ECOLE
POLYTECHNIQUE
DE BRUXELLES

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Building an integrated framework for the automatic modular design of robot swarms

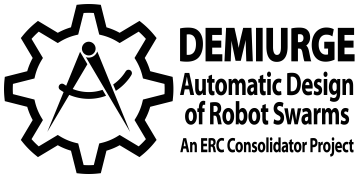
Mémoire présenté en vue de l'obtention du diplôme
d'Ingénieur Civil en informatique à finalité spécialisée

Ammar Hasan

Directeur
Professeur Mauro Birattari

Superviseur
David Garzón Ramos

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

Acknowledgments

I thank my thesis promoter, Prof. Mauro Birattari, introducing me to the world of swarm robotics and for trusting me with the interesting project that turned into my master thesis. I would also thank the help of the members of IRIDIA for the assistance during the development of this thesis. My sincere gratitude goes also to my family, who was there in every moment to support me. The completion of this master thesis could not have been possible without the assistance of my supervisor David Garzon Ramos. I thank him for his guidance and advice that carried me through all the stages of writing my thesis. I would also like to thank Jonas Kuckling for the time he gave me.

This master thesis has been developed under the framework of the DEMIURGE project, funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681872).

Contents

1 Introduction **1**

2 State of the art **4**

3 An integrated framework for the automatic design of robot swarms **8**
3.1 AutoMoDe: automatic modular design of robot swarms 9
3.1.1 Robot platform and simulation environment 9
3.1.2 Set of software modules 11
3.1.3 Control architecture 13
3.1.4 Design process 13
3.2 SwarmDesign 14
3.2.1 Account 15
3.2.2 Project creation 15
3.2.3 Mission selection 15
3.2.4 Manual design of robot swarms 15
3.2.5 Design budget 16
3.2.6 Automatic design of robot swarms 16
3.2.7 Monitoring and results of the design process 16
3.2.8 Testing the design 16
3.2.9 Help 17
3.3 Designing robot swarms 17

4 Experimental setup **18**
4.1 Experiment 1: software Validation 18
4.1.1 AutoMoDe-Chocolate 18
4.1.2 AutoMoDe-Maple 19
4.1.3 AutoMoDe-TuttiFrutti 20
4.1.4 AutoMoDe-Coconut 21
4.1.5 AutoMoDe-Arlequin 22
4.2 Experiment 2: combining AutoMode's subsets of modules and architectures 23
4.2.1 Comparing probabilistic finite-state machines (FSM) vs behavior trees (Bt) 24

5 Results and discussion **25**
5.1 Results of software Validation 25
5.2 Results of combining AutoMode's subsets of modules and architectures . . 28
5.2.1 AAC mission 28
5.2.2 Foraging mission 30

5.2.3 Aggregation mission 32
5.2.4 Grid exploration mission 34
5.2.5 Aggregation-xor mission 36
5.2.6 Experiment 2: General remark 38
5.3 Results of comparing FSM vs BT 39
5.3.1 AAC mission 39
5.3.2 Foraging mission 40
5.3.3 Aggregation mission 40
5.3.4 Grid exploration mission 41
5.3.5 Aggregation-xor mission 41
5.3.6 Experiment 3: General remark 42

6 Conclusions **43**

Abstract

For some decades now, swarm robotics has been attracting more and more attention of the scientific community. This is due to the unique set of properties that it offers. Indeed, swarm robotic systems are based on local interactions between the robots, and robots and the environment. Thanks to this paradigm, they are rely on a distributed control system. Swarm robotics systems are known to be robust, flexible, scalable and fault-tolerant. The main challenge of swarm robotics is the indirect nature of the design process. Indeed, to design a robot swarm the robots must be programmed individually. However, it is difficult to know beforehand what a robot should do so that a collective behavior emerges at the swarm level. Historically, the design of robot swarms has been mainly done manually with a trial-and-error approach. The lack of a general methodology for designing robot swarms is one of the reasons that hinder the applications in the real-world. To overcome this problem, a new approach has been introduced: the automatic modular design (AutoMoDe). AutoMoDe is an offline automatic design method that allows the design of robot swarms via optimization. AutoMode is a family of methods, that include variants such as AutoMode- Chocolate, AutoMoDe-Maple, AutoMoDe-Tuttifrutti, AutoMoDe-Coconut, AutoMoDe- Harlequin and others. In this thesis, a framework that allows to automatically design and evaluate robot swarms by integrating AutoMoDe method is proposed. The framework simplifies the design and the variety of the possible control software that can be produced. This framework combines the modules of the five variants of AutoMoDe named above. By integrating these methods, it enables the experimentation and the production of control software that was not possible before. It does so by enabling the combination of various specialized software modules and architectures. The framework is validated through independent experiments that reproduce previous results with AutoMoDe. And also, it is used to extend these experiments with new combinations of modules.

Keywords: AutoMoDe, automatic design, robot swarms.

Résumé

Depuis quelque décennies maintenant, la robotique en essaim attire de plus en plus l'attention de la communauté scientifique. Cela est dû à l'unique ensemble de propriétés qu'elle offre. En effet, les systèmes robotiques en essaim ne se reposent pas sur une unité de contrôle centralisée ou sur une infrastructure externe. Au contraire, les interactions locales entre les robots et les interactions locales entre les robots et l'environnement aboutissent plutôt à un système de contrôle distribué. Les systèmes robotiques en essaim sont connus pour être robustes, flexibles, évolutifs et ainsi que tolérants aux éventuelles pannes. Le défi principal de la robotique en essaim est la conception indirecte d'essaims de robots. Effectivement, pour concevoir l'essaim, il faut programmer les robots. Cependant, il est compliqué de discerner le comportement d'un robot de l'objectif de l'essaim. Jusqu'à présent, la conception d'essaims de robots se fait principalement manuellement avec une approche par essais et erreurs. Le manque d'une méthodologie générale pour la conception d'essaims de robots est la cause principale aux manques d'applications dans le monde réel de la robotique en essaim. Pour pallier ce problème, une nouvelle approche a été introduite. AutoMoDe est une méthode de conception automatique hors ligne qui permet de concevoir des essaims de robots. Ce dernier a plusieurs variants tels que AutoMoDe-Chocolate, AutoMoDe-Maple, AutoMoDe-Tuttifrutti, AutoMoDe-Coconut, AutoMoDe-Arlequin et d'autres.

Dans cette thèse, un framework permettant automatiquement de concevoir des essaims de robots sera développé. Il permettra de simplifier la conception et d'augmenter la qualité des essaims de robots ainsi produit. Ce framework combinera les modules des cinq variants d'AutoMoDe cités plus haut. Cela permettra aux framework de produire des essaims de robots jamais produits au part avant. La première expérience validera le bon fonctionnent du framework. La deuxième expérience permettra au framework de pouvoir combiner différents sous-ensembles de modules. Et la dernière expérience permettra d'analyser plus finement les performances de deux types d'architecture logicielle de contrôle utilisés dans les robots. Les résultats ont démontrés le bon fonctionnement du framework ainsi que l'amélioration de performances dû aux combinaisons des sous-ensembles de modules. Une présence importante des machines à états finis a été remarquée aux détriment des arbre des comportements.

Mots-clés : AutoMoDe, conception automatique, essaims de robots, Framework, Robotic en essaim.

Chapter 1

Introduction

Robots are not alone anymore they are coming in swarms. From homes to the rest of the world, robots are everywhere. Indeed, from some decades now we have seen a huge amount of research devoted to robotics [Yang et al., 2018]. This effort leads us to the creation of multiple types of robots for different tasks. There are a lot of variety of robots such as industrial robots, cleaning robots, car driver robots, robots that can interact with humans and many more. For almost every human task a robot can be realised. Their presence makes live easier by automating tasks.

Together, with the evolution of robotics a new field has emerged. Since several decades now, the field of swarm robotics is getting an increasing attention from the scientific community [Dorigo et al., 2021, Dorigo et al., 2020]. This field finds its origin not just in robotics but also in biology where the inspiration has been taken from social ants and animals [Dorigo et al., 2014]. It is now an engineering field of research. Indeed, swarm engineering's literature have shown several concepts and ideas that could be relevant to tackle real-world applications [Brambilla et al., 2013]. The day where the first industrial application of a swarm of robots will arise is not so far anymore [Dorigo et al., 2020].

Swarm robotics focuses on the design of robot swarms [Dorigo et al., 2014]. A swarm is composed of multiple relatively simple robots. Together, those robots have to accomplish a mission. The mission should be complicated enough so that it cannot be accomplished efficiently by just one individual robot. In this sense, something impossible for the individual robot is been made possible through the swarm. The swarm should operate without relying to any form of centralized control or external infrastructure. Indeed, a collective behavior is suppose to emerge from the swarm through the local interactions between the robots. The emergence of the collective behavior is also based on the interaction between the robots and the environment. Swarm intelligence principles [Dorigo and Birattari, 2007] are assets used to design robot swarm that are robust, flexible, scalable and fault tolerant. Those properties combined distinguish swarm robotics from other fields. It makes swarm robotics a promising candidate solution in tasks such as search and rescue, underwater or planetary exploration, and surveillance [Dorigo et al., 2021, Dorigo et al., 2020].

Swarm robotics is now becoming engineering field that is focused on the development of tools and methods to solve real world problems in a near future [Brambilla et al., 2013, Schranz et al., 2020, Hamann et al., 2020]. The main challenge of designing a robot swarm comes from the indirect nature of the design problem. One has to design an individual robot to obtain the desired swarm-level behavior [Francesca et al., 2014a, Birattari et al.,

2021]. Decomposing a global behavior into small interaction rules is not an easy task. For this reason, swarms are mostly designed by hand via an trial and error approach. Until now, swarm experiments have been conducted in a controled environment and do not have real-world applications [Schranz et al., 2020]. Real-world applications of robot swarm are prevented by its design methodology. Manually designing robot swarm is costly, time consuming, it is not predictable and it depends highly on the quality of the designer. To be more reliable, a general design methodology should be defined. It should give some guarantees on the performance of the swarm and it should not rely on any human expert [Francesca and Birattari, 2016, Garzón Ramos et al., 2021].

AutoMoDe stands for Automatic Modular Design [Francesca et al., 2014b, Birattari et al., 2021]. It is a novel approach to design robot swarms. This design method generates control software for robots by assembling and fine-tuning preexisting parametric modules into a control architecthre prior the deployment of the robots. To do so, an optimization process samples candidate control software, evaluates them in simulation, and selects those that perform significatively better than others. In this sense, it is an off-line approach. AutoMode is a promising approach to design robot swarms. It has been proved that this designing method can efficiently deal with the reality gap and that it has outperform the design of some human experts [Francesca et al., 2014b]. AutoMode is has several variants such as AutoMoDe-TuttiFrutti [Garzón Ramos and Birattari, 2020], AutoMode-Maple [Ligot et al., 2020b], AutoMode-Coconut [Spaey et al., 2020] and many more. All flavours share their modular nature, but they vary with respect their subset of parametric modules, the robot capabilities, the control architecture they rely on, or the optimization algorithm that conducts the design process. The concept of automatic modular design, and therefore the AutoMoDe family, has grown largely and to this day it has been proven to work under various design problems.

This master thesis brings together a set of the various design methods proposed in the AutoMoDe family under a unified framework. So far, all methods in the AutoMoDe family have been used independently and they are specialized in addressing different classes of missions. However, it remains still unknown whether the modularity of the individual methods makes them suitable to operate altogether. Several questions arise from this idea: are all methods compatible with each other? would the optimization algorithm be able to successfully combine the modules of different methods? are there software architectures that are not compatible with some of the modules? To address this questions, in this work a general methodology for designing robot swarms will be put in place in the form of a framework for the automatic design of robot swarms. This framework will rely on five implementations of methods of the AutoMoDe family in order to produce the control software for the robots. In this work, it is consider the subset of modules, robot platform, and control architecture of AutoMoDe-Chocolate [Francesca et al., 2015], AutoMoDe-TuttiFrutti [Garzón Ramos and Birattari, 2020], AutoMode-Maple [Ligot et al., 2020b], AutoMode-Coconut [Spaey et al., 2020] and AutoMode-Arlequin [Ligot et al., 2020a].

Two experiments are conducted to evaluate the proposed framework and to investigate the aforementioned questions. First, each independent subset of modules is tested within the original experiment conditions with which it was proposed in the literature. That is, each AutoMoDe method in the framework is tested independently with a single mission extracted from its original publication, while following the same experimental protocol (optimization budget, configuration of the design process, etc). This first experiment aims to determine if the integration has been done correctly and that it is possible to

reproduce previous results. In a second experiment, the framework (in full) is used to design control software for the five missions extracted from the literature. In these sense, the design process has freedom to select among the subset of modules of each AutoMoDe method included, and also between the control architectures available. This second experiment aims to determine if the modules are compatible with each other and if the design process can select new combinations of them to address missions proposed in the literature. Finally, as an extention to the second experiment, the impact of selecting an appropiate control architecture is investigated. Most methods in the AutoMoDe family produce control software in the form of probabilistic finite-state machines [Francesca et al., 2015, Garzón Ramos and Birattari, 2020, Spaey et al., 2020, Ligot et al., 2020a], and few of them produce them as behavior trees [Ligot et al., 2020b]. Little research has been done to evaluate if the most recent set of modules produced for finite-state machines are suitable to be used in behavior trees. Thanks to the proposed framework, this research can now be conducted and it is part of the contribution of the thesis.

The results obtained through the framework will be analyzed and compared to the results of other automatic design methods. It has been shown that this framework is able to reproduce the work of other automatic design methods. It can also combine different subsets of modules to generate control software. Those control software are successfully performing the mission for which they have been designed. In some cases a significant increase in performance is observed. The last experiment has shown that finite state machines are producing better collective behaviour than behaviour trees.

This thesis is structure as follows. Chapter 2 will cover the state of the art of the automatic design of robot swarms. The third chapter is about the development of SwarmDesign a framework for the automatic design of swarms of robots. The fourth chapter is describing the experimental setup of the three experiments realised. The fifth chapter contains the results of those experiments and a discussion of it. The last chapter is dedicated to the conclusion of this thesis.

Chapter 2

State of the art

Evolutionary robotics [Trianni, 2008], and in particular neuroevolution, is the typical approach to the design of autonomous robots swarms. It uses an evolutionary algorithm to generate the control software for the robots. Largely inspired in biology processes, this method generates and fine-tunes artificial neural networks to control the behavior of robots. This neurocontroller uses the readings of the robot's sensors as inputs, and outputs a signal used to send motor commands. The control software generated by this method are not human readable and more importantly they are suffering from the reality gap [Hasselmann et al., 2021]—that is, the unavoidable differences between the simulation and reality. Indeed, there is a significant drop of performance when robots go from simulation to reality [Floreano et al., 2008, Ligot and Birattari, 2020]. In the context of swarm robotics, the reality gap issue has to be considered and efficiently addressed. By doing so, a first step towards the real world application of robot swarms will be done.

AutoMoDe is an automatic modular design method. This method generates collective behavior of robot swarms by combining and fine tuning preexisting parametric modules. The control software generated by this method is typically in a form of a probabilistic final state machine, although more recent studies have proposed the design of other architectures like behavior trees. AutoMoDe methods have been demonstranted to be more robust to the reality gap than those in the the evolutionary approach [Hasselmann et al., 2021]. This point makes AutoMoDe a promising approach for the design of robot swarms. AutoMode-Vanilla [Francesca et al., 2014b] is the proof-of-concept version of this method. It has been shown that, for tasks such as foraging and aggregation AutoMoDe-Vanilla generates better results than the evolutionary approach when the control software is deployed in physical robots. The performance of AutoMode-Vanilla in simulation and in reality are comparable while the evolutionnary aproach is clearly suffering from the reality gap issue [Francesca et al., 2014b].

In the introduction of AutoMode-Vanilla, two studies were conducted. In the first study, a comparison between 4 different types of design methods has been made. Those methods are AutoMoDe-Vanilla, EvoStick—an evolutionary method, C-Human and U-human. The first two design methods are automatic methods, while the last two are methods to design the control software manually. In C-Human, the human expert is constrained to use AutoMoDe-Vanilla's modules and architecture to design the control software. This is not the case in U-Human where the human expert is unconstrained in the programing of the robots. The comparison between those methods has been done on 5 different missions. Under the experimental conditions considered, the results of the

first study show that AutoMoDe-Vanilla outperform EvoStick. But AutoMoDe-Vanilla is outperformed by C-Human. Since AutoMoDe-Vanilla and C-Human share the same software modules, the reasonable assumption was that C-Human could combine them better than AutoMoDe-Vanilla. This results suggest that the optimization algorithm of AutoMoDe-Vanilla—F-Race [Birattari et al., 2002a]—although sufficient, still lacked performance with respect to the hability of human designers.

AutoMoDe-Chocolate [Francesca et al., 2015] is an improved version of AutoMoDe-Vanilla. Indeed the inefficient optimization algorithm of AutoMode-Vanilla has been replaced by a more robust algorithm namely Iterated F-Race [Balaprakash et al., 2007]. The second study in the paper is a comparison between AutomMoDe-Chocolate, C-human and AutoMoDe-Vanilla. The comparison has been done on the same fives missions as the first study. The results of the second study show that AutoMoDe-Vanilla is outperformed by C-Human as expected but C-human is outperformed by AutoMoDe-Chocolate. AutoMode-Chocolate becomes the first automatic method to design robot swarms better than human experts under the considered conditions. And also, it is the baseline to which many of the new methods in the AutoMoDe family are compared.

AutoMoDe-Maple [Ligot et al., 2020b] is a variant of AutoMoDe-Chocolate. Instead of generating control software by combining parametric modules into finite state machines, AutoMoDe-maple combines those modules into behavior trees. A study compared the performances of three automatic design methods namely AutoMode-Chocolate, AutoMoDe-Maple and EvoStick. Those design methods have been run for two different missions such as foraging and aggregation. It appears that AutoMoDe-Chocolate and AutoMode-Maple had a similar performance. Both have successfully cross the reality gap issue and both have outperformed the traditional evolutionary approach. Indeed, EvoStick suffered considerably more to the reality gap than the two other approaches.

AutoMoDe-TuttiFrutti [Garzón Ramos and Birattari, 2020] is another variant of AutoMoDe-Chocolate. This time, its peculiarity is that it can generate control software that are sensitive to colored lights. Indeed, by enhancing the ability of the e-puck robots so that they can perceive and display colors using their RGB LEDs and omnidirectional camera. This method can design collective behavior based on information propagated through light. The RGB LEDs of the robots can be used to communicate whit other robots, to navigate or to handle events. AutoMode-TuttiFrutti has been assessed on missions where the environments have some colored information and on which the robots should react. The results show that AutoMoDe-TuttiFrutti is able to generate efficient control software which can react and propagate colored information.

AutoMoDe-Coconut [Spaey et al., 2020] is another variant of AutoMoDe-Chocolate. The difference between both design methods relies on the exploration scheme of the robot swarms. Indeed, AutoMode-Chocolate has a basic exploration scheme such as random walk while AutoMoDe-Coconut is providing multiple configurable ones. By doing so, AutoMoDe-Coconut generates robot swarms which are exploring the environment in different manners. In order to study the impact of the exploration schemes on the performance of the swarm some experiments have been conducted. Swarms of robots have been designed from missions such as Aggregation, Foraging and Grid exploration. Those experiments have been done in workspaces that were both bounded and unbounded. A comparison has been done between the performance of AutoMoDe-Coconut and AutoMoDe-Chocolate. The results show that both design methods are successfully building control software for those missions. The swarm level performance of both design methods are sim-

Table 2.1: AutoMoDe family of methods for the automatic design of robot swarms

AutoMoDe family			
Method	Year	Peculiarity	Publication
Vanilla	2014	Proof of concept	[Francesca et al., 2014b]
Chocolate	2015	Working concept	[Francesca et al., 2015]
Gianduja	2018	One message communicate	[Hasselmann and Birattari, 2020]
Maple	2018	Behavior Trees	[Ligot et al., 2020b]
IcePop	2019	Simulated Annealing	[Kuckling et al., 2020b]
Waffle	2019	Hardware configuration	[Salman et al., 2019]
TuttiFrutti	2020	Display and Perceive Colors	[Ramos and Birattari, 2020]
Coconut	2020	Exploration Schemes	[Spaey et al., 2020]
Arlequin	2020	Neural Networks	[Ligot et al., 2020a]
Mate	2022	Spacial organization	[Mendiburu et al., 2022]
Cedrata	2022	Behavior Trees and Communication Capabilities	[Kuckling et al., 2022]

ilar. This shows that different exploration schemes can be used to design robot swarms. But the collective behavior of the swarm is more correlated to the local interaction between robots than to the exploration schemes.

AutoMoDe-Arlequin [Ligot et al., 2020a] is another variant of AutoMoDe-Chocolate. This variant is able to transform a traditional evolutionary approach into an AutoMoDe approach. Indeed, this variant produces control software by automatically combining neural-network modules into probabilistic finite-state machines. Those modules are generated by EvoStick a neuro-evolutionary approach originally introduced as a baseline with AutoMoDe-Vanilla [Francesca et al., 2014b]. AutoMoDe-Arlequin is thus able to design robot swarms controlled by neural network combined in a modular manner. In order to assess its performance this variant has been compared to AutoMoDe-Chocolate and to EvoStick. This comparison has been done on two missions such as Foraging and Aggregation. The results show that Evostick considerably suffers from the reality gap. AutoMode-Arlequin is satisfactorily crossing the reality gap and has a better performance than EvoStick. AutoMoDe-Chocolate is efficiently crossing the reality gap and has a better performance than its variant. AutoMoDe-Arlequin is a promising approach, it has reduced the reality gap issue and enhance the performance of the evolutionary approach.

AutoMoDe has also some other variants such as AutoMoDe-Gianduja [Hasselmann and Birattari, 2020], AutoMoDe-Waffle [Salman et al., 2019], AutoMoDe-Mate [Mendiburu et al., 2022] and AutoMoDe-IcePop [Kuckling et al., 2020b]. Each flavour has its own peculiarity. AutoMode is a family of automatic offline design methods. Which modules are conceived in a mission agnostic manner. Those modules are general enough to be used in different missions. By combining them, AutoMoDe is able to generate control software with a good performance and that can cross the reality gap satisfactorily. The current known methods from the AutoMoDe family are listed in Table 2.1.

In this thesis a framework for designing swarms of robots will be developed. This framework will automatically generate control software for a given mission. In order to do so, the framework needs to rely on an automatic design method. A new design method

has been developed for that purpose. This new design method is part of the AutoMoDe family. It uses subsets of modules of 5 different AutoMoDe design methods namely AutoMoDe-Chocolate, AutoMoDe-Maple, AutoMoDe-TuttiFrutti, AutoMoDe-Coconut and AutoMoDe-Arlequin. Those 5 subsets have been chosen for the ability that they bring to the robots. The subset of modules of AutoMoDe-Chocolate have been chosen since AutoMoDe-Chocolate is the baseline and state of the art automatic design method. The subset of modules of AutoMode-Maple have been chosen to allow the new design method to produce control software of two different architecture, probabilistic finite state machine and behavior tree. The subset of AutoMoDe-TuttiFrutti are not just allowing the communication between robots trough colored light but they are also allowing a communication between the robots and a colored light emitting environment. The modules of AutoMoDe-Coconut can help to choose the best exploration schemes in various arenas. AutoMode-Arlequin's subset is the one who is able to have the closest behavior to the one of the evolutionary robotics approach.

Chapter 3

An integrated framework for the automatic design of robot swarms

Real-world applications of robot swarms are prevented by the lack of an engineering methodology of designing swarms of robots. The optimization-based design process proposed with AutoMoDe is a step further to facilitate the deployment of robot swarms. AutoMoDe is a family of offline automatic design methods which are proven to be robust against the reality gap issue. Those methods can design robot swarms that are successfully fulfilling a class of missions. From an industrial point of view, using AutoMoDe's design methods is not simple. In order to design a swarm, a first step is to choose one design method from the AutoMoDe family. A first challenge is choosing the most adequate one. Although possible in practice, this selection requires large domain knowledge. A better approach could possibly be to unify the design methods, and let the optimization process to select among them. A second issue to be addressed is the technical challenge to prepare and deploy instances of the AutoMoDe family. The current implementations require manual setup, experiment preparation, results retrieving, and deployment of the robots. This makes the automatic design methods less accessible and difficulties extremely the reproduction of the experiments.

The typical preparations of an AutoMoDe method proceed as follows. When a design method is chosen, it should be first install. Having a working AutoMoDe installation can be very hard. It requires a working installation of ARGoS and some other dependencies. Irace, the optimization algorithm needs also to be installed. Once the installations have been done, the design process can begin. To do so, a mission should be provided. A mission is composed of two different files, an argos file and a loopfunction file. Those files are linked, they need to be placed in a specific manner—whose typically known only by the developers. The second file must be compiled and installed to be used by ARGoS during the design process. Before running the design process, some parameters need to be set such as the design budget. Those steps requires some knowledge about the internal functioning of AutoMoDe. The design process can be run on a local computer but it is more efficiently done on a cluster. Those two different manners to run the design process have not exactly the same procedure. All those steps make the design of swarms of robots complicated. To entirely take advantage of an automatic design method, a framework needs to be defined. This framework will make the design of robot swarms more practical while facilitating the reproduction and possibly the extention of the experiments.

In the following, it is detailed first the components of the automatic modular design

process followed in the development presented in the thesis. Afterward, there is an introduction to SwarmDesign, the framework to automatically design control software for robot swarms that integrates the various methods from the AutoMoDe family.

3.1 AutoMoDe: automatic modular design of robot swarms

There are four main components to the automatic design of robot swarms with the AutoMoDe approach: the robot platform, the set of software modules, the control architecture, and the design process. The robot platform defines the robot for which an specific implementation of AutoMoDe can design control software. Indeed, by defining the target robot it is possible to define the set of capabilities that individual robots in the swarm can have. The second important component is the set of software modules from which an AutoMoDe method can combine and create the modular control software. Different implementations of AutoMoDe have defined sets of specialized modules to address particular classes of missions. For example, as observed in the studies with AutoMoDe-TuttiFrutti [Garzón Ramos and Birattari, 2020], the modules are specialized to perceice, produce, and react to color signals—capabilities that are selected according to the robot that is used. The third component is the control architecture into which the software modules are assembled. Different control architectures can provide different design patterns and, therefore, enable the design of collective behaviors from which different properties emerge [Ligot et al., 2020b,Kuckling et al., 2022]. The two architectures that are currently used in the AutoMoDe family are probabilistic finite-state machines and behavior trees. Finally, the last main component in an AutoMoDe method is the design process itself. The design process is driven by an optimization algorithm that fine-tunes and combines the software modules to produce the control software for the robots. These components are further discussed in the following.

3.1.1 Robot platform and simulation environment

All methods in the AutoMoDe family have been conceived to produce control software for an extended version of the e-puck robot [Mondada et al., 2009, Garattoni et al., 2015]. The e-puck is a mobile robot originally conceived to be used in educational robotics. The robot is two-wheeled platform with differential driving, it is about 3.5 cm tall, and it has a diameter of about 5.5 cm. The e-puck is possible the most used robot in swarm robotics research [Dorigo et al., 2021] due to its small form factor and because it allows for extension modules that enlarge the capabilities of the robot—for example with direct [Gutiérrez et al., 2009] or indirect communication modules [Salman et al., 2020]. Although methods in the AutoMoDe family typically target physical robots, the work conducted in thesis targets only a simulated version of the e-puck. However, the software libraries [Pinciroli et al., 2012, Garattoni et al., 2015] that has been used during the simulations are the same as those that are used during the deployment of physical robots. This gives a certain degree of certainty that, in future work, the control software produced with the SwarmDesign can be ported to the physical robot without requiring further modification.

SwarmDesign has been conceibed to operate with a model of the e-epuck that integrates the sensors and actuators defined in the specification of AutoMoDe-Chocolate,

Table 3.1: Reference model of the e-puck considered in SwarmDesign

Input	**Value**	**Description**
$prox_{i\in\{1,...,8\}}$	[0, 1]	reading of proximity sensor i
$light_{i\in\{1,...,8\}}$	[0, 1]	reading of light sensor i
$gnd_{j\in\{1,...,3\}}$	$\{black, gray, white\}$	reading of ground sensor j
n	$\{0, ..., 20\}$	number of neighboring robots detected
V_n	$([0.5, 20]; [0, 2]\pi rad)$	their relative aggregate position
$cam_{c\in\{R,G,B,C,M,Y\}}$	$\{yes, no\}$	colors perceived
$V_{c\in\{R,G,B,C,M,Y\}}$	$\{1.0; [0, 2])\pi rad$	their relative aggregate direction
Output	**Value**	**Description**
$v_{k\in\{l,r\}}$	$[-0.12, 0.12]m/s$	target linear wheel velocity
$LEDs$	$\{\emptyset, C, M, Y\}$	color displayed by the LEDs

AutoMoDe-TuttiFrutti, AutoMode-Maple, AutoMode-Coconut and AutoMode-Arlequin. The methods of the AutoMoDe family characterize the capabilities of the robot platform by using specific reference models (RM). The *reference model* [Francesca and Birattari, 2016] is a concept that allows for a formal definition and specification of the characteristics of the platform for which it is possible to design control software with a particular methods. The reference model defines the characteristics of the robot platform both in simulation and reality. That is, it makes explicit the inputs and outputs of the control software, and maps their relationship with the hardware of the robot. The methods of the AutoMoDe family have considered robots with a variety of reference models for the e-puck [Hasselmann et al., 2018]. AutoMoDe-Chocolate, AutoMode-Maple, AutoMode-Coconut and AutoMode-Arlequin operate on the basis of the reference model RM1.2. AutoMoDe-TuttiFrutti operates on the basis of the reference model RM3.0.

The framework proposed in this thesis unifies the software produced in the aforementioned design methods. For this reason, the version of the e-puck for which SwarmDesign can produce control software is defined by a combination of the reference models RM1.2 and RM3.0. Instead of allowing for an individual selection of reference models during the design process, the approach followed in SwarmDesign is to consider a single version of the e-puck that contains all the hardware capabilities defined in the past. Table 3.1 defines the reference model of the e-puck considered in SwarmDesign. The version of the e-puck considered has 8 proximity sensors to determine the presence of nearby (3 cm) objects, 8 light sensors that sense the intensity of ambient light, three ground sensors that sense the color of the floor in gray scale basis, a range-and-bearing board that allows the robot to locate other robots in a 50 cm neighborhood, an omni-directional camera to perceive color signals in a 50 cm neighborhood, RGB LEDs to emit color signals, and two wheels that can be controled independently. A detailed description of the most relevant reference models conceived alongside AutoMoDe methods is available in [Hasselmann et al., 2018].

Simulation environment

The implementation of the methods in the AutoMoDe family considered in this thesis have been developed for ARGoS3 [Pinciroli et al., 2012]—a multi-robot multi-physics simulator specialized for swarm robotics. ARGoS is a free and open-source simulator that, in comparison to other robotics simulators such as Gazebo [Koenig and Howard,

2004] and V-Rep [Rohmer et al., 2013], it can simulate a large number of robots with a relatively lower computational power. This is a fundamental property for the research on the automatic design of orobot swarms. In the methods of the AutoMoDe family, the control software is produced with large number of simulations—ranging from tens of thousands to the hundres of thousands. ARGoS allow for the execution of those number of simulations in a relatively short time when used a computational cluster—few hours for the experiments presented in this thesis.

3.1.2 Set of software modules

As mentioned before, the main characteristic of methods in the AutoMoDe family is that they produce control software by fine-tuning and combining parametric software modules. The software modules in each method have been conceived in a mission agnostic manner. That is, they are meant to be used in a number of missions and not only in one of them. The class of missions that a robot swarm can execute is characterized by the capabilities of robot platform to be used. In this thesis, the sets of modules of AutoMoDe-Chocolate, AutoMoDe-TuttiFrutti, AutoMode-Maple, AutoMode-Coconut and AutoMode-Arlequin have been unified in a superset. By doing so, it is expected that SwarmDesign can address in a unified manner all the classes of missions for which the aforementioned methods were conceived. In this sense, the modules to be integrated are not anymore restricted to be a specializtion of AutoMoDe for a certain class of missions. They are rather all modules available that can be combined according the hardware capabilities of the e-puck.

In the AutoMoDe methods, there are two types of software modules: low-level behaviors and condition transitions [Birattari et al., 2021]. The low-level behaviors are software modules that enable actions that the robot can execute and the condition transitions are software modules with events that are triggered by changes in the sensors of the robot. The transitions conditions are used to swicht between the execution of low-level behaviors at runtime. SwarmDesign can combine the software modules in two architectures: probabilistic finite-state machines and behavior trees. Table 3.2 lists all the modules integrated in SwarmDesign—which correspond to the implementations of the five AutoMoDe flavours considered during the creation of the framework. Table 3.3 describes the basic functioning of the modules. Extended descriptions and parametrization of each of the corresponding modules are available in the original publications for each method—AutoMoDe-Chocolate [Francesca et al., 2015], AutoMoDe-TuttiFrutti [Garzón Ramos and Birattari, 2020], AutoMode-Maple [Ligot et al., 2020b], AutoMode-Coconut [Spaey et al., 2020] and AutoMode-Arlequin [Ligot et al., 2020a].

The challenge when enable the combination of different software modules is that each software module might exist, not exists, or exists as a variation in the implementation of the different AutoMoDe methods. That is, for example, modules GoToColor and GoAwayFromColor are low-level behaviors that only exist on AutoMoDe-TuttiFrutti, and therefore, they do not overlap with modules in the other flavors. On the contrary, the modules originally introduced with AutoMoDe-Chocolate are the same as those used in AutoMoDe-Maple. A third case also occurs with modules that are similar for the most part but that have extra parameters to be tuned during the design process. For example, the implementation of Stop in AutoMoDe-Chocolate, AutoMoDe-Maple and AutoMoDe-Coconut is the same. Although the implementation of Stop in AutoMoDe-TuttiFrutti is also the same for the most part, in AutoMoDe-TuttiFrutti there is an extra parameter

Table 3.2: Index of the software modules integrated in SwarmDesign. The Table shows the IDs for each method, low-level behavior and condition transition.

Indexation of the set of modules					
Method	ID	Behavior	ID	Condition	ID
Chocolate	1	Exploration	1	Black floor	0
TuttiFrutti	2	Stop	2	Grey floor	1
Coconut	3	Photoaxis	3	White floor	2
Maple	4	Anti-Photoaxis	4	Neighbors count	3
Arlequin	5	Attraction	5	Inverted neighbors count	4
		Repulsion	6	Fixed probability	5
		Go to color	8	Color probability	7
		Go away color	9		

Table 3.3: Description of the software modules integrated in SwarmDesign. The table describes the general functioning of the modules across all methods. * Alongside the described behavior, the modules integrate the specific capabilities endowed in their original AutoMoDe methods. AutoMoDe-TuttiFrutti modules also allow for setting the emition of light signals with the LEDs of the robot. AutoMoDe-Coconut modules allow for the selection of independent exploration schemes for the movement of the robots.

Low-level behaviors *	Description
Exploration	movement by random-walk
Stop	standing still position
Photoaxis	movement towards ambient light sources
Anti-Photoaxis	movement away from ambient light sources
Attraction	movement towards neighboring peers
Repulsion	movement away from neighboring peers
Go to color	movement towards specific color signals
Go away color	movement away from specific color signals
Transition condition *	**Description**
Black floor	detected black floor
Grey floor	detected gray floor
White floor	detected white floor
Neighbors count	neighboring peers over a certain threshold
Inverted neighbors count	neighboring peers less than a certain threshold
Fixed probability	transition with a fixed probability
Color probability	detected light signal with a specific color

that allows to set the colors of the LEDs of the robots. In order to maintain the definition of the modules as they were presented in the introduction of the design methods, the framework allows for the existence of different instances of the same module but with different implementations. This is achieved by indexing the software modules with a two digit identifier (ID)—see Table 3.2. The first digit denotes the AutoMoDe method from which the implementation is taken, and the second digit indicates the specific software module. The modules that only exist in a single AutoMoDe method are only indexed with the ID that corresponds that particular implementation.

3.1.3 Control architecture

The control architecture provides the infrastructure into which the software modules are combined to produce control software for the robots. The two control architectures explored so far in the AutoMoDe family are probabilistic finite-state machines and behavior trees. Probabilistic finithe-state machines are a control architecture composed by nodes and transitions. In AutoMoDe methods, the nodes are the low-level behaviors and the transitions are the conditions that trigger behavior changes. The transitions are triggered with a certain probability when the condition that applies is reached—the change typically happens by stimuli registered in the sensors of the robot. This control architecture is characterized because it allows for multiple transitions modules departing and arriving to the same node. In this cases, the transition that will be triggered is the first that satisfy the transition condition. Behaviors trees are substantially different from the probabilistic finite-state machines. The architecture takes the form of a tree in which the leaves represent the low-level behaviors and the condition transitions. In the behavior trees there exist intermediate nodes that control the order with which the possible many leafs of the tree are executed. In this sense, the execution of the control software happens sequentially across all the leafs.

Finite-machines and behavior trees do not necessarily can represent the same control software. In particular, as the behavior trees are executed sequentially, there is no possible representation that allow for multiple incoming and ongoing transition conditions from a low-level behavior. This is probably the most important architectural difference for the AutoMoDe methods. There is no clear advantage yet that defines whether one architecture is more appropriate than the other for the automatic design of robot swarms [Kuckling et al., 2020a]. For this reason, SwarmDesign embeds the two architectures: this will allow for further investigation on the impact of the selection of the control architecture. As will be shown in the experiment section, in this thesis, for the first time it is the optimization process which decides on the control architecture to be used to build the control software.

3.1.4 Design process

Swarm design produces control software following the same procedure originally conceived for AutoMoDe-Chocolate [Francesca et al., 2015], which has been also used in the realization of robot swarms with the other AutoMoDe methods considered in the thesis. SwarmDesign belongs to the automatic design approach, and therefore, to operate it, the problem of designing the control software is cast into an optimization process. SwarmDesign starts from the specification of a mission that a robot swarm must perform—which considers the scenario where the robots operate and a performance metric expressed in the

form of an objective function to be optimized. The optimization algorithm iterative samples combinations of the software modules, tunes their parameters, and assembles them in a control architecture. In this sense, the control software produced with SwarmDesign is the result of an optimization process that maximizes the performance of the swarm according to the mission-specific objective function. The control software produced is passed to all the robots without being modified.

The architecture of the control software that SwarmDesign produces is restricted to the same number of low-level behaviors and condition transitions defined in the original publications of AutoMoDe-Chocolate and AutoMoDe-Maple, see AutoMoDe-Chocolate [Francesca et al., 2015], AutoMode-Maple [Ligot et al., 2020b]. The control software is produced offline. That is, the modules and their parameters are selected prior the deployment of the robot swarm on its operational environment. In AutoMoDe studies, the control software is produced first in simulation and is tested after in physical robots. In this thesis, the assessment is also conducted in simulation. The process to automatically design robot swarms with SwarmDesign is conducted with Iterated F-Race [López-Ibáñez et al., 2016]: a multi-purpose optimization algorithm frequently used in the automatic configuration of algorithms, and that is an extentsion of F-Race [Birattari et al., 2002b]. Iterated F-Race looks into the design space for possible configurations of the control software. During the design process, it evaluates the finite-state machines or behavior trees with simulations in ARGoS3. The duration of the optimization process is constrained by a pre-defined budget of simulations that Iterated F-race can use to produce a solution. The design process ends when the optimziation algorithm runs out of its simulations budget and it returns the best configuration so far.

3.2 SwarmDesign

SwarmDesign is a framework for designing robot swarms. This framework is developed to ease the development of swarms of robots. SwarmDesign embedds a command line interface written in Python. It is conceived to run on the Ubuntu 20.04.3 operating system. It provides all the necessary commands to design collective behaviors for a class of missions. To do so, swarmDesign is relying on a new automatic design method. This new design method uses the subset of modules of the five following design methods: AutoMoDe-Chocolate, AutoMoDe-Maple, AutoMoDe-TuttiFrutti, AutoMoDe-Coconut and AutoMoDe-Arlequin. The original implementation that integrates the design methods is extended in this thesis from previous work developed in IRIDIA, the Artificial Intelligence lab from the Université Libre de Bruxelles. SwarmDesign provides to the user a general procedure to automatically design robot swarms in simulation. It automatically handles all the installation that a design process needs in order to be executed. The execution of the design process can be done on a local computer or on a cluster. It provides several missions for which the design process can be run. Design monitoring is also made possible through this framework. SwarmDesign enables the designer to express himself by providing a interface where he can manually design swarms of robots and observe the swarm in action. Due to its automatic design method, SwarmDesign is able to generate control software of different architecture such as probabilistic finite-state machines or behavior trees. The control software produced, as it combines the modules of different flavors of AutoMoDe, can design swarms of robots that are sensible to colored light, they can have a special exploration scheme or/and they can have some neural

networks modules. In the following sections the different parts of the framework will be explained.

3.2.1 Account

In order to use the framework, a user first needs to create an account. SwarmDesign uses a relational database where the information of the users are stored in a table called usersdata. This table is storing information such as username, password and space. The last element is referring to the workspace of the user. The workspace of a user is the path of the folder where we can find all his local projects. The commands related to the account management are the following: sign-up, sign-in, exit and show-users.

3.2.2 Project creation

After signing into his account, the user is able to create a project. Users are allowed to create as many projects as wanted. The project information are stored in the projects table of the relational database. This table stores information such as the project's name, its owner, the path where it is located and a Boolean variable which indicates if the project is on the cluster or not. Another table is also used to store cluster connection information. A user can decide to create a project on his local computer or on a cluster. By creating a project a user is installing all the necessary material needed to run a design process. More specifically, it means that ARGoS3 beta48 will be locally installed. The plugins ARGoS-Epuck and ARGoS-Arena will be installed. The package demiurge-epuck-dao and experiments-loop-functions will be added. The new design method will be compiled. If the project is on a local computer the AutoMoDe Visualization Tool [Kuckling et al., 2021] will be installed. And finally, the version 2.2 of Irace will be installed. The commands related to project management are the following: newProject, active-project, show-projects and select-project.

3.2.3 Mission selection

Once the project has been created, the user is able to chose a mission. SwarmDesign has for instance nine different missions. This set of missions contains missions such as Foraging, Aggragation-Xor, Shelter with constrained access (SCA), Coverage with forbidden areas (CFA), Grid exploration and more. This set of missions can be manually extended by adding more missions to the experiments-loop-functions repository. Once a user has made his choice, the selected mission path is given to the AutoMoDe-Editor. The framework will also prepare the selected mission for the design process by automatically disabling the visualization of the mission. To select a mission, the user must use the select-mission command which will list all available missions and request for the number of the chosen mission.

3.2.4 Manual design of robot swarms

After the selection of a mission and if the project is installed on the local computer, the user is able to manually design swarms of robots. By using the following command line 'swarmDesign run-exp', a web editor will be running on http://localhost:8080 in a browser. From there, the user is able to manually design control software for robot swarms. He can

create control software of different architectures such as behavior trees or probabilistic finite state machines. He has at his disposal five subsets of modules from the AutoMoDe family. By clicking on the exec button on the web editor, the user can observe the created control software in action in the selected mission.

3.2.5 Design budget

Before running a design process the user needs to set the design budget. The design budget represents the number of execution of ARGoS that can be used to produce the control software. It can be set by the following command, where the number argument represents the desired budget: 'swarmDesign set-budget number'.

3.2.6 Automatic design of robot swarms

Once the mission is selected, the user is able to run a design process to automatically generate the control software. The user can also customize his design process. Indeed, the automatic design method used by this framework is combining modules from five different subsets of modules. A user can decide which subsets of modules can be used by the automatic design method to design the collective behavior. This is done by providing options to the design command such as –fsm , –bt, –chocolate, –maple, –coconut, –tuttifrutti, and –arlequin. For instance in order to only used the modules of chocolate and the one of coconut, the following command must be used :
'swarmDesign run-design –fsm –chocolate –maple'.

3.2.7 Monitoring and results of the design process

A user can monitor and get the results of his design process. To do so, the 'status' command from swarmDesign must be used. While the design is running, this command will give to the user some relevant information about the design process. The information provided are the total number of iterations, the current iteration, the number of experiments used so far, the remaining budget, the current budget, the number of configurations and a string representation of the best configuration so far. Those information are taken from the Rdata provided by Irace. A complete details of the design process can also be requested by using the option '–complete'. In this case, the whole Irace output file will be displayed.

3.2.8 Testing the design

A user can test the control software generated by the automatic design method. For that, he needs to save the string representing the control software obtained from the design process. If the initial project is on the cluster, he needs to create a new local project. In this new local project, he needs to set the mission for which the control software has been designed. Then the user needs to run the web editor by using the 'swarmDesign run-exp' command. From there he needs to paste the representation of the control software and click on the exec button. A window with the ARGoS simulator will open where the user will be able to run the mission with the designed collective behavior.

3.2.9 Help

SwarmDesign is command line interface. Some commands are not directly available. For instance, a user needs to sign in before creating a project. This framework has several commands, each command has its own set of arguments and options. To ease the utilisation of SwarmDesign, the framework is providing documentation for all its commands. The user will be provided with more information about a command by using the '–help' option. By doing so, a description of the command and its arguments and options will be displayed.

3.3 Designing robot swarms

The procedure to automatically design robot swarms has been simplified through this framework. After creating an account, these are the steps one has to follow in order to design a robot swarm:

(1) swarmDesign newProject cluster PROJECT_NAME CLUSTER_USERNAME

(2) swarDesign set-mission

(3) swarDesign set-budget NUMBER

(4) swarDesign run-design –fsm –chocolate –tuttifrutti

(5) swarDesign status

(6) swarDesign status –complete

This instance, is creating a project on the cluster and it is designing the swarm by using the modules of AutoMoDe-Chocolate and AutoMoDe-TuttiFrutti. SwarmDesign is providing a simple procedure to design robot swarms. In the next chapter, this framework will be tested on five different missions and in different experimental setups.

Chapter 4

Experimental setup

Two experiments have been conducted. The first one was to assess the reliability of the framework. The second one was to assess and analyze the control software generated by swamrDesign when the framework is combining different subsets of modules of AutoMoDe. Based on the observations of the second experiment, a third experiment has been added. The third experiment is a comparison between two different control software architecture such as probabilistic finite state machines and behavior trees. All conducted experiments are described in the following sections.

4.1 Experiment 1: software Validation

The first experiment is conducted to validate swarmDesign. This experiment is composed of five different parts. Indeed, swarmDesign combines subsets of modules of five different AutoMoDe's flavours: Chocolate, Maple, TuttiFruiti, Coconut and Arlequin. Each part of this experiment is dedicated to one AutoMoDe flavour. For that particular flavour a mission will be chosen from its original paper. This mission will be described and the experiment concerning it will be reproduce in simulation with the exact same experimental setup. But this time the control softwares will be generated by swarmDesign. The framework will be restricted to only use the subset of modules of the flavour in question. To be reliable the framework should be able to reproduce the results of pass experiments of each flavour.

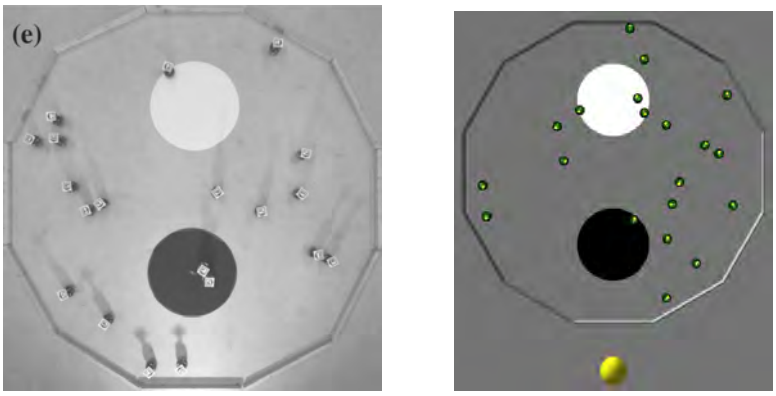
4.1.1 AutoMoDe-Chocolate

In [Francesca et al., 2015] the authors have introduced AutoMoDe-Chocolate for the first time. They have assessed its performance on 5 different missions. One of those missions was the Aggregation with ambient cues (AAC). The goal of the following experiment is to reproduce through the framework the result obtained in simulation for this mission.

Mission description

In the Aggregation with ambient cues mission the robots have to aggregate on the black spot. For that purpose, the arena is composed of two circular regions. Those two regions have a radius of 0.3m but they are of different colors. The first one is black and the second is white. A light source is present on the south side of the arena near the black

region. Those elements can be used by the robots to orientate themselves. A measure of performance is define through an objective function. The objective function to maximise is the following : $F_{AAC} = \sum_{t=1}^{T} N(t)$, where N(t) is the number of robots on the black region at time t.



(a) Arena in reality (b) Arena in simulation

Figure 4.1: Figure (a) is the arena from the original paper. Figure (b) is the arena used for designing and assessing the control software generated during the experiment of AutoMoDe-Chocolate.

Protocol

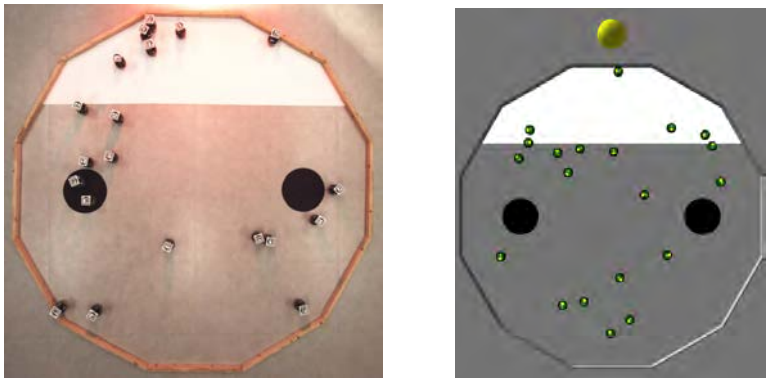
To reproduce the results obtained in the paper a similar experimental setup has been made. The duration of the mission is limited to 120 s. The robotic platform targeted is the RM1 [Hasselmann et al., 2018]. The simulator used to evaluate the design candidate is ARGoS [Pinciroli et al., 2012]. The design budget is of 200,000 executions of ARGoS. The 2.2 version of irace is used.

4.1.2 AutoMoDe-Maple

AutoMoDe-Maple has been introduced for the first time in [Ligot et al., 2020b]. In this paper, the authors assessed the quality of the produced control software on two different missions: Aggregation and Foraging. The goal of the following experiment is to reproduce through the framework the results obtained in simulation. This experiment will only consider the Foraging mission.

Mission description

The arena of the Foraging mission is composed of a nest and two sources areas. The ground color of the nest region is white while the sources areas have a black ground. A light source is present behind the nest. It can help the robots to navigate to the nest. The goal of the swarm is to retrieve as many objects as possible. A robot entering the nest after passing through a sources area is considered as an object retrieved. The performance measure for this mission is the following objective function : $F_F = N_i$, where N_i is the number of objects retrieved.



(a) Arena in reality (b) Arena in simulation

Figure 4.2: Figure (a) is the arena from the original paper. Figure (b) is the arena used for designing and assessing the control software generated during the experiment of AutoMoDe-Maple.

Protocol

The protocol adopted for this experiment is similar to the one in [Ligot et al., 2020b]. The software produces control software for the Foraging mission. The duration of the mission is limited to 120 s. The swarm is composed of 20 e-pucks robots. Due to the stochasticity of the design process, it is run 10 times and produces 10 instances of control software. The design budget allocated to the mission is 50 000 simulation runs. The performance of each instance is assessed once in simulation. The simulator used is ARGoS, beta 48 [Pinciroli et al., 2012].

4.1.3 AutoMoDe-TuttiFrutti

AutoMoDe-TuttiFrutti has been introduced for the first time in [Garzón Ramos and Birattari, 2020]. The authors of the paper have assessed AutoMoDe-TuttiFrutti in three different missions: Foraging, Stop and Aggregation. In those missions an import role is played by colors displayed in the environment. The goal of the following experiment is to reproduce through the software the results obtained in simulation of the Aggregation mission.

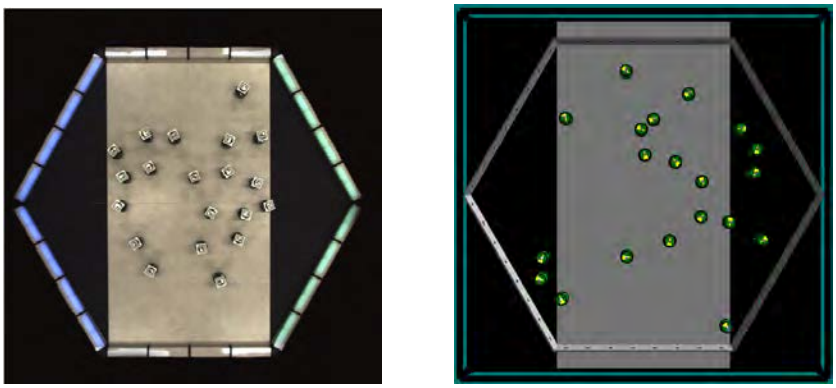
Mission description

The Aggregation mission takes place in an hexagonal arena of about 2.60 m^2. This arena is made up of three different zones. The first one is located at the left side of the arena. It has a triangle shape and a black ground color. The two walls of this zone are displaying a blue color. The second zone is located in the middle of the arena. It has a rectangular shape and a grey ground color. The last zone is located at the right side of the arena. It has the same shape and ground color as the first one. The only difference is the green color displayed by the walls. Initially the robots are randomly distributed in the second zone. Their mission is to aggregate as soon as possible in the first zone. The performance measure of the swarm is define by an objective function:

$$C_a = \sum_{t=1}^{T} \sum_{i=1}^{N} I_i(t)$$

$$I_i(t) = \begin{cases} 1, \text{if robot i is not in the aggregation area at time t;} \\ 0, \text{otherwise.} \end{cases}$$

C_a indicates the time that the robots spend outside of the blue zone. N and T represent the number of robots and the duration of the mission, respectively. The lower this objective function is, the better the swarm is performing.



(a) Arena in reality (b) Arena in simulation

Figure 4.3: Figure (a) is the arena from the original paper. Figure (b) is the arena used for designing and assessing the control software generated during the experiment of AutoMoDe-TuttiFrutti.

Protocol

The protocol is similar to the one defined in [Garzón Ramos and Birattari, 2020]. The swarm is composed of twenty e-pucks robots. The mission has a limited time of T = 120 s. The control software is produced by the framework. The design budget allocated to this experiment is 100 K. Due to the stochasticity of the design process, ten designs of the control software have been produced for the same mission. The results of the design processes through the framework have been assessed by testing each design once in simulation. The simulator used is ARGoS, beta 48 [Pinciroli et al., 2012].

4.1.4 AutoMoDe-Coconut

In [Spaey et al., 2020] the authors have introduced AutoMoDe-Coconut for the first time. They have designed robot swarms for a set of missions. Those missions took place in both bounded and unbounded workspaces. The Grid exploration mission was part of the study. The goal of the following experiment is to reproduce, through the software, the results obtained in simulation for the bounded Grid exploration mission.

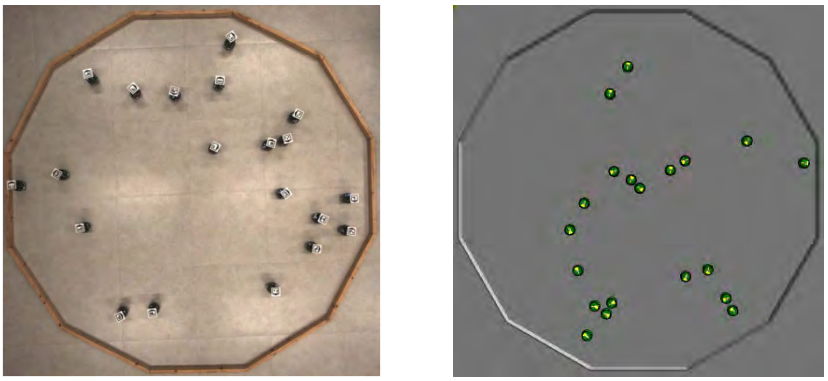
Mission description

The Grid exploration mission takes place in a workspace. This workspace is surrounded by walls placed to form a dodecagonal shape of 4.91 m^2. The ground color of the space

is gray. Initially the robots are randomly distributed in the workspace. Their mission is to explore and to cover as much space as possible. The performance metric is build as follow. The arena is divided in a gird of size 10 x 10. For each space in the grid, we count the time t for which the space has been unoccupied by a robot. When a robot occupies a space, the time t of this space is reset to zero. The swarm performance is the sum over all control cycles of the opposite of the average time t over all the spaces. It can be formalized as follow :

$$F_{gridexploration} = \sum_{i=1}^{N_{cc}} \left(\frac{1}{N_{spaces}} \sum_{j=1}^{N_{spaces}} -t_{ij} \right)$$

Where N_{cc} is the number of control cycles for the whole experiment, N_{spaces} is the number of spaces and t_{ij} is the time, at the control cycle i, since the space j was crossed by a robot.



(a) Arena in reality (b) Arena in simulation

Figure 4.4: Figure (a) is the arena from the original paper. Figure (b) is the arena used for designing and assessing the control software generated during the experiment of AutoMoDe-Coconut.

Protocol

The protocol that was followed is the one described in [Spaey et al., 2020]. The mission is performed by a swarm of 20 e-puck robots. The duration of the mission is limited to 120 s. The design budget allocated to the mission is 100 000 simulation runs. Due to the stochasticity of the design process, the design is executed 10 times for the same mission. It will produce 10 instances of control software. Each of these instances is then evaluated once in simulation. The simulator used is ARGoS, beta 48 [Pinciroli et al., 2012].

4.1.5 AutoMoDe-Arlequin

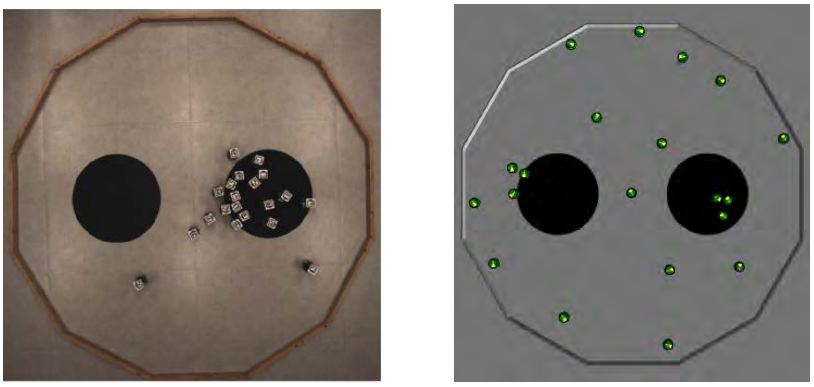
AutoMoDe-Arlequin is introduced for the first time in [Ligot et al., 2020a]. In this paper, the authors assessed the quality of the produced control software on two different missions: Foraging and Aggregation-xor. The goal of the following experiment is to reproduce through the software the results obtained in simulation for the Aggregation-xor mission.

Mission description

The aggregation-xor mission takes place in a dodecagonal arena of 4.91 m2 surrounded by walls. The arena has a gray ground color and it has two circular black regions. The mission of the robots is to aggregate on one of the two black regions. The performance of the swarm is measured by the following function:

$$F_A = \frac{\max(N_l, N_r)}{N}$$

where N_l and N_r are the number of robots located on each of the two black area and N is the total number of robots.



(a) Arena in reality (b) Arena in simulation

Figure 4.5: Figure (a) is the arena from the original paper. Figure (b) is the arena used for designing and assessing the control software generated during the experiment of AutoMoDe-Arleyquin.

Protocol

The protocol adopted is the one described in [Ligot et al., 2020a] The mission is performed by a swarm of 20 e-puck robots. The duration of the mission is limited to 180 s. The design budget allocated is 200 000 simulation runs. Due to the stochasticity of the design process, the design is executed 10 times for the same mission. It will produce 10 instances of control software. Each of these instances is then evaluated once in simulation. The simulator used is ARGoS, beta 48 [Pinciroli et al., 2012].

4.2 Experiment 2: combining AutoMode's subsets of modules and architectures

The second experiment is about combining the five subsets of modules of AutoMoDe. The previous experiment aimed to determine if the framework is able to reproduce the functioning of each of its component flavours. In this experiment SwarmDesign will use all the five subsets of modules available to design control software for five different missions. These missions are the same as those described in the first experiment. The same experimental setups and protocols as the first experiment were performed and followed in the second experiment. The goal of this experiment is to study the impact of combining different subsets of modules on the results of the design process in simulation. Both

results results, the one obtained by combining all five flavours and the results obtained with a single flavour have been plotted.

4.2.1 Comparing probabilistic finite-state machines (FSM) vs behavior trees (Bt)

This experiment has been conduct is an extension to the second experiment. During the investigation, it was noted that, in the control software produced, there was a considerable majority of probabilistic finite states machines to the detriment of behavior trees. Indeed, the ten design processes performed for each of the 4 last missions have resulted to a probabilistic finite states machine architecture. On the contrary, the ten design processes performed for the AAC mission led to one finite state machine and nine behavior trees. To better understand these results, the following experiment has been conduct. For each of the five missions, twenty design processes have been performed. The ten first design processes were constrained to produce probabilistic finite state machines and the others were constrained to produce behavior trees. The budget allocated to these design processes was 200.000 simulations. The performances of the produced instances were assessed once in simulation. The performance results of probabilistic finite state machines and behavior trees have been plotted.

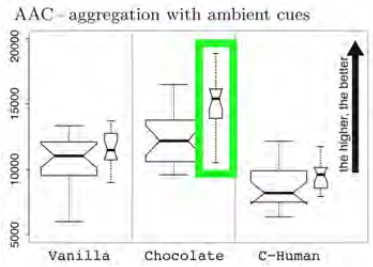
The objective of this experiment is to determine if separating the control architecture gives an advantage at combining the set of modules. A priory, it was thought that a possible reason to the aforementioned results is that finite-state machines inherently produce better performing control software. However, it was also of interest to investigate if, on the contrary, the results were caused because the finite-state machine converge faster to a reasonably good solution, and this biases the optimization process towards this type of architecture.

Chapter 5

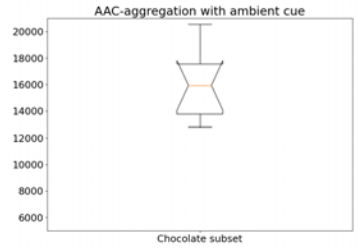
Results and discussion

5.1 Results of software Validation

The validation experiment was conducted to proof that swarmDesign is able to reproduce the results obtained by each of its composing AutoMoDe flavour.

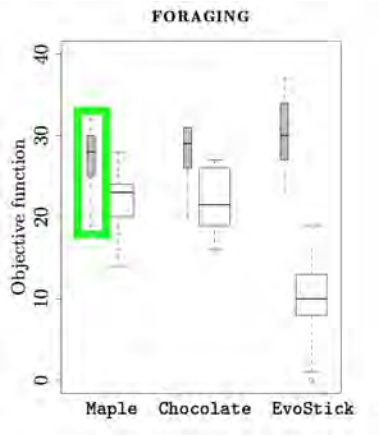


(a) Original paper's results

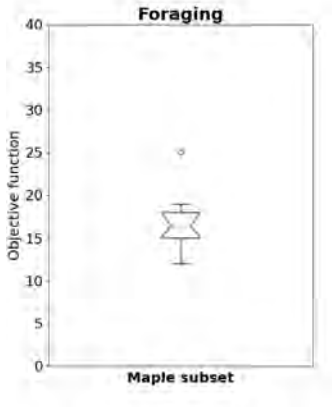


(b) Validation experiment's results for AutoMoDe-Chocolate.

Figure 5.1: (a) AutoMoDe-Chocolate's results in simulation are represented by the narrow box in the middle plot. (b) Reproduction of the original results through swarmDesign.

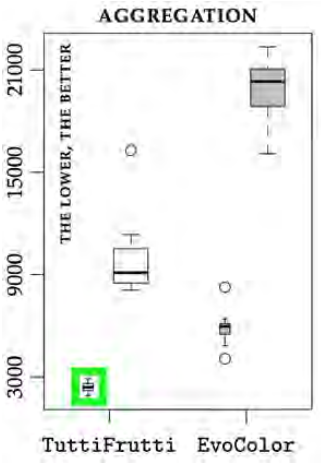


(a) Original paper's results

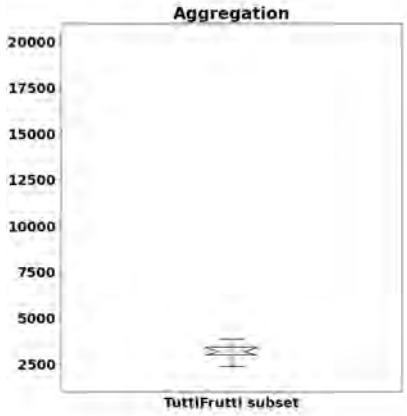


(b) Validation experiment's results for AutoMoDe-Maple.

Figure 5.2: (a) AutoMoDe-Maple's results in simulation are represented by the narrow box on the left side of the plot. (b) Reproduction of the original results through swarmDesign.

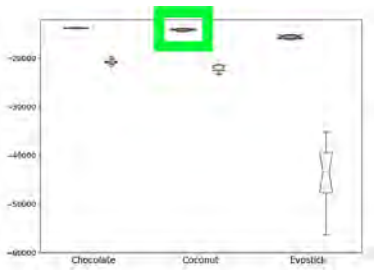


(a) Original paper's results

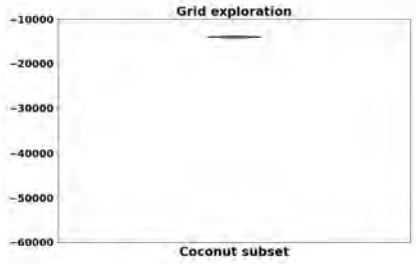


(b) Validation experiment's results for AutoMoDe-TuttiFrutti.

Figure 5.3: (a) AutoMoDe-TuttiFrutti's results in simulation are represented by the narrow box on the left side of the plot. (b) Reproduction of the original results through swarmDesign.

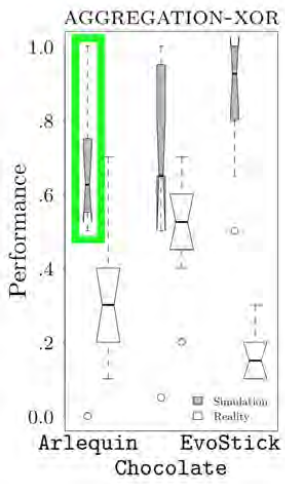


(a) Original paper's results

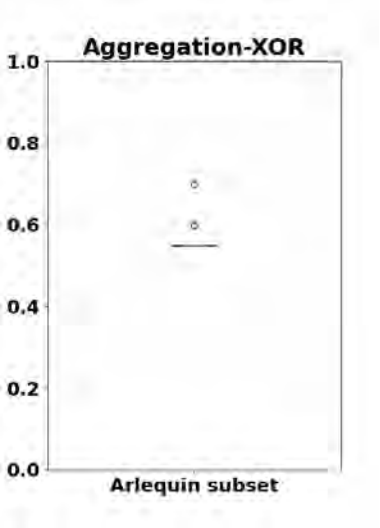


(b) Validation experiment's results for AutoMoDe-Coconut.

Figure 5.4: (a) AutoMoDe-Coconut's results in simulation are represented by the narrow box in the middle of the plot. (b) Reproduction of the original results through swarmDesign.



(a) Original paper's results



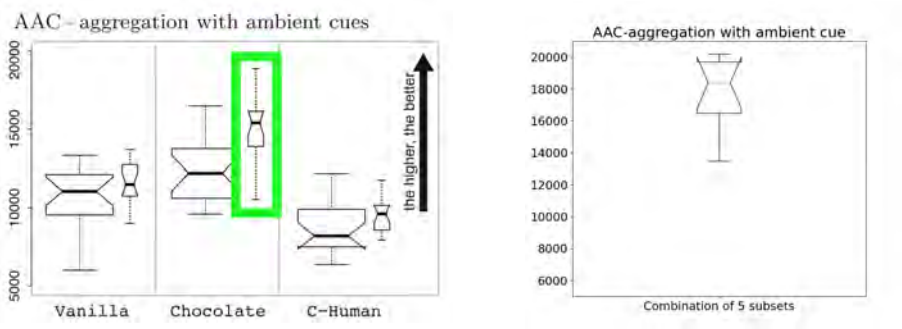
(b) Validation experiment's results for AutoMoDe-Arlequin.

Figure 5.5: (a) AutoMoDe-Arlequin's results in simulation are represented by the narrow box on the left side of the plot. (b) Reproduction of the original results through swarmDesign.

The results of the validation experiment are showing that swarmDesign is able to reproduce the results of AutoMoDe-TuttiFrutti and AutoMoDe-Coconut. An increase and a decrease in performance have been observed when swarmDesign is reproducing the results obtained respectively by AutoMoDe-Chocolate and by AutoMoDe-Arleyquin. This can be explained by the stochasticity of the process. A more significant decrease in performance is observed in the validation experiment of AutoMoDe-Maple. A more in-depth study must be conduct to insure that this is due to the stochastic. Nevertheless, swarmDesidn is able to reproduce the results of those 5 AutoMoDe's flavours with a relatively similar performance.

5.2 Results of combining AutoMode's subsets of modules and architectures

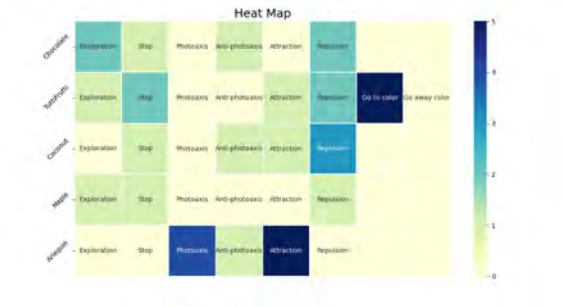
5.2.1 AAC mission



(a) Original paper's Results

(b) Results of the combination of the five flavours.

Figure 5.6: (a) AutoMoDe-Chocolate results in simulation are represented by the narrow box in the middle plot. (b) Represents the results obtained in simulation by combining the subsets of modules of all five flavours. Both experiments have been conduct with a budget of 200K on the AAC mission.

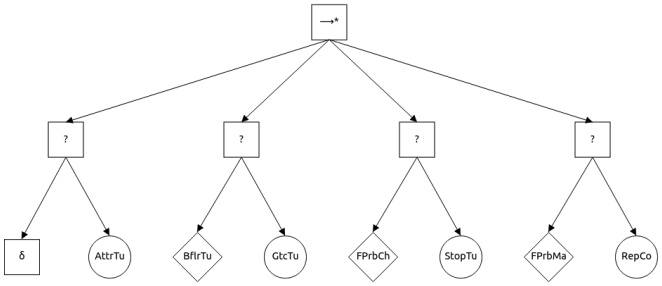


(a) Heat map of the behavior modules.

(b) Heat map of the condition modules.

Figure 5.7: Heat maps of the modules present in the ten different control software designed for the AAC mission.

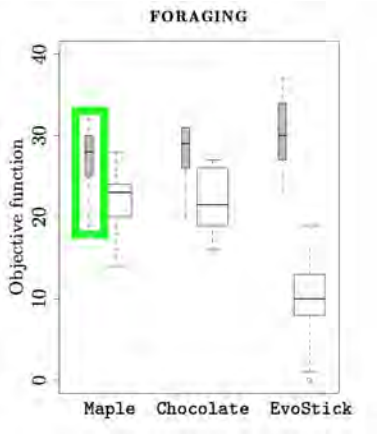
Compared to the performance of AutoMoDe-Chocolate in simulation, a significant increase in performance can be observed on figure 5.6 b. This increase in performance can be explained by the behavior observed in simulation. Indeed, an analysis of the collective behavior has shown that the best instances of control software were using colored information. By communicating through their LED's, the robots were able to attract or to repulse each other. Whether blinking or emitting a constant signal, robots were able to efficiently aggregate on the black spot. Even if the combination of the five subsets of module has increase the search space, the optimization algorithm is able to find better candidates in the same budget. Another observation is the unusual convergence of Irace. Indeed, it appears that for this particular mission the design process led us 9 times out of ten to a behavior tree architecture. In all the other missions Irace has always converged to a probabilistic final state machine architecture. This point has been further investigated in the last experiment.
The analysis of the heat map shows that modules from different subsets have been successfully combined to perform the mission. The heat map of the behavior modules is showing an important presence of the go to color module from TuttiFrutti, the Attraction and the Photoaxis module from arlequin, and the repulsion module from coconut.



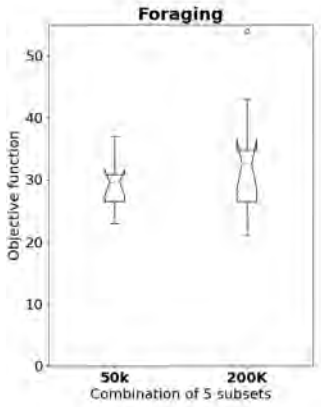
(a) Representation of one of the ten control software generated.

Figure 5.8: This control software uses the attraction, the black floor, the go to color and the stop modules from TuttiFrutti. The fixed probability module from chocolate and from Maple are also used. And It uses also the repulsion module from coconut. All those modules are combined in a behavior tree.

5.2.2 Foraging mission

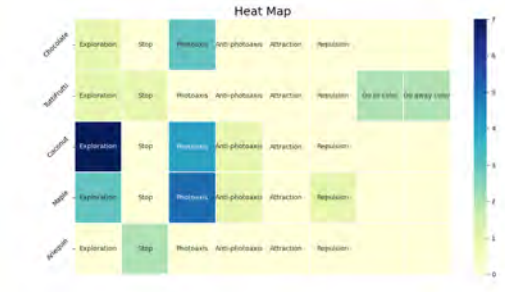


(a) Original paper's results

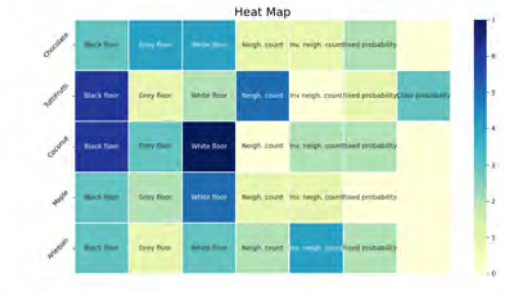


(b) Results of the combination of the five flavours.

Figure 5.9: (a) AutoMoDe-Maple's results in simulation are represented by the narrow box on the left side of the plot. (b) Represents the results obtained in simulation by combining the subsets of modules of all five flavours. Both experiments have been conducted with a budget of 50K on the Foraging mission.



(a) Heat map of the behavior modules.

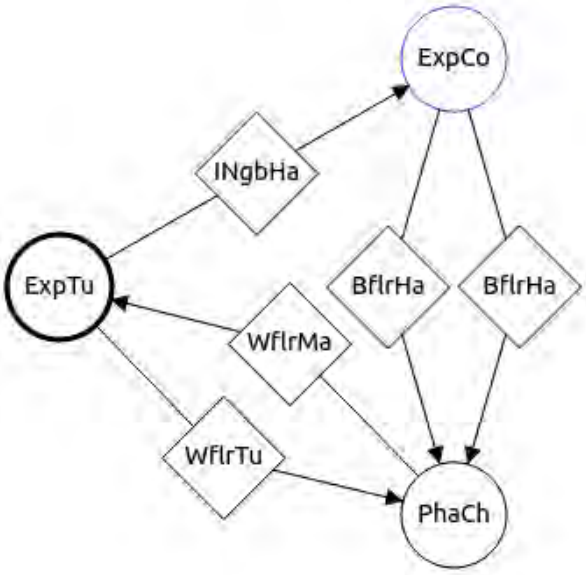


(b) Heat map of the condition modules. generated.

Figure 5.10: Heat maps of the modules present in the ten different control software designed for the Foraging mission.

For the Foraging mission combining subsets of modules does increase a little bit the overall performance. An unexpected observation is that for a small budget of 50k and for a

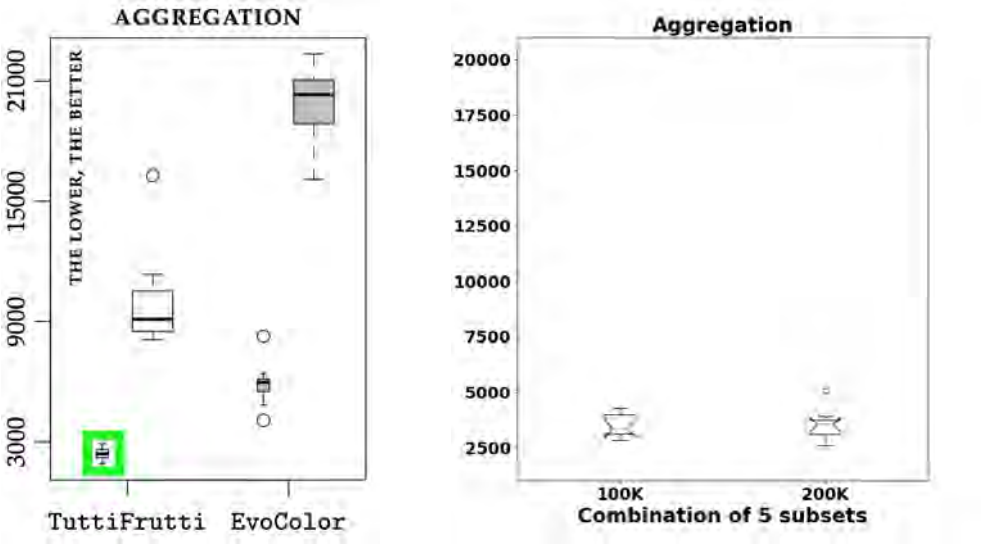
search space of five subsets of modules, the performance is better than a simple design method. In this mission, increasing the budget does increase a bit the performance of the swarm. By analysing the heat map of the behavior modules, the following strategy can be deduced. Robots are exploring the arena, they have the ability to go toward the source light and they can communicate through color light. The black and the white floor are impacting their behaviors. The ten instances generated by irace are all probabilistic finite state machines. A further study has been conducted, in the last experiment, to compare the performance between this architecture and the behavior tree architecture.



(a) Representation of one of the ten control software designed.

Figure 5.11: This control software uses the inverted neighbors count and the black floor modules from Arlequin. It uses also the exploration and the White floor modules from TuttiFrutti. The white floor module of Maple is also used. And It uses the photoaxis module from Chocolate. All those modules are combined in a probabilistic finite state machine.

5.2.3 Aggregation mission

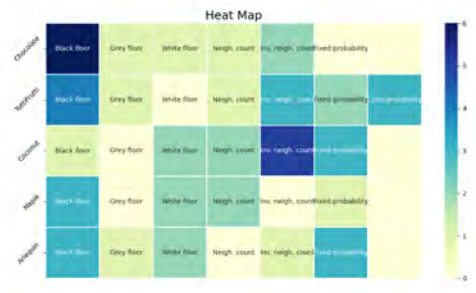


(a) Original paper's results

(b) Results of the combination of the five flavours.

Figure 5.12: (a) AutoMoDe-TuttiFrutti's results in simulation are represented by the narrow box on the left side of the plot. (b) Represents the results obtained in simulation by combining the subsets of modules of all five flavours. Both experiments have been conducted with a budget of 100K on the Aggregation mission.

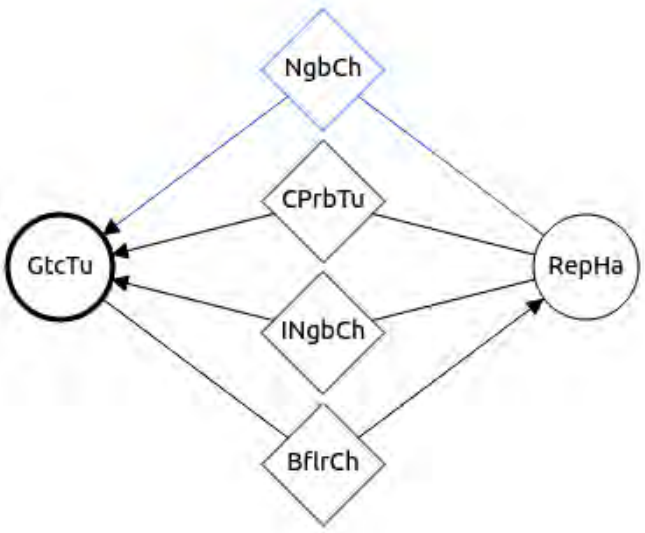
(a) Heat map of the behavior modules.



(b) Heat map of the condition modules.

Figure 5.13: Heat maps of the modules present in the ten different control software designed for the Aggregation mission.

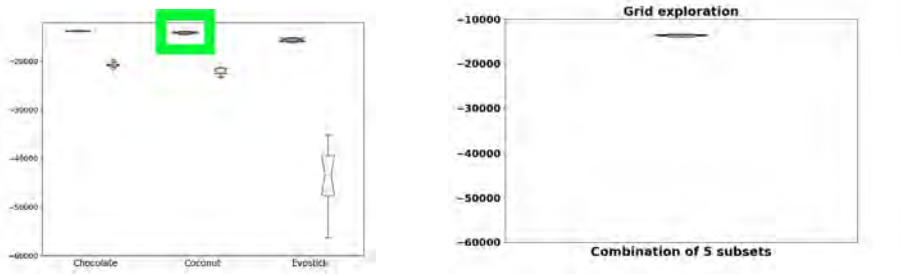
The results of the aggregation mission show that the method of combining subsets of modules has a similar performance as AutoMoDe-Tuttifrutti. Doubling the design budget does not have a significant impact on the performance. After an observation of the robots in simulations, it appears that the robots cannot do much better. Indeed, in less than 500 time steps all the robots are aggregated in the target zone. Their mission then is to stay in this area. This mission is already performed efficiently by the swarms designed by AutoMoDe-TuttiFrutti. Nevertheless, swarmDesign is not doing worse than AutoMoDe-TuttiFrutti. With a budget of 100k and a search space of 5 subsets of modules, irace is able to find the most adequate modules to use. The heat map is showing the important presence of the go to color module. This module is very efficient in a colored light emitting environment. The Black floor and the inverted neighbors count are also useful condition modules for this mission. The ten control software generated were probabilistic finite state machines. A further comparison between the performance of this architecture and the one of the behavior tree architecture has been analysed in the third experiment.



(a) Representation of one of the ten control software designed.

Figure 5.14: This control software uses the Go to color and the Color probability modules from TuttiFrutti. It uses also the Neighbors count, the Inverted neighbors count and the Black floor modules from Chocolate. The Repulsion module from Arlequin is also used. All those modules are combined in a probabilistic finite state machine.

5.2.4 Grid exploration mission



(a) Original paper's results

(b) Results of the combination of the five flavours.

Figure 5.15: (a) AutoMoDe-Coconu's results in simulation are represented by the narrow box in the middle of the plot. (b) Represents the results obtained in simulation by combining the subsets of modules of all five flavours. Both experiments have been conducted with a budget of 100K on the Grid exploration mission.

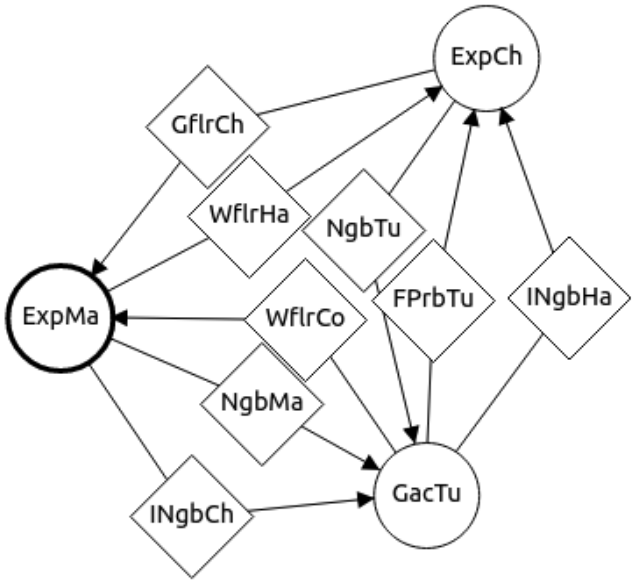
(a) Heat map of the behavior modules.

(b) Heat map of the condition modules.

Figure 5.16: Heat maps of the modules present in the ten different control software designed for the Grid exploration mission.

The results of both design methods are similar for the Grid exploration mission. The same observation as the previous mission can be made. This mission is simple enough to be efficiently performed with a single subset of modules. A significant increase in performance is not possible. In simulations, the robots explore the arena well and avoid each other. However, the importance of the go away color module, neighbors count and the inverted neighbors count is shown by the heat map. By analysing the heat maps

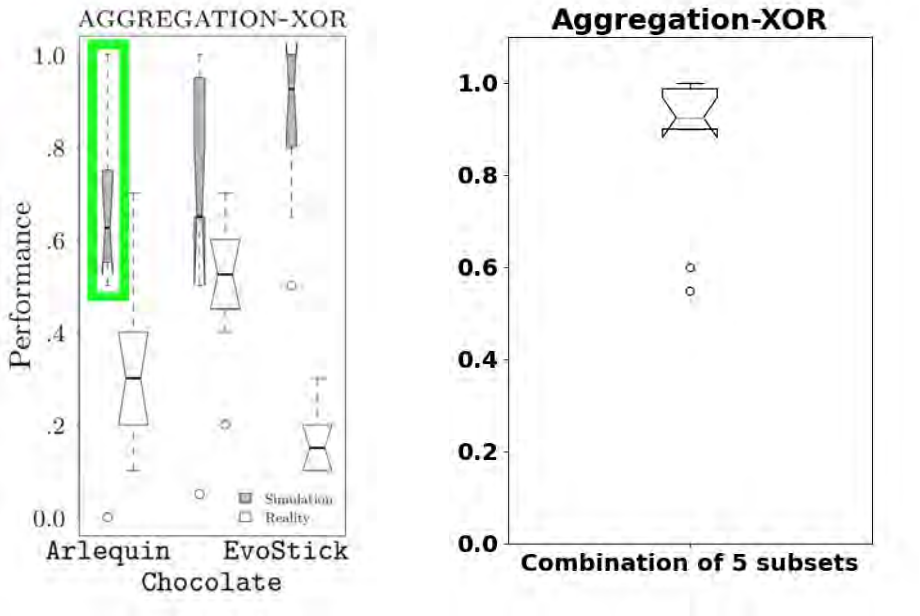
the strategy of the robots can be deduced. As usual, the final 10 instances of the design processes are probabilistic finite state machines. A comparison of the performance of both control software architecture for this mission has been done in the third experiment.



(a) Representation of one of the ten control software generated.

Figure 5.17: Ten different design process have been run to produce ten instances of control software for the Grid exploration mission. Those control software are using modules from 5 different flavours. Figure (b) represents one of those ten control software. This control software uses the Neighbors count, the Go away color and the Fixed probability modules from TurriFrutti. It uses also the Inverted neighbors count and the White floor modules from Arlequin. The Inverted neighbors count, Exploration and Grey floor modules from Chocolate are also used. From Coconut, the White floor module is used. And It uses the Neighbors count and the Exploration modules from Maple. All those modules are combined in a probabilistic finite state machine.

5.2.5 Aggregation-xor mission

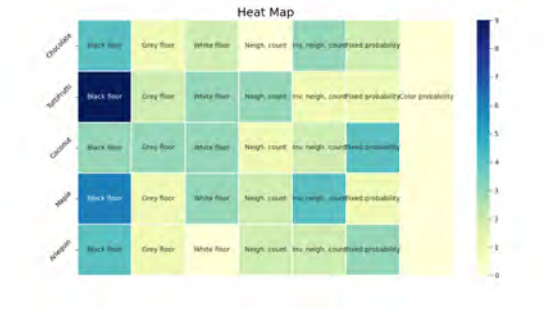


(a) Original paper's results

(b) Results of the combination of the five flavours.

Figure 5.18: (a) AutoMoDe-Arlequin's results in simulation are represented by the narrow box on the left side of the plot. (b) Represents the results obtained in simulation by combining the subsets of modules of all five flavours. Both experiments have been conducted with a budget of 200K on the Aggregation-xor mission.

(a) Heat map of the behavior modules.

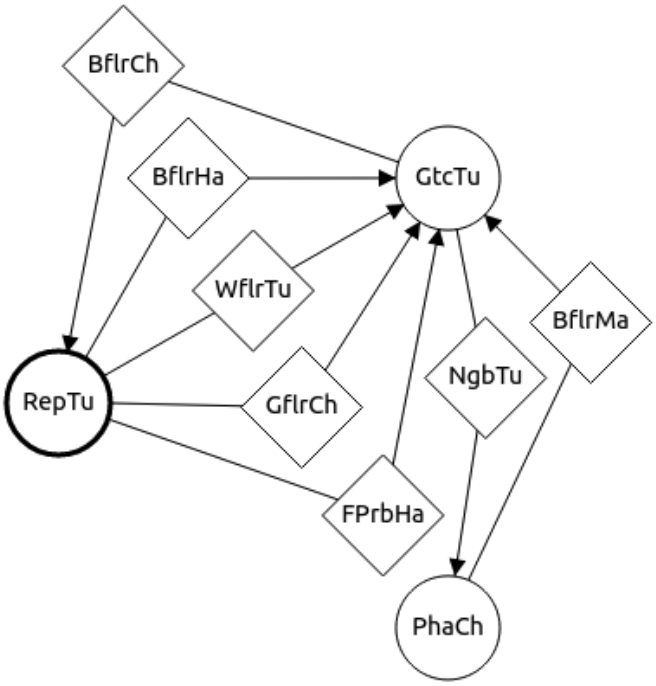


(b) Heat map of the condition modules.

Figure 5.19: Heat maps of the modules present in the ten different control software designed for the Aggregation-xor mission.

The results of the Aggregation-xor mission are as expected. The combination of AutoMoDe subsets led to a significantly better performance than AutoMoDe-Arlequin in simulation. The results achieved cannot be further improved considerably. Indeed, the swarms are performing near optimally for this mission. A performance equal to one means that all the robots have decided to aggregate on the same black spot. A further analysis of the robots in simulations have shown an emergence of the following particular collective behavior. The robots start by exploring the map. When a robot is sensing a black floor and if the robot is not detecting a certain number of other robots, this robot will leave the black spot. In the other case, if the robots detects some other robots this robot will start emitting a colored light signal. This signal is used to attract robots outside of the black region. When several robots are present on the black region the following behavior is observed. Robots are attracting each other at a certain frequency. By doing so, the robots are able to stay together when they are present in sufficient quantity otherwise they will leave the black region. Those observations are strengthen by the heat maps. The most present behavior modules are Attraction, repulsion, Go to color and go away color from TuttiFrutti. The heat map of the condition modules is showing an important presence of modules such as Black floor, neighbors count and inverted neighbors count.

The ten control software obtained for this mission are probabilistic state machines. The third experiment will compare the performance of both architecture for the Aggregation-xor mission.



(a) Representation of one of the ten control software designed.

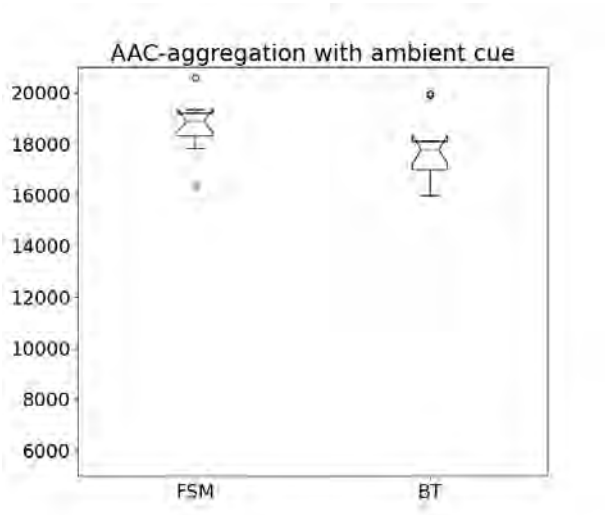
Figure 5.20: This control software uses the Repulsion, the White floor, the Neighbors count and the Go to color modules from TuttiFrutti. It uses also the Black floor, Grey floor and the Photoaxis modules from Chocolate. The Black floor and the Fixed probability modules of Arlyquin are also used. And It uses the Black floor module from Maple. All those modules are combined in a probabilistic finite state machine.

5.2.6 Experiment 2: General remark

It has been shown through the results of this experiment that combining AutoMoDe's subsets of modules and architecture is a viable approach. In some cases, it has led to a better solution. Combining those modules has increase the ability of the swarm. But some missions were to simple to entirely take advantage of this combinations of modules. However, the framework has successfully designed the control software for five different missions. It has used all the modules available. The optimisation algorithm has shown its efficiency by designing swarms in a budget such as 50K, 100K and 200K. The majority of the design processes led to finite state architecture to the detriment of behavior trees.

5.3 Results of comparing FSM vs BT

5.3.1 AAC mission

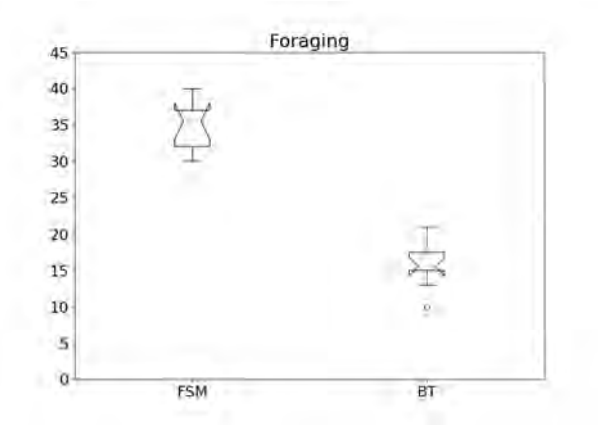


(a) Results of the comparison between fsm and bt.

Figure 5.21: Comparison between fsm and bt on the AAC mission

In experiment 2, the AAC mission has been designed with a budget of 200k without any restrictions. The results of this experiment have shown that nine designs out of ten have produced control software with a behavior tree architecture. The results obtained on figure 5.21 are not expected. They are showing that probabilistic finite state machines have better performances then behavior trees. This is may be due to the allocated design budget. Indeed, this phenomenon has already been observed in another study [Kuckling et al., 2018]. It has been explained that with behavior trees it is possible to quickly find a reasonable good solution. When this solution is found, the performance will not remain the same. On the Contrary, probabilistic finite state machines need some more time to achieve good performances. In this case, a considerable budget of 200K is allocated which produces finite state machines with good performances.

5.3.2 Foraging mission

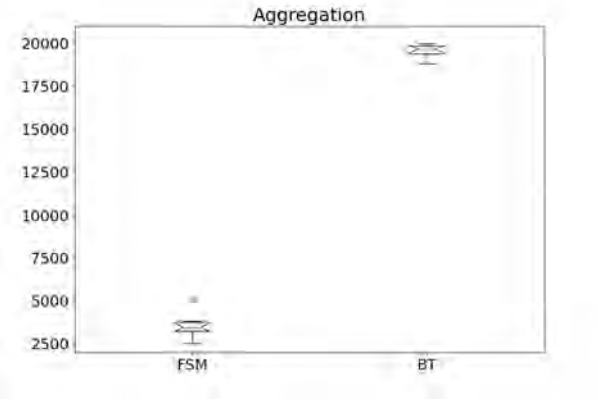


(a) Results of the comparison between fsm and bt.

Figure 5.22: Comparison between fsm and bt on the Foraging mission

The results of the Foraging mission are showing a significant difference in the performances of both architectures. This justifies the observation of experiment 2, where all the design processes have led to finite state machines. This plot is clearly showing that the combination of modules within behavior trees are not as efficient as they are for the finite state machines.

5.3.3 Aggregation mission

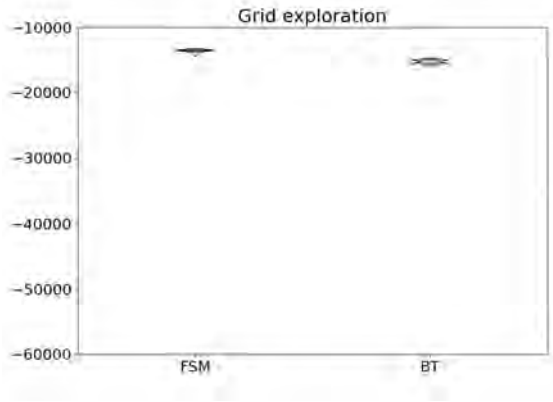


(a) Results of the comparison between fsm and bt.

Figure 5.23: Comparison between fsm and bt on the Aggregation mission

In the Aggregation experiment of experiment 2, the ten design processes have led to probabilistic finite state machines. The result obtained in figure 5.22 were expected. Fsm are performing much better than behavior trees. The gap between both performances is considerable. The aggregation mission can efficiently be done with the modules of AutoMoDe-Tuttifrutti. But it appears that behaviors trees are not able to take advantage of them.

5.3.4 Grid exploration mission

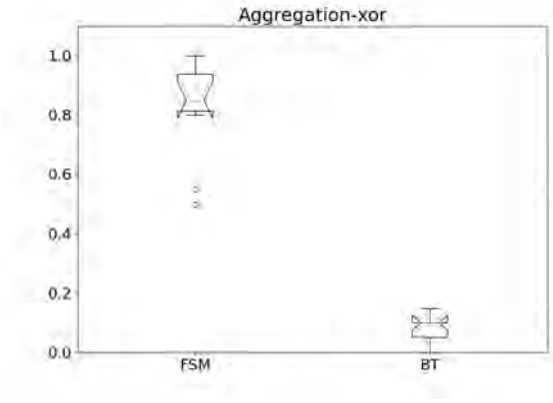


(a) Results of the comparison between fsm and bt.

Figure 5.24: Comparison between fsm and bt on the Grid exploration mission.

The results of the Grid exploration mission are showing a slightly better performance for the finite state machines. But due to the scale of the Y-axis, this difference can be significant. This will justify the observations of experiment 2, where all designed control software were finite state machines. However, finite state machines are more efficient for the grid exploration mission.

5.3.5 Aggregation-xor mission



(a) Results of the comparison between fsm and bt.

Figure 5.25: Comparison between fsm and bt on the Aggregation-xor mission.

The results of the Aggregation-xor mission are showing that finite state machines are efficiently performing the mission. On the contrary, the results are showing that the behavior tree architectures are not suitable to perform the aggregation-xor mission.

5.3.6 Experiment 3: General remark

Finite state machines have proved their efficiency in all the five missions. This control software architecture, is able to combine modules from different subsets and to perform several missions. In all the missions, finite state machines have outperformed the behavior trees. The behavior tree software architecture has shown some difficulties to combine subsets of modules. It appears that they are not as efficient as the finite states machines. A further study should be conducted to find the reason behind these results.

Chapter 6

Conclusions

This master thesis presents SwarmDesign, a framework for the automatic modular design of robot swarms has been developed. Trough simple commands, this framework has defined a procedure to automatically design robot swarms. SwamrDesign is built on the basis of methods in the AutoMoDe family, and it designs control software following the modular approach. SwamrDesign combines several subsets of modules of five methods: AutoMoDe-Chocolate, AutoMoDe-Maple, AutoMoDe-TuttiFrutti, AutoMoDe-Coconut and AutoMoDe-Arlequin. This enables the experimentation the automatic design of robot swarms by combining the specialized modules, control architectures, and hardware capabilities that were originally conceived as independent. A first experiment has been conducted to validate the framework. This was done by restricting the design process to one subset of modules and to compare the results obtained with the results of the AutoMoDe method using the same subset of modules. To conduct these evaluation, in the thesis it was considered a group of five missions originally introduced with the aformentioned AutoMoDe methods. A second experiment has been conducted on the same missions, but this time SwarmDesign had no restrictions in the subset of modules or control architecture that it could use to produce the control software. In this manner, the framework was able to combine modules from different subsets and produce control software that was never conceived before: for example, software modules designed for AutoMoDe-TuttiFrutti now assembled in a behavior tree conceived for AutoMoDe-Maple. The results of the experiments has been analyzed with respect to the performance of the swarm, their collective behaviors, and constituent software modules of the control software produce. A special attention has been accorded to the architecture of the control software. In the last experiment, a comparison between finite state machines and behavior trees has been done. The results of the first experiment has shown that the developed framework is an efficient tool for the design of robot swarms in simulation. It is a reliable framework which can reproduce the results obtained by the automatic design method who shares a subset with it. The second experiment has shown that combining AutoMode's subsets of modules is a viable approach. It has been tested on five different missions and it has not been outperformed. On the contrary, this approach has made it possible to achieve on some missions a performance never achieved before. Some missions were to simple to take full advantage of the designing capabilities of SwarmDesign. The last experiment has shown that whit a budget of 200k, finite state machines have an advantage on the behavior trees. But this topic deserves to be studied more specifically.

In the following, there are some partial answers to the questions originally proposed

in the research and that are detailed in the introduction. (i) *Are AutoMoDe methods compatible with each other?:* SwarmDesign has successfully designed collective behaviors for different types of missions. In order to do so, it has used modules from every subsets. Some modules of subset have been used more than others. For instance modules from TuttiFrutti are the most present one as it appears that the automatic design process can better exploit its properties to address the set of missions. However, each subset has its impact on the design process. Mostly represented in the transition conditions. (ii) *Would the optimization algorithm be able to successfully combine the modules of different methods?* This framework has been used to design robot swarms with different designing budgets such as 50K, 100K and 200K. In every configuration, it has been able to produce control software having at least a similar performance than the other designing method. Even if the search space has been significantly extended due to the multiple subsets, the optimisation algorithm is still able to converge towards a reasonable good solution. We argue that the optimization algorithm selected is sufficient to produce reasonably good control software. However, as no other study exists on this scale, there is no certainty on whether the combination of modules could lead to better performing results. This thesis serves as a baseline from which to extend the studies on the automatic modular design of robot swarms while considering large subsets of software modules. (iii) Aare there software architectures that are not compatible with some of the modules? For each mission, the design processes have led to at least one finite state machine. This is a prove that the modules are compatible with this type of software architecture. In contrary, behavior trees are rarely appearing in the final results of a design process. A considerable drop of performance has been observed on the aggregation mission. This mission is efficiently handled by the modules of Tuttifrutti. It appears that although the modules and the behavior trees are compatible, the optization algorithmn is not capable of finding good performing solutions when they operate together. The reason for which this happens cannot clearly identified from the results obtained in this thesis, and it is a topic to be addressed in future work.

All in all, it is possible to conclude that SwarmDesign is a reliable framework which is able to automatically design robot swarms by combining different subsets of modules from the AutoMode-familly. It enables for new investigations with the combination of sofware modules with new architectures. It eases the deployment of experiments through a user-friendly interface to conduct and monitor the automatic design process. And provides a basic structure to which add new AutoMoDe implementations.

Bibliography

[Balaprakash et al., 2007] Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In Bartz-Beielstein, T., Blesa, M. J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., and Sampels, M., editors, *Hybrid Metaheuristics*, volume 4771 of *Lecture Notes in Computer Science*, pages 108–122. Springer, Heidelberg.

[Birattari et al., 2021] Birattari, M., Ligot, A., and Francesca, G. (2021). AutoMoDe: a modular approach to the automatic off-line design and fine-tuning of control software for robot swarms. In Pillay, N. and Qu, R., editors, *Automated Design of Machine Learning and Search Algorithms*, Natural Computing Series, pages 73–90. Springer, Cham, Switzerland.

[Birattari et al., 2002a] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002a). A racing algorithm for configuring metaheuristics. In Langdon, W. B. et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA.

[Birattari et al., 2002b] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002b). A racing algorithm for configuring metaheuristics. In Langdon, W. B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M. A., Schultz, A. C., Miller, J. F., Burke, E. K., and Jonoska, N., editors, *GECCO'02: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 11–18, San Francisco, CA, USA. Morgan Kaufmann Publishers.

[Brambilla et al., 2013] Brambilla, M., Ferrante, E., Birattari, M., and Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41.

[Dorigo and Birattari, 2007] Dorigo, M. and Birattari, M. (2007). Swarm intelligence. *Scholarpedia*, 2(9):1462.

[Dorigo et al., 2014] Dorigo, M., Birattari, M., and Brambilla, M. (2014). Swarm robotics. *Scholarpedia*, 9(1):1463. revision #138643.

[Dorigo et al., 2020] Dorigo, M., Theraulaz, G., and Trianni, V. (2020). Reflections on the future of swarm robotics. *Science Robotics*, 5:eabe4385.

[Dorigo et al., 2021] Dorigo, M., Theraulaz, G., and Trianni, V. (2021). Swarm robotics: past, present, and future [point of view]. *Proceedings of the IEEE*, 109(7):1152–1165.

[Floreano et al., 2008] Floreano, D., Husbands, P., and Nolfi, S. (2008). *Evolutionary Robotics*, pages 1423–1451.

[Francesca and Birattari, 2016] Francesca, G. and Birattari, M. (2016). Automatic design of robot swarms: achievements and challenges. *Frontiers in Robotics and AI*, 3(29):1–9.

[Francesca et al., 2015] Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Mascia, F., Trianni, V., and Birattari, M. (2015). AutoMoDe-Chocolate: automatic design of control software for robot swarms. *Swarm Intelligence*, 9(2–3):125–152.

[Francesca et al., 2014a] Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., and Birattari, M. (2014a). AutoMoDe: A novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2):89–112.

[Francesca et al., 2014b] Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., and Birattari, M. (2014b). AutoMoDe: a novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence*, 8(2):89–112.

[Garattoni et al., 2015] Garattoni, L., Francesca, G., Brutschy, A., Pinciroli, C., and Birattari, M. (2015). Software infrastructure for e-puck (and TAM). Technical Report TR/IRIDIA/2015-004, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

[Garzón Ramos and Birattari, 2020] Garzón Ramos, D. and Birattari, M. (2020). Automatic design of collective behaviors for robots that can display and perceive colors. *Applied Sciences*, 10(13):4654.

[Garzón Ramos et al., 2021] Garzón Ramos, D., Bozhinoski, D., Francesca, G., Garattoni, L., Hasselmann, K., Kegeleirs, M., Kuckling, J., Ligot, A., Mendiburu, F. J., Pagnozzi, F., Salman, M., Stützle, T., and Birattari, M. (2021). The automatic off-line design of robot swarms: recent advances and perspectives. In De Masi, G., Ferrante, E., and Dario, P., editors, *R2T2: Robotics Research for Tomorrow's Technology*.

[Gutiérrez et al., 2009] Gutiérrez, Á., Campo, A., Dorigo, M., Donate, J., Monasterio-Huelin, F., and Magdalena, L. (2009). Open e-puck range & bearing miniaturized board for local communication in swarm robotics. In Kosuge, K., editor, *2009 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3111–3116, Piscataway, NJ, USA. IEEE.

[Hamann et al., 2020] Hamann, H., Schranz, M., Elmenreich, W., Trianni, V., Pinciroli, C., Bredeche, N., and Ferrante, E. (2020). Editorial: designing self-organization in the physical realm. *Frontiers in Robotics and AI*, 7:164.

[Hasselmann and Birattari, 2020] Hasselmann, K. and Birattari, M. (2020). Modular automatic design of collective behaviors for robots endowed with local communication capabilities. *PeerJ Computer Science*, 6:e291.

[Hasselmann et al., 2018] Hasselmann, K., Ligot, A., Francesca, G., Garzón Ramos, D., Salman, M., Kuckling, J., Mendiburu, F. J., and Birattari, M. (2018). Reference models for AutoMoDe. Technical Report TR/IRIDIA/2018-002, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.

[Hasselmann et al., 2021] Hasselmann, K., Ligot, A., Ruddick, J., and Birattari, M. (2021). Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms. *Nature Communications*, 12:4345.

[Koenig and Howard, 2004] Koenig, N. P. and Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference On Intelligent Robots And Systems (IROS)*, volume 3, pages 2149–2154, Piscataway, NJ, USA. IEEE.

[Kuckling et al., 2021] Kuckling, J., Hasselmann, K., Pelt, V. V., Kiere, C., and Birattari, M. (2021). Automode editor: a visualization tool for automode. *Tech. Rep. TR/IRIDIA/2021-009, IRIDIA, Brussels*.

[Kuckling et al., 2018] Kuckling, J., Ligot, A., Bozhinoski, D., and Birattari, M. (2018). *Behavior Trees as a Control Architecture in the Automatic Modular Design of Robot Swarms: 11th International Conference, ANTS 2018, Rome, Italy, October 29–31, 2018, Proceedings*, pages 30–43.

[Kuckling et al., 2020a] Kuckling, J., Stützle, T., and Birattari, M. (2020a). Iterative improvement in the automatic modular design of robot swarms. *PeerJ Computer Science*, 6:e322.

[Kuckling et al., 2020b] Kuckling, J., Ubeda Arriaza, K., and Birattari, M. (2020b). AutoMoDe-IcePop: automatic modular design of control software for robot swarms using simulated annealing. In Bogaerts, B., Bontempi, G., Geurts, P., Harley, N., Lebichot, B., Lenaerts, T., and Louppe, G., editors, *Artificial Intelligence and Machine Learning: BNAIC 2019, BENELEARN 2019*, volume 1196 of *Communications in Computer and Information Science*, pages 3–17. Springer, Cham, Switzerland.

[Kuckling et al., 2022] Kuckling, J., van Pelt, V., and Birattari, M. (2022). AutoMoDe-Cedrata: automatic design of behavior trees for controlling a swarm of robots with communication capabilities. *SN Computer Science*, 3:136.

[Ligot and Birattari, 2020] Ligot, A. and Birattari, M. (2020). Simulation-only experiments to mimic the effects of the reality gap in the automatic design of robot swarms. *Swarm Intelligence*, 14:1–24.

[Ligot et al., 2020a] Ligot, A., Hasselmann, K., and Birattari, M. (2020a). AutoMoDe-Arlequin: neural networks as behavioral modules for the automatic design of probabilistic finite state machines. In Dorigo, M., Stützle, T., Blesa, M. J., Blum, C., Hamann, H., Heinrich, M. K., and Strobel, V., editors, *Swarm Intelligence: 12th International Conference, ANTS 2020*, volume 12421 of *Lecture Notes in Computer Science*, pages 109–122, Cham, Switzerland. Springer.

[Ligot et al., 2020b] Ligot, A., Kuckling, J., Bozhinoski, D., and Birattari, M. (2020b). Automatic modular design of robot swarms using behavior trees as a control architecture. *PeerJ Computer Science*, 6:e314.

[López-Ibáñez et al., 2016] López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., and Stützle, T. (2016). The irace package: iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

[Mendiburu et al., 2022] Mendiburu, F. J., Garzón Ramos, D., Morais, M. R. A., Lima, A. M. N., and Birattari, M. (2022). AutoMoDe-Mate: automatic off-line design of spatially-organizing behaviors for robot swarms. *Swarm and Evolutionary Computation*, 74:101118.

[Mondada et al., 2009] Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In Gonçalves, P., Torres, P., and Alves, C., editors, *ROBOTICA 2009: Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pages 59–65, Castelo Branco, Portugal. Instituto Politécnico de Castelo Branco.

[Pinciroli et al., 2012] Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G. A., Ducatelle, F., Birattari, M., Gambardella, L. M., and Dorigo, M. (2012). ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295.

[Ramos and Birattari, 2020] Ramos, D. G. and Birattari, M. (2020). Automatic design of collective behaviors for robots that can display and perceive colors. *Applied Sciences*, 10(13):4654.

[Rohmer et al., 2013] Rohmer, E., Singh, S. P. N., and Freese, M. (2013). V-REP: a versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference On Intelligent Robots And Systems (IROS)*, volume 3, pages 1321–1326, Piscataway, NJ, USA. IEEE.

[Salman et al., 2020] Salman, M., Garzón Ramos, D., Hasselmann, K., and Birattari, M. (2020). Phormica: photochromic pheromone release and detection system for stigmergic coordination in robot swarms. *Frontiers in Robotics and AI*, 7:195.

[Salman et al., 2019] Salman, M., Ligot, A., and Birattari, M. (2019). Concurrent design of control software and configuration of hardware for robot swarms under economic constraints. *PeerJ Computer Science*, 5:e221.

[Schranz et al., 2020] Schranz, M., Umlauft, M., Sende, M., and Elmenreich, W. (2020). Swarm robotic behaviors and current applications. *Frontiers in Robotics and AI*, 7:36.

[Spaey et al., 2020] Spaey, G., Kegeleirs, M., Garzón Ramos, D., and Birattari, M. (2020). Evaluation of alternative exploration schemes in the automatic modular design of robot swarms. In Bogaerts, B., Bontempi, G., Geurts, P., Harley, N., Lebichot, B., Lenaerts, T., and Louppe, G., editors, *Artificial Intelligence and Machine Learning: BNAIC 2019, BENELEARN 2019*, volume 1196 of *Communications in Computer and Information Science*, pages 18–33. Springer, Cham, Switzerland.

[Trianni, 2008] Trianni, V. (2008). *Evolutionary Swarm Robotics*. Springer, Berlin, Germany.

[Yang et al., 2018] Yang, G.-Z., Bellingham, J., Dupont, P. E., Fischer, P., Floridi, L., Full, R., Jacobstein, N., Kumar, V., McNutt, M., Merrifield, R., Nelson, B. J., Scassellati, B., Taddeo, M., Taylor, R., Veloso, M., Wang, Z. L., and Wood, R. (2018). The grand challenges of Science Robotics. *Science Robotics*, 3(14):eaar7650.