every message from every node publishing to that specific topic.

A helpful tool of ROS is called rosbag, which can record transmitted messages into a .rosbag file. Rosbag can also allow replaying the messages recorded in the .rosbag file.

3.3 ARGoS

ARGoS [51] is multi-physics simulator designed for swarm robotics. ARGoS can be used with e-pucks for simulating robot swarm missions (Fig 3.2) through a dedicated plug-in [27]. The simulator has been used in many studies to evaluate the AutoMoDe [24] among other control software. Each experiment runs in ARGoS is described by an XML file (.argos), where all information about how ARGoS will run the experiments.

The .argos file, describing the robot swarm mission, always contains the following XML tags:

— The Framework tag contains the mission's length in seconds, the « ticks-per-second » which is the frequency of the update cycle of ARGoS, and the random seed of the mission.

— The Loop functions tag contains the path to the loop functions script, which is responsible for the floor colour, the placement of the robot and the computation of the objective function in ARGoS. During my master thesis, I also put in this XML tag information that can be parsed by Orchestra (see Section 3.4) like the information of the elements composing the arena.

— The Controllers tag contains the information about the control software design and the robot controller used in the experiments.

— The Arena tag contains information about the specifications of the arena, the number of walls forming the arena and their specifications, the number of eventual obstacle and their specifications, and the eventual presence of the light source. The Arena tag also contains the E-puck tag containing all relevant information for ARGoS about the robots, like how to distribute them in the arena, their quantity or which is their controller.

— The Physics engines tag only contains the information of which physic engine ARGoS is using.

— The Media tag contains information about the LEDs and the range-and-bearing of the e-puck.

— The Visualisation tag enables the visualisation of ARGoS.

3.4 Orchestra

Orchestra is a human/swarm robot interface developed at IRIDIA using the game engine Unity. The goal of Orchestra is to monitor a robot swarm during a mission. All communications between the robot swarm and Orchestra are done through ROS.

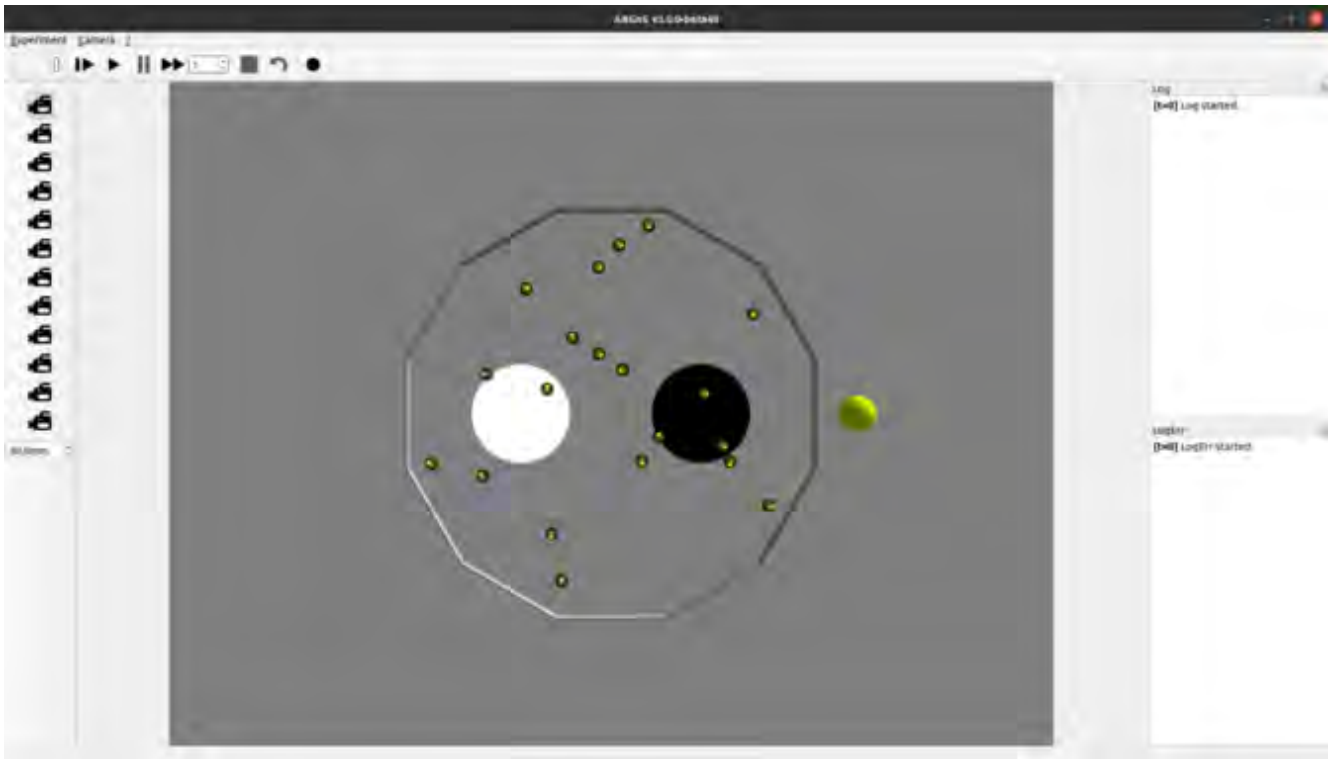


Figure 3.2 – Snapshot of ARGoS with an example of a robot swarm mission with 20 e-pucks

The Orchestra user can retrieve information from monitored robots, like their sensors' value, which is visualised in real-time in the software as they move during the swarm mission. The user can also directly command one or a group of robots by driving them or sending them behavioural commands. The ROS communication also allows multiple users to operate on the same swarm.

Figure 3.3 shows the first screen of Orchestra, where a .argos file can be entered to generate the mission setup and the IDs of the e-pucks to be monitored.

3.5 Support Vector Machine

Support vector machines (SVMs) are a family of machine learning algorithms used to solve classification and regression problems in some feature space [35] [61]. As shown in Figure 3.4, the principle of an SVM is to separate two classes of points. This separation is done with a hyperplane in the feature space such that the « margin » between the two clusters is maximal. This margin is the distance between the two boundary hyperplane of the two classes. The points on these boundary hyperplanes are called the support vectors.

Mathematically, the SVM classifies two classes of point defined in a feature space and labeled $y_i \in \{-1, 1\} \quad \forall x_i$, where $x_i \quad \forall i = 1, ..., n$ are the points in the feature space. The SVM will classify a point x as part of class 1 if $w \cdot x - b \geq 1$ or, -1 if $w \cdot x - b \leq -1$. The vector w is the orientation vector of the separating hyperplane. the support vectors are the points where $w \cdot x - b = 1$ and $w \cdot x - b = -1$. The distance between the hyperplanes intercepting the support vectors, called the margin, equals $2/||w||$. The best hyperplane to separate the two classes is such

Figure 3.3 – Principal menu of Orchestra

that the margin is maximal. Hence, the SVM solves the following optimisation problem:

$$\begin{aligned} \max_{w} \quad & \frac{2}{||w||} \\ \text{s.t.} \quad & y_i(w.x_i - b) \geq 1 \qquad \forall i \end{aligned} \tag{3.2}$$

The problem 3.2 has a similar formulation as the IRL problem 2.4. Hence an SVM can be used to solve that optimisation problem.

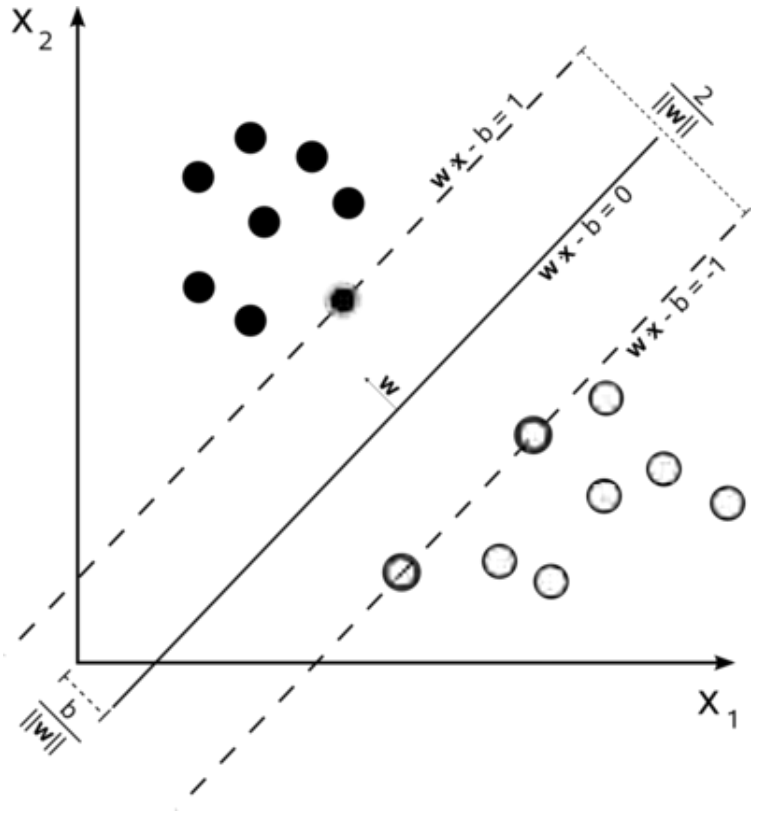


Figure 3.4 – This picture summarises the main elements of the classification problem solved by the SVM. The support vectors are the point intercepted by the frontier of the classification boundary. The goal of the SVM is to find the hyperplane with the maximal boundary distance. Credit: Wikimedia [2]

Chapter 4

Arena and Demonstration Builder

In my master thesis, I propose a complete automatic modular design process, from building the mission's arena in simulation to computing control software for robot swarms to accomplish the mission. Hence, I extend Orchestra to no longer only serve as an interface but also to allow building mission's arena and swarm behaviours.

The automatic modular design I propose in this master thesis requires the swarm mission's arena setup and some swarm behaviour demonstrations in place of an explicit objective function to produce control software. To specify the swarm mission arena, Di Ruscio *et al.* propose a domain-specific language [16]. The inconvenience with specifying missions with domain-specific language is that the user needs to know the language and its grammar. Hence to specify swarm missions, I implemented within Orchestra a graphical user interface (GUI) in a « what you see is what you get » (WYSIWYG) fashion so I can build swarm mission's arena. I detail how I typically build an arena below.

Once on Orchestra first menu (see Figure 3.3), the user can click on the « make demo » button for building the arena and later demonstrate some robot swarm behaviour. Once in the arena builder screen, several options are available to build an arena for the missions I choose in Section 6.4.

The user can change the arena shape by selecting the one they want from the top-left drop-down menu (see Figure 4.1). Then, the user can place different elements in the arena by checking one of the checkboxes in the list of elements on the left of the screen.

The « Disk » and « Quad » elements correspond to the floor patches in circular and rectangular shapes, respectively. Once placed in the arena, the patch's parameters can be changed. The patch's menu opens when the patch is clicked on. In the patch's menu, the colour of the patch, the diameter of the circular patch, or the height, width and orientation of the rectangular patch can be modified (see Figure 4.2 and Figure 4.3). The placed patch can be erased at any moment by pressing the « Delete » button from the patch's menu.

Special patches, called « SpawnDisk » and « SpawnQuad » can be used exactly like the « Disk » and « Quad » patches, except their purpose, is to define the position area of the robot swarm at the beginning of the mission. Their menu is the same as the ones of « Disk » and « Quad » except that they don't have colors (see Figure 4.4).

Similarly to the patches, obstacles can be placed in the arena and adapted. Again, when pressed on, the obstacle's menu opens up and allows to change the parameters of the obstacle like

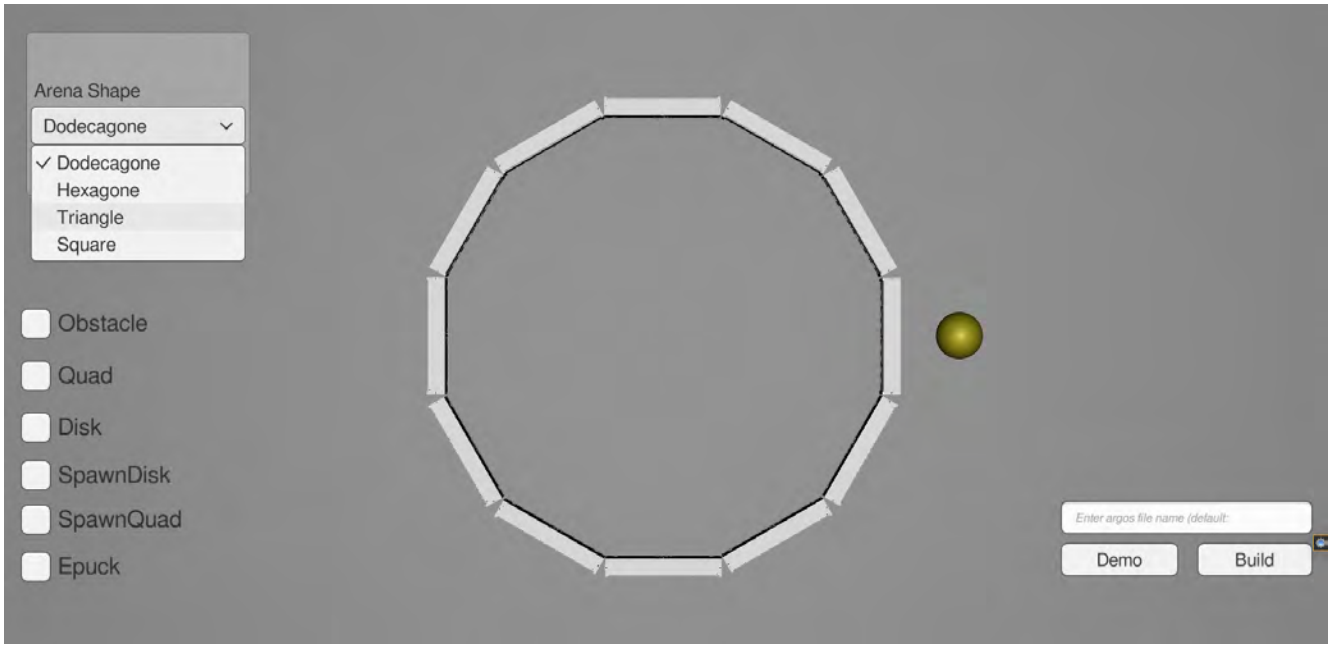


Figure 4.1 – Screenshot from the arena builder of Orchestra. When first entering the arena builder screen, the arena is empty and has the shape of a dodecagon. The user can change the arena's shape by selecting the wanted shape from the screen's drop-down menu at the top-left corner.

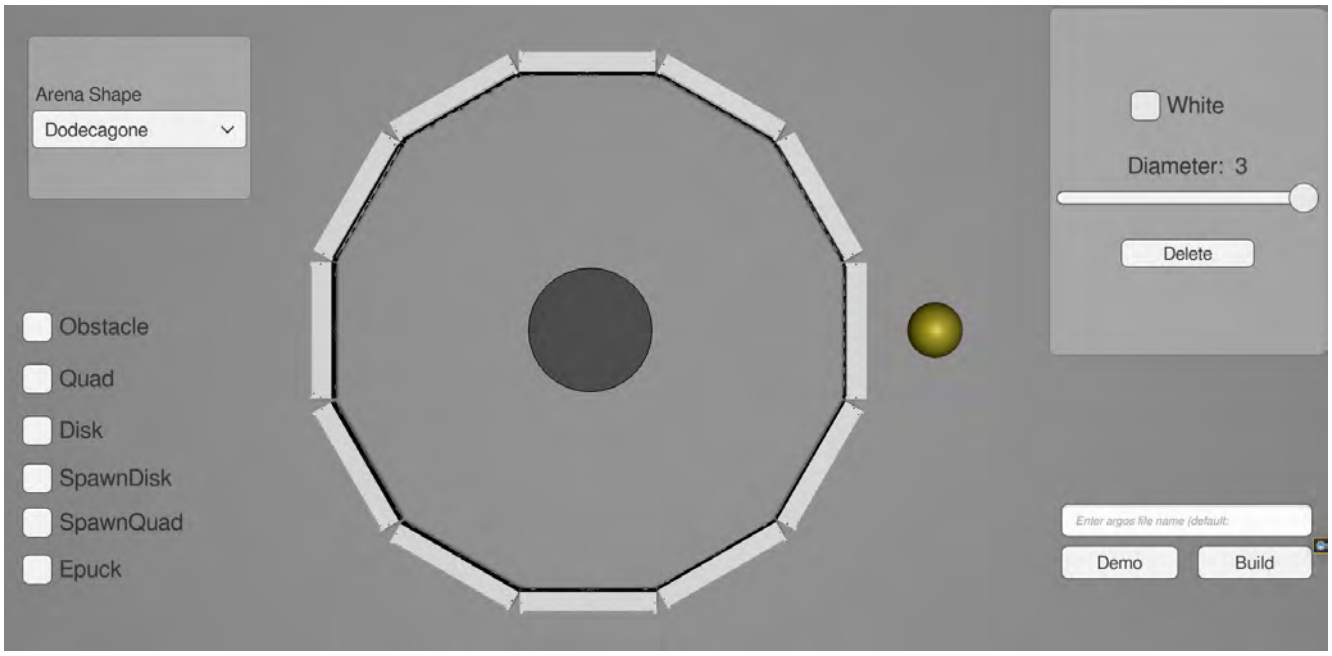


Figure 4.2 – Screenshot from the arena builder of Orchestra. The user can place a circular floor patch by checking the « Disk » box from the list of elements on the left side of the screen. Once checked, the user can point and click to place the floor patch in the arena's desired location. Clicking on a placed floor patch will open its menu where its parameters can be changed. For the circular floor patch, the user can change its diameter and colour.

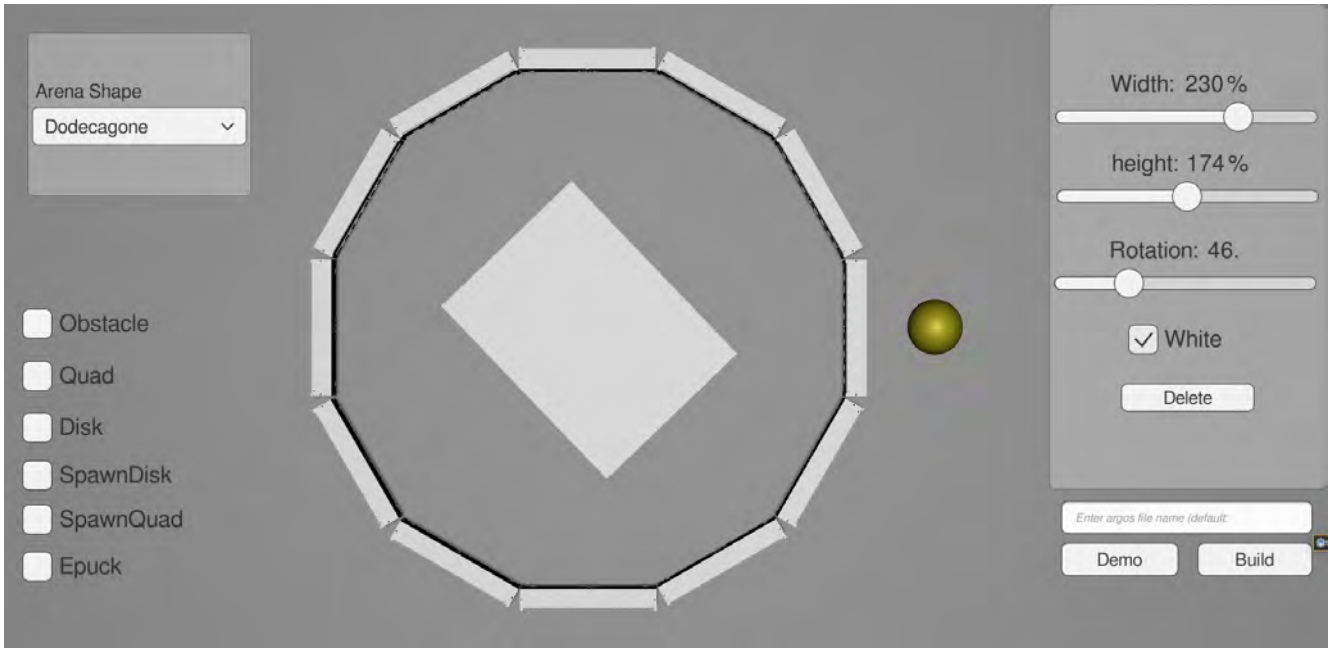


Figure 4.3 – Screenshot from the arena builder of Orchestra. The user can place a rectangular floor patch by checking the « Quad » box from the list of elements on the left side of the screen. Once checked, the user can point and click to place the floor patch in the arena's desired location. Clicking on a placed floor patch will open its menu where its parameters can be changed. For the rectangular floor patch, the user can change its width, height, orientation and colour.

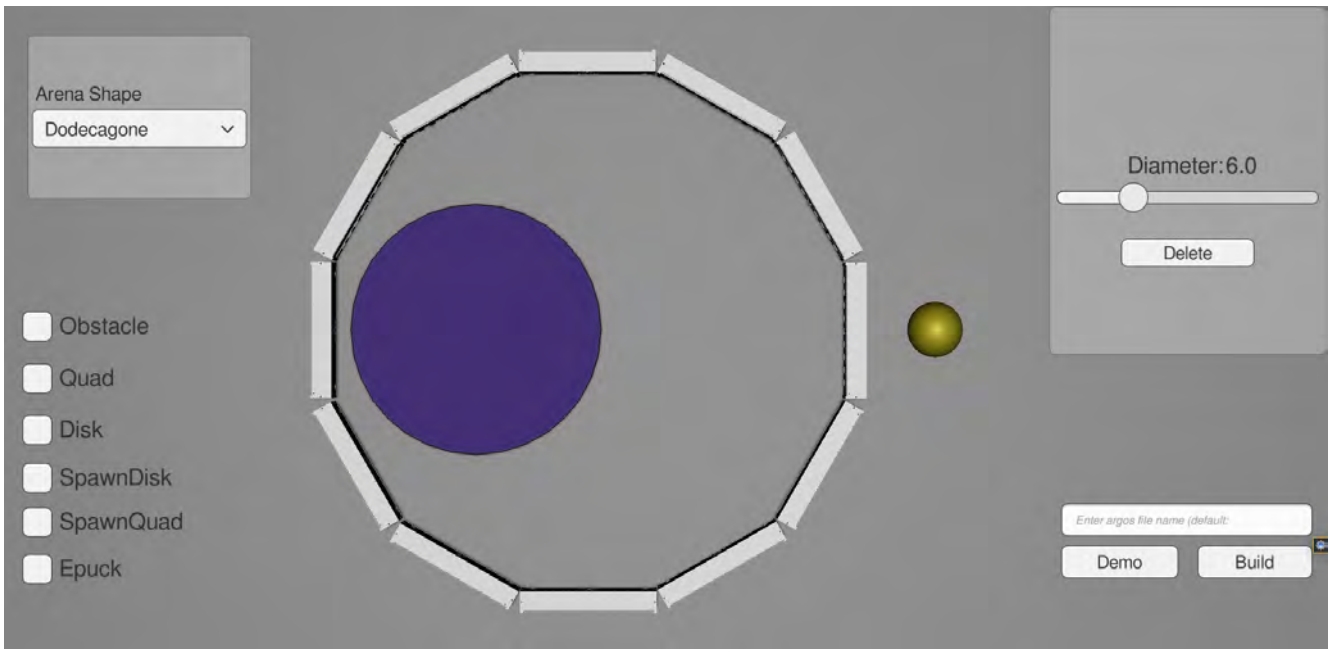


Figure 4.4 – Screenshot from the arena builder of Orchestra. The user can define the starting area for the robot swarm positions at the beginning of the swarm mission. The user can place either a circular or a rectangular starting area by checking the « SpawnDisk » or the « SpawnQuad » box from the list of elements on the left side of the screen. Once checked, the user can point and click to place the starting area in the arena's desired location. Clicking on a starting area patch will open its menu where its parameters can be changed. For the circular starting area, the user can change its diameter. For the rectangular starting area, the user can change its width, height, and orientation.

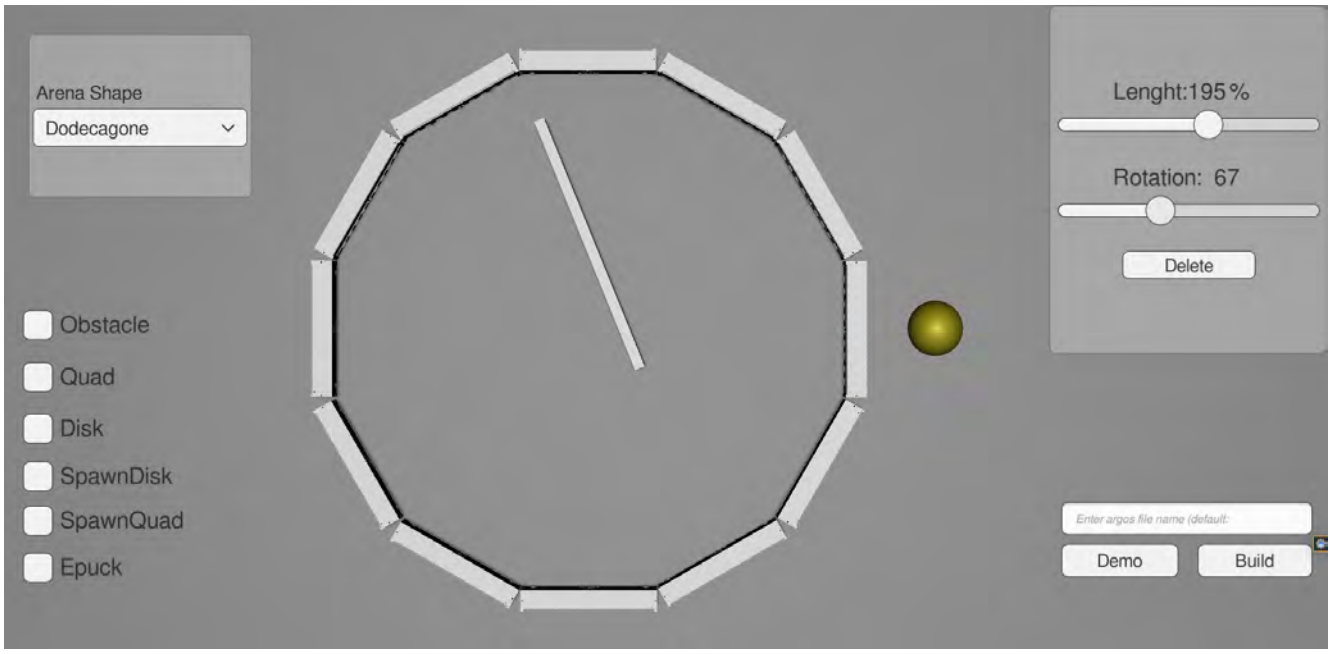


Figure 4.5 – Screenshot from the arena builder of Orchestra. The user can place an obstacle by checking the « Obstacle » box from the list of elements on the left side of the screen. Once checked, the user can point and click to place the obstacle at a desired location in the arena. Clicking on an obstacle will open its menu, where its parameters can be changed. For the obstacle, the user can change its length and orientation.

its length and orientation.

This GUI also allows demonstrating robot swarm behaviour. Once every element is placed into the mission arena, the user can build the demonstration. Hence, by checking the box of the "Epuck" element from the list of elements, the user can place as many robots as wanted by clicking on location in the arena (see Figure 4.6). To save a demonstration, the user can press the "Demo" button at the bottom right corner of the screen. The placed e-puck will disappear, and the user can make as many new demonstrations in the arena he builds as he wants.

Figure 4.7 represent a typical scene of Orchestra's mission/demonstration builder. When the user is done with building the arena and demonstrating robot swarm behaviours, he can save his work into a .argos file. To do so, the user needs to provide a file name in the text field at the bottom-right corner of the screen and then press the "Build" button. The saved .argos file will then be provided to Demonstration-`Cho` to produce control software in the next step of my method.

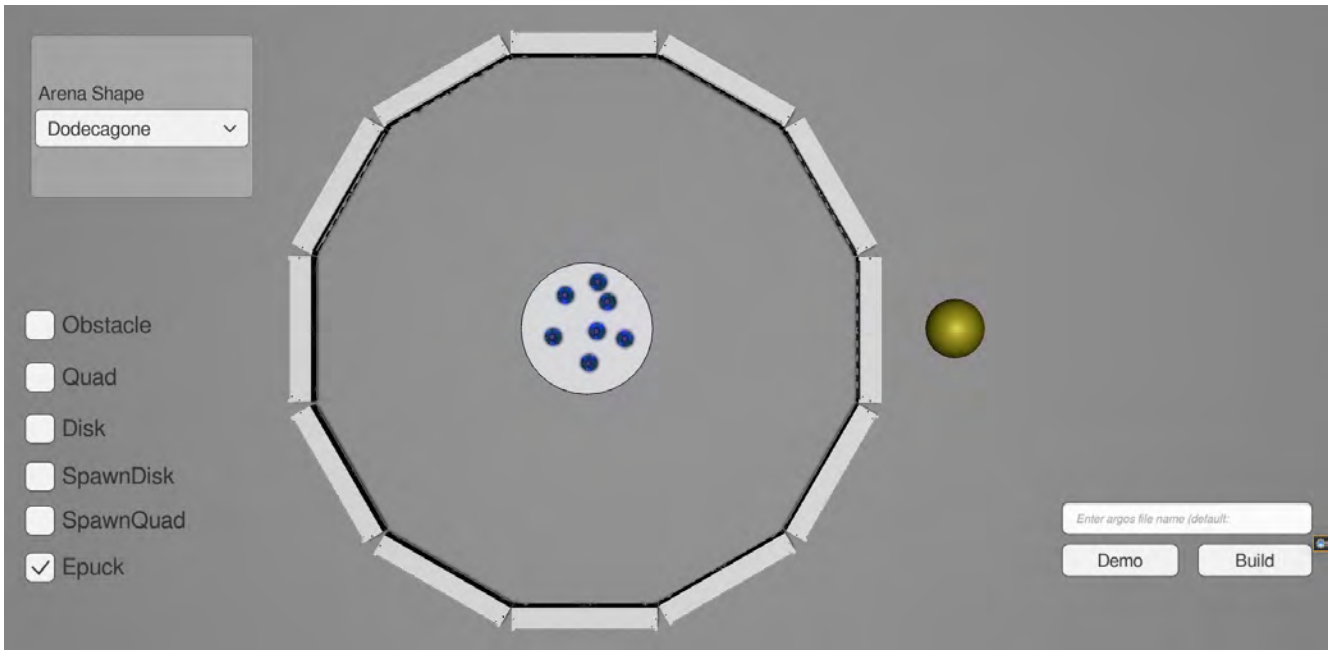


Figure 4.6 – Screenshot from the arena builder of Orchestra. The user can place the robots by checking the "Epuck" box from the list of elements on the left side of the screen. Once checked, the user can point and click to place the robots at the desired locations in the arena to demonstrate a robot swarm behaviour. Pressing the "Demo" button at the bottom-right of the screen will save the robot positions and remove them from the screen. The user can repeat the process of making and saving a demonstration as often as possible.

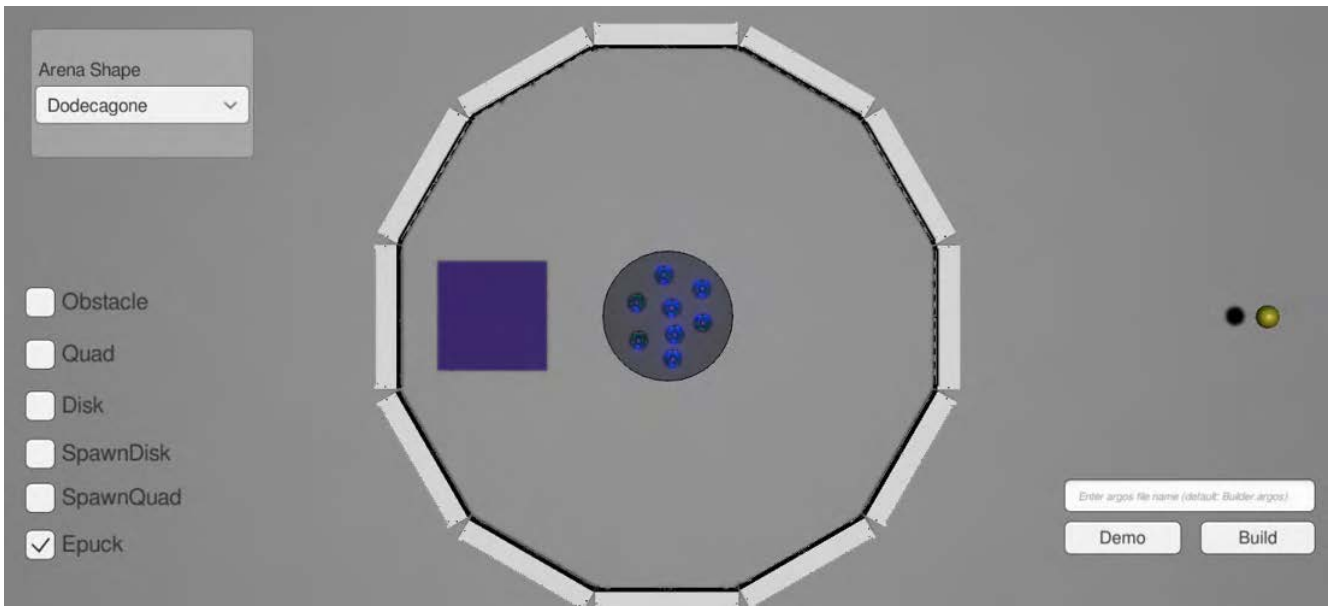


Figure 4.7 – Snapshot of the mission builder of Orchestra where the user can construct the setting of the mission and place the robots to make their demonstration of where they should be and how they should gather like

Chapter 5

APRIL

The first method I worked on during this master thesis is an adapted version of the Active Preference Learning-Based reinforcement learning algorithm (APRIL) [2]. The objective of my master thesis is to learn control software for robot swarm without specifying any explicit objective functions. One of APRIL's advantages is that it is an RL technique that does not require any explicit reward function. Instead, it learns the expert's preference thanks to the expert's ranking of proposed policies. By refining its model of the expert's preference, APRIL can propose policies closer to the expert's preference. Hence, by adapting APRIL to the context of swarm robotics, I may be able to fulfil the goal of my master thesis.

As a reminder, APRIL is composed of three phases: the demonstration phase, the ranking phase, and the update phase. I adapted each phase so APRIL could produce control software for robot swarms.

During the demonstration phase, I ran the candidate PFSM in ARGoS for the considered mission. I then look at how the robots perform in the mission.

In the ranking step, I decide if I prefer the performance of the candidate PFSM I just witnessed during the demonstration step over the performance of my prefered PFSM so far.

The third step is the updating phase, where all the learning is done. This step is divided into two sub-steps. First, APRIL updates its model of the expert's preference based on the current ranking of the policies demonstrations. This model of the expert's preference is called the Approximated Expected Utility of Selection (AEUS) criterion (see Eq.2.8). The AEUS criterion can be interpreted as the reward function the following policy needs to optimize. Hence, once the AEUS criterion is updated, APRIL uses an RL technique with the AEUS criterion as a reward function to produce a new policy. That produced policy is the one optimizing the current model of APRIL of the expert's preference. Then the newly produced policy is demonstrated to the expert, looping back to the first step of APRIL. Hence, APRIL is repeated until it produces a policy that satisfies the expert, for example, by performing well in regards to a validation metric.

Although in the original paper [2], APRIL is meant for single agent training. Hence to apply APRIL in the context of swarm robotics, I needed to do some modifications to the original APRIL algorithm.

The first modification I made is to the definition of a *policy trajectory* (see Section 2.6). Indeed, in the original paper [2], the *policy trajectory* is the unit vector over all *sensori-motor states* (SMS). For single robot learning, SMS is easily defined as the combination of all its sensors' and

actuators' values. For swarm robotics, defining SMS is more challenging. Indeed, the robot's behaviour in the swarm is represented by PFSM, where the robots can have the same combination of sensor value/actuator value for different states. For example, knowing the value of the robot's wheel speed and sensors' readings at a given moment does not allow to differentiate if the robot is in the « Exploration » or « Anti-phototaxis » state described in Table 2.1. Hence, I decided to describe an SMS of the robots as the number of times a triggered termination of a behaviour in the PFSM due to a transition condition happens. For example, the number of termination of the « Phototaxis » behaviour due to the transition condition « Black-floor » corresponds to one SMS. For example, cases where robots never leave a behaviour until the end of the mission are also considered SMS. Furthermore, I define the SMS of the robot swarm as the average over the SMS of each robot. Hence, I define the *policy trajectory* as the unit vector over all SMS of the robot swarm.

I made the second modification to the original APRIL algorithm during the learning phase. As I have presented above, once the AEUS criterion has been updated, APRIL uses an RL algorithm with the AEUS criterion as a reward function to produce the following policy. Because I consider swarm robotics with PFSM control software, I cannot use a classical RL technique. Instead, I use the automatic modular design process AutoMoDe-`Chocolate` with the AEUS criterion as an objective function. Using AEUS as the objective function will make AutoMoDe-`Chocolate` produce a control software optimizing the current model of the expert's preference.

Once I implemented the two modifications I presented above, I was ready to test my version of APRIL for swarm robotics. Although, I rapidly encounter a significant issue.

To evaluate APRIL, I have constructed a simple aggregation-like mission where the robot only has to aggregate on a black spot in the centre of the arena. Hence, I started to rank the candidate behaviour I witnessed on ARGoS regarding my appreciation when preferring a behaviour over another. The issue was that my version of APRIL only produced basic control software, with often only one or two SMS. Hence, I never succeed in producing satisfactory control software, even for this simple mission.

I believe that the problem is my definition of SMS. Indeed, in the features space, the SVM classify behaviour into two categories: the candidate behaviour the expert prefers from the current-best one and the complementary class of behaviour the expert most likely does not prefers. However, the points in that feature space constructed over my definition of SMS seem impossible to separate linearly. It follows that the AEUS criterion does not provide relevant information on the behaviour I most likely will prefer next.

To fix my version of APRIL, I see two options. The first one is to redefine the SMS to construct a feature space where the two classes of points can be linearly separable. The second option is to use another classifier than the classical SVM.

Hence, due to the complication I encountered, I, unfortunately, had to leave my work on adapting APRIL for swarm robotics for the moment.

Chapter 6

Apprenticeship Learning

The method I developed during this master thesis is an automatically modular design for robot swarms by learning from human demonstrations I call Demonstration-`Cho`. More specifically, Demonstration-`Cho` is a mix between AutoMoDe-`Chocolate` [24] and the apprenticeship learning algorithm [1]. The core idea of my method is not to provide any explicit objective function to AutoMoDe-`Chocolate`. To do so, Demonstration-`Cho` learns an implicit objective function of the swarm mission by providing demonstrations of swarm behaviour accomplishing the specified mission to the apprenticeship learning algorithm. This implicit objective function will then be used by AutoMoDe-`Chocolate` to produce control software.

The apprenticeship learning algorithm I use in this master thesis is the one described in the paper of Pieter Abbeel, and Andrew Y. Ng [1]. Although, I needed to assume some hypotheses to adapt the apprenticeship learning algorithm to the scope of swarm robotics. I will explain my working hypothesis in Section 6.1.

During the development of Demonstration-`Cho`, I encountered the problem of how to represent a robot swarm demonstration in the feature space. In section 6.2, I will detail my final version of the features representation of a robot swarm demonstration and how I end up with this features representation.

In Section 6.3, I detailed the algorithm behind Demonstration-Cho.

At IRIDIA, a lot of robot swarm mission has been proposed. To test my method, I needed to select some missions Demonstration-`Cho`, AutoMoDe-`Chocolate`, and `Evo` could address. Section 6.4 presents an extensive list of the swarm mission developed at IRIDIA. I will also present the four missions I use for my experiments and why I choose them.

6.1 Hypothesis

I had to make two hypotheses to adapt the Apprenticeship Learning algorithm for single agent learning to swarm robotics.

Firstly, in my method, a swarm behaviour demonstration is nothing more than the positions of the robots at the end of the robot swarm mission. In the paper of Abbeel and Ng [1], the value of a policy is the expected discounted reward collected by the policy over time for all initial states, implying that the demonstration is defined as a sequence of action for all time. For simplicity, I only consider the reward collected at the last step of the mission. Hence I only consider the final

position of the robots as robot swarm behaviours.

Secondly, I assume that the objective function specifies that the robot swarm mission is a linear combination of the features vector of the robot swarm behaviour demonstration.

6.2 Features Representation

Abbeel and Ng define the feature vector $\phi(x)$ representing the agent behaviour as the result of the function $\phi : S \to [0,1]^k$, where S is the set of states and k is the number of features, to the state x [1]. For example, the feature vector could be the vector of desirable features of a self-driving car, such as the number of collisions or the lane the self-driving car rides on.

One fundamental difference is that Abbeel and Ng consider a single agent training, and I use the algorithm for a swarm of agents, but I consider a swarm of robots. Hence, the features vector no longer only represents the features of a single agent but rather the features of a swarm of agents. The question is how to construct that feature vector for a robot swarm.

In the beginning, I needed to decide between two ways of representing the features vector. One possibility was to think about swarm-level features. This way, we might consider the robot swarm as a whole as a single agent. The other possibility is to use the features of the individual robots, and I would concatenate all the features together in the features vector. Ultimately, I have chosen the second option because defining a swarm-level feature is not easy compared to defining the individual robot features.

Once I decided on how to represent the features vector, I had now to define what the features were. As I presented in Section 6.1, I consider the robot swarm behaviour as the position of the robots at the end of the mission. The features must then be related to the position of the robots.

From here, I will designate the features vector by μ and not ϕ by abuse of language. Indeed, Abbeel and Ng define μ as the expected features vector of the robot swarm behaviour. To compute it, I repeat the robot swarm mission multiple times and retrieve as many feature vectors ϕ. Then, I take the average over all ϕ to compute the expected features vector μ. Because μ is the average over multiple ϕ, replacing ϕ by μ will not impact the formula I will present below.

The first feature I defined is a function of the position of the robots in regards to the robot's position from the expert demonstration. More specifically, I defined the distance from the expert demonstration features vector as follows:

$$\mu = (\mu_{Hungarian_1}, ..., \mu_{Hungarian_n}) \quad s.t. \quad \mu_{Hungarian_i} = e^{-\frac{2.\ln 10}{d^2}.d.x_{Hungarian_i}} \quad \forall i = 1, ..., n \tag{6.1}$$

where n is the number of robots, and d is the diameter of the inner circle of the arena. The variable $x_{Hungarian_i}$ represents the *ith* smallest distance from the assignment between the positions of the robots resulting from the automatic design and the positions of the robots from the expert's demonstration. This assignment results from the Hungarian algorithm minimizing the sum of all robot-to-robot distances between the two groups. The reason I use an assignment algorithm is to break the ordering dependency the features vector has if I consider the robots in the same order every time I need to compute the distance between the robots and the expert's demonstration.

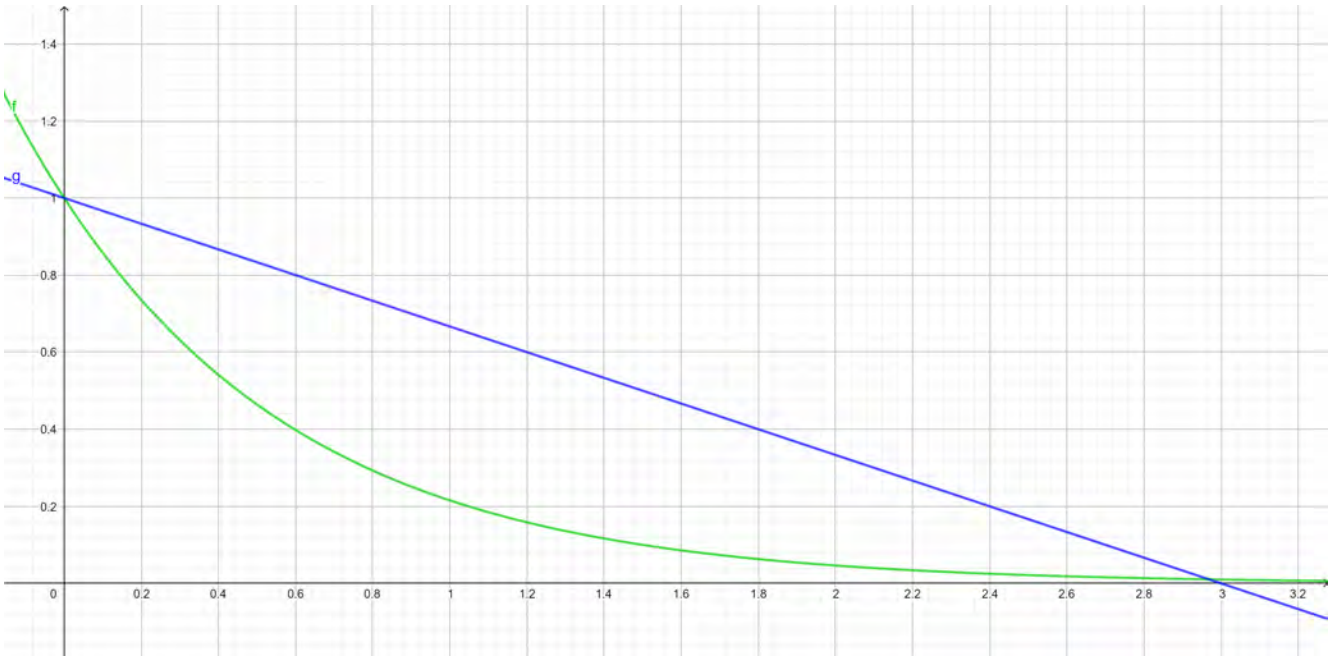


Figure 6.1 – The x-axis is the distance between one robot position resulting from the automatic design process and one robot from the expert's demonstration. The y-axis is the value of μ in the function of the distance between the positions of the robot. The green curve is the exponential function described in Eq.6.1, and its decreasing speed is such that when the distance is more significant than half of the arena's inner radius (equal to 3 meters in this graph), the value of μ is already below 0.1. The penalization of the exponential function when the robot from the automatic design is far from the robot from demonstration is more severe than if the linear function represented by the blue curve was used. This penalization matters because in the robot swarm experiments, we are only interested when the robots end up close to the robots of the demonstration.

The exponential function in Eq.6.1 has two purposes. First, it keeps the value of the $\mu_{Hungarian}$ in the range of $[0, 1]$ with the value 1 reached when a robot is precisely at the position of a robot from the expert demonstration. The second purpose of the exponential decrease is to penalize the robots the farthest they are from the demonstration. Indeed, in the context of the robot swarm missions, I have chosen (see Section 6.4), a robot's behaviour is considered poorly performing regardless of the distance from the demonstration from the point it is more significant than half the size of the arena. Figure 6.1 shows the difference between decreasing speed of the exponential function (in green) and the linear function (in blue). The exponential function's factors from Eq.6.1 are such that $\mu_{Hungarian} = 0.1$ when the distance between the robot and the demonstration is equal to the radius of the inner circle of the arena.

The problem with defining the feature vector is only based on the distance from the demonstration is that the resulting implicit objective function may lack specific information about the mission. Indeed, I noticed when experimenting with that features vector representation that sometimes the behaviour produced by my method ignored some element of the arena, resulting in non-optimal behaviour. For example, in the SAC mission (see Section 6.4), the access to shelter is constrained by walls. Hence, by only considering the distance from the arena, aggregating close to the shelter's wall out of the arena is considered a « good » behaviour by my automatic design process. However, it should not be the case. If the features could integrate the obstacle or floor patches, more information about the mission would result in better control software.

Hence, I drop the idea of defining the features vector in the distance function to the expert demonstration. Instead, I defined multiple features per robot, all in function of the distance to the « landmarks » in the arena. Indeed, in all missions listed in Section 6.4, the same elements appeared regularly, which I defined as landmarks: the circular and the rectangular floor patches, in black and white. The features vector at this point is defined as follows:

$$\mu = (\mu_{patch_{11}}, ..., \mu_{patch_{1n}}, ..., \mu_{patch_{kn}}) \tag{6.2}$$

$$s.t. \quad \mu_{patch_{ij}} = \begin{cases} e^{-\frac{2.\ln 10}{d^2}.d.x_{patch_{ij}}} & \forall i = 1, ..., n \quad , \quad \text{if the robot is outside patch j, } \forall j = 1, ..., k; \\ 1, & \text{otherwise} \end{cases} \tag{6.3}$$

where n is the number of robots in the swarm, k is the number of floor patches in the arena and d is the diameter of the inner circle of the arena. The variable $x_{patch_{ij}}$ represents the *ith* smallest distance from a robot of the swarm to the *jth* floor patch. The computation of the μ_{patch} is the same as in Eq.6.1 except that this time $\mu_{patch} = 1$ as soon as the robot is on the floor patch.

Now, the features vector considers the element specific to the robot swarm mission. Although, more information from the mission can be considered besides the floor patches. Indeed, this new representation has the same issue with the shelter's walls of the SAC mission I have mentioned above. For other missions like CFA (see Section 6.4), a feature about the swarm density could also be helpful. Hence, for my final version of the features vector, I have made one modification and one addition to Eq.6.4. To consider the wall, I modify the μ_{patch} such that $\mu_{patch} = 0$ if there is an obstacle between the robot's final position and the obstacle. This modification is meant to penalize a robot outside the shelter when the demonstration robot is in the shelter. Finally, I added the feature relative to the distance to the closest robot neighbour to provide information about the robot swarm density. Hence the final version of my features vector is defined as follows:

$$\mu = (\mu_{patch_{11}}, ..., \mu_{patch_{1n}}, ..., \mu_{patch_{kn}}, \mu_{neigh_1}, ..., \mu_{neigh_n}) \tag{6.4}$$

$$s.t. \quad \mu_{patch_{ij}} = \begin{cases} 1, & \text{if robot i is inside patch j;} \\ 0, & \text{if an obstacle is between robot i and patch j;} \\ e^{-\frac{2.\ln 10}{d^2}.d.x_{patch_{ij}}} & , \text{ otherwise; } \forall i = 1, ..., n \text{ and } \forall j = 1, ..., k; \end{cases} \tag{6.5}$$

$$\mu_{neigh_i} = e^{-\frac{2.\ln 10}{d^2}.d.x_{neigh_i}} \quad \forall i = 1, ..., n \tag{6.6}$$

where x_{neigh_i} is the *ith* smallest distance between one robot and its closest neighbor in the swarm.

6.3 Demonstration-`Cho`

My method for robot swarm mission specification and production of control software for that mission is composed of two steps. The first step is to build the arena and make some swarm robot demonstration with Orchestra as presented in the previous Section.

The second step is to adapt the Apprenticeship Learning algorithm [1] to produce control software for robot swarms. I did three modifications to the original Apprenticeship Learning algorithm. The first one is to adapt the features vector to describe the robot swarm. I provide the

details about my definition of features vector for robot swarm in the context of my master thesis in Section 6.2.

The second modification concerns the stopping criterion of the Apprenticeship Learning algorithm. As a reminder, the goal of the apprenticeship learning is to produce a policy performing as good as the expert policy on an implicit reward function $R = w.\mu$, where μ is the features vector and w a weight vector. This implicit reward function is learned iteratively by solving Prob 2.4 with an SVM. The Figure 6.2 schematically summarizes the apprenticeship learning. The expert features vector (in green) is the only point in the class « +1 » and all feature vectors produced so far by the apprenticeship algorithm belong to the class « -1 » (in red). The SVM is fitted to separate the two classes by maximizing the margin t. The vector w is then used to learn the implicit reward function $R = w.\mu$. In the original Apprenticeship Learning algorithm, [1], the stopping criterion is when the value of the max-margin t is below an arbitrary threshold. If the max-margin t is small, it means that one policy has a features vector close to the features vector of the expert policy in the feature space. In my experiments, there are as many feature spaces as there are missions, so the value of the max-margin is not necessarily on the same scale. Hence in Demonstration-`Cho`, I specify a fixed number of iterations for the algorithm to terminate.

The last modification I made to the algorithm of Abbeel and Ng [1] is in the policy learning step. Indeed, the apprenticeship learning algorithm is used in the context of single agent learning. Hence, once the implicit reward function is learned, the original paper uses an RL algorithm to learn a policy that would maximize the implicit reward function. In my master thesis, I work in the context of swarm robotics, so I cannot use a conventional RL technique to learn a swarm policy. Instead of the RL algorithm, I use AutoMoDe-`Chocolate` with the learn implicit reward function to compute the control software in a probabilistic finite state machine (PFSM) that would optimize the implicit objective function.

The following pseudo-code describes how Demonstration-`Cho` computes a control software for a robot swarm mission by providing an expert robot swarm demonstration:

1. Provide the arena and the expert demonstration from Orchestra, and the number of Demonstration-`Cho` iterations N.

2. Compute the features vector of the expert's demonstration and label it as +1 in the PFSM history.

3. Set i = 0 and generate a random initial $PFSM_i$ and compute the features vector of $PFSM_i$ and label it as -1 in the PFSM history.

4. Compute t_i and w_i by fitting the SVM to the PFSM history.

5. If $t_i < t_{i-1}$ for i = 1,...,N, store $PFSM_i$ as the best PFSM and if i = N, return the stored best PFSM and terminate.

6. Run AutoMoDe-`Chocolate` with $R = w_i.\mu$ in ARGoS with the arena from Orchestra, increment I and compute new $PFSM_i$.

7. Compute the features vectors of the new $PFSM_i$ and label it as -1 for the PFSM history.

8. Go back to step 3.

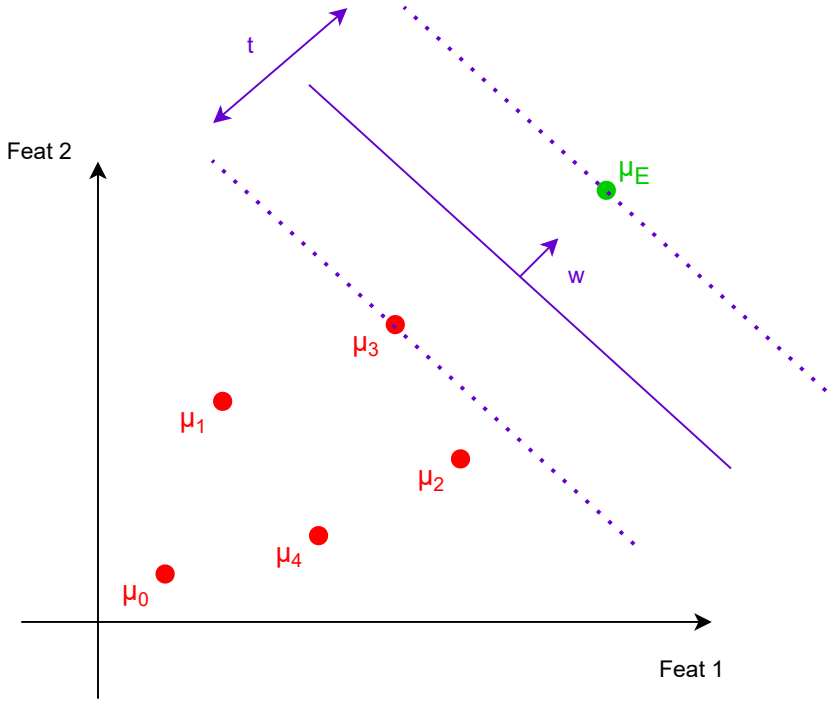


Figure 6.2 – The graph represents the fourth iteration of the apprenticeship learning in a feature space of two dimensions. The SVM is fitted to separate the features vector of the already four produced policies (in red) from the features vector of the expert policy (in green). The apprenticeship learning algorithm terminates upon the value of the max-margin t goes below a threshold. This means that a features vector from the policy produced by the algorithm is close to the features vector of the expert policy in the feature space.

6.4 Selection of missions

Hereafter is an extensive list of the mission in IRIDA's literature. In the first part of the list, for each class of missions, I provide a short explanation of their objectives. Each class of missions have some variant missions that I will mention. From this first part of the list, I will select candidate missions that can be addressed by Demonstration-`Cho`. Finally, I will present my selection of the missions I will use to compare the performance of Demonstration-`Cho` with Objective-`Cho` and Objective-`Evo`.

From the literature of the IRIDIA lab, I found the following class of missions:

— `Stop` : The robots must stop as soon as a signal is emitted. It can be that one robot steps on floor patches or the walls emit a stop signal. [32] [28] [31] [40] [41]

— `Foraging`: The robots need to forage resources. Some variants of this mission exist, like choosing the best source of objects to gather. [46] [47] [66] [48] [45] [68] [50] [67] [52]

— `Aggregation`: The swarm of robots need to aggregate at some specific location [46] [32] [47] [48] [68] [28] [31]. Many variants of this class of mission exist:

  - XOR: The swarm must aggregate on one floor patch or the other but not both at the same time.[50] [33] [52]
  - AAC (with ambient cues): The swarm have access to a light source to guide them to gather on the floor patch. [43] [45] [24]
  - End-Time: The robot must aggregate on a floor patch by the end of the time of the mission. The performance is measured only at the end of the mission. [66]
  - Marker: The robot must aggregate within a dotted area which is not visible to the robot. A floor patch is in the aggregation area to be used as a marker by the swarm. [40] [41]

- `Shelter`: The swarm must find and aggregate inside of a shelter [33]. This class of mission has some variants:
  - SCA (with constrained access): The shelter is surrounded by walls and is only accessible by one side. A light source is present outside the arena on the side of the arena's opening to guide the swarm. [43]
  - SAC (with ambient cues): Half of the arena is covered by a black floor patch. The arena is the same as SCA (paper yet to be published).
  - guided: The swarm have two ambient cues to guide them to the shelter: a light source and a conic region providing a path to the shelter [43].

- `Directional-Gate`: The robots need to pass through a ≪ gate ≫ represented by a floor patch. The robots must pass through the gate in a correct sense. [33]

- `Homing`: The robots start at one side of the arena and must aggregate on a floor patch on the other side. There is no light source to guide the swarm. [33] [52]

- `Coverage`: The robot swarm has to spread as much as possible in the arena. Some variants of the class of mission exist:
  - Phormica: The arena is divided by cell units, and the robots must at least visit every cell once. The robots are capable of leaving pheromones to communicate through stigmergy [67].
  - CFA (with forbidden areas): Three circular floor patches are placed in the arena. The robot swarm must cover the arena but needs to avoid the floor patches [24].

  - LCN (largest covering network): While trying to cover the arena, the robots have to maintain connected to each other [23].

  - SPC (surface and perimeter coverage): The goal of the mission is to cover the area of a white sqaure floor patch and the perimeter of a black circle floor [23].
- `Tasking`: Gray floor patches and LEDs represent
textquoteworkstations where the robots need to perform the task. A task considered as performed is the robot spin in the workstation. The robots must visit a workstation only once. The Phormica variant of this mission involves pheromones to mark the workstation already visited [67].

- `Decision`: A circular floor patch is placed in the centre of the arena and can be white or black. A light source is present outside of the arena. Initially, the robot is placed randomly across the arena. The task requires that the robots relocate to one-half of the arena if the floor patch is black and to the other half part of the arena if the floor patch is white [32] [49].

— `Anytime-Selection`: Two circular floor patches are placed in the arena. The robots must aggregate in one of the floor patches. The performance is measured over time. So the robots must aggregate in one of the patches and stay on it until the end of the mission [66].

The candidate missions should be feasible for AutoMoDe-`Chocolate` as Demonstration-`Cho` uses this automatic modular design. The mission requiring seeing LED colours or leaving pheromones to set stigmergy communication has to be discarded as no modules in AutoMoDe-`Chocolate` allow any of that. The mission that involves only the spatial positioning of the robots also needs to be discarded. Indeed, I defined my features vector relative to the position only, so they cannot represent any time-based mission like Foraging. Hence, I broke down the above list into the following list of candidate missions:

— `Aggregation`:
  — XOR
  — End-Time
  — AAC
  — Marker

— `Shelter`:
  — SCA
  — SAC

— `Homing`

— `Coverage`:
  — CFA

I selected one variant mission from the four remaining feasible classes of missions. Hence, the missions I will use for my experiments are AAC, SAC, Homing and CFA.

Chapter 7

Experimental Protocol

This chapter presents the experimental protocol for the experiments conducted in simulations and with real robots.

In the Experiments, I will compare Demonstration-`Cho` with Objective-`Cho` and Objective-`Evo`. Demonstration-`Cho` is the design method described in Chapter 6.3, Objective-`Cho` is AutoMoDe-`Chocolate` and Objective-`Evo` is EvoStick.

The experimental protocol for all missions is as follows:

1. With the mission builder of Orchestra, I build the arena of the chosen mission.

2. When the arena is constructed, I demonstrate five demonstrations of the robot swarm's final positions for the chosen mission in the Demonstration builder of Orchestra.

3. I compute ten instances of control software of Demonstration-`Cho` with the arena specification and the demonstrations from Orchestra. For each fifty iteration of each instance of Demonstration-`Cho`, I use a design budget of 10,000 executions of ARGoS.

4. I compute ten instances of control software of Objective-`Evo` with the arena specification from Orchestra and the original objective function, as defined in the paper proposing the mission. For each instance of Objective-`Evo`, I use a design budget of 10,000 executions of ARGoS.

5. I compute ten instances of control software of Objective-`Cho` with the arena specification from Orchestra and the original objective function, as defined in the paper proposing the mission. For each instance of Objective-`Cho`, I use a design budget of 10,000 executions of ARGoS.

6. For each generated instance of control software, I evaluate the performance once in simulation with ARGoS.

7. For each generated instance of control software, I evaluate the performance once in reality with a real arena at IRIDIA lab.

8. I report the results in the form of boxplots.

The reason for Demonstration-`Cho` running multiple iterations in opposition to one for Objective-`Evo` and Objective-`Cho` is because it is learning the implicit objective function specifying the mission along each iteration. On the contrary, Objective-`Evo` and Objective-`Cho` only run once because they have access to the explicit objective function.

I describe each mission's objective, arena specification, and explicit objective function in the following sections. I also show the demonstrations I made with Orchestra for each mission. I also show the explicit arena built at IRIDIA for each mission.

7.1 Aggregation with Ambient Cues

For this mission, the arena is composed of twelves walls of length equal to 66 cm forming a regular dodecagon. Within the arena are one white and one black circle floor patches of radius equal to 30 cm. The black circle is placed to the side of the arena, the closest to the light source placed outside of the arena. The white circle is placed on the opposite side of the arena, next to the side the farthest to the light source.

In the mission Aggregation with Ambient Cues mission (AAC), the robots swarm gather on the black spot as quickly as possible. The robots have access to the light source outside of the arena to guide them. At the beginning of the mission, the robots are randomly distributed in the arena.

The fives pictures from Figure 7.1 show the arena configuration for AAC as I constructed with Orchestra, with a different example of a demonstration of the final positions of the robot swarm I made. Figure 7.2 shows the actual arena built at IRIDIA with the robots placed randomly distributed in the whole area of the arena as starting position for the mission.

The objective function is defined as follows [24]:

$$F_{AAC} = \sum_{t=1}^{T} N(t)$$

where $N(t)$ is the number of robots on the black spot at time t and $T = 180$ seconds.

7.2 Homing

For this mission, the arena is composed of twelves walls of length equal to 66 cm forming a regular dodecagon. Within the arena is one black circle floor patch of radius equal to 30 cm. The black circle is placed next to one side of the arena.

In the Homing mission, the robots swarm gather on the black spot as quickly as possible by the end of the available time. No light source is available for guiding the robot. The robots must only rely on local information about the ground colour and their immediate neighbourhood density. At the beginning of the mission, the robots are placed gathered next to the opposite side of the black spot.

The fives pictures from Figure 7.3 show the arena configuration for the Homing mission as I constructed with Orchestra, with a different example of a demonstration of the final positions of the robot swarm I made. Figure 7.4 shows the actual arena built at IRIDIA with the robots

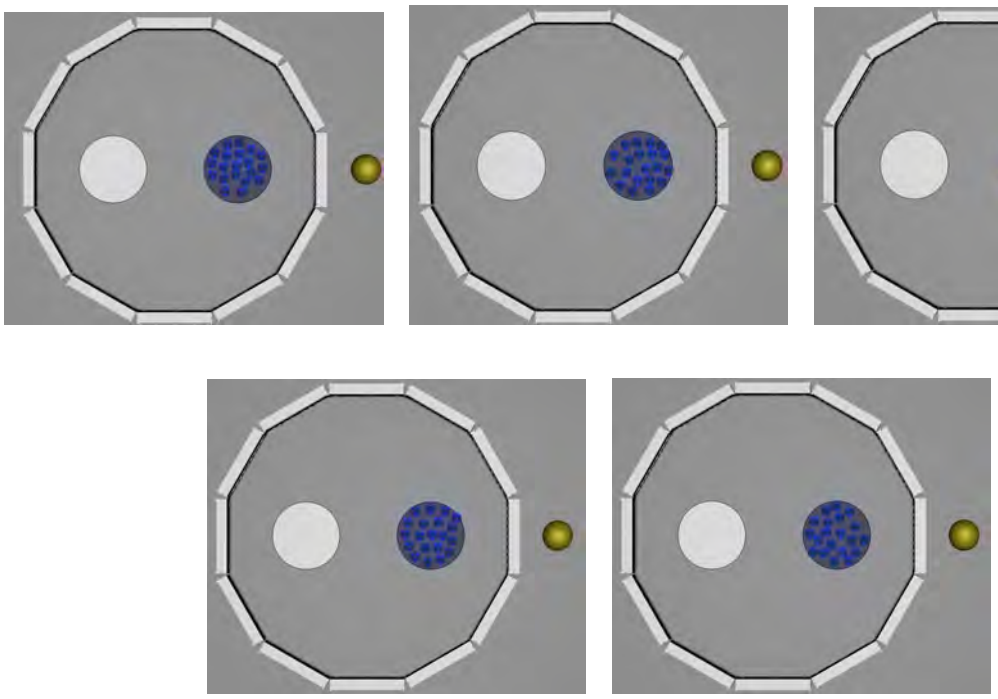


Figure 7.1 – The five pictures represent the arena for AAC constructed in Orchestra. In each of the five pictures, I make one different demonstration of the end position of the robots I want the produced control software from my method to replicate.

placed randomly distributed in the whole area of the arena as starting position for the mission.

The objective function is defined as follows [33]:

$$F_{Homing} = \sum_{i=1}^{N} I_i(T); \quad I_i(T) = \begin{cases} 1, & \text{if if robot i is in the black area at time T;} \\ 0, & \text{otherwise} \end{cases}$$

where $N = 20$ is the number of robots and $T = 120$ seconds.

7.3 Sheltering with Ambient Cues

For this mission, the arena is composed of twelves walls of length equal to 66 cm forming a regular dodecagon. A light source is placed outside of the arena. Within the arena is one white rectangle floor patch of 25 cm in width and 15 cm in height. Two walls of 35 cm in length and one of 50 in length are placed around the white floor patches, leaving an opening on the side of the white floor patch the closest to the light source. The three walls and the white floor patches compose the shelter together. The arena is covered by a black area on all its height, from its border the farthest to the light source to the border of the shelter the closest to the light source.

In the Sheltering with Ambient Cues mission (SAC), the robot swarm are tasked to gather on the white shelter as quickly as possible. The robots have access to the light source outside of the arena to guide them. At the beginning of the mission, the robots are randomly distributed in the arena.

The fives pictures from Figure 7.5 show the arena configuration for SAC as I constructed with Orchestra, with a different example of a demonstration of the final positions of the robot swarm

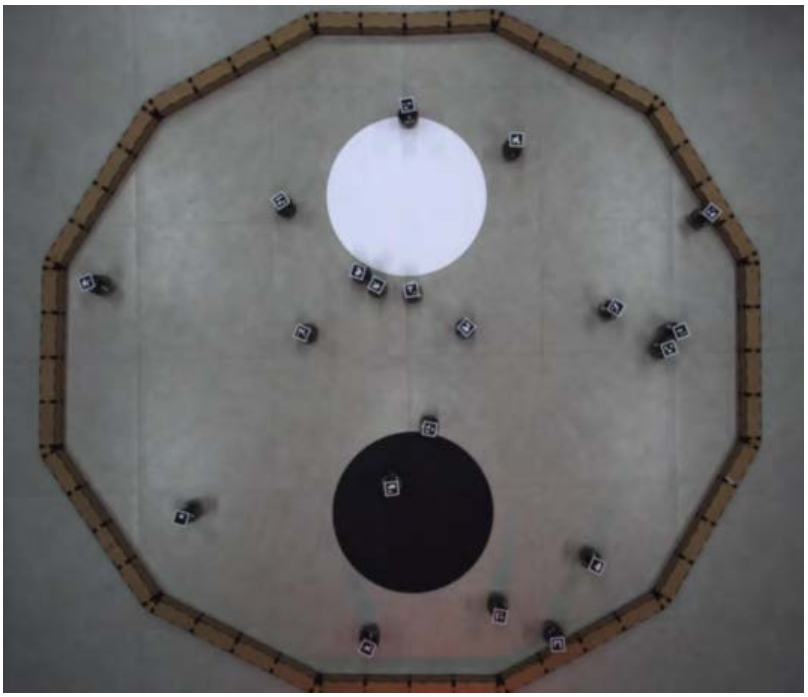


Figure 7.2 – Picture of the arena built at IRIDIA for the AAC mission. The robots are randomly placed in the arena at the start of the mission.

I made. Figure 7.6 shows the actual arena built at IRIDIA with the robots placed randomly distributed in the whole area of the arena as starting position for the mission.

The objective function is defined as follows (paper yet to be published):

$$F_{SAC} = \sum_{t=1}^{T} N(t)$$

where $N(t)$ is the number of robots in the shelter at time t and $T = 180$ seconds.

7.4 Coverage with Forbidden Areas

For this mission, the arena is composed of twelves walls of length equal to 66 cm forming a regular dodecagon. Within the arena are three black circle floor patches of a radius equal to 30 cm. The black circles are placed all placed close to a side of the arena, forming a triangular formation around the arena's centre.

In the Coverage with Forbidden Areas mission (CFA), the robots need to spread as much as possible across the arena by avoiding the black spots at the available time to perform the task. No light source is available for guiding the robot. The robots must only rely on local information about the ground colour and their immediate neighbourhood density. At the beginning of the mission, the robots are randomly distributed in the arena.

The fives pictures from Figure 7.7 show the arena configuration for the CFA mission as I constructed with Orchestra, with an example of a demonstration of the final positions of the robot swarm I made. Figure 7.8 shows the actual arena built at IRIDIA with the robots placed randomly distributed in the whole area of the arena as starting position for the mission.

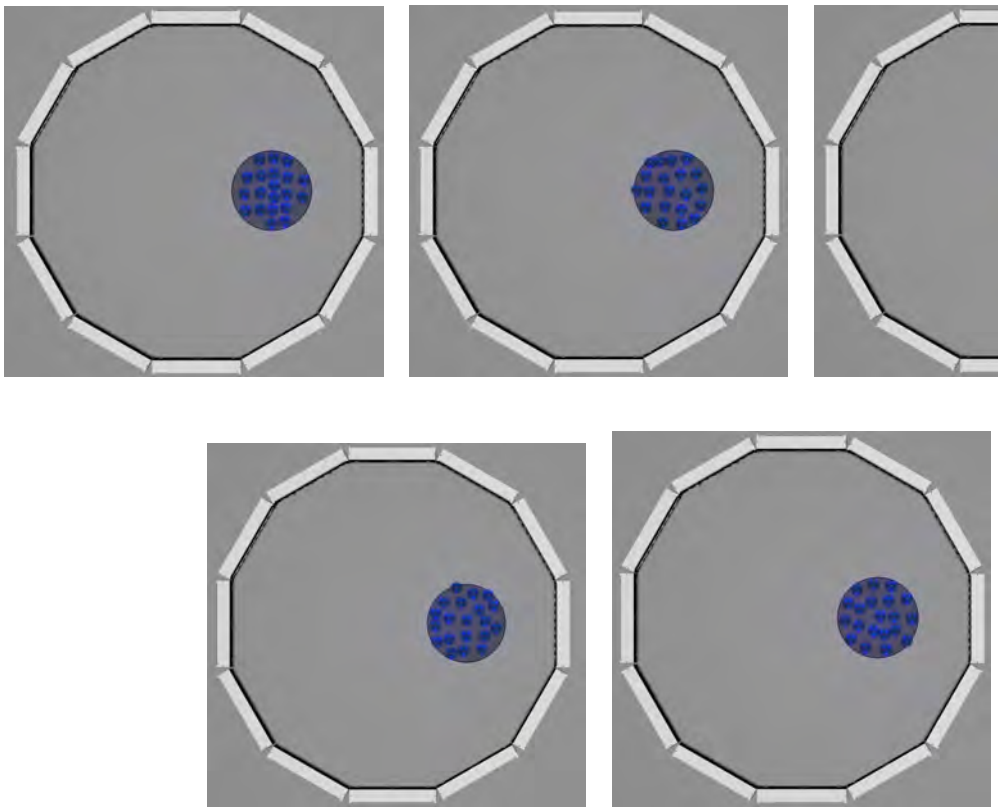


Figure 7.3 – The five pictures represent the arena for the Homing mission constructed in Orchestra. In each of the five pictures, I make one different demonstration of the end position of the robots I want the produced control software from my method to replicate.

The objective function is defined as follows [24]:

$$F_{CFA} = 25000 - E[d(T)]$$

where $E[d(T)]$ is the expected distance between a generic point in the arena and the closest robots not on a black spot, at the end of T, and $T = 180$ seconds. I changed the original definition of the objective function, so the objective function needs to be maximised like the other three missions.

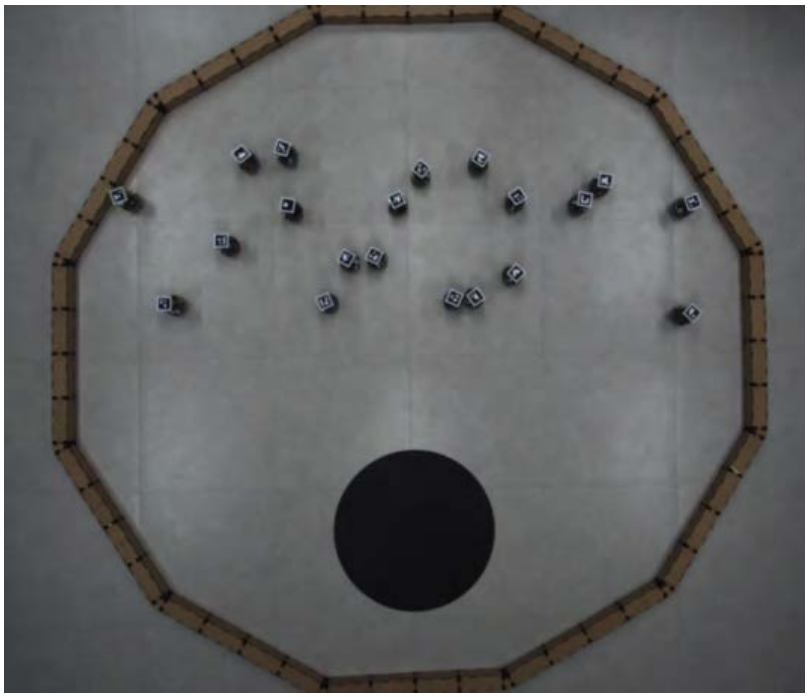


Figure 7.4 – Picture of the arena built at IRIDIA for the Homing mission. The robots are randomly placed in a rectangular area on the opposite side of the arena from the black patch at the start of the mission.

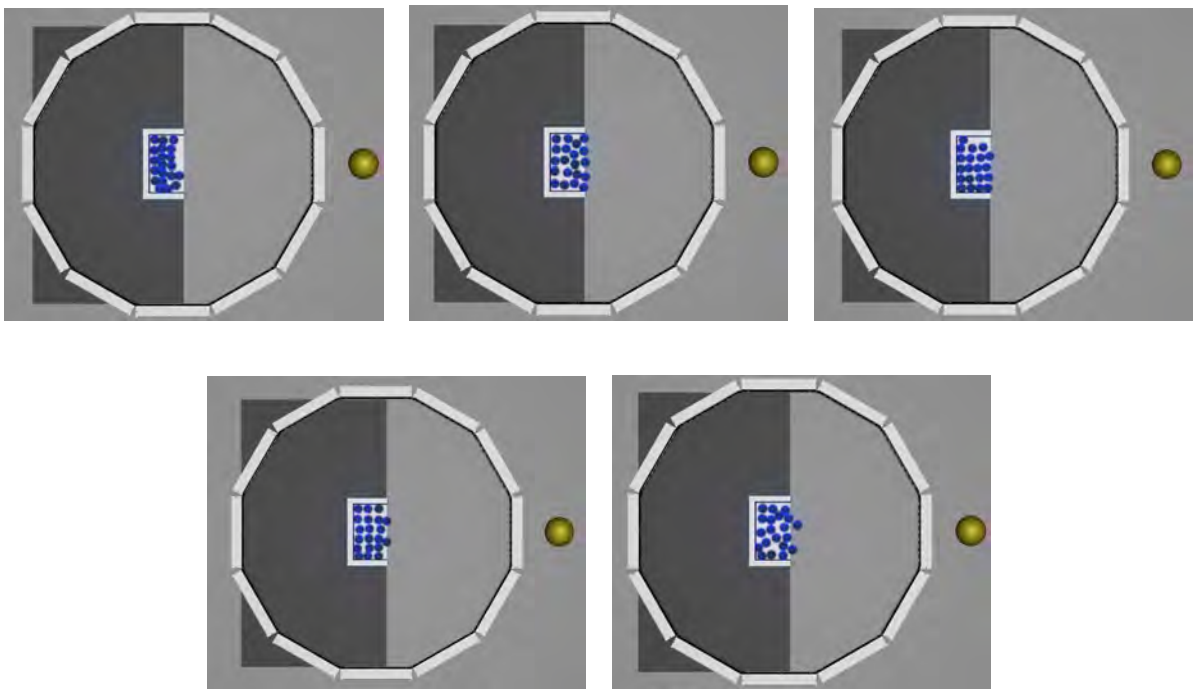


Figure 7.5 – The five pictures represent the arena for SAC constructed in Orchestra. In each of the five pictures, I make one different demonstration of the end position of the robots I want the produced control software from my method to replicate.

Figure 7.6 – Picture of the arena built at IRIDIA for the SAC mission. The robots are randomly placed in the arena at the start of the mission.

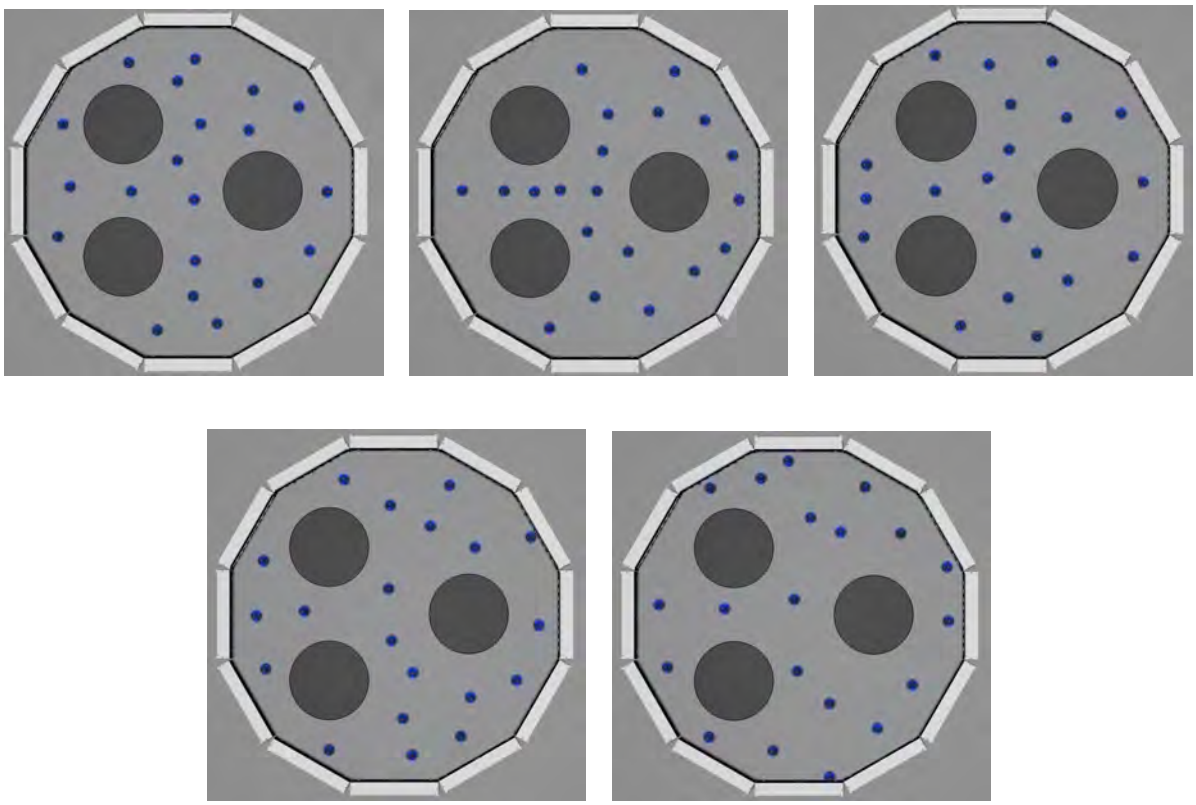


Figure 7.7 – The five pictures represent the arena for CFA constructed in Orchestra. In each of the five pictures, I make one different demonstration of the end position of the robots I want the produced control software from my method to replicate.

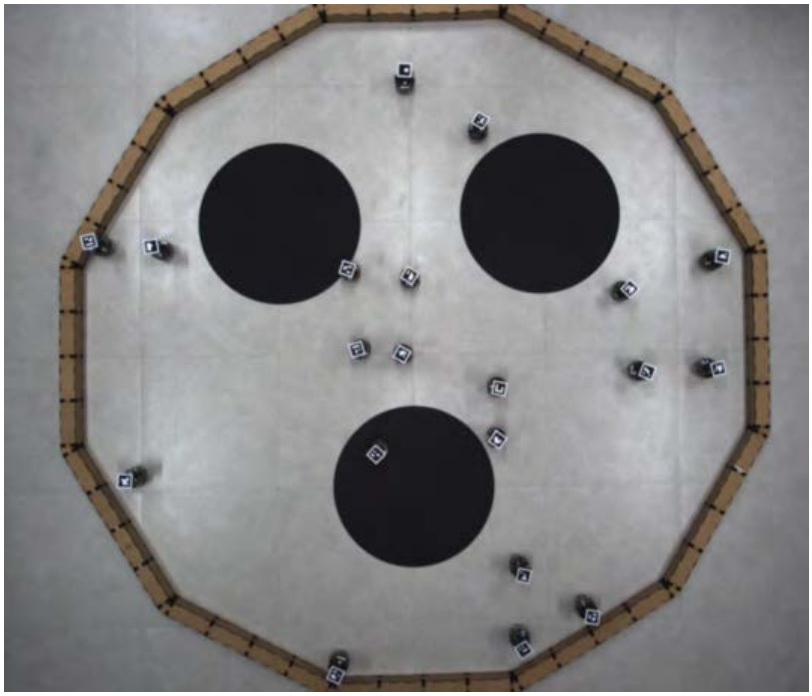


Figure 7.8 – Picture of the arena built at IRIDIA for the CFA mission. The robots are randomly placed in the arena at the start of the mission.

Chapter 8

Results

This Chapter presents the results of the four experiments in simulation and with real robots, I presented in the previous Chapter. For every mission, the performance of Objective-`Evo`, Objective-`Cho` and Demonstration-`Cho` are presented as notched boxplots. The points composing the notched boxplots are computed as explained in Chapter 7. The notched boxplots allow comparing the performances between the three automatic design processes for robot swarm with statistical relevance. Indeed, the notch part of a notched boxplot indicates the 95% confidence interval of the median value. Hence, if the notches of two boxplots do not overlap, then there is a statistically significant difference between the performance of the two methods.

Objective-`Evo` and Objective-`Cho` run each one iteration of design process for a computational budget of 10,000 executions of ARGoS to produce one instance of control software. For Demonstration-`Cho`, I run 50 iterations of the design process with a computational budget of 10,000 executions of ARGoS per iteration to produce one design process.

For every mission, ten control software are generated per control design process. Each notched boxplot is constructed with the score of one run of each ten control software for each design process. Half of the boxplots are the results of the experiments in simulation (in blue), and the other half are from the experiments with real robots (in green)

The ten instances of Demonstration-`Cho` produce together a t-plot for each mission. In the t-plot, each curve represents the value of the max-margin t of the SVM (or the t-value) over the design process iterations for each instance (called experience) of Demonstration-`Cho`. The experience with the lowest t-value among all ten experiences is related to the best-generated control software. Indeed, the best control software for Demonstration-`Cho` means that in the feature space, the features vector generated by Demonstration-`Cho` pushed the most the boundary between all features vectors in the dataset of the SVM and the features vector of the expert's demonstration. This best control software is represented below in the form of a probabilistic finite state machine (PFSM) for every mission.

The experiments were recorded on videos accessible in this git repository: [GIT LINK].

The following sections present all the results from the simulation and reality experiments for each mission.

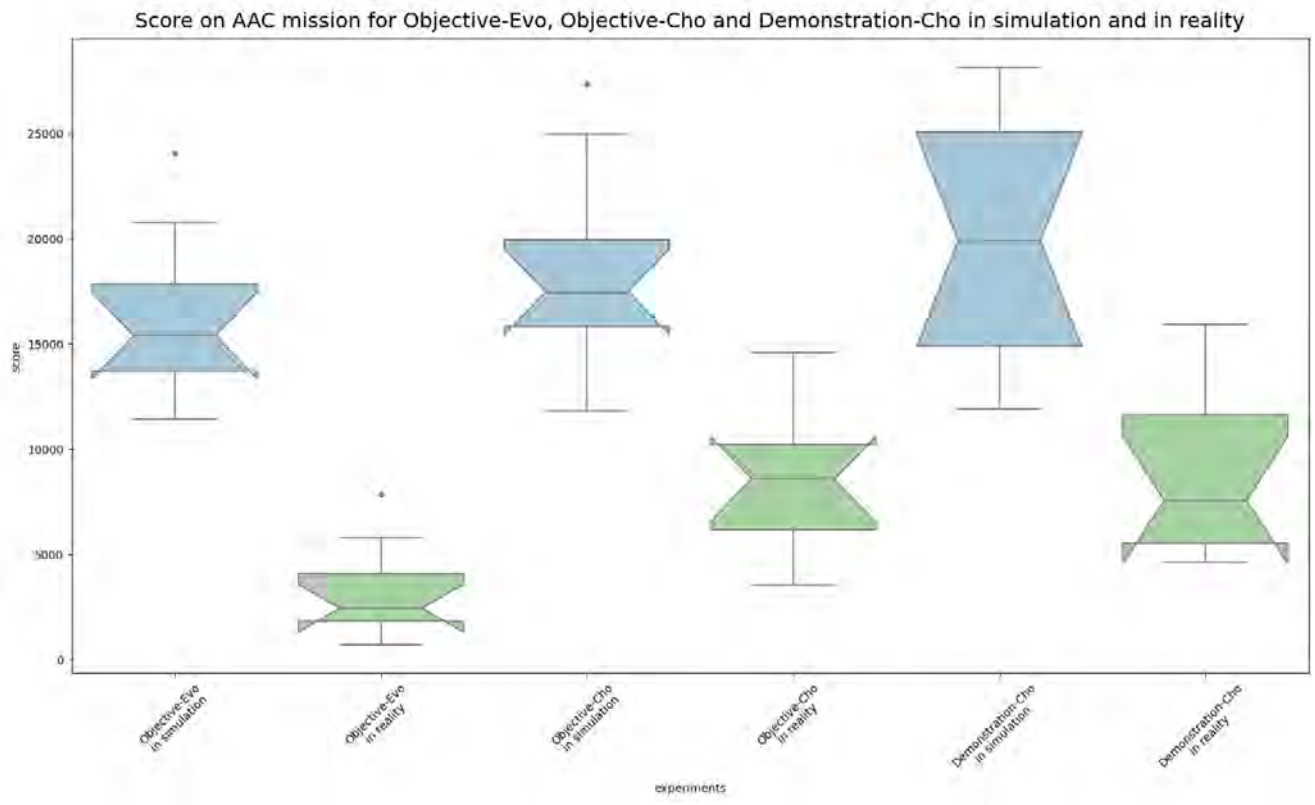


Figure 8.1 – Comparison between the three control process design in terms of performance in both simulation (in blue) and reality (in green). Each notched boxplot is computed from the result of one single run for every ten produced control software per control software design process.

8.1 Aggregation with Ambient Cues

Figure 8.1 shows the notched boxplots of the three design processes for the AAC mission in both simulation and reality. The notch part of the three design processes' boxplots in simulation overlap, meaning that the three methods have similar performance in simulation. When assessed in reality, all design Methods suffer from a drop in performance. Objective-`Evo` is the one who suffered the most from the reality gap as the average performance loss from simulation to reality is 13158, in contrast to an average of 11370 for Demonstration-`Cho` and 9913 for Objective-`Cho`. Hence, Objective-`Cho` and Demonstration-`Cho` are more robust to the reality gap as their reality boxplot upper quartile overlap with their simulation counterpart boxplot lower quartile. The notch part of the reality boxplots of Objective-`Cho` and Demonstration-`Cho` overlap, meaning they perform similarly in real experiments. Furthermore, the notch part of the reality boxplot of Objective-`Evo` is below both the notch part of Objective-`Cho` and Demonstration-`Cho` reality boxplots, meaning that Objective-`Evo` is less efficient in reality than the other two design processes for the AAC mission. The fact that Demonstration-`Cho` has similar performance as the other two methods in simulation and reality is fascinating. Indeed, the objective function of AAC is continuously evaluated. However, even if Demonstration-`Cho` only designs control software evaluated at the last step of the mission, it performs reasonably well even when evaluated at every step.

The t-plot from Figure 8.2 shows that all ten runs of Demonstration-`Cho` for the AAC mission rapidly improve within the first iterations to reach their lower max-margin value at very different iterations. Experience 1 reaches the lowest value of t at iteration 23, meaning that the generated control software for this iteration is the one reducing the max-margin the most for all experiences.

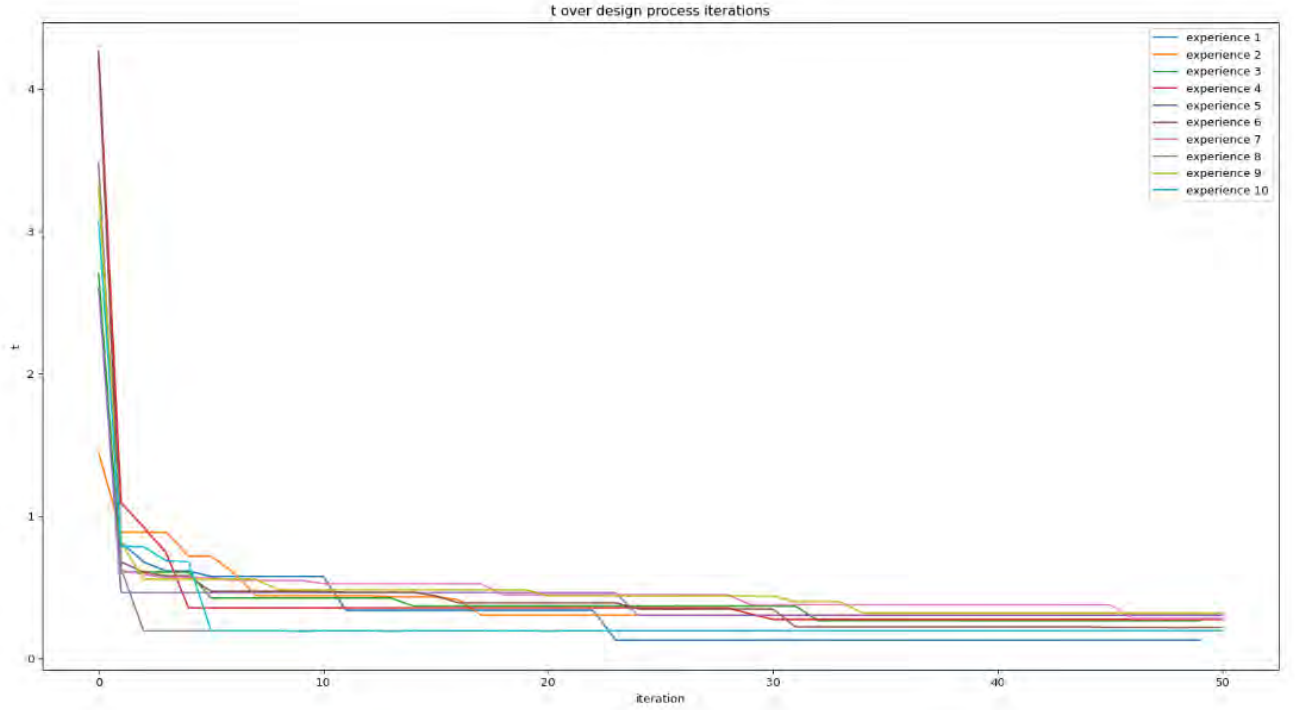


Figure 8.2 – The graph shows ten curves representing the evolution of the SVM max-margin over the ten design process iterations over the AAC mission, called experience. The best behaviour is associated with the experience that reached the global minimum value of t.

Figure 8.3 shows the PFSM related to iteration 23 of experience 1 of the t-plot from Figure 8.2. To accomplish the AAC mission, Demonstration-`Cho` has constructed behaviour that gathers the robots together and brings them toward the light source until they eventually reach the black spot. Once on the black spot, the robots alternate between approaching and receding from the light source and staying grouped to keep the swarm within the black spot. Hence, the robot swarm stays inside the black spot until the end of the allowed time.

8.2 Homing

Figure 8.4 shows the notched boxplots of the three design processes for the Homing mission in both simulation and reality. The notch parts of the Objective-`Cho` and Demonstration-`Cho` boxplot overlap in simulation. This means that the two methods have similar performance in simulation. The notch part of the boxplot of Objective-`Evo` in simulation is slightly below the two other design processes, meaning that it is slightly less efficient for the Homing mission in simulation than the other two. When assessed in reality, all design methods suffer a drop in performance. Objective-`Evo` is the one who suffered the most from the reality gap. The average performance loss from simulation to reality is about 0.39, in contrast to an average of 0.36 for Demonstration-`Cho` and 0.32 for Objective-`Cho`. Objective-`Cho` and Demonstration-`Cho` suffered less from the reality gap as their reality boxplot upper quartile overlapped with their simulation counterpart boxplot lower quartile. The notch part of the reality boxplots of Objective-`Cho` and Demonstration-`Cho` overlap, meaning they perform similarly in real experiments. Furthermore, the notch part of the reality boxplot of Objective-`Evo` slightly overlaps the notch part of both notch part of Objective-`Cho` and Demonstration-`Cho` reality boxplots from the bottom, meaning that Objective-`Evo` is slightly less efficient in reality than the other two design processes for the Homing mission in reality.

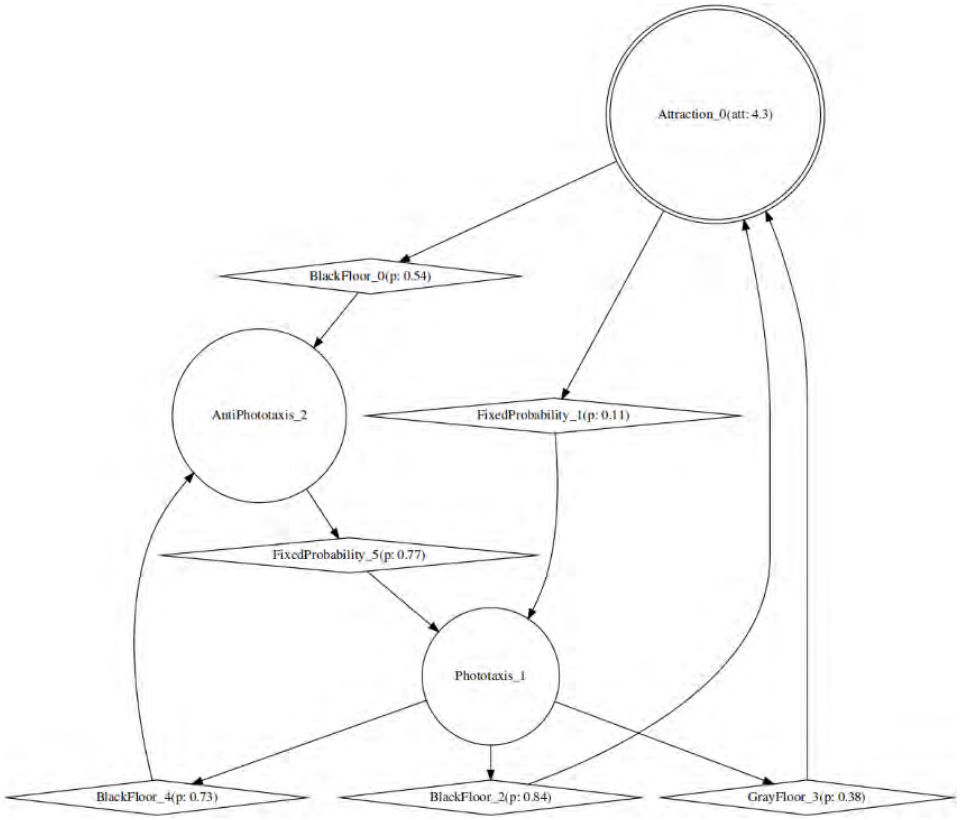


Figure 8.3 – The figure represents the probabilistic finite state machine of the best behaviour for the AAC mission chosen among the ten experiences of Demonstration-`Cho` in simulation.

Figure 8.5 shows the t-plot for the ten instances of Demonstration-`Cho`. For all experience, the t-value decrease rapidly for the first iterations to stay relatively constant for the rest of the iterations. There is also a wider range of value of the max-margin among the experiences. Here again, the experiences reach they lower t-value at very different iterations. The lowest t-value is reached by experience 10 at iteration 44.

Figure 8.6 represents the control software generated at iteration 44 of experience 10. The strategy of that PFSM is to explore the arena and aggregate with the other robots once on the black spot. Noticeably, the « Stop » behaviour and condition above the initial behaviour in Figure 8.6 is never reached.

8.3 Sheltering with Ambient Cues

Figure 8.7 shows the notched boxplots of the three design processes for the SAC mission in both simulation and reality. The notch part of the Objective-`Evo` and Objective-`Cho` boxplots in simulation overlap, meaning that the two methods have similar performance in simulation. The notched boxplot of Demonstration-`Cho` in simulation is noticeably very narrow, with its notch part overlapping with the top of the notch part of the other two design processes' boxplot, meaning that Demonstration-`Cho` performed slightly better on SAC in simulation than both Objective-`Evo` and Objective-`Cho`. When assessed in reality, all design Methods suffer from a drop in performance. Demonstration-`Cho` is this time the one suffering the most from the reality gap. Indeed, there is a considerable gap between Demonstration-`Cho` simulation and reality boxplot. The performance

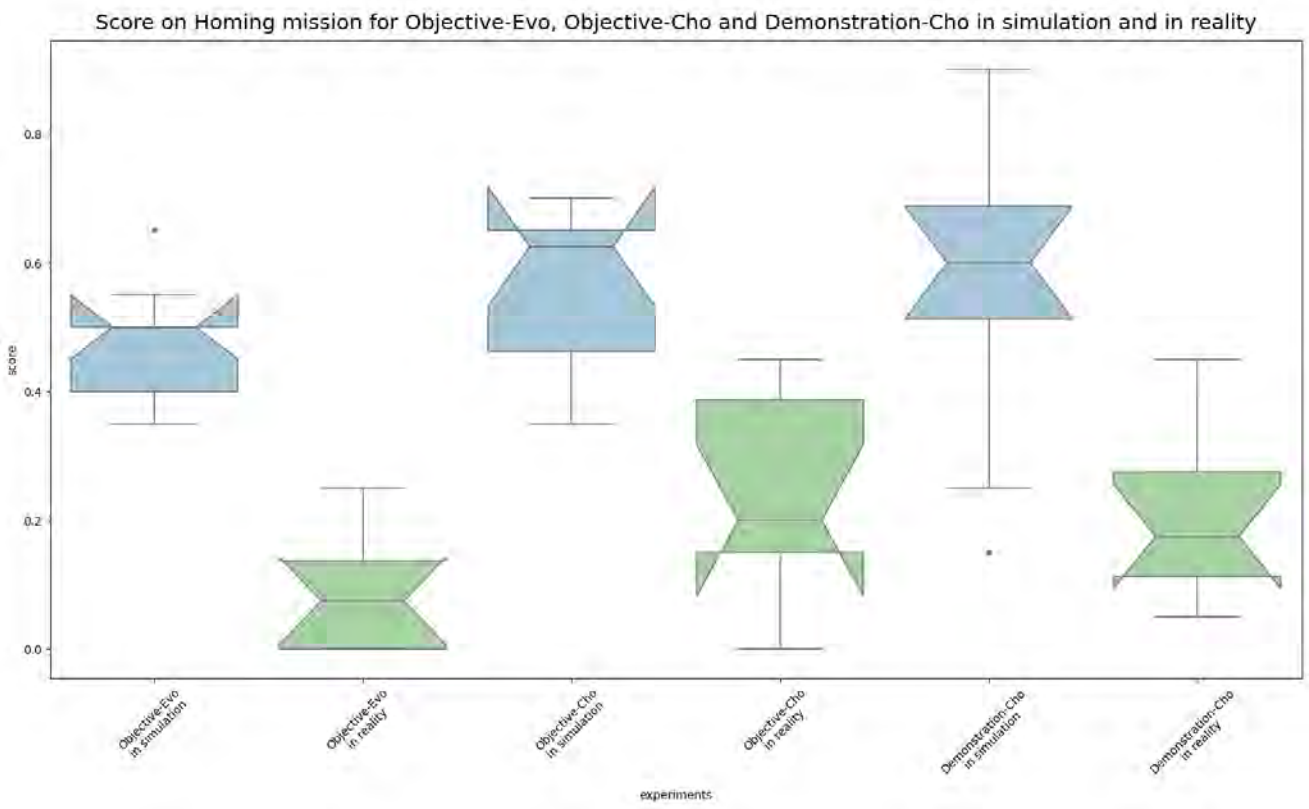


Figure 8.4 – Comparison between the three control process design in terms of performance in both simulation (in blue) and reality (in green). Each notched boxplot is computed from the result of one single run for every ten produced control software per control software design process.

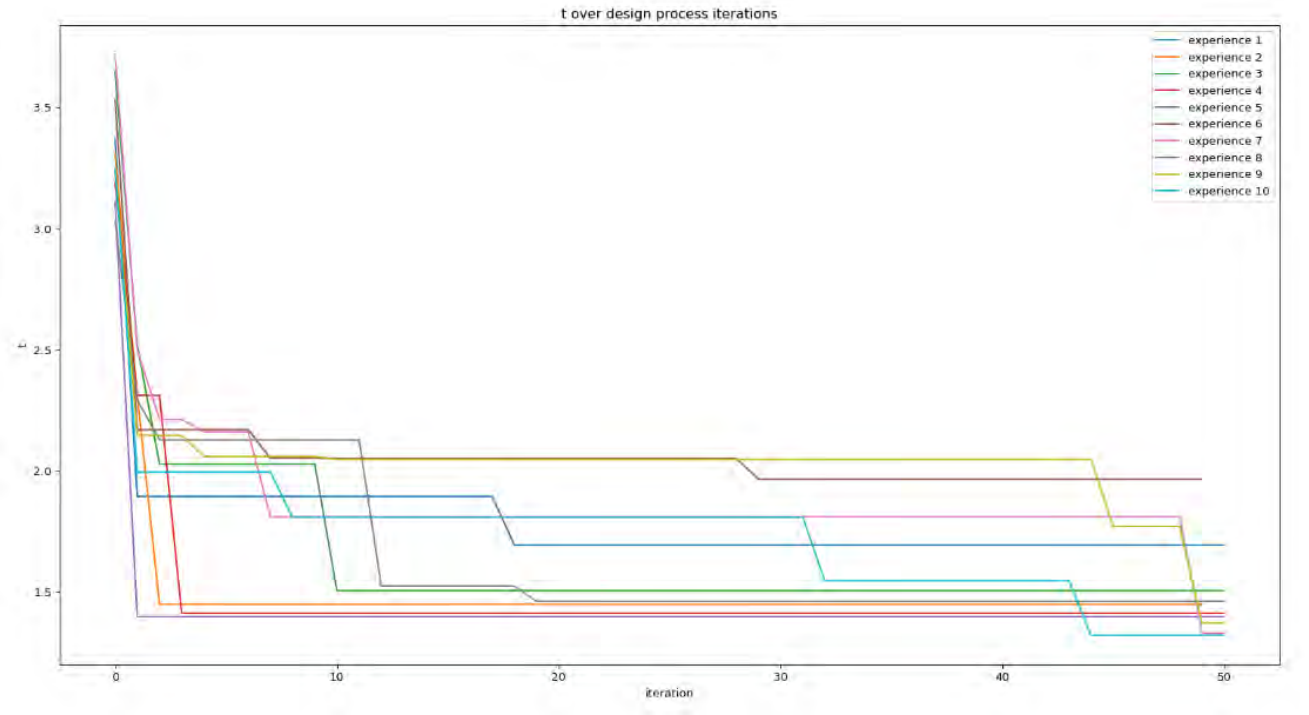


Figure 8.5 – The graph shows ten curves representing the evolution of the SVM max-margin over the ten design process iterations for the Homing mission, called experience. The best behaviour is associated with the experience that reached the global minimum value of t.

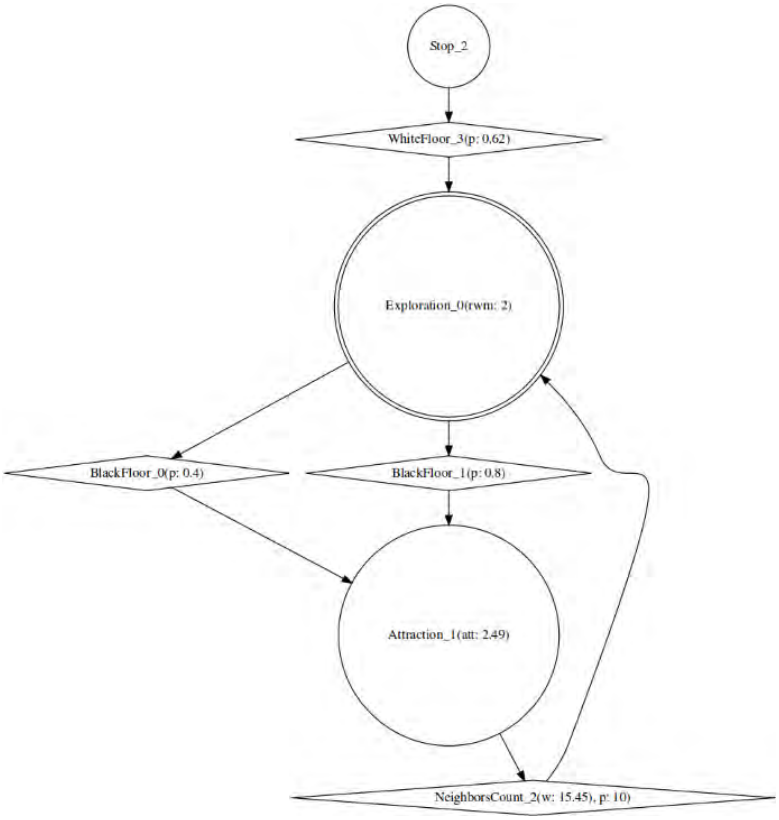


Figure 8.6 – The figure represents the probabilistic finite state machine of the best behaviour for the Homing mission chosen among the ten experiences of Demonstration-`Cho` in simulation.

loss of Demonstration-`Cho` is 11675, against an average of 9218 and 6572 for respectively Objective-`Evo` and Objective-`Cho`. Objective-`Evo` also suffers from the reality gap with its simulation and reality compared to Objective-`Cho`. Hence, Objective-`Cho` is the most robust method for SAC. The notch part of the reality boxplots of the three design processes overlaps, meaning they have similar performance in real experiments for the SAC mission. The fact that Demonstration-`Cho` has similar performance as the other two methods in simulation and reality is fascinating. Indeed, the objective function of SAC is continuously evaluated. However, even if Demonstration-`Cho` only designs control software evaluated at the last step of the mission, it performs reasonably well even when evaluated at every step.

The t-plot of Figure 8.5 shows the ten experiences with Demonstration-`Cho` on SAC in simulation. For all experience, the t-value decrease rapidly for the first iterations to stay relatively constant for the rest of the iterations. The experiences reach they lower t-value at very different iterations once again. The lowest t-value is reached by experience 7 at iteration 11.

Figure 8.9 represents the PFSM generated at iteration 11 of experience 7. The PFSM displays a straightforward strategy where the robot goes to the light source to move away from the light when a certain number of neighbours is present nearby. Either the robot will enter the shelter or touch the black area, so it will fall back to its initial state and repeat the process until it enters the shelter.

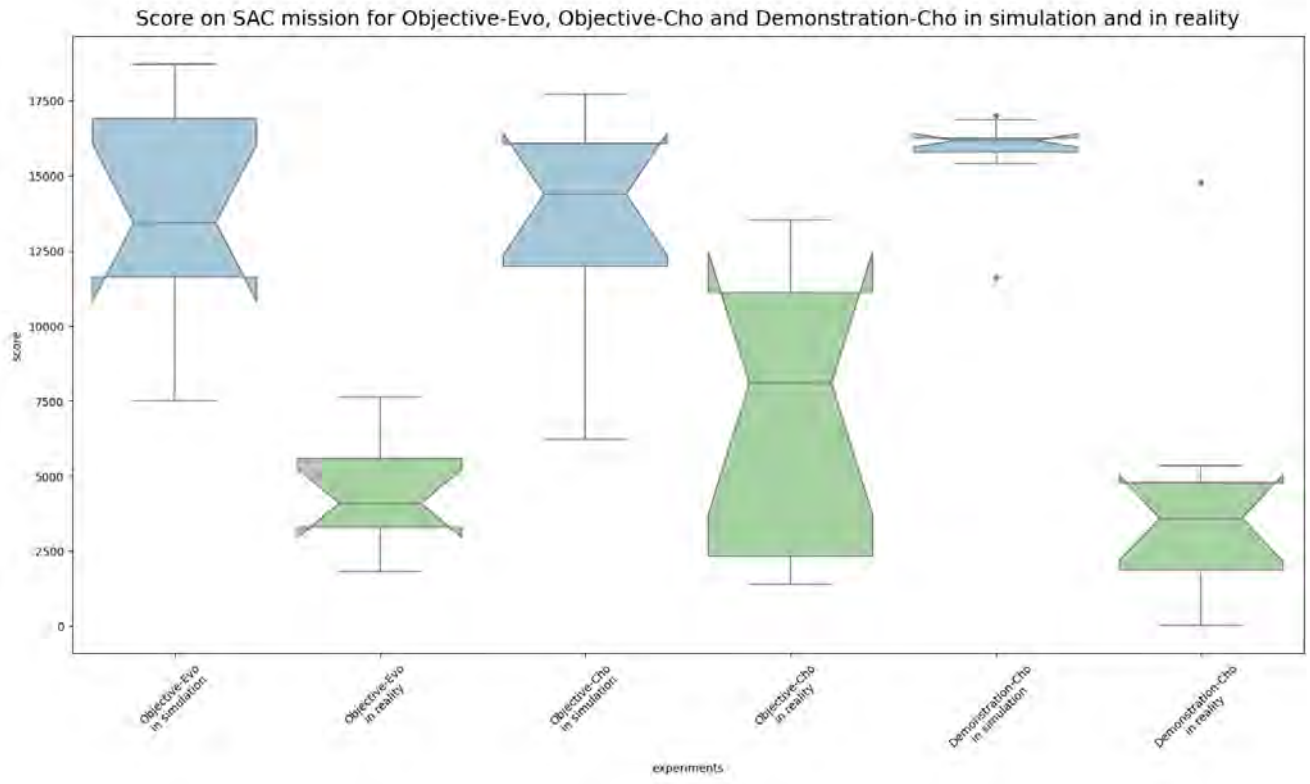


Figure 8.7 – Comparison between the three control process design in terms of performance in both simulation (in blue) and reality (in green). Each notched boxplot is computed from the result of one single run for every ten produced control software per control software design process.

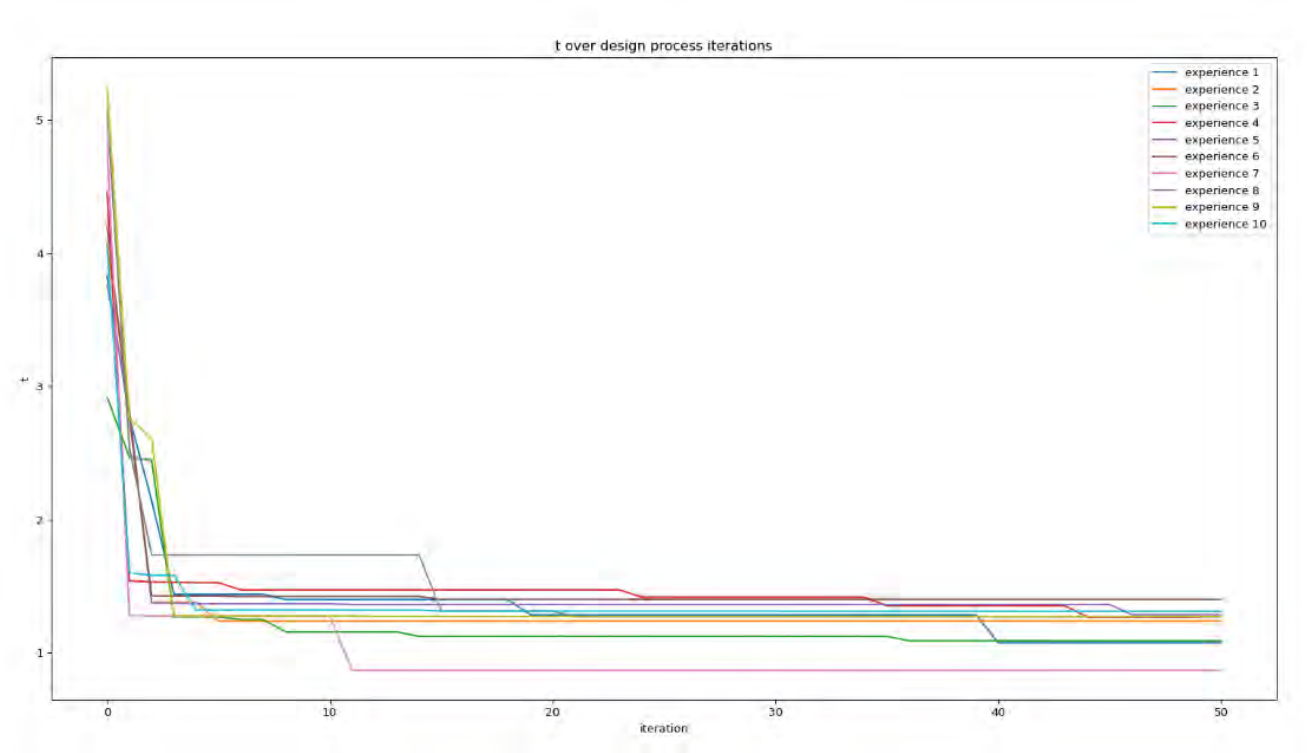


Figure 8.8 – The graph shows ten curves representing the evolution of the SVM max-margin over the ten design processes iterations for the SAC mission, called experience. The best behaviour is associated with the experience that reached the global minimum value of t.

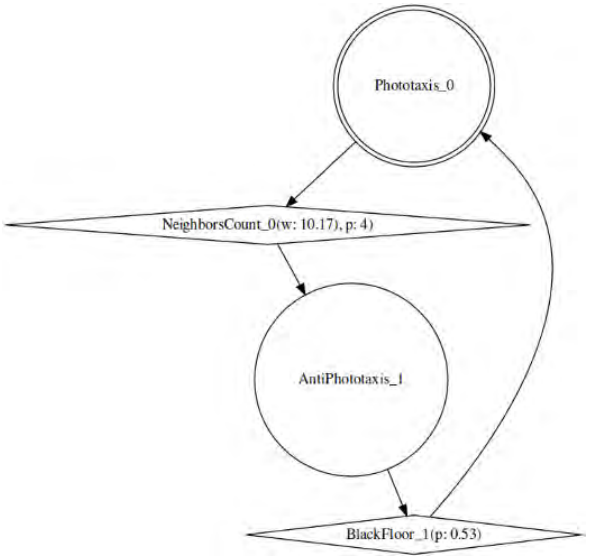


Figure 8.9 – The figure represents the probabilistic finite state machine of the best behaviour for the SAC mission chosen among the ten experiences of Demonstration-`Cho` in simulation.

8.4 Coverage with Forbidden Areas

Figure 8.10 shows the notched boxplots of the three design processes for the CFA mission in both simulation and reality. The notch part of the boxplots of the three design processes in simulation overlap, meaning that the three methods have similar performance. Thus, the notch part of the boxplot of Objective-`Cho` in simulation is slightly above the ones of the other two design processes' in reality, meaning that Objective-`Cho` performs better than the other two design processes for CFA in simulation. This time, Objective-`Cho` suffered the most from the reality gap with an average drop of performance from simulation to reality of 6, against 2 for Objective-`Evo`. Furthermore, Demonstration-`Cho` actually improved in reality compare to the simulation with a average increase of 2. We can also see that the three design processes are just as efficient as each other for CFA in reality as the notch part of their reality boxplot overlap.

The t-plot of Figure 8.11 shows the ten experiences with Demonstration-`Cho` on SAC in simulation. The max-margin value of every experience is in a small range in the stationary part of the curves. For all experiences, the t-value decreases rapidly for the first iterations to stay relatively constant for the rest of the iterations. The experiences reach they lower t-value at very different iterations once again. The lowest t-value is reached by experience 4 at iteration 38.

Figure 8.12 represents the PFSM generated at iteration 38 of experience 4. The strategy of the PFSM is to switch between the « Repulsion » behaviour for covering the whole arena and the « Exploration » behaviour to escape from the black patches. Noticeably, some transition conditions involve white patches even though the mission does not contain any.

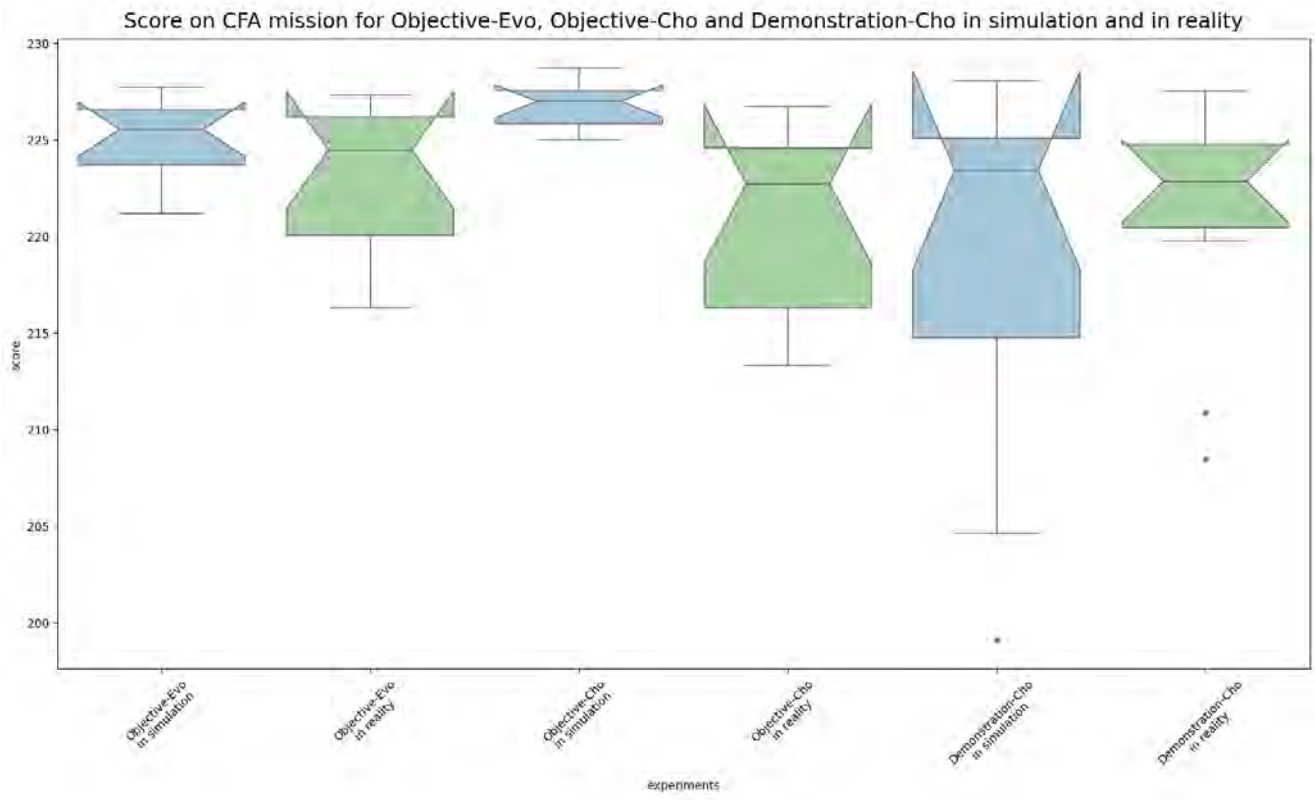


Figure 8.10 – Comparison between the three control process design in terms of performance in both simulation (in blue) and reality (in green). Each notched boxplot is computed from the result of one single run for every ten produced control software per control software design process.

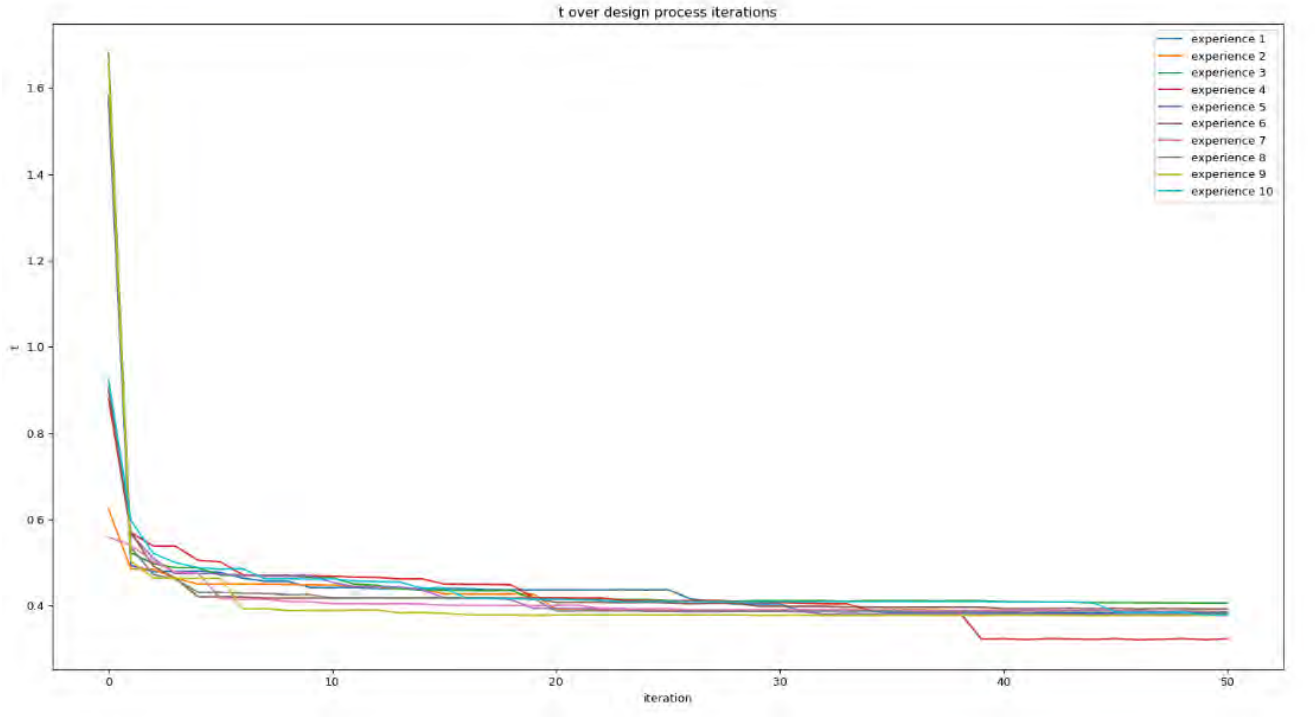


Figure 8.11 – The graph shows ten curves representing the evolution of the SVM max-margin over the ten design process iterations for the CFA mission, called experience. The best behaviour is associated with the experience that reached the global minimum value of t.

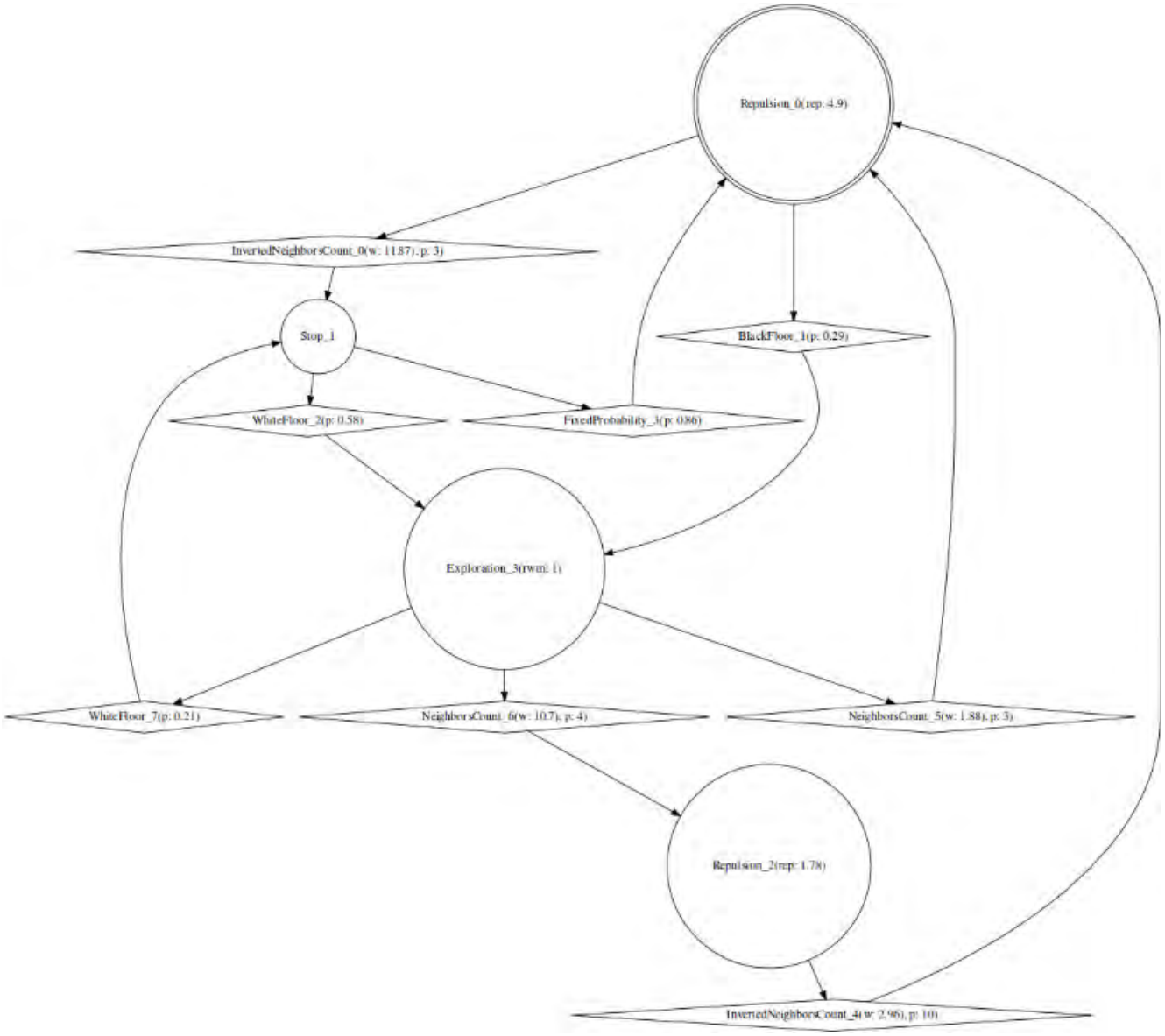


Figure 8.12 – The figure represents the probabilistic finite state machine of the best behaviour for the CFA mission chosen among the ten experiences of Demonstration-`Cho` in simulation.

Chapter 9

Discussion and Further Work

The goal of my master thesis is to propose an automatic modular design producing control software to accomplish robot swarm missions without specifying any explicit objective function of the robot swarm mission. In this Chapter, I discuss the success of my master thesis based on the results I presented in Chapter 8. I also provide some insights about Demonstration-Cho and the future work that needs to be considered to improve Demonstration-Cho.

The previous Chapter's results indicate that Demonstration-Cho can display similar performances as Objective-Cho both in simulation and with real robots. This implies that Demonstration-Cho is similarly robust to the reality gap as Objective-Cho. Furthermore, Demonstration-Cho achieve to produce reasonably good control software for missions which are continuously evaluated, even though the design process of Demonstration-Cho only evaluates Control Software at the final time step..

Indeed, the only time Demonstration-Cho suffered heavily from the reality gap was for the SAC mission. I explain Demonstration-Cho not being able to cross the reality gap for the SAC mission by the fact that Demonstration-Cho use only variants of one specific strategy. This strategy, represented by the PFSM in Figure 8.9, is very efficient in simulation, as clearly shown by the narrow notched boxplot of Demonstration-Cho in simulation from Figure 8.7. Although well-performing, when applying this strategy, it appears that the transition from the « Anti-Phototaxis » to the « Phototaxis » behaviour is triggered too often. Indeed, due to a potential misreading of the floor colour of the arena, the robots read the arena's floor as black before entering the black area. Objective-Cho for some runs with the real robots also displays the same strategy and suffers from the same issue. Although one strategy produced by Objective-Cho suffers from the reality gap, Objective-Cho displays more than one strategy, explaining how it performs better than Demonstration-Cho for real experiments. However, in general, the results from Chapter 8 show that Demonstration-Cho achieves similar performance as Objective-Cho for the four missions considered, both in simulation and in reality. Demonstration-Cho also showed to be robust to the reality gap for three out of four missions.

I have some insights on the control software produced by Demonstration-Cho. The first insight is that Demonstration-Cho does not diversify its strategy, leading to issues like the reality gap problem with SAC explained above. The other insight is how some PFSMs produced by Demonstration-Cho have useless parts. For example, the PFSM from Figure 8.6 and Figure 8.12 both have unreachable behaviour or transition conditions impossible to satisfy in the scope of the considered mission. I would argue that the production of PFSM by Demonstration-Cho should be improved both in diversification and optimisation.

I also have an insight into the scalability of Demonstration-`Cho` regarding missions it can address. Indeed, for the four considered missions, Demonstration-`Cho` seems to efficiently produce satisfactory control software, even without the explicit objective function of the mission. Although, only the positions of the robot are taken into account to achieve the mission I selected in Section 6.4. Hence, I have chosen the mission that the feature space I have defined can represent. However, for the `Foraging` mission, for example (see section 6.4), the features should take into account more information than only the final positions of the robot. In order to address missions such as foraging, the feature space Needs to be adapted. Furthermore, Orchestra will also need to be extended to allow the expert to demonstrate other swarm behaviour than only the final position of the robots.

For the four missions I consider, the feature vectors of the expert demonstration usually have a value close to 1 or 0, meaning that the features vector of the expert's demonstration is usually at the border of the definition domain of the feature space. Hence, it is usually always possible to linearly separate the features vector of the expert's demonstration from the features vectors of the PFSM produced by Demonstration-`Cho`. However, it is not guaranteed that in the future, the features vector of the expert's demonstration will always be linearly separable from the features vector produced from Demonstration-`Cho`. This means that the SVM must be carefully used, and a potentially more suited classifier might be considered in the future.

Finally, I think that addressing the issue with my version of APRIL (see Chapter 5) by investigating how to define SMS for robot swarm is interesting as it would lead to an alternative to Demonstration-`Cho`.

Chapter 10

Conclusion

In this master thesis, I investigate how to specify a robot swarm mission without constructing any explicit objective function. As a first step, I implemented an arena builder in Orchestra. The arena builder of Orchestra allows to construct the setup for some robot swarm missions from the IRIDIA lab literature and to save the arena specification from being used with the robot simulator ARGoS.

As a second step, I investigate how to produce control software for robot swarms to achieve the mission without specifying any explicit objective function. By reviewing the literature, I found two interesting reinforcement learning algorithms: APRIL, a Preference-based Learning algorithm; Apprenticeship learning, an Inverse Reinforcement Leaning algorithm.

I first adapted the Active Preference Learning-Based reinforcement learning algorithm (APRIL). Unfortunately, I did not have positive results with that approach as the *sensori-motor states* used by APRIL in the scope of single agent learning are difficult to define for robot swarms.

Then, I adapted the apprenticeship learning algorithm to be usable in swarm robotics. The apprenticeship learning algorithm requires expert demonstrations of the desired behaviour to produce a policy to replicate that behaviour. In order to provide These demonstrations, I added a feature to Orchestra allowing the user to place the robots in their final positions. I also defined a feature function, mapping the state of the swarm into a Vector usable by the apprenticeship learning algorithm. As the last adaptation, I modify the apprenticeship learning algorithm's policy generation phase. Instead of generating a policy for a single learning agent with a reinforcement learning algorithm, I generate a control software for robot swarms with the automatic modular design method AutoMoDe-`Chocolate`.

Once I implemented Demonstration-`Cho`, I asses its performance in simulation and reality experiments with physical robots on missions I selected from the literature. The performances of Demonstration-`Cho` were compared with two automatic design processes called Objective-`Cho` and Objective-`Evo`. Objective-`Cho` and Objective-`Evo` are respectively AutoMoDe-`Chocolate` and `EvoStick` whose have access to the explicit objective functions of each missions.

The results from my experiments show that Demonstration-`Cho` achieves similar performance for all missions, both in simulation and in reality, as Objective-`Cho`. Hence, I succeed in proposing a new automatic modular design to produce robot swarm control software without the need to provide an explicit objective function.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. « Apprenticeship learning via inverse reinforcement learning ». In: *ICML '04: Proceedings of the twenty-first international conference on Machine learning*. NewYork$_N Y_U SA$, 2004, p. 1. DOI: 10.1145/1015330.1015430.

[2] Riad Akrour, Marc Schoenauer, and Michèle Sebag. « April: Active preference learning-based reinforcement learning ». In: *2012 Joint European conference on machine learning and knowledge discovery in databases*. 2012, pp. 116–131. DOI: https://doi.org/10.1007/978-3-642-33486-3_8.

[3] Riad Akrour, Marc Schoenauer, and Michèle Sebag. « Preference-based policy learning ». In: *2011 Joint European conference on machine learning and knowledge discovery in databases*. 2011, pp. 12–27.

[4] Saurabh Arora and Prashant Doshi. « A survey of inverse reinforcement learning: Challenges, methods and progress ». In: *Artificial Intelligence* 297 (2021), pp. 103–500. DOI: https://doi.org/10.1016/j.artint.2021.103500.

[5] Jan C. Barca and Y. A. Sekercioglu. « Swarm robotics reviewed ». In: *Robotica* 31.3 (2013), pp. 345–359. DOI: 10.1017/S026357471200032X.

[6] Mikhail Belkin et al. « Reconciling modern machine-learning practice and the classical bias–variance trade-off ». In: *Proceedings of the National Academy of Sciences of the United States of America* 116.32 (2019), pp. 15849–15854. DOI: 10.1073/pnas.1903070116.

[7] Gerardo Beni. « From swarm intelligence to swarm robotics ». In: *Swarm Robotics: SAB 2004 International Workshop*. Ed. by Erol Şahin and William M. Spears. Vol. 3342. Lecture Notes in Computer Science. Berlin, Germany: Springer, 2005, pp. 1–9. DOI: 10.1007/978-3-540-30552-1_1.

[8] Raffaele Bianco and Stefano Nolfi. « Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce ». In: *Connection Science* 16.4 (2004), pp. 227–248. DOI: 10.1080/09540090412331314759.

[9] Mauro Birattari, Antoine Ligot, and Gianpiero Francesca. « AutoMoDe: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms ». In: *Automated Design of Machine Learning and Search Algorithms*. Ed. by Nelishia Pillay and Rong Qu. Natural Computing Series. Cham, Switzerland: Springer, 2021, pp. 73–90. DOI: 10.1007/978-3-030-72069-8_5.

[10] Mauro Birattari, Antoine Ligot, and Ken Hasselmann. « Disentangling automatic and semi-automatic approaches to the optimization-based design of control software for robot swarms ». In: *Nature Machine Intelligence* 2.9 (2020), pp. 494–499. DOI: 10.1038/s42256-020-0215-0.

[11] Mauro Birattari et al. « F-Race and Iterated F-Race: an overview ». In: *Experimental Methods for the Analysis of Optimization Algorithms*. Ed. by Thomas Bartz-Beielstein et al. Berlin, Germany: Springer, 2010, pp. 311–336. DOI: 10.1007/978-3-642-02538-9_13.

[12] Darko Bozhinoski and Mauro Birattari. « Designing control software for robot swarms: software engineering for the development of automatic design methods ». In: *RoSE'18: Proceedings of the 1st International Workshop on Robotics Software Engineering*. New York, NY, USA: ACM, 2018, pp. 33–35. DOI: 10.1145/3196558.3196564.

[13] Manuele Brambilla et al. « Swarm robotics: a review from the swarm engineering perspective ». In: *Swarm Intelligence* 7.1 (2013), pp. 1–41. DOI: 10.1007/s11721-012-0075-2.

[14] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. « Multi-agent reinforcement learning: An overview ». In: *Innovations in multi-agent systems and applications-1* 297 (2010), pp. 183–221. DOI: https://doi.org/10.1007/978-3-642-14435-6_7.

[15] Lucian Buşoniu, Bart De Schutter, and Robert Babuška. « Decentralized reinforcement learning control of a robotic manipulator ». In: ed. by Uwe Aßmann and Gerd Wagner. Vol. 1319. IEEE, 2006, pp. 1–6. DOI: 10.1109/ICARCV.2006.345351.

[16] Davide Di Ruscio, Ivano Malavolta, and Patrizio Pelliccione. « A family of domain-specific languages for specifying civilian missions of multi-robot systems ». In: *MORSE 2014 Model-Driven Robot Software Engineering: Proceedings of the 1st International Workshop on Model-Driven Robot Software Engineering*. Ed. by Uwe Aßmann and Gerd Wagner. Vol. 1319. Aachen, Germany: CEUR Workshop Proceedings, 2014, pp. 13–26.

[17] Marco Dorigo and Mauro Birattari. « Swarm intelligence ». In: *Scholarpedia* 2.9 (2007), p. 1462. DOI: 10.4249/scholarpedia.1462.

[18] Marco Dorigo, Mauro Birattari, and Manuele Brambilla. « Swarm robotics ». In: *Scholarpedia* 9.1 (2014), p. 1463. DOI: 10.4249/scholarpedia.1463.

[19] Marco Dorigo, Guy Theraulaz, and Vito Trianni. « Swarm robotics: past, present, and future [point of view] ». In: *Proceedings of the IEEE* 109.7 (2021), pp. 1152–1165. DOI: 10.1109/JPROC.2021.3072740.

[20] Miguel Duarte et al. « Evolution of collective behaviors for a real swarm of aquatic surface robots ». In: *PLOS ONE* 11.3 (2016), e0151834. DOI: 10.1371/journal.pone.0151834.

[21] Dario Floreano, Phil Husbands, and Stefano Nolfi. « Evolutionary robotics ». In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer Handbooks. First edition. Berlin, Germany: Springer, 2008, pp. 1423–1451. DOI: 10.1007/978-3-540-30301-5_62.

[22] Gianpiero Francesca and Mauro Birattari. « Automatic design of robot swarms: achievements and challenges ». In: *Frontiers in Robotics and AI* 3.29 (2016), pp. 1–9. DOI: 10.3389/frobt.2016.00029.

[23] Gianpiero Francesca et al. « An experiment in automatic design of robot swarms: AutoMoDe-Vanilla, EvoStick, and human experts ». In: *Swarm Intelligence: 9th International Conference, ANTS 2014*. Ed. by Marco Dorigo et al. Vol. 8667. Lecture Notes in Computer Science. Cham, Switzerland: Springer International Publishing, 2014, pp. 25–37. DOI: 10.1007/978-3-319-09952-1_3.

[24] Gianpiero Francesca et al. « AutoMoDe-Chocolate: automatic design of control software for robot swarms ». In: *Swarm Intelligence* 9.2–3 (2015), pp. 125–152. DOI: 10.1007/s11721-015-0107-9.

[25] Gianpiero Francesca et al. « AutoMoDe: a novel approach to the automatic design of control software for robot swarms ». In: *Swarm Intelligence* 8.2 (2014), pp. 89–112. DOI: 10.1007/s11721-014-0092-4.

[26] Lorenzo Garattoni and Mauro Birattari. « Swarm robotics ». In: *Wiley Encyclopedia of Electrical and Electronics Engineering*. Ed. by John G. Webster. Hoboken, NJ, USA: John Wiley & Sons, 2016, pp. 1–19. DOI: 10.1002/047134608X.W8312.

[27] Lorenzo Garattoni et al. *Software infrastructure for e-puck (and TAM)*. Tech. rep. TR/IRIDIA/201 004. Brussels, Belgium: IRIDIA, Université Libre de Bruxelles, 2015.

[28] David Garzón Ramos and Mauro Birattari. « Automatic design of collective behaviors for robots that can display and perceive colors ». In: *Applied Sciences* 10.13 (2020), p. 4654. DOI: 10.3390/app10134654.

[29] Roderich Groß and Marco Dorigo. « Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling ». In: *Adaptive Behavior* 16.5 (2008), pp. 285–305. DOI: 10.1177/1059712308090537.

[30] Heiko Hamann. *Swarm robotics: a formal approach*. Cham, Switzerland: Springer, 2018. ISBN: 978-3-319-74526-8. DOI: 10.1007/978-3-319-74528-2.

[31] Ken Hasselmann and Mauro Birattari. *Modular automatic design of collective behaviors for robots endowed with local communication capabilities: supplementary material*. http://iridia.ulb.ac.be/supp/IridiaSupp2019-005/. 2019.

[32] Ken Hasselmann, Frédéric Robert, and Mauro Birattari. « Automatic design of communication-based behaviors for robot swarms ». In: *Swarm Intelligence: 11th International Conference, ANTS 2018*. Ed. by Marco Dorigo et al. Vol. 11172. Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2018, pp. 16–29. DOI: 10.1007/978-3-030-00533-7_2.

[33] Ken Hasselmann et al. « Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms ». In: *Nature Communications* 12 (2021), p. 4345. DOI: 10.1038/s41467-021-24642-3.

[34] Ken Hasselmann et al. *Reference models for AutoMoDe*. IRIDIA, 2019.

[35] Marti A. Hearst et al. « Support vector machines ». In: *IEEE Intelligent Systems and their Applications* 13.4 (1998), pp. 18–28. DOI: 10.1109/5254.708428.

[36] Verena Heidrich-Meisner and Christian Igel. « Neuroevolution strategies for episodic reinforcement learning ». In: *Journal of Algorithms* 64.4 (2009). Special Issue: Reinforcement Learning, pp. 152–168. DOI: 10.1016/j.jalgor.2009.04.002.

[37] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. « Multilayer feedforward networks are universal approximators ». In: *Neural Networks* 2.5 (1989), pp. 359–366. DOI: https://doi.org/10.1016/0893-6080(89)90020-8.

[38] Nick Jakobi, Phil Husbands, and Inman Harvey. « Noise and the reality gap: the use of simulation in evolutionary robotics ». In: *Advances in Artificial Life: Third European Conference on Artificial Life*. Ed. by F. Morán et al. Vol. 929. Lecture Notes in Artificial Intelligence. Berlin, Germany: Springer, 1995, pp. 704–720. DOI: 10.1007/3-540-59496-5_337.

[39] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. « Reinforcement learning: a survey ». In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 237–285. DOI: 10.1613/jair.301.

[40] Jonas Kuckling, Vincent van Pelt, and Mauro Birattari. « Automatic modular design of behavior trees for robot swarms with communication capabilities ». In: *Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021*. Ed. by Pedro A. Castillo and Juan Luis Jiménez Laredo. Vol. 12694. Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2021, pp. 130–145. DOI: 10.1007/978-3-030-72699-7_9.

[41] Jonas Kuckling, Vincent van Pelt, and Mauro Birattari. « AutoMoDe-Cedrata: automatic design of behavior trees for controlling a swarm of robots with communication capabilities ». In: *SN Computer Science* 3 (2022), p. 136. DOI: 10.1007/s42979-021-00988-9.

[42] Jonas Kuckling, Vincent van Pelt, and Mauro Birattari. *AutoMoDe-Cedrata: automatic design of behavior trees for controlling a swarm of robots with communication capabilities: supplementary material*. http://iridia.ulb.ac.be/supp/IridiaSupp2021-004/. 2021.

[43] Jonas Kuckling, Thomas Stützle, and Mauro Birattari. « Iterative improvement in the automatic modular design of robot swarms ». In: *PeerJ Computer Science* 6 (2020), e322. DOI: 10.7717/peerj-cs.322.

[44] Jonas Kuckling, Keneth Ubeda Arriaza, and Mauro Birattari. « AutoMoDe-IcePop: automatic modular design of control software for robot swarms using simulated annealing ». In: *Artificial Intelligence and Machine Learning: BNAIC 2019, BENELEARN 2019*. Ed. by Bart Bogaerts et al. Vol. 1196. Communications in Computer and Information Science. Cham, Switzerland: Springer, 2020, pp. 3–17.

[45] Jonas Kuckling, Keneth Ubeda Arriaza, and Mauro Birattari. « Simulated annealing as an optimization algorithm in the automatic modular design of robot swarms ». In: *Proceedings of the Reference AI & ML Conference for Belgium, Netherlands & Luxemburg, BNAIC/BENELEARN 2019*. Ed. by Katrien Beuls et al. Vol. 2491. Aachen, Germany: CEUR Workshop Proceedings, 2019.

[46] Jonas Kuckling et al. « Behavior trees as a control architecture in the automatic modular design of robot swarms ». In: *Swarm Intelligence: 11th International Conference, ANTS 2018*. Ed. by Marco Dorigo et al. Vol. 11172. Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2018, pp. 30–43. DOI: 10.1007/978-3-030-00533-7_3.

[47] Antoine Ligot and Mauro Birattari. *On mimicking the effects of the reality gap with simulation-only experiments*. Tech. rep. TR/IRIDIA/2018-003. Brussels, Belgium: IRIDIA, Université Libre de Bruxelles, 2018.

[48] Antoine Ligot and Mauro Birattari. « Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms ». In: *Swarm Intelligence* 14 (2020), pp. 1–24. DOI: 10.1007/s11721-019-00175-w.

[49] Antoine Ligot, Ken Hasselmann, and Mauro Birattari. « AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines ». In: *Swarm Intelligence: 12th International Conference, ANTS 2020*. Ed. by Marco Dorigo et al. Vol. 12421. Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2020, pp. 109–122. DOI: 10.1007/978-3-030-60376-2_21.

[50] Antoine Ligot, Ken Hasselmann, and Mauro Birattari. *AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines: supplementary material*. http://iridia.ulb.ac.be/supp/IridiaSupp2020-005/index.html. 2020.

[51] Antoine Ligot et al. *AutoMoDe, NEAT, and EvoStick: implementations for the e-puck robot in ARGoS3*. Tech. rep. TR/IRIDIA/2017-002. Brussels, Belgium: IRIDIA, Université Libre de Bruxelles, 2017.

[52] Antoine Ligot et al. « Towards an empirical practice in off-line fully-automatic design of robot swarms ». In: *IEEE Transactions on Evolutionary Computation* (2022). DOI: 10.1109/TEVC.2022.3144848.

[53] Hod Lipson and Jordan B. Pollack. « Automatic design and manufacture of robotic lifeforms ». In: *Nature* 406 (2000), pp. 974–978. DOI: 10.1038/35023115.

[54] Manuel López-Ibáñez et al. « The irace package: iterated racing for automatic algorithm configuration ». In: *Operations Research Perspectives* 3 (2016), pp. 43–58. DOI: 10.1016/j.orp.2016.09.002.

[55] Manuel López-Ibáñez et al. *The irace package: user guide*. IRIDIA, Université Libre de Bruxelles, Brussels, Belgium. Version 3.4.1. 2020.

[56] Fernando J. Mendiburu et al. « AutoMoDe-Mate: Automatic off-line design of spatially-organizing behaviors for robot swarms ». In: *Swarm and Evolutionary Computation* 74.4 (2022), pp. 101–118. DOI: https://doi.org/10.1016/j.swevo.2022.101118.

[57] Volodymyr Mnih et al. « Human-level control through deep reinforcement learning ». In: *Nature* 518.7540 (2015), pp. 529–533. DOI: https://doi.org/10.1038/nature14236.

[58] Francesco Mondada et al. « The e-puck, a robot designed for education in engineering ». In: *ROBOTICA 2009: Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*. Ed. by Paulo Gonçalves, Paulo Torres, and Carlos Alves. Castelo Branco, Portugal: Instituto Politécnico de Castelo Branco, 2009, pp. 59–65.

[59] Andrew Y. Ng and Stuart Russel. « Algorithms for inverse reinforcement learning ». In: *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*. Vol. 1. 2000, p. 2.

[60] Andrew Y. Ng et al. « Autonomous inverted helicopter flight via reinforcement learning ». In: *Experimental robotics IX*. Springer, 2006, pp. 363–372. DOI: https://doi.org/10.1007/11552246_35.

[61] William S. Noble. « What is a support vector machine? » In: *Nature biotechnology* 24.12 (2006), pp. 1565–1567. DOI: https://doi.org/10.1038/nbt1206-1565.

[62] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. First. A Bradford Book. Cambridge, MA, USA: MIT Press, 2000.

[63] Stefano Nolfi et al. « How to evolve autonomous robots: different approaches in evolutionary robotics ». In: *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*. Ed. by Rodney Allen Brooks and Pattie Maes. A Bradford Book. Cambridge, MA, USA: MIT Press, 1994, pp. 190–197. DOI: 10.7551/mitpress/1428.003.0023.

[64] Paolo Pagliuca and Stefano Nolfi. « Robust optimization through neuroevolution ». In: *PLOS ONE* 14.3 (2019), e0213193. DOI: 10.1371/journal.pone.0213193.

[65] Morgan Quigley et al. « ROS: an open-source Robot Operating System ». In: *2009 IEEE International Conference on Robotics and Automation (ICRA)*. Ed. by Kinugawa Kosuge. Vol. 3. 3.2. Piscataway, NJ, USA: IEEE, 2009, p. 5.

[66] Muhammad Salman, Antoine Ligot, and Mauro Birattari. « Concurrent design of control software and configuration of hardware for robot swarms under economic constraints ». In: *PeerJ Computer Science* 5 (2019), e221. DOI: 10.7717/peerj-cs.221.

[67] Muhammad Salman et al. « Phormica: photochromic pheromone release and detection system for stigmergic coordination in robot swarms ». In: *Frontiers in Robotics and AI* 7 (2020), p. 195. DOI: 10.3389/frobt.2020.591402.

[68] Gaëtan Spaey et al. « Comparison of different exploration schemes in the automatic modular design of robot swarms ». In: *Proceedings of the Reference AI & ML Conference for Belgium, Netherlands & Luxemburg, BNAIC/BENELEARN 2019*. Ed. by Katrien Beuls et al. Vol. 2491. Aachen, Germany: CEUR Workshop Proceedings, 2019.

[69] Gaëtan Spaey et al. « Evaluation of alternative exploration schemes in the automatic modular design of robot swarms ». In: *Artificial Intelligence and Machine Learning: BNAIC 2019, BENELEARN 2019*. Ed. by Bart Bogaerts et al. Vol. 1196. Communications in Computer and Information Science. Cham, Switzerland: Springer, 2020, pp. 18–33. DOI: 10.1007/978-3-030-65154-1_2.

[70] Vito Trianni. « Evolutionary robotics: model or design? » In: *Frontiers in Robotics and AI* 1 (2014), p. 13. DOI: 10.3389/frobt.2014.00013.

[71] Vito Trianni. *Evolutionary Swarm Robotics*. Berlin, Germany: Springer, 2008. DOI: 10.1007/978-3-540-77612-3.

[72] Yukiya Usui and Takaya Arita. « Situated and embodied evolution in collective evolutionary robotics ». In: *Proceedings of the 8th International Symposium on Artificial Life and Robotics*. Ed. by Masanori Sugisaka and Hiroshi Tanaka. Tokyo, Japan: International Society of Artificial Life and Robotics (ISAROB), 2003, pp. 212–215.

[73] Christopher Watkins and Peter Dayan. « Q-learning ». In: *Machine Learning* 8 (1992), pp. 279–292. DOI: https://doi.org/10.1007/BF00992698.

[74] Tian Yu et al. « Apply incremental evolution with CMA-NeuroES controller for a robust swarm robotics system ». In: *2014 Proceedings of the SICE Annual Conference (SICE)*. Tokyo, Japan: The Society of Instrument and Control Engineers (SICE), 2014, pp. 295–300. DOI: 10.1109/SICE.2014.6935195.

[75] Shao Zhifei and Er Meng Joo. « A survey of inverse reinforcement learning techniques ». In: *International Journal of Intelligent Computing and Cybernetics* 5.3 (2012), pp. 293–311. DOI: https://doi.org/10.1108/17563781211255862.