Evolutionary strategies in swarm robotics controllers

Alexandre Valério Rodrigues

Master in Computer Engineering

Supervisor:
Phd Luís Miguel Nunes, Associate Professor
ISCTE-IUL

Co-Supervisor:
Phd Sancho Moura Oliveira, Associate Professor
ISCTE-IUL

November, 2022

TECNOLOGIAS
E ARQUITETURA

Department of Technology and Architecture

Evolutionary strategies in swarm robotics controllers

Alexandre Valério Rodrigues

Master in Computer Engineering

Supervisor:
Phd Luís Miguel Nunes, Assistant Professor
ISCTE-IUL

Co-Supervisor:
Phd Sancho Moura Oliveira, Assistant Professor
ISCTE-IUL

November, 2022

Acknowledgments

I would like to thank Micaela for all the love during this period and for being my pillar of strength. Your continuous support was essential in giving me the motivation to finish this journey.

I would like to acknowledge my Family, for their continued support, love, and comprehension.

I would also like to acknowledge my supervisors, Luís Nunes and Sancho Oliveira. The path toward a master's degree, while working simultaneously was challenging, and their patience and continuous guidance were crucial factors for my success.

I would also like to acknowledge my Portuguese Navy fellow comrades, especially in the Portuguese Hydrographic Institute, for their comprehension and motivation.

This work was partly funded through national funds by FCT - Fundação para a Ciência e Tecnologia, I.P. under project UIDB/04466/2020 (ISTAR). I would also like to acknowledge the Portuguese Telecommunication Institute.

Resumo

Atualmente os Veículos Não Tripulados (VNT) encontram-se difundidos por todo o Mundo. A maioria destes veículos requerem um elevado controlo humano, e o sucesso das missões está diretamente dependente deste fator. Assim, é importante utilizar técnicas de aprendizagem automática que irão treinar os controladores dos VNT, de modo a automatizar o controlo, tornando o processo mais eficiente.

As estratégias evolutivas podem ser a chave para uma aprendizagem robusta e adaptativa em sistemas robóticos. Vários estudos têm sido realizados nos últimos anos, contudo, existem lacunas que precisam de ser abordadas, tais como o *reality gap.* Este facto ocorre quando os controladores treinados em ambientes simulados falham ao serem transferidos para VNT reais.

Este trabalho propõe uma abordagem para a resolução de missões com VNT, utilizando um simulador realista e estratégias evolutivas para treinar controladores. A arquitetura escolhida é facilmente escalável para sistemas com múltiplos VNT.

Nesta tese, é apresentada a arquitetura e configuração do ambiente de simulação, incluindo o modelo e software de simulação do VNT. O modelo de VNT escolhido para as simulações é um modelo real e amplamente utilizado, assim como o software e a unidade de controlo de voo. Este fator é relevante e torna a transição para a realidade mais suave. É desenvolvido um algoritmo evolucionário para treinar um controlador, que utiliza *behavior trees,* e realizados diversos testes.

Os resultados demonstram que é possível evoluir um controlador em ambientes de simulação realistas, utilizando um VNT simulado mas real, assim como utilizando as mesmas unidades de controlo de voo e software que são amplamente utilizados em ambiente real.

Palavras-chave: Algoritmos evolucionários, Veículos não tripulados, Simulações realistas, controladores com *behavior trees*.

Abstract

Nowadays, Unmanned Vehicles (UV) are widespread around the world. Most of these vehicles require a great level of human control, and mission success is reliant on this dependency. Therefore, it is important to use machine learning techniques that will train the robotic controllers to automate the control, making the process more efficient.

Evolutionary strategies may be the key to having robust and adaptive learning in robotic systems. Many studies involving UV systems and evolutionary strategies have been conducted in the last years, however, there are still research gaps that need to be addressed, such as the reality gap. The reality gap occurs when controllers trained in simulated environments fail to be transferred to real robots.

This work proposes an approach for solving robotic tasks using realistic simulation and using evolutionary strategies to train controllers. The chosen setup is easily scalable for multi-robot systems or swarm robots.

In this thesis, the simulation architecture and setup are presented, including the drone simulation model and software. The drone model chosen for the simulations is available in the real world and widely used, such as the software and flight control unit. This relevant factor makes the transition to reality smoother and easier. Controllers using behavior trees were evolved using a developed evolutionary algorithm, and several experiments were conducted.

Results demonstrated that it is possible to evolve a robotic controller in realistic simulation environments, using a simulated drone model that exists in the real world, and also the same flight control unit and operating system that is generally used in real world experiments.

Keywords: Evolutionary algorithms, Unmanned vehicles, Realistic simulations, behavior tree controllers.

Contents

CHAPTER 1

Introduction

The study of automatic devices goes back to 10-70 A.D. during the influence of Hellenistic thinking and knowledge of science, with creations such as human statues that moved to imitate humans or devices to open and close temple doors. Since then several inventions were created, and it is known that Leonardo da Vinci (1452 – 1519) designed a programmable robot and moving devices (Rossi et al., 2009). Nowadays, Unmanned Vehicles (UV) have an important role in our lives and the investment in this field is growing each day. In 2021 Unmanned Aerial Vehicles (UAV) overall market was estimated to reach 27.4 billion dollars, and it is expected to reach 58.4 billion dollars in 2026, resulting in a compound annual growth rate of 16.4% (Markets and Markets, 2021).

Several tasks can be performed using UV or even Swarm Robots (SR), where multiple robots work together by building beneficial structures and behaviors that are comparable to those seen in natural systems. The application of these systems can improve areas such as surveillance, search and rescue, military applications, agriculture, among others [3][4]. UV operation, and specifically, SR, brings numerous advantages. Some of the benefits of SR are flexibility, material risk instead of human, scalability, robustness, decentralized control, and self-organization (Oh & Suk, 2013).

Most UVs still require a great level of human control, and mission success is directly reliant on this dependency. Examples of indirect control methods are usually based on manual programming and behavioral models. For specific and simple tasks this approach may have good results, however, with the increase in task complexity manual programming may not be viable. The disadvantages of the dependency on human control are even more evident in the usage of SR.

Swarm intelligence is present in nature, as shown in the collective activities of insects and other swarming animals. Applications of SR inspired by this phenomenon have proven good results in several applications (Oh & Suk, 2013). One of the most important aspects of these systems is their ability to cooperate and evolve dynamically, understanding the surrounding environment and adapting. Therefore, artificial intelligence began to have a great impact on SR.

Evolutionary strategies may be the key to having robust and adaptative UV and SR. The evolutionary approach in single UV was presented in 1992 with the presentation of a neural network to control a walking robot (Lewis et al., 1992). Since then, many other studies involving SR and evolutionary strategies were conducted, resulting in promising outcomes (Rutkowski et al., 2012). Despite the potential of Evolutionary Robotics (ER), many issues still need to be studied and solved (Silva et al., 2015). ER has proven capable of performing simple tasks but fails when complexity scales.

This thesis proposes an approach for solving robotic tasks using realistic simulation and physics software, using controllers and evolutionary strategies, that can be easily scaled for multi-robot systems or SR. The present chapter introduces this thesis motivation, objectives, research method, and structure.

1.1. Motivation

There are still many research gaps regarding ER. The first ER studies focused on developing evolutions online, in robotic hardware (Floreano & Mondada, 1999). This approach has benefits, such as the reliability on each robot and not on external entities to perform the evolution, and also the ability to modify the behavior if environmental or task conditions change. However, online evolution is still difficult to perform in real robotic hardware, as good solutions require large amounts of computation and time. Therefore, controllers need to be evolved in realistic simulations and then transferred to reality.

One of the main issues related to offline evolution is the reality gap. Because they use elements from simulated worlds that are different from the real world, controllers developed in simulations may become ineffective when transferred to the robotic hardware (Silva et al., 2015). One of the approaches proposed to solve this issue is the usage of minimalist simulations and large levels of noise, in order to avoid reaching simulation parameters that are not close to reality. Although this approach has been used in the last decades, it still has not proven to be a good general solution. However, one other approach aims to do the opposite, which is to perform accurate simulations, being as close to reality as possible to minimize the gap that occurs when the controller is transferred to reality. This latter approach still needs to be further analyzed and investigated. Other issues related to ER are the bootstrap problem and deception, where candidate solutions may stagnate, leading to limited controllers and unsuccessful tasks.

Work has been done to solve these issues, such as the decomposition of complex tasks into simple ones, to perform better evolutions (Duarte, 2015). Also, cooperation between each robot of the swarm has been used to achieve good collective results (Romano, 2018). However, both studies need further research, such as the need to increase task and controller complexity to have a better approximation to reality. Most of these works are done following a minimal simulations approach, where many of the simulation features are treated as noise, to prevent the reality gap. This work aims to perform the opposite, which is to conduct accurate simulations and find the balance between large simulation time and the number of generations necessary to achieve good results. By using base behaviors and evolving only its composition, this study aims to understand the sensibility of this approach regarding the reality gap.

1.2. Research Questions

The focus of this thesis is to approach the issues regarding ER and realistic simulation environments, specifically, to evolve controllers and perform accurate simulations, analyzing its impact on the reality gap, where the proposed solution should be easily scaled for multi robot systems or SR. Therefore, this work aims to answer the following research questions:

RQ1: Is it possible to evolve robotic controllers using close to reality simulators and evolutionary algorithms?

RQ2: Is it possible to do so using the same real software that will be used in reality, to make the transition to reality smoother, minimizing the reality gap?

RQ3: Can this solution be used or easily scaled for multi-robot or swarm robot systems?

1.3. Objectives

The main objective of this thesis is to study the evolution of controllers for robotic systems in accurate simulation models, operating on complex tasks and using ER strategies. Also, analysis should be conducted on whether it is possible to use the same software that is used in real drones, and also, if it is scalable to solve different tasks with multi robot systems. This thesis aims to continue the work produced by Miguel Duarte and Pedro Romano[10] [11]. The novelty of the proposed approach is the conduction of these strategies in an accurate simulation model, with the same software and control that is used in reality. Results must be analyzed in order to understand the advantages and disadvantages of this solution. Research should be conducted by following a research methodology that guides researchers to develop effective work.

1.4. Research Methodology

In order to develop relevant and effective research, it is important to follow a methodological approach. The objective of Design Science Research (DSR) is to provide a conceptual process and mental model to create and evaluate artifacts (Peffers et al., 2006)(Peffers et al., 2007). It allows researchers to solve problems, make research contributions, and communicate results appropriately. Figure 1.1 describes the model presented by Peffers et al., which is divided into six activities.

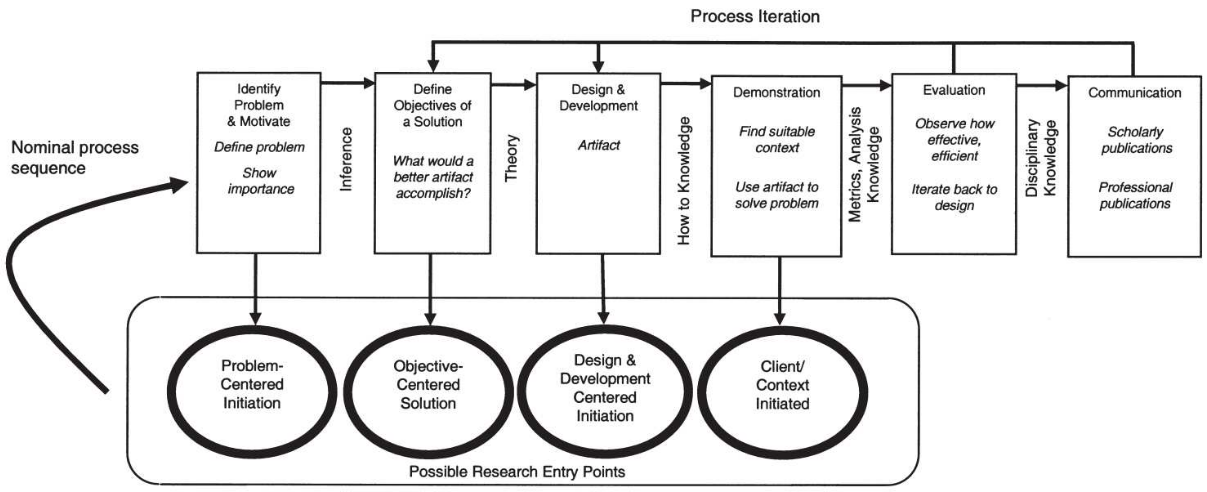

Process Iteration
Nominal process sequence
Identify Problem & Motivate
Define problem
Show importance
Inference
Define Objectives of a Solution
What would a better artifact accomplish?
Theory
Design & Development
Artifact
How to Knowledge
Demonstration
Find suitable context
Use artifact to solve problem
Metrics, Analysis Knowledge
Evaluation
Observe how effective, efficient
Iterate back to design
Disciplinary Knowledge
Communication
Scholarly publications
Professional publications
Problem-Centered Initiation
Objective-Centered Solution
Design & Development Centered Initiation
Client/ Context Initiated
Possible Research Entry Points

Figure 1.1 - DSR methodology process model (Peffers et al., 2007)

The first activity consists of problem identification and motivation. The objectives are defined based on the conclusions of this step. Both these activities are conducted throughout chapter 1, where the problem is stated, research questions are presented so as the objectives.

The third activity is design and development, where the artifact is created. This activity should start by determining the artifact's functionality and architecture, and then its creation. After that, activity four is the demonstration, where experiments should be performed in order to prove that the artifact can solve the problem.

Evaluation is the fifth activity, where observations should be performed to analyze if the artifact can solve the problem. It can use items from the proposed objectives in activity 2. Also, this is an iterative process, as researchers can decide to go back to activity 3 to develop new artifacts or other versions. The final activity is communication, which is accomplished by thesis and paper publications.

Although the methodology is presented in sequential order, it is not necessary nor it is expected that researchers should follow it strictly. As represented in Figure 1.1, a DSR process can start with any one of the first four activities. In this thesis, the DSR starts with activity one, being a problem-centered initiation.

1.5. Thesis structure

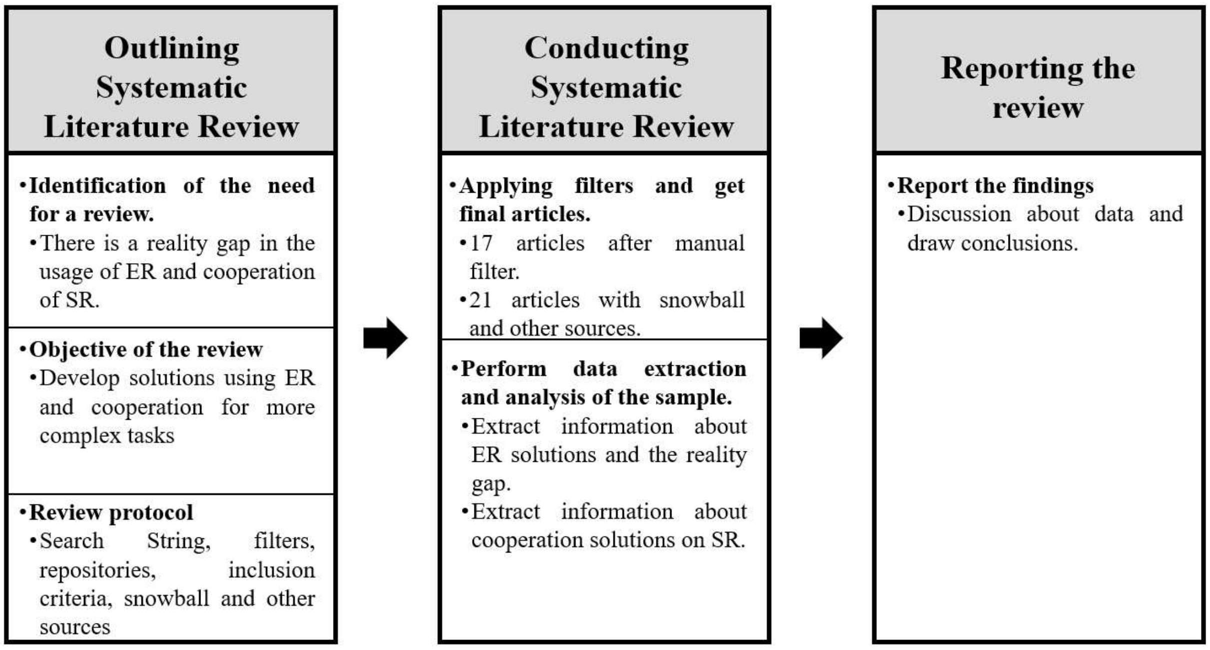
This section provides an overview of the thesis structure. In Chapter 2, a literature review is conducted based on the method presented in the last section. Chapter 3 presents the methodology adopted for the experiments, where the simulation environments are defined. Chapter 4 demonstrates the experiments done, starting with the simple tests with no evolution, and finishing with a parallel simulation engine. Chapter 5 presents the results and analysis of the evolutionary tests. Finally, conclusions are presented, such as suggestions for further work.

CHAPTER 2

State of the Art

This chapter presents the state of the art regarding swarm robotics, evolutionary robotics and cooperation, and its recent research and challenges. The research method used in this process is the Systematic Literature Review (SLR), complementing it with different sources and also a snowball approach. The SLR has proven to be an extremely useful method for researchers, being an organized and evidence-based method. Specifically, SLR usage is growing in software engineering, and studies show that this method is improving the quality of research (Kitchenham et al., 2009). This method can be complemented by building knowledge from combining different sources and findings.

The snowball approach demonstrated to be a good complement to an SLR, having good results in finding relevant literature (Wohlin, 2014). The term snowball refers to the practice of identifying additional articles using an article's reference list or citations. Other sources are also included in the research, such as articles provided by thesis supervisors. Figure 2.1 presents the method applied for this research.



Outlining Systematic Literature Review
•Identification of the need for a review.
•There is a reality gap in the usage of ER and cooperation of SR.
•Objective of the review
•Develop solutions using ER and cooperation for more complex tasks
•Review protocol
•Search String, filters, repositories, inclusion criteria, snowball and other sources
Conducting Systematic Literature Review
•Applying filters and get final articles.
•17 articles after manual filter.
•21 articles with snowball and other sources.
•Perform data extraction and analysis of the sample.
•Extract information about ER solutions and the reality gap.
•Extract information about cooperation solutions on SR.
Reporting the review
•Report the findings
•Discussion about data and draw conclusions.

Figure 2.1 - Research method for state of the art, SLR and snowball

As it is presented in Figure 2.1, the SLR strategy focuses on answering research questions by having a review protocol applied on several highly regarded databases among the scientific community. Research questions for this thesis were already provided in the previous chapter. In order to answer the research questions, the following search string was chosen: "*(("swarm" OR "multi") AND ("robot*" OR "UAV*" OR "drone*")) AND ("evolution*" AND ("cooperati*" OR "control*"))*". The choice of the search string was completed after several trials in numerous scientific databases, resulting in several articles related to the theme. The filters applied to the research are presented in Table 2.1.

Table 2.1 - Filters applied during SLR method

Filter 1	Query All Metadata, all documents
Filter 2	Query Abstracts, all documents
Filter 3	Query Title, all documents
Filter 4	Relevant (inclusion/exclusion criteria)
Filter 5	Remove Duplicate
Filter 6	Irrelevant topics (manual filter)

Inclusion and exclusion criteria are presented in Table 2.2. These criteria allow to have a good amount of papers for the research and simultaneously to keep the latest articles documented in the scientific databases.

Table 2.2 - Inclusion and exclusion criteria for SLR method

Inclusion criteria	Exclusion criteria
Papers written in English	Papers not written in English
Papers since 2012	Papers before 2012
Identified authors	Unidentified authors

Table 2.3 presents the final research table for the SLR, resulting in 37 references after the automatic filters, and 17 after filter 6 which was a manual filter. These were then complemented with 28 references from the snowball approach and other sources. This method resulted in a total of 45 relevant references for the thesis.

Table 2.3 - References for SLR and snowball approach

Database	**Filter 1**	**Filter 2**	**Filter 3**	**Filter 4**	**Filter 5**	**Filter 6**
IEEEXplore	1,559	526	34	16	16	5
EBSCOHost	20,639	102	3	1	0	0
webofscience	1,726	597	40	23	14	7
ACM Digital Library	1,806	9	0	0	0	0
Scopus	115,723	1,768	70	34	7	5
SLR Total	**141453**	**3002**	**147**	**74**	**37**	**17**
		Complement from snowball and other sources				**28**
		Total				**45**

2.1. Swarm robotics

SR can have a huge impact on worldwide economies, due to its numerous advantages, such as its scalability, flexibility, cost reduction, among others. Nowadays, these systems are subject to study by several research groups across the world, and the number of applications is growing. Traditionally robot behaviors have been programmed manually, with methods where the majority of the information a robot need to complete a task is known. However, with the increase of task complexity, it is essential to have controllers that understand and adapt according to the surrounding environment. Thus, it is important to study ways for controlling the systems, and evolutionary algorithms have proven to have good results in this field.

SR usage aims to solve problems collectively, with teams composed of multiple UVs. These systems can operate in several applications, such as search and rescue, military applications, agriculture, environmental monitoring, communications, and others (Grocholsky et al., 2006)(Zhao & Lu, 2012)(Sabino & Grilo, 2018)(Albani et al., 2017).

In military operations, SR is being intensively studied due to its numerous advantages. UV systems may be deployed into dangerous missions and have good performances, resulting in material risk and not human risk. Furthermore, these systems may reduce costs in comparison with traditional methods. One example of this is the usage of SR for large areas coverage, and also search and rescue operations. In this field, several studies show that SR systems are beneficial and deliver promising results (Grocholsky et al., 2006; Zhao & Lu, 2012). Specifically, authors have proven that cooperative and evolutionary approaches may be the key to task success. However, most of these studies were done in basic simulations, and therefore, further research must be done to achieve effective results in more complex simulations and obviously, in the real world.

The Swarm Robotics for Agricultural Applications (SAGA) experiment presented SR systems for agriculture applications, using cooperative approaches and artificial intelligence(Albani et al., 2017). The task was based on field monitoring and detection of weed presence and location, and the study demonstrates that SR operation has several benefits. Some of these benefits are efficiency, robustness, flexibility and scalability. Individuals of the SR had to follow patrolling strategies and analyze images. Usually, this patrol is planned *a priori*, however, this approach does not allow the system to adapt to different circumstances. This project also aimed to perform this patrol in a decentralized way, exploiting visual feedback to direct patrol strategies of each individual. Although the paper provided baseline results that prove that SR systems are beneficial and promising for agricultural applications, there is still a long way in order to bring these experiments to more accurate simulations and into the real world.

Several studies have also been conducted in the communication field, forming network extensions or compensations. Specifically, the usage of multiple UAVs as flying access points in order to create a mesh network (Sabino & Grilo, 2018). The objective of SR application in this area is to optimize the positioning of UAVs to provide the best possible communication coverage. In order to achieve the goal, the authors proposed an evolutionary algorithm to get the best efficiency, deploying the minimum number of UAVs possible. Results have shown that the usage of evolutionary algorithms optimized the SR positioning. However, the results were once more a starting point for the study, and further research and more complex simulations need to be conducted.

2.2. Evolutionary robotics and cooperation

The ER approach is based on the Darwinian principle of selective reproduction of the fittest, but, in this case, a robotic control system is evolved (Floreano & Nolfi, 2000). Usually, evolutionary algorithms are composed of a population with several candidates. Candidates are evaluated, and the best ones are chosen. After this, the best candidates typically suffer variations, which can be mutations or crossover. This process continues with several iterations until a certain condition is reached.

Controllers' evolution is usually done through genetic algorithms and Artificial Neural Networks (ANN). ANN can be generalized as a model that mimics the nervous system, since they have several layers, connected by neurons and synapses. In this case, ANN are evolved and parameters such as the number of neurons, weights, among others, are mutated to find optimal controller systems (Yao & Liu, 1998).

There are relevant beneficial properties when using ANN in robotic systems. Floreano and Mondada presented some of these objectives, which are presented in Table 2.4 (Floreano & Mondada, 1999).

Table 2.4 - ANN properties and their benefits for robotic controllers (Floreano & Mondada, 1999)

ANN properties	Benefits for robotic controllers
Flexibility	ANN flexibility allow the controller to adapt to environmental changes.
Robustness	Missing network links or hardware malfunction do not have a huge impact on the controller.
Noise tolerance	Noise tolerance is an essential property for SR controllers due to their sensors and actuators' intrinsic noise in the real world.
Scalability	ANN can have several different architectures and connections, being extremely relevant for complex mappings and tasks.

Although ER was initially used in individual robots, several studies have been conducted for its application on SR, and some of these are presented bellow. In SR it is also important to achieve cooperation between the elements of a group. Understanding and adapting according to the surrounding environment is the key to having robust and successful controllers.

Whether robotic controllers should be evolved online or offline is a question that has been studied in the last decades. One of the initial ER studies conducted an online evolution, which is the development and evaluation onboard the robot (Floreano & Mondada, 1999). This approach has the main advantage of altering the robots' behavior according to environmental or task changes. However, in ER, the generations and candidates necessary for good solutions are typically large, requiring a fair amount of time and computing power. Furthermore, robots usually have limited computing power and evolution onboard may be challenging (Silva et al., 2015).

Nonetheless, studies regarding onboard evolution have been presented in the last few years (Weel et al., 2012). Specifically, embodied evolution has been approached, where robots test and share several candidates with each other (O'Dowd et al., 2011). However, several evaluations are needed so as a large amount of computing power and time. Hence, online evolution has still remained unfeasible (Silva et al., 2015).

On the other hand, offline evolution is an approach where controllers are synthesized offline, i.e. in simulation. After that, they are transferred to real robots, avoiding the computing and time limitations required during evolution. However, there are still issues with this approach (Silva et al., 2015).

One of the main issues is the reality gap, as controllers evolved in simulation use elements that are different or may not even exist at all in the real world, resulting in several limitations (Jakobi, 1997). Figure 2.3 illustrates the reality gap problem, where a minimalist simulation founds an optimal solution in the simulated environment but had a performance loss when transferred to reality (Koos et al., 2013). Although the reality gap is a problem related to evolving in simulations, Francesca et al. stated that this issue can also be seen in manual design controllers (Francesca et al., 2014). Approaches to solving this problem have been presented in the scientific community, including the addition of noise in simulations, the concept of minimal simulations (Jakobi, 1997), and also the transferability approach (Koos et al., 2013). However, these approaches have limitations and there is no generalized solution to solve the reality gap.

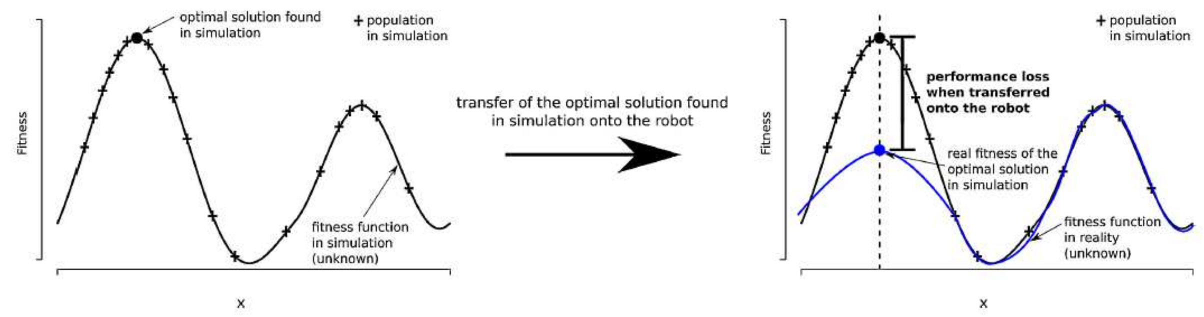


Figure 2.3 - Reality gap problem in a minimalist simulation model (Koos et al., 2013)

As previously referred, the bootstrap problem and deception are also issues related to ER, but not directly related to offline evolution. The bootstrap problem may occur when the mission is too difficult for the fitness function to impose any meaningful selection pressure on a randomly generated population of initial candidate solutions. This can lead the evolution to undesired regions of the search space. As for deception, it may occur when the fitness function fails to establish a gradient that drives to a global optimum solution, but instead, it leads to local optimum solutions. This may stagnate the controller, leading to unsuccessful tasks (Silva et al., 2015).

2.3. ER recent challenges

As previously referred, even though there are studies that demonstrate good results with ER, there are still challenges that need to be addressed. The difficulty to transfer control from simulation environments to reality is still present. Also, ER prove to have good performances in simple tasks but fails in more complex ones. Finally, the bootstrap problem and deception are still issues that may lead to undesired solutions and stagnate controllers. Therefore, it is important to study the state of the art regarding these issues, in order to understand their limitations and present a possible generalized solution.

Koos et al. presented the transferability approach as a method to cross the reality gap (Koos et al., 2013). By stating that in simulation, the most efficient solutions frequently make use of poorly parametrized phenomena in order to accomplish high fitness values with unrealistic behaviors, the authors aimed to optimize fitness and transferability. Unlike several approaches with minimalist simulations, this approach intends to add and make more complex simulations. The objective is to increase controller transfer quality, which can be defined as a measure that compares simulated and real behaviors. This is done by exploiting the best modeled parameters of the simulator and making use of them in the controller. The results demonstrated that this approach can be a promising and relevant method to cross the reality gap in ER. However, there are still work and tests that need to be conducted, such as having a simulation model that works for several tasks and is not task specific. Also, it is important to continue the study of evolving in more accurate simulators, to understand if it is possible to have a few experiments.

Complex problems decomposition into smaller ones has been investigated as an approach to reducing controllers' complexity (Costa, 2018; Duarte, 2015; J. Yang et al., 2012). Regarding this matter, Cooperative Co-evolutionary Algorithms (CCAs) have been used, by dividing a problem into several subcomponents and creating different populations for each one, which is then evolved using ANNs. Results in both simulated and real environments demonstrated that synthesizing cooperative behaviors in addition to ER is a valid and promising approach (J. Yang et al., 2012). However, the scenarios created were minimalist, and it is necessary to validate this approach for more complex tasks.

A hybrid control for large swarms of aquatic drones was presented by Duarte (Duarte, 2015) and Duarte et al. (Duarte et al., 2014), where behaviors were synthesized and evolved. Once more, breaking down a complex task into simpler ones allowed a combination of evolved and traditional preprogrammed control. One of the advantages is the ability to perform better regarding the reality gap, because preprogrammed control can be used when certain actions are too difficult to perform. A hierarchical combination among individual controllers was done, and each controller was composed of both evolved and preprogrammed nodes. Also, the architecture allowed behaviors addition or removal based on the task characteristics, resulting in an adaptative approach. Using this approach resulted in promising results for both the reality gap and the bootstrap problem.

The research done by Tian Yu et al. presented solutions for the food foraging problem, based on ER controllers (Yu et al., 2014). In this case, three evolutionary algorithms were applied and compared. Incremental evolution was also implemented analyzed and compared, which consists in dividing a complex task into several manageable tasks. This latter approach was applied to solve the bootstrap problem in certain cases. Results have proven that neuroevolution based on Covariance Matrix Adaptation Evolution Strategy (CMA-ES) with incremental evolution had the best results. Although this work has demonstrated good results, the reality gap was not addressed, and once more, the simulation was minimalist.

Other approaches for cooperative object transportation and cooperative control using ER have been presented in the last decade (Alkilabi et al., 2015)(Juang et al., 2020)(Jhang et al., 2019). In the work presented by Alkilabi et al., a group of robots had the task of moving an object that was too heavy to be pushed by a single one (Alkilabi et al., 2015). Being aware that one of ER's main issues is the transfer to real hardware, this work was carefully developed in physics engine simulators, and initial tests demonstrated successful results when transported to real robots. Also, all sensors and motors were subjected to a high amount of random noise to ensure that the controllers translated to the real robot had minimal performance degradation. The strategies adopted in this work were successful, additionally, unexpected and intelligent behaviors emerged during evolution. Specifically, robots' tendencies for a recruitment behavior, where one robot approached the closest mate in order to get help to push the object. This suggests that ER not only offers successful expected behaviors, but also boosts systems adaptability. Even though the results were satisfactory, training the controller still took a long time, and work needs to be done to boost efficiency.

Embodied evolution has been presented in the scientific community as an approach to accelerate online controllers' evolution. In this method, each individual belonging to an SR can evaluate several candidates and then transmit the best ones to their mates. Although it was first presented as a possible answer for online evolution, recent studies demonstrated that this approach can be adapted for offline evolution (Trueba et al., 2017). Results proved that embodied evolution is a possible approach for SR optimization. However, once more, the study was limited in complexity. Other approaches regarding online evolution have been presented, such as particle swarm optimization, differential evolution, and multi-objective cooperative co-evolution algorithm have been presented for robot joint control (Roy et al., 2016)(Zheng et al., 2014)(T. Yang & Jiang, 2015)(Xiang-Yin & Hai-Bin, 2012).

Vaughan presented an approach to improve cooperation in SR tasks (Vaughan, 2018). Specifically, the approach improves communication between the elements of a group and uses Finite State Machine to define different navigation behaviors. Comparisons were made with other non-communicating approaches, and the presented work outperformed the traditional one. Regarding limitations, this research was modeled in a simple 2D simulation. Also, the author found counter-productive communications, causing problems and repeating actions between the elements, resulting in diminished efficiency.

Table 2.5 presents the summary of the most relevant articles presented in the recent challenges.

Table 2.5 - Summary of the presented works

References	**Research**	**Limitations**
(Koos et al., 2013)	Transferability approach, more accurate simulations in order to cross the reality gap.	Needs further research, such as in a generalized simulator for several tasks.
(Costa, 2018; Duarte, 2015; Duarte et al., 2014)	Behaviors synthetization and hierarchical control with combination of evolved and preprogrammed nodes.	Needs to be studied in more complex environments and robots.
(Alkilabi et al., 2015)	ER and cooperation strategies with noise addition in order to minimize the reality gap.	Long computation time.
(Trueba et al., 2017)	Adapting embodied evolution for offline approaches.	Limited, minimal simulation.
(Roy et al., 2016)(Zheng et al., 2014)(T. Yang & Jiang, 2015)(Xiang-Yin & Hai-Bin, 2012)	Online evolution strategies for path planning, formation control, among others.	Computing and time limitations regarding online evolution in SR.
(Vaughan, 2018)	Swarm communication in virtual environment combined with FSM navigation control.	Minimal simulation, communication limitations.

Even though these approaches have benefits, none of them prove to be a general solution, as there is still no widespread consensus on how to cross the reality gap in complex environments and tasks. There is still a long way to have ER strategies widespread in real world applications. Thus, this thesis aims to develop a generalized approach to evolve controllers, to operate in robotic tasks in more accurate simulation environments. It aims to continue research related to evolution in accurate simulations and also the behaviors synthetization and hierarchical control.

CHAPTER 3

Methodology

As stated in previous chapters, there are no widespread developments in the usage of accurate simulation models to test ER strategies. This chapter describes the methodology that will be used to test the approach, including the physics simulation software, drone simulation software and drone operating system. Comparisons between different software are done and the global architecture is defined. Finally, the ER strategy that will be tested is presented, including its advantages and disadvantages.

This chapter is divided into different sections as follows: (i) physics simulation software; (ii) drone operating system and simulation software; (iii) drone controller model; (iv) summary.

3.1 Physics Simulation Software

Physics simulators allow users to test their architectures and proposals cheaply and safely, without breaking or degrading physical platforms. This is especially important when one is using ER strategies, as many hundred simulations must be performed to reach a final solution.

Collins et. al. conducted an extensive review of the most used simulators (Collins et al., 2021), comparing several features and also their citations in scientific papers. For the purpose of this thesis, physics simulators for robot applications are defined as an application that runs at least the following functionalities: (i) physics engine for accurate physical modeling; (ii) collision detection functionalities; (iii) Graphical User Interface (GUI); (iv) options to import different scenarios and models; (v) API for at least the most used robotic programming languages (C++/Python). Through the analysis of the extensive work produced by Collins et. al., it is possible to compare several simulators and choose the one that has the best pros and cons concerning the usage of ER and robotic learning strategies.

Table 3.1 presents a summary of the principal simulators and some of their features, especially the ones that are important for ER (Collins et al., 2021). Having open-source availability is important, as it allows users to know the base code, facilitating development and software innovation. It also provides code reuse and recycling, making collaboration easier, normally resulting in higher communities and support. Vision and LIDAR sensors are often an important requirement for almost any complex task, as these are important features for a simulator. To overcome the reality gap, an essential factor is the possibility to vary between each individual simulation, also called domain randomization. This can be done by applying small changes in each simulation, such as random forces to the robot or even random sensor failure.

Some of the simulators, such as Gazebo and CoppeliaSim, support multiple physics engines, which is another randomization technique that may prevent over-fitting to the physics engine. Allowing the use of fluid mechanics is not an ER related feature, but one that can be explored in more complex tasks, allowing future scalability of the present architecture. Finally, Robotic Operating System (ROS) is a widespread open-source framework that allows developers to build and reuse robotic applications. PX4 and Ardupilot are open-source flight controllers used in most of the existing drones. Hence, the ability to have ROS, PX4 and Ardupilot support is also a great feature in a simulator, as it makes the transition from simulation models to real to robots much smoother, making it also an advantage to cross the reality gap.

Table 3.1 - Summary of the most used simulators for ER strategies and its features(Collins et al., 2021)

Simulator	**Open-source**	**Vision and LIDAR sensors**	**Multiple Physics Engines**	**Applying random forces/failures**	**Allow fluid mechanics**	**ROS / Px4 / Ardupilot Support**
Raisim	No	Yes	No	Yes	No	No
Gazebo	Yes	Yes	Yes	Yes	Yes	Yes
Nvidia Isaac	Yes	Yes	No	Yes	No	Yes
MuJoCo	Yes	Yes	No	Yes	Limited	No
PyBullet	Yes	Yes	No	Yes	No	No
CARLA	Yes	Yes	No	No	No	Yes
Webots	Yes	Yes	No	Yes	Limited	Limited
CoppeliaSim	Yes	Yes	Yes	Yes	No	Yes

Throughout the analysis of Table 3.1, Gazebo may be considered the physics simulator that is best fitted for ER strategies. Gazebo is an open-source robotics simulator used in a wide range of mobile ground, aerial, and water vehicles, designed to have 3D dynamic environments and realistic physics engines (Koenig & Howard, 2004). It has large support and an active community, and allows the usage and creation of several vehicles, sensors and other models. It fully supports ROS, Ardupilot and PX4, which can be integrated for Software In The Loop (SITL) and Hardware In The Loop (HITL) simulations, i.e. simulation modes where normal PX4 or Ardupilot run on real flight controller software or hardware, respectively (Ardupilot Dev Team, n.d.; PX4, 2022). Both SITL and HITL are of extreme importance when crossing the reality gap, as the software or hardware that runs in real drones is used in simulation.

Gazebo has multiple packages and it is also used in many research about online evolution and also in reinforcement learning techniques (Collins et al., 2021; Nordmoen et al., 2018). However, to the date of this research, there is still no generalized solution for the usage of Gazebo for offline ER controllers strategies, hence, to the best of our knowledge, this thesis proposes a novelty approach to this matter. This simulator also has in-built support, such as the ability to reset a simulation environment programmatically. All these features make Gazebo the appropriate choice for the architecture presented in this thesis.

3.2 Drone Operating System and Flight Controller

Choosing the right drone operating system and flight controller is also an important decision in the architecture, as it directly impacts the transition from simulation environments to reality. The principal concern is to have a general operating system that allows vehicles for ground, aerial and water environments, and that has extensive support and compatibility with several robotic frameworks.

As previously stated, ROS is an open-source software development kit for robotic applications. It was designed based on the following principles: peer-to-peer, as it does not rely on a single central server, resulting in better scalability for large groups of robots; tools-based, resulting in a simpler operating system design; multi-lingual, providing support for four different programming languages, including Python and C++; thin, as it places the complexity in libraries, encouraging users to develop in small standalone libraries and resulting in easier code extraction and reuse; free and open-source, with full source code being publicly available, facilitating debugging at several levels (Quigley et al., 2009). Due to all these characteristics, ROS has a large community all over the world, with support and compatibility with several external libraries, being the choice for the present architecture.

The choice of the appropriate flight controller for this architecture is also a vital decision, as it should be compatible with the simulator and operating system. PX4 and Ardupilot are both compatible with Gazebo and ROS and are widespread in the robotic market. They also provide SITL and HITL tools, which are ideal to use in simulation environments and to attempt to minimize the reality gap in the transition to the real world. SITL is used to run a flight controller and simulate a drone, without hardware, being a cost-effective option to test software and catch bugs. It is usually employed when testing specific parts of the code, e.g. the controller code. On the other hand, HITL is a methodology that allows the connection of controller hardware, to the simulation environment. This hardware is called a companion computer and there are several examples such as a *Raspberry Pi*, or an UP companion computer, which is represented in Figure 3.1. It simulates the electrical behavior of sensor the and actuators' Input/Output to check the controllers' results. Thus, SITL is normally used in the early stages of code development, while HITL is typically used in later stages (Syed Ahamed et al., 2018).

As previously stated, this thesis aims to develop an architecture to develop ER strategies in SR robots. Because it aims to develop controller code, including evolutionary and decision making algorithms, SITL will be used. Due to its compatibility with Gazebo and ROS, Ardupilot's SITL is the choice for the architecture. One of the main advantages of this software is that it can simulate many different types of vehicles, such as multi-rotor aircraft, fixed-wing aircraft, ground vehicles, and underwater vehicles, among others (Ardupilot Dev Team, n.d.). It also provides several sensors such as optical sensors, lidars, and allows the addition of new vehicles or sensor types.

Figure 3.1 - UP Companion computer, could be used in this architecture with HITL (UP Shop, 2022)

The communication between the Ardupilot and ROS is done via MavLink, which is a drone lightweight communication system. ROS treats MavLink messages using the MAVROS package, which makes the interface for sending and receiving data between the Flight Control Unit (FCU) and ROS. MAVROS can be used in a companion computer in the drone, but also in the Ground Control Station (GCS) (Open Robotics, 2018).

3.3 Drone controller model

The drone controller is where the decisions will be made and evolved. As previously stated, hierarchical control brings several advantages, as it simplifies complex tasks reducing them into several simpler ones. Some examples of these decisions can be high-level decisions such as an order to return to base, or low-level decisions such as drone rotation or altitude change.

Several models can be used to control drones. Some of the most used are Finite State Machines, Decision Trees, and Behavior Trees.

Finite State Machines' advantages include a common structure and the fact that they are simple to understand and implement. On the other hand, they are hard to scale, and they lack in reactiveness to changes. Also, another disadvantage is its reusability, as it is difficult to reuse the same sub-Finite State Machine in multiple scripts.

Decision trees represent a list of sequential if-then decisions used to reach a final action. Modularity, hierarchical structure and being intuitive are some of their benefits. However, tree nodes do not retrieve any information, making them not so reactive to changes or failures. Figure 3.2 presents an example of a decision tree for a robotic task (Colledanchise & Ögren, 2017).

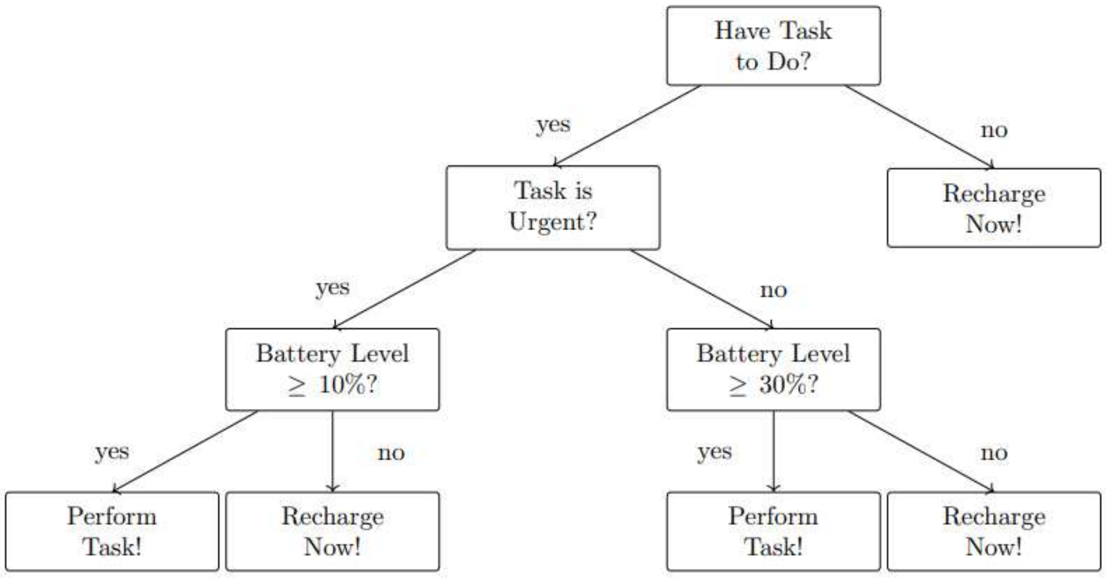


Figure 3.2 - Example of a Decision Tree for task or recharge actions (Colledanchise & Ögren, 2017)

Behavior Trees can be seen as a derivation from Decision Trees and Finite State Machines, with significant improvements (Colledanchise & Ögren, 2017). Some of its improvements are the reaction to failures and modularity. Reactivity is an important characteristic of robotic applications, as in the real world the environment is always changing and being affected by external agents. Behavior trees are naturally reactive, as their nodes can return different states and they are constantly connected with the world state through condition nodes. Behavior tree modularity allows them to be separated and recombined simply, making them a good choice for hierarchical structure models. On the other hand, behavior trees can be complex to implement, because of their ability to have information flowing from nodes. However, the engine only has to be built once, and there are several open-source libraries with its implementation that can be used (Colledanchise & Ögren, 2017). Hence, due to its modularity, reactive and hierarchical structure, behavior trees will be used to implement our controllers.

Behavior Trees are initialized with a root node that generates signals called *ticks* with a specific frequency. Each child node is executed only when it receives *ticks*. Then, the child node returns to its parent node the action status *Running* if the action is being executed, *Success* if the action was successfully executed, or *Failure* if the action failed. In traditional behavior tree implementation, there are four flow control nodes: *Sequence*, *Fallback*, *Parallel,* and *Decorator*. Additionally, there are also two execution nodes: *Actions* and *Conditions* (Colledanchise & Ögren, 2017).

Sequence control node is where child nodes are executed from left to right until one of them returns *Failure* or *Running*. When this happens, the next child node is not executed and the *Sequence* flow control node also returns *Failure* or *Running*, respectively, to its parent. This flow control node only returns *Success* when all its children have also returned *Success*. Typically, the *Sequence* flow control node is represented by a "->" character.

The *Fallback* node *ticks* the child nodes from left to right, until one of them returns *Success* or *Running*, returning this information to its parent node. This flow control node returns *Failure* only if all its children have also returned *Failure*. The *Fallback* node only *ticks* the next child when the present one returns *Failure*. Usually, its symbol is the character "?".

When the *Parallel* node runs, it *ticks* all its children. If a certain number of children nodes return *Success*, the *Parallel* node also returns *Success*. Otherwise, if another number of children nodes return *Failure*, this flow control node also returns *Failure*. Both of these numbers are specified by the developer. If none of these conditions happen, the node returns *Running*.

Action nodes perform a certain command, such has landing or turning left. They return *Success*, *Failure,* or *Running*, and are usually represented by a box. *Condition* nodes check a certain expression or state, and only return *Success* or *Failure*. A *Decorator* node is used to manipulate the return of a certain child, e.g. a *Decorator* node that inverts the return of a certain child.

Table 3.2 presents a summary of behavior tree structure and symbology.

Table 3.2 - Summary of behavior tree nodes and symbology (Colledanchise & Ögren, 2017)

Node	Symbology	Success	Failure	Running
Sequence	→	When all children succeed	When one child fails	When one child returns running
Fallback	?	When one child succeeds	When all children fail	When one child returns running
Parallel	⇉	When X child succeeds (defined by the developer)	When Y children fail (defined by the developer)	If it didn't return Success or Failure
Decorator	◇	Developer option	Developer option	Developer option
Action	▭	When completed	If failed to complete	When running
Condition	◯	If True	If False	Does not run

Figure 3.3 demonstrates the same mission executed in the decision tree represented in Figure 3.2, but in this case, with a behavior tree (Colledanchise & Ögren, 2017). It is important to refer that in this structure the robot does not simply execute tasks, but it continually monitors the tree, at *tick* frequency defined by the developer, checking all conditions and executing other actions if necessary. One of the advantages is that the behavior tree structure allows the robot to turn back in certain actions if a more important condition happens. For instance, in the structure shown in Figure 3.3, the robot may be performing an urgent task, as its battery is above 10%. However, if during the task execution the robots' battery reaches 10% or below, the robot will stop the task and start to recharge. The same would not happen in the decision tree structure.

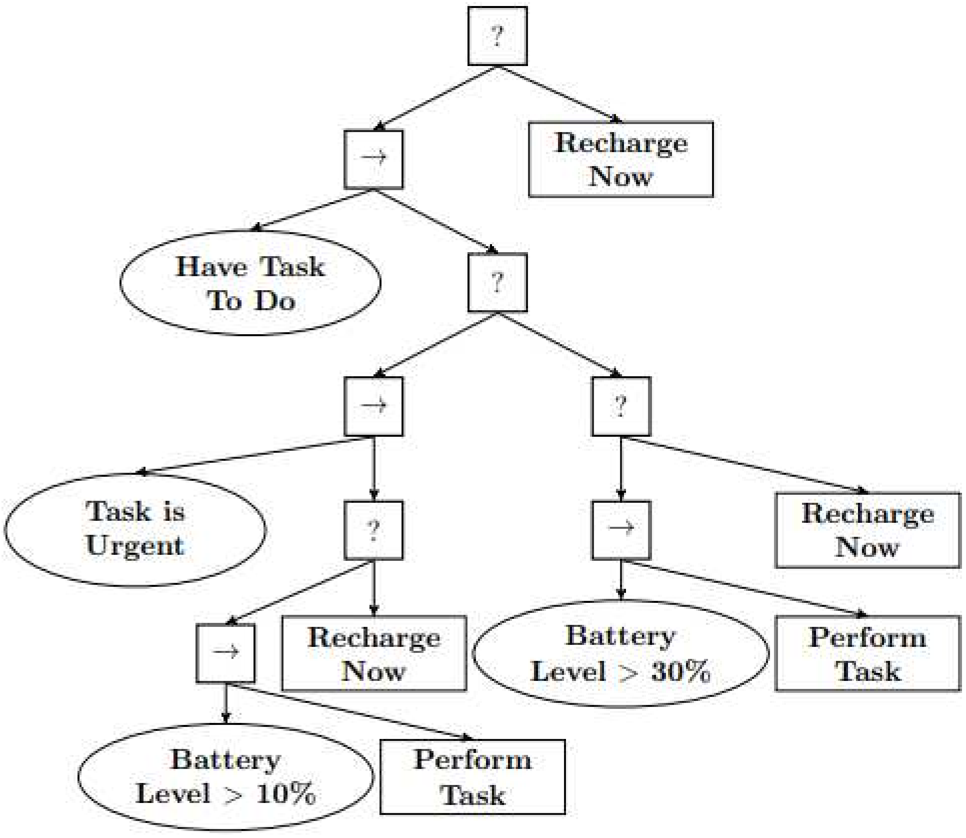


Figure 3.3 - Example of a Behavior Tree for task or recharge actions (Colledanchise & Ögren, 2017)

3.3.1 Behavior tree evolution

Knowing the behavior tree architecture, it is important to understand how can they be evolved using evolutionary algorithms. There are different methods to use evolutionary algorithms to train behavior trees, and the difference is the following: (i) keeping the tree structure and evolving tree parameters; (ii) turning the tree dynamic and applying crossover between tree nodes.

In the first method, the behavior tree structure is static, i.e. the nodes don't change their place in the structure. The tree structure is initially defined by the user, and the evolution will occur in the node parameters. For instance, in a node that will change the altitude, the parameter to be evolved can be the altitude value, in meters. This type of evolution is smoother and simpler to analyze, as it is possible to directly compare parameter values between individuals during all the generations and to obtain conclusions about these values. Because of this fact, this method will be used in the experiments described in the following chapters.

In the second method, the behavior tree is dynamic, as the nodes may change their place in the tree or subtrees. The most basic evolution can occur when doing crossover between subtrees, where an individual's subtree is switched with another individual's subtree. This can be seen in Figures 3.4 and 3.5. It is also possible to evolve behavior trees by randomly deleting one or more nodes. This is especially important when trees start to increase in size significantly, becoming complex to read and slower to run. Additionally, behavior trees can also be evolved by checking for empty operation regions, which are regions that will not be executed because the condition is already met in previous nodes.

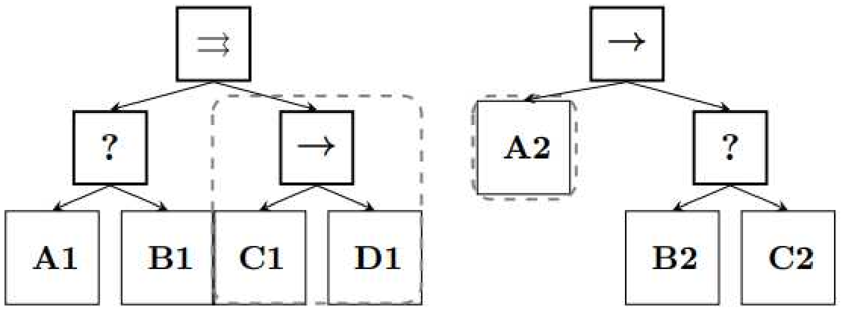


Figure 3.4 - Behavior trees before crossover (Colledanchise & Ögren, 2017)

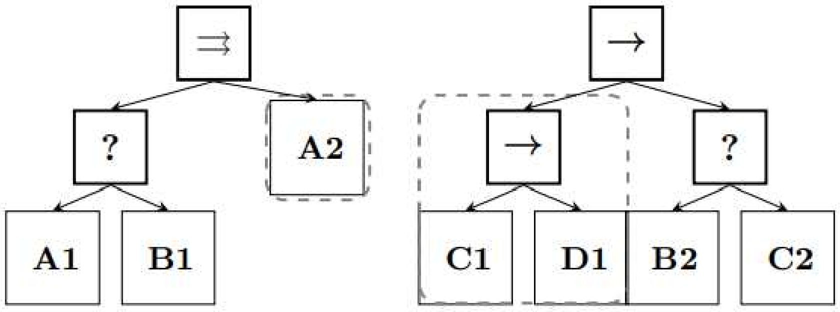


Figure 3.5 - Behavior trees after crossover (Colledanchise & Ögren, 2017)

3.4 Summary

To summarize, Figure 3.6 presents the general architecture defined for this thesis. This architecture allows the creation of robotic applications, such as evolutionary algorithms scripts or drone controllers, using ROS and MAVROS interface, and testing it in simulation using Gazebo and Ardupilot SITL. The behavior tree model and an evolutionary algorithm will be implemented in the ROS environment, and the evolution will occur in the node parameters, keeping the same behavior trees' structure in all individuals.

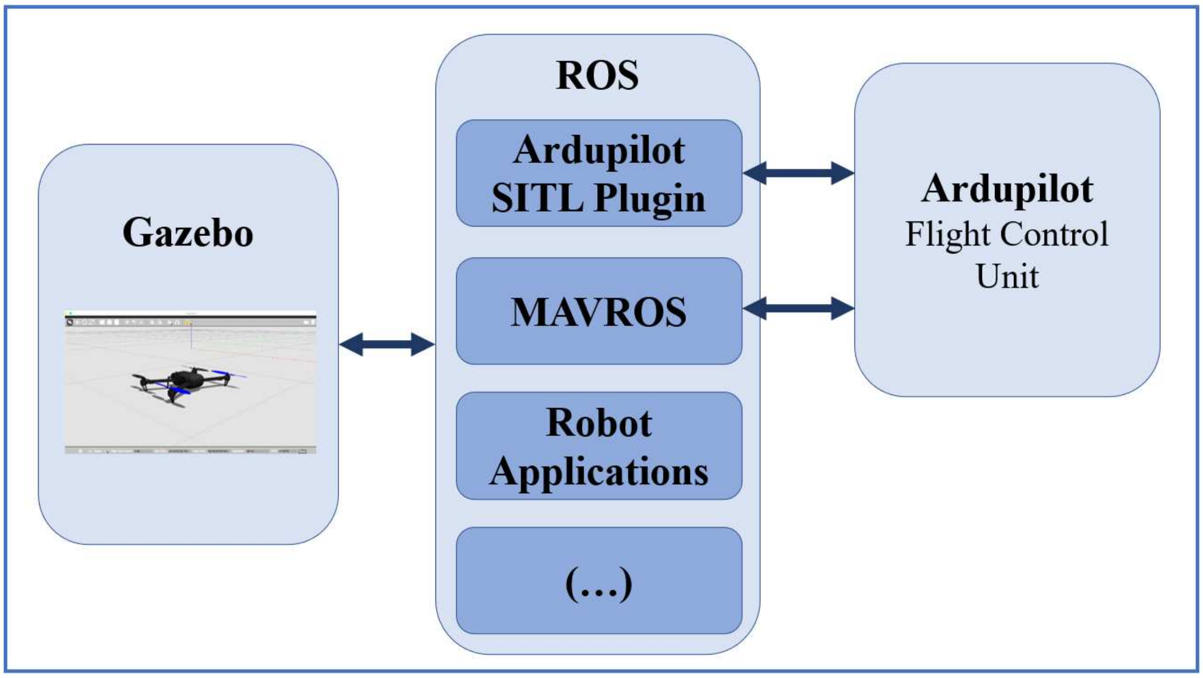


Figure 3.6 - Architecture for the experimental setup

CHAPTER 4

Experimental Setup

The previous chapter presented the architecture that will guide the experiments. Thus, this chapter presents the controller setup and design, and also the specific experiments conducted, from the initial to the final setup.

It is divided into different sections as follows: controller setup and design, where the behavior tree controllers are explained; experimental setups, where the performed experiments are explained.

4.1 Controller setup and design

As stated before, behavior trees will be used to control each drone. In this case, one individual will represent one robotic controller, which is one behavior tree. Each one will have certain actions that can be parameterized. For instance, one behavior tree with the objective of returning to the base can have a move forward action, turn left, turn right, among others. All these actions can have specific values to move forward (in meters), to turn left or right (in degrees), respectively. These will be the values that will differentiate each individual, and the ones that will be evolved and evaluated.

4.1.1 Behavior tree design

There are several methods to design behavior trees. One of the most known is backchaining, where the robot is deliberative and completes certain actions until the goal is reached. This method proposes to work the trees backward from the goal, with a Postcondition-Precondition-Action (PPA) approach, represented in Figure 4.1 (Colledanchise & Ögren, 2017).

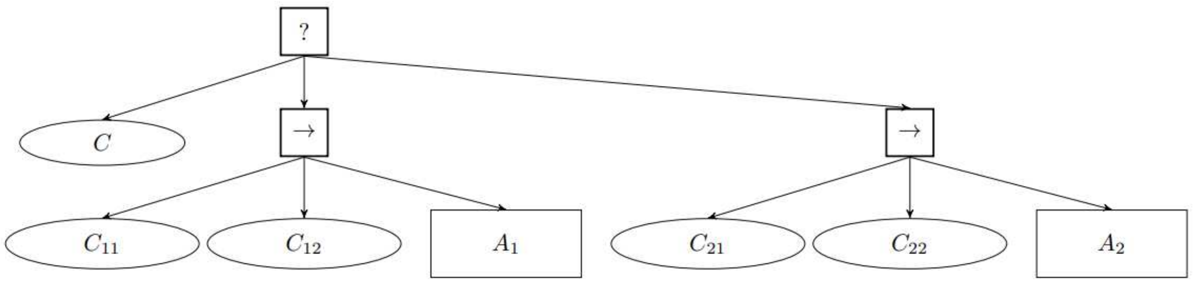


Figure 4.1 - Behavior tree design with Postcondition (C), Precondition (Cij), and Action (Ax) (Colledanchise & Ögren, 2017)

In this approach, postcondition C can be achieved by one of the actions presented by Ax. These actions are defined in a *sequence* node, having its preconditions Cij. To increase efficiency, actions with higher success probabilities should be placed first, avoiding unnecessary failures (Colledanchise & Ögren, 2017).

Figure 4.2 presents an adaptation of the previous method. This behavior tree will be used in the first evolution experiments, and its objective is to return to the base. As previously stated, the use of a behavior tree will provide modularity to the controller, and therefore, this tree can be used as a smaller subtree for a more complex mission. It also provides a clearer and simpler representation of the structure.

As previously referred, although the behavior tree engine may be complex to implement, there are several open-source engines that can be used to fit user needs. In this case, it was used the *task_behavior_engine*, developed by the Toyota Research Institute (Toyota Research Institute, n.d.). Regarding the high-level robotic and navigation tasks, it was used the *iq_gnc* and *iq_sim* packages (Johnson, n.d.).

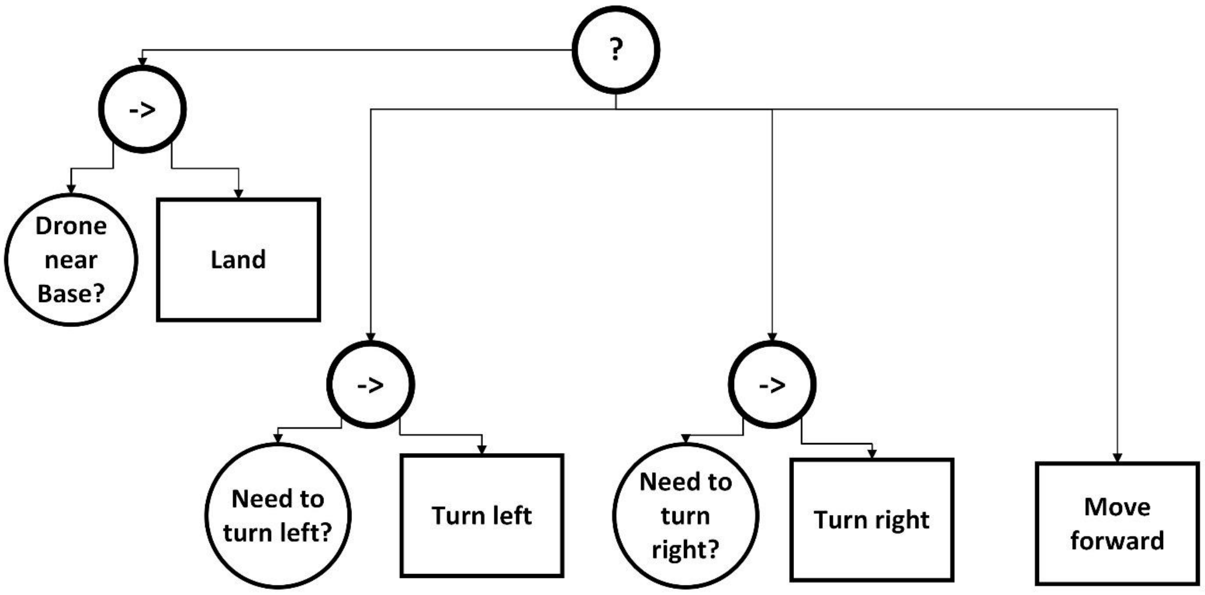


Figure 4.2 - Behavior tree structure to return to the base mission

Because it must be evolved, it is necessary to represent each individual in a simple way, that can be manipulated by the evolutionary algorithm. In this thesis, the evolution will occur only in the actions' parameters, specifically in their values, and hence, each individual will have the same behavior tree structure. For instance, in the example shown in Figure 4.2, the only difference between individuals will be the evolvable parameters, which are the degrees to turn left, degrees to turn right, and degrees to move forward.

Therefore, since the behavior tree presents the same structure in each individual, it can be represented by a list, where each column is the value of the parameter. It is also important to normalize the evolvable parameters, so the evolutionary algorithm can perform calculus equally for each parameter. Then, the parameters are passed to the behavior tree and transformed, based on the minimum and maximum values defined on the behavior tree nodes. The normalization equation is based on the min-max equation, and it is represented in equation (4.1). Hence, each tree is defined in a list, such as the one exemplified in Table 4.1.

$$Xnorm = \frac{X - Xmin}{Xmax - Xmin} \tag{4.1}$$

Table 4.1 - Representation of the behavior tree in the first return to the base task. In this case, there are three evolvable parameters.

	Individual Nr.	**Turn left (Degrees)**	**Turn right (Degrees)**	**Move forward (meters)**	**Fitness (evaluated after an individual run)**
Normalized	0-Nr. of individuals	0-1	0-1	0-1	0-1
Real values	0-Nr. of individuals	1-40 degrees	1-40 degrees	1-40 meters	0-1

4.1.2 Evolutionary algorithm design

To fit the simulation setup, an evolutionary algorithm was designed. The script was created in *Python*, and it performs the following actions:

1. Create a population with a certain number of individuals, specified by the user. Each individual begins with random evolvable parameters;
2. Runs and evaluates each individual, based on a fitness function;
3. Orders each individual based on fitness values;
4. Selects a certain percentage of the best individuals, in these experiments, it will be 10%. Adds these individuals in the next generation;
5. The rest 90% of individuals are created through mutations on the best 10% of individuals. These mutations are done with small changes based on their parent. In this case, one random parameter is chosen, and its value is randomly increased or decreased by a random value between zero and 10%, i.e. 0-0.1 in normalized values;

6. Iterates the previous steps 2 to 6 until the evolution reaches the number of generations indicated by the developer.

4.2 Experimental simulation setup

As previously stated, the Gazebo simulator allows developers to have closed to reality simulations, including the simulation of the physics forces applied to the robot. The first action developed in the experimental setup was the creation of a ROS workspace. A ROS workspace is a set of directories where packages are installed, created, modified and built. The official ROS workspace is called Catkin Workspace (D. Thomas, 2017).

The catkin workspace is where all the robotic applications are built, so as the drone models and world files. Because gazebo and ROS are completely compatible, it is possible to launch gazebo world models and drone models through the catkin workspace.

Gazebo world models can be created from an empty world, or one could also modify the existing ones. In the world files, it is possible to add shapes such as boxes or spheres, but it is also possible to import several models that exist in the gazebo database, such as houses, persons, house objects, and much more buildings or objects that may exist in a real environment. These objects may be added to the world file and parameterized in their position, static or dynamic behavior, and collision with other models, among others.

The vehicle model is also defined in the world file. In this case, the vehicle model should be using ardupilot flight control, so as the gazebo-ardupilot SITL plugin. Using gazebo, it is possible to insert and control several autonomous vehicles in only one world, which is an extremely important factor for multi-robot systems and SR, making the solution scalable for these systems. Figures 4.3 and 4.4 present the gazebo world scenario, where the first evolution tests, presented in Subsection 4.2.2, will run.

It is important to mention that it is possible to create any drone model in gazebo. This feature is of extreme importance when crossing the reality gap, as it is possible to replicate the real world drone in simulation, making the simulation closer to reality.

Hence, in this thesis, the drone model that will be used is the Iris Quadcopter model, which is a computational model from the real copter, available in on the market (ArduCopter UK, n.d.). The real Iris Quadcopter model is presented in Figure 4.5, and, on the other hand, the simulation drone model is presented in Figure 4.6. This model runs the ardupilot software in simulation, but also in the real world. As previously mentioned, this fact is extremely significant, as the drone in simulation is using the same exact software that would run in realty, making the transition to the real world smoother. Naturally, this is also an essential advantage when crossing the reality gap.

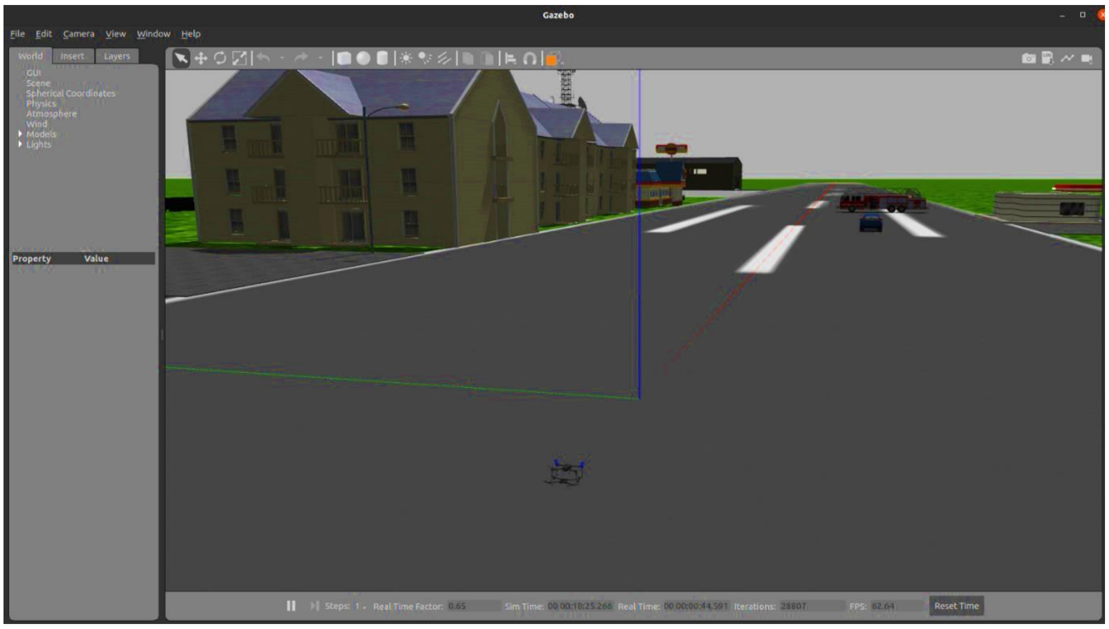


Figure 4.3 - Gazebo simulated world, with visible Iris drone model

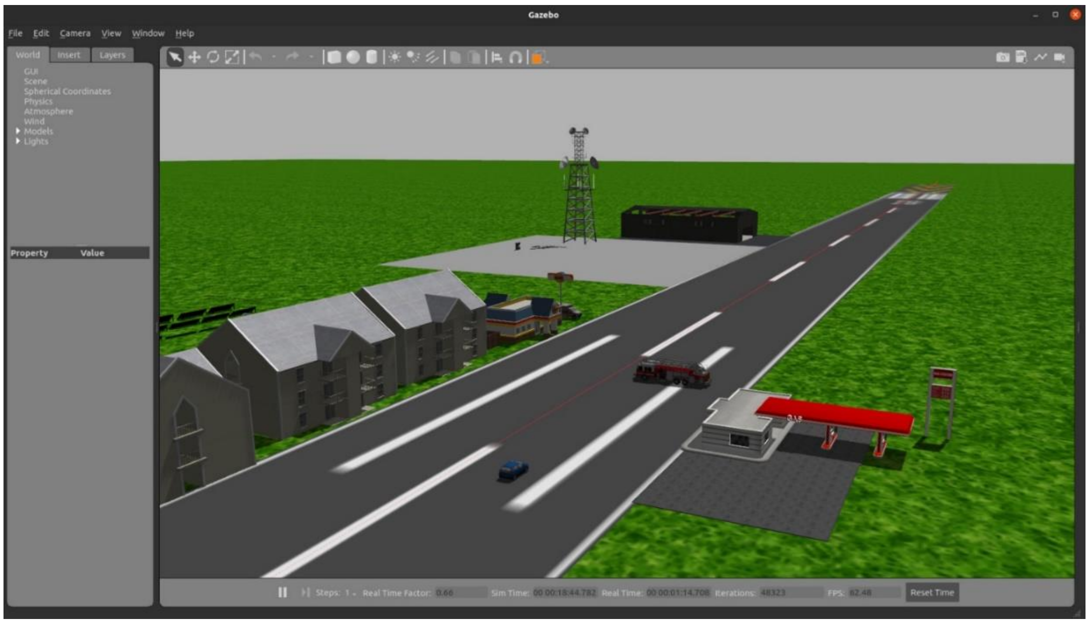


Figure 4.4 - Gazebo world model with another view

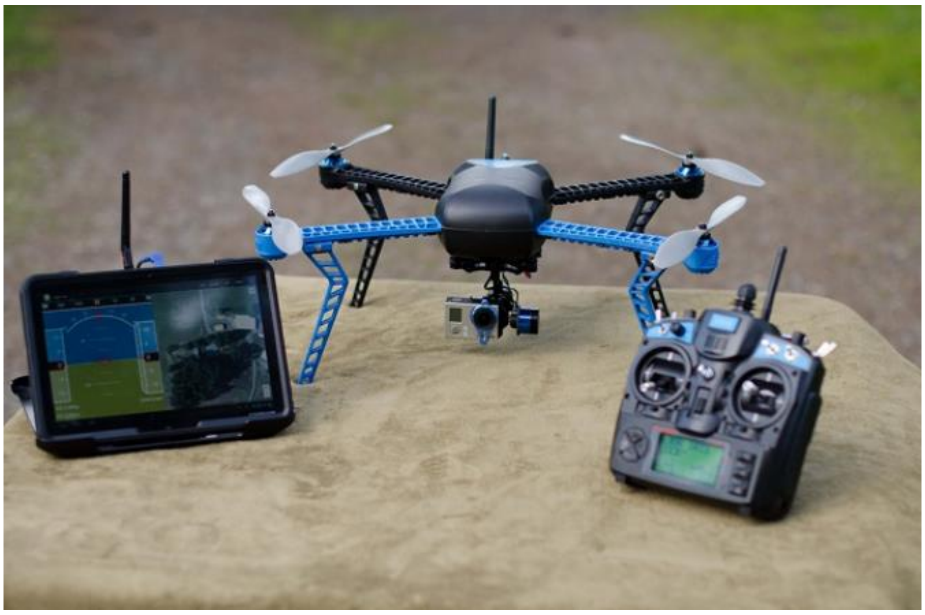


Figure 4.5 - Real Iris Quadcopter Model, which runs ardupilot (A. Thomas, 2016)

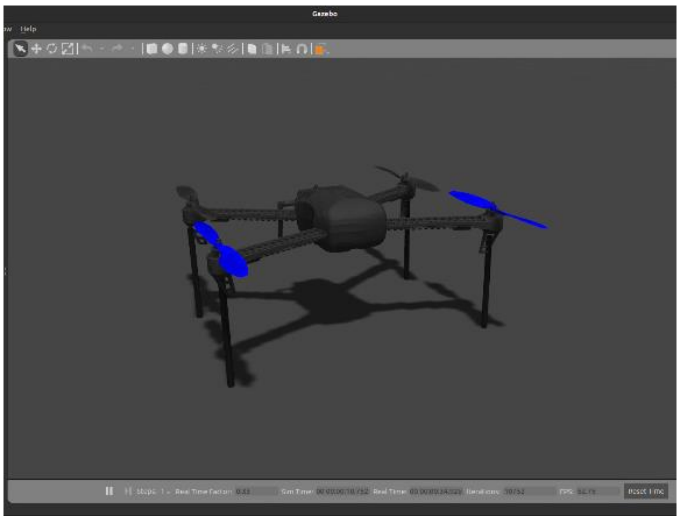


Figure 4.6 - Iris drone model in Gazebo world simulation

4.2.1 Initial experiments with no evolution

The main objective of the initial tests that will be conducted in the proposed architecture is to test its functionalities and possible missions to be evolved. The first missions performed simple actions with one drone without evolutionary algorithms, such as taking off, doing a certain movement pattern, and then landing.

Then, task complexity increased and more drones and sensors were added, such as cameras and lidars. More complex missions were conducted, including search and rescue, where a drone would takeoff, do a search and rescue pattern, look for a person and then land. Tests have also been done using lidar sensors, specifically, using two drones and testing collision avoidance algorithms (Ribeiro, 2005).

For image detection on camera and object identification, it was used the *darknet_ros* package (Bjelonic, 2018). This is an open-source ANN framework that can be run on CPU and GPU. It has a light version package for the case that is used in a drone companion computer. Additionally, the ANN can also be trained to detect the objects the developer wants.

These initial tests were positive as they allowed the execution of several libraries and tasks. They also allowed to test several Python scripts that will be essential to define the experiments that will be evolved with the evolutionary algorithms.

4.2.2 Experiments with evolution

After the initial experiments, it is important to test the evolutionary experimental setup. As previously stated, gazebo provides tools to reset the world and its models programmatically. It also permits the developer to speed up the simulation, according to the simulation speed allowed by the drone models and by the ardupilot SITL plugin. This is especially important in evolutionary algorithms, as it is necessary to run several simulations to have evolution.

Regarding ardupilot SITL, there is no built-in method to reset the simulator, making evolution more difficult. Hence, a python script was made in order to handle this action, by killing the specific Linux processes associated with ardupilot SITL. Then, several experiments were conducted to verify the script result.

Having the gazebo and ardupilot reset working, it was necessary to test the setup in order to verify the simulation speed and how many generations would be completed in a specific time. It was verified that one single simulation environment takes too much time to evolve. Specifically, a return to the base mission was tested, where the robot had to perform certain actions to complete it. This mission's average execution time was 5 minutes, depending on the generation, as initial generations would take longer. For instance, if one would decide to evolve this mission in 10 generations, with 500 individuals in each one, the execution time would be approximately 17 days and 8 hours. This execution time is not viable for evolution, as several experiments need to be done to minimize it.

There are two options to reduce the execution time: making the single simulation faster enough to reduce the time significantly; creating an engine with several parallel simulations. The first studies focused on making one simulation faster.

Gazebo provides a method to increase the simulation speed at more than real time speed. It also provides a parameter called Real Time Factor (RTF), which relates the simulation and real time. If RTF equals one, the simulation time is running at real time speed. On the opposite, if RTF is higher than one, the simulation time is faster than real time. Gazebos' parameters were defined to have the higher RTF possible, specifically *time update rate* and *max step size.* Regarding ardupilot SITL, it is also possible to define the *SIM_SPEEDUP.* This parameter increases the backend velocity of the SITL vehicle simulator.

However, when changing both these parameters, results showed that the simulation RTF was still limited. After several analyses, it was found that ardupilot SITL – gazebo plugin, which makes the connection between SITL and gazebo, was slowing down the simulation, creating a bottleneck in RTF. This was related to gazebo version and its ardupilot SITL plugin, as it is expected to be treated in the next versions[1]. As it was not possible to speed up the simulation at a significant value, the next try focuses on creating a parallel simulations engine.

4.2.3 Experiments with multiple parallel simulations

Running several parallel simulations requires an engine that will receive data and coordinate the evolution. Thus, a script was created, named *evol_engine,* with the following purposes:

1. Creates the first random individuals, for example, 600 individuals per generation. As mentioned before, one individual is one behavior tree controller that will be experimented and evaluated in simulation;
2. Separates the individuals according to the number of parallel simulations, for example, 6 parallel simulations, 100 individuals per parallel simulation;
3. Launches each simulation script, where the individuals will run. The fitness of each individual will be calculated in each parallel simulation, after the respective individual run. For example, the first parallel simulation will run its first individual, evaluate its fitness, and continue to the next individual;

[1] A new gazebo version was released in the final period of this master thesis, Gazebo garden, released in September 2022 (Open Robotics, 2022). Tests were done by the support community with gazebo and ardupilot SITL plugin, resulting in higher RTF when comparing with classic gazebo version (more than 2x higher).

4. Waits for the conclusion of every parallel simulation, read results, and writes them in a *.csv* file. In the previous example, it waits for the conclusion of 600 individuals and reads their fitness;
5. Joins all 600 individuals and orders them by their fitness, choosing a certain percentage of the best ones. For instance, it chooses the best 10%, which is 60 individuals;
6. Adds the best ones to the next generation and also generates 90% of mutated individuals, based on the best ones. For example, it adds the 60 best individuals to the next generation. It also takes these 60 and mutates them, making small changes, resulting in 540 mutated individuals. The result is 600 new individuals for generation number two;
7. Iterates the previous steps 2 to 6 until the evolution reaches the number of generations indicated by the developer.

4.3 Summary

To summarize, this chapter focused on the controller design and experimental setup. Firstly, it was explained the controller design, which in this case is a behavior tree. It also mentioned the representation and normalization method that would be conducted to apply a machine learning algorithm to these trees. Then, the developed machine learning algorithm, which in this thesis is an evolutionary algorithm, was explained. Finally, the simulation experiments were presented, starting with the simpler ones and finishing with the more complex ones. The obstacles found along this development and simulations were also presented, so as the methods to mitigate them. The next chapter will present the results and analysis obtained.

CHAPTER 5

Results and Analysis

Chapter 5 presents the results and analysis of the experiments where the controller was evolved. The significant experiments were done in three different cases: the first one with only one simulation environment, in a return to the base task; the second one with the same previous task, but with six parallel simulations; the third with a more complex task, search and rescue, where the robot looks for missing persons.

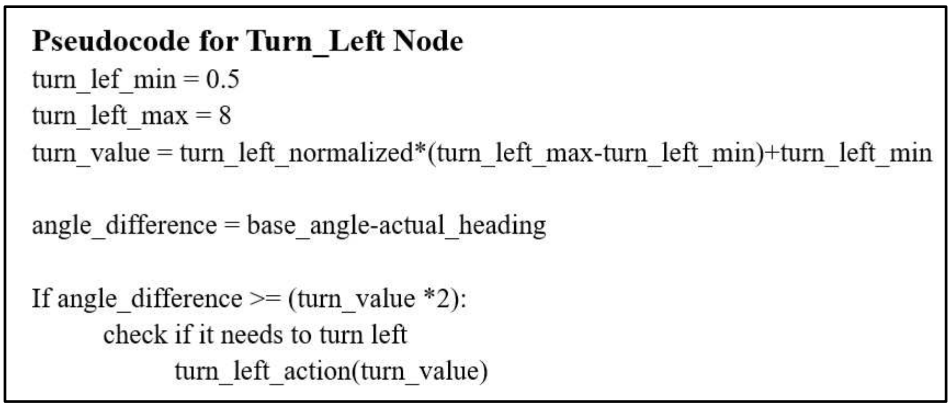
5.1 Return to the base task – one single simulation

As previously mentioned, the first evolutionary tests were conducted with one single simulation. The objective of these tests was to evolve the task of returning to base, where the base is represented by a point in the gazebo world, with specific coordinates. This scenario was shown in Figure 4.4.

The behavior tree designed for this objective had four principal nodes:

1. *CheckBase*, which verifies if the drone is near the base. If so, it lands, otherwise, it returns *failure*. There are no parameters to be evolved in this node;
2. *TurnLeft,* where the drone will check if it is aligned with the base or if it needs to turn left. This calculus is done based on the drone's current position and base coordinates; The parameter to be evolved is the number of degrees to turn left. This parameter is also used in the verification if whether the drone is aligned with the base or not. The pseudocode for this node is shown in Figure 5.1. For this initial test, the minimum and maximum value is defined as 0.5 degrees and 8 degrees, respectively;
3. *TurnRight,* same as the previous node, but in this case, it will check if it needs to turn right. Additionally, in this case, the parameter to be evolved is the *turn_right_normalized*;
4. *MoveForward*, which is the last action, and will move forward if all the other nodes fail, meaning the drone should be aligned with the base according to previous conditions, but it is not near the base yet. In this node, the parameter to be evolved is the number of meters to move forward, where the minimum distance is 1 meter and the maximum distance is 20 meters.

The controller and evolutionary algorithm parametrization are presented in Table 5.1.



```
Pseudocode for Turn_Left Node
turn_lef_min = 0.5
turn_left_max = 8
turn_value = turn_left_normalized*(turn_left_max-turn_left_min)+turn_left_min

angle_difference = base_angle-actual_heading

If angle_difference >= (turn_value *2):
    check if it needs to turn left
        turn_left_action(turn_value)
```

Figure 5.1 - Pseudocode for Turn_left node, in the initial evolution experiments

Table 5.1 – Controller / ER parametrization for the first return to base evolutionary experiments

Controller / ER algorithm setup	**Values / Other commentaries**
Parameters to be evolved	Turn Left (min. 0.5, max. 8), Turn Right (min. 0.5, max. 8), Move Forward (min. 1, max. 20)
Starting individuals	Randomly chosen 1 parameter in each individual, and attributed a random normal value between 0 and 1. The rest of the attributes received a value of 0.5.
Nr. of individuals/generations	20 individuals and 12 generations, a total of 240 individuals
Fitness	Fitness was calculated based on the time drone would take to return to base. High fitness means lower time to reach base. Fitness is 0 when the individual could not return to base after 10 minutes.
Best individuals' percentage	10% of individuals are chosen for the next generation
Mutation	Generates 90% of mutated individuals for the next generation. Mutation occurs by choosing one random parameter and randomly adding/subtracting a value between 0 and 0.1.

The results presented in Figure 5.2 and Table 5.2 show the average fitness and maximum value for every generation, where each one has 20 individuals. The execution time of these 240 individuals was 36 hours.

It is important to mention that the cases where the drone can't arm or takeoff because of any bug on the reset, are declared with 0 fitness. These cases are rare (happens in 2% to 3% of individuals) but have an impact when the number of individuals is low, such as in this case, where there are 20 individuals per generation. Because there are few individuals in each generation, the graphic shows fitness fluctuations in some of the generations. This is especially significant in the transition from generation 5 to 6.

Table 5.2 - Average fitness results and max values for experiments with one simulation, return to the base mission

Generation	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Av. Fitness	0.68	0.82	0.89	0.87	0.88	0.93	0.83	0.85	0.84	0.84	0.84	0.89
Max. Value	0.90	0.913	0.925	0.93	0.935	0.948	0.95	0.953	0.95	0.952	0.953	0.95

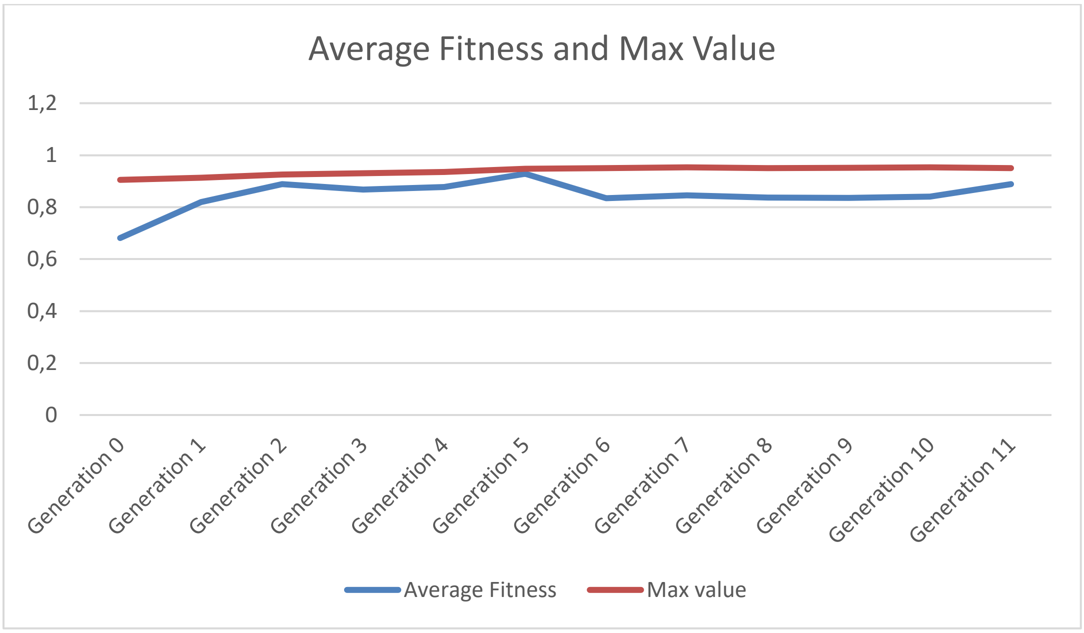


Figure 5.2 - Average and max fitness for the first return to base ER experiment, one simulation, and return to the base objective

Additionally, because the task is relatively simple, high maximum fitness values were obtained right in the first generations. Regarding the maximum fitness value, it was found in generation 10, with 0.953 fitness. The individual is number 211, and the result of the parameters that were evolved was the following:

1. *Turn_left,* with 0.59 normalized value, which equals 4.925 degrees in scaled value;
2. *Turn_right* with 0.65 normalized value, which means 5.375 degrees in scaled value*;*
3. *Move_forward* with 1.128 value, which means 22.432 meters in scaled value.

These results demonstrated that the parameter with the most evolution was *move_forward.* Also, the evolutionary algorithm found the best value to be above the maximum value defined by the user, which was 20 meters. These results are consistent with the simulation scenario, as the base was far away from the drone, and the longer the drone would move forward, the faster he would return to the base. Also, the fact that the individuals started with most of the parameters with 0.5 normalized value helped the fitness and evolution, as they already had an advantage at the beginning of the simulation. Because of this, the individuals in the next tests will be started with all random parameters.

5.2 Return to the base task - multiple parallel simulations

As the tests with single simulations can not run several individuals in an efficient time, it is important to conduct experiments with parallel simulations. The engine used is the one specified in Chapter 4. The objective of this test is to perform a mission to return to base, but in this case, with six parallel simulations where each simulation will run 100 individuals.

To add complexity to the mission, one other node was added to the task, which was altitude order. Also, as previously mentioned, each individual will start with all random parameters, avoiding an initial advantage on the controllers.

The behavior tree design is shown in Figure 5.3, and it has four principal nodes:

1. *CheckBase*, same as previous tests, no parameters to be evolved in this node;
2. *TurnLeft,* same as previous, but this time, minimum and maximum values are defined as 0.1 degrees and 50 degrees, respectively;

3. *TurnRight,* same as previous, but this time, minimum and maximum values are defined as 0.1 degrees and 50 degrees, respectively;
4. *ChangeAltitude,* which verifies if the drone is at the altitude value ordered by the user. The parameter to be evolved is the altitude, in meters, and minimum and maximum values are defined as 1 meter and 80 meters, respectively;
5. *MoveForward*, same as previous, but in this case, minimum and maximum values are defined as 0.1 and 15 meters, respectively.

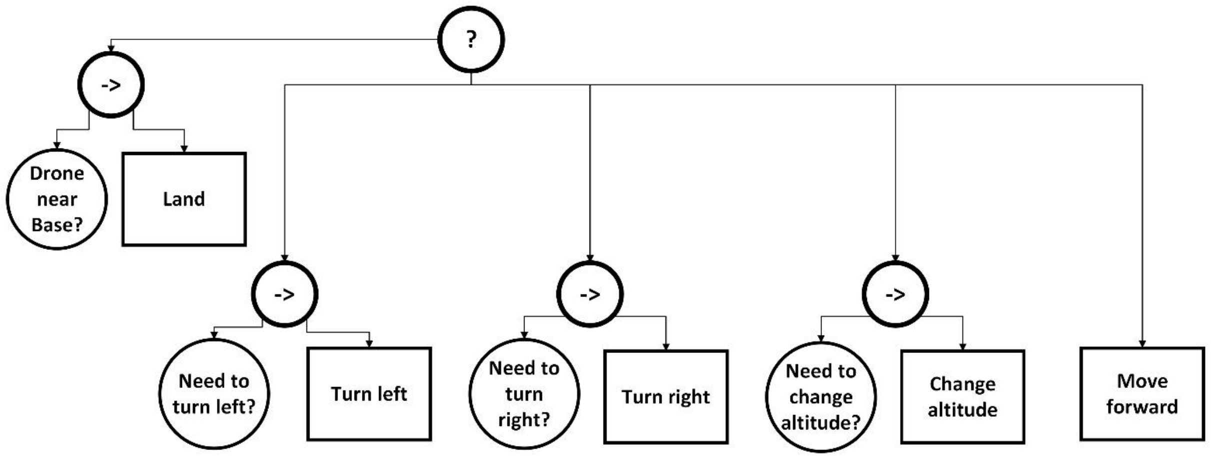


Figure 5.3 - Behavior tree for return to the base mission, with four parameters to be evolved

The controller and evolutionary algorithm parametrization are defined in Table 5.3.

Table 5.3 - Controller and ER parametrization for return to base tests, with multiple simulations

Controller / ER algorithm setup	**Values / Other commentaries**
Parameters to be evolved	Turn Left (min. 0.1, max. 50), Turn Right (min. 0.1, max. 50), ChangeAltitude (min 1, max 80), Move Forward (min. 0.1, max. 15).
Starting individuals	All four individual parameters are started with random values between 0 and 1.
Nr. of individuals/generations	6 parallel simulations, 600 individuals, 100 individuals per generation, 11 generations, total of 6600 individuals.
Fitness	Fitness was calculated based on the time drone would take to return to base. High fitness means lower time to reach base.

	Fitness is 0 when the individual could not return to base after. 10 minutes.
Best individuals' percentage	10% of individuals are chosen for the next generation.
Mutation	Generates 90% of mutated individuals for the next generation. Mutation occurs by choosing one random parameter and randomly adding/subtracting a value between 0 and 0.1.

The results of this controller evolution experiment are presented in Table 5.4 and Figure 5.4. The execution time was approximately 72 hours, which means approximately 4 minutes per individual. If the same would have been done with only one single simulation, it would take about 18 days. Once again, this fact shows the vast advantage of having an engine running parallel simulations.

Table 5.4 - Average fitness and maximum value for parallel simulation tests, return to the base mission

Generation	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Av. Fitness	0.199	0.759	0.792	0.823	0.825	0.868	0.879	0.886	0.875	0.904	0.904
Max. Value	0.944	0.939	0.939	0.942	0.944	0.944	0.925	0.942	0.939	0.944	0.944

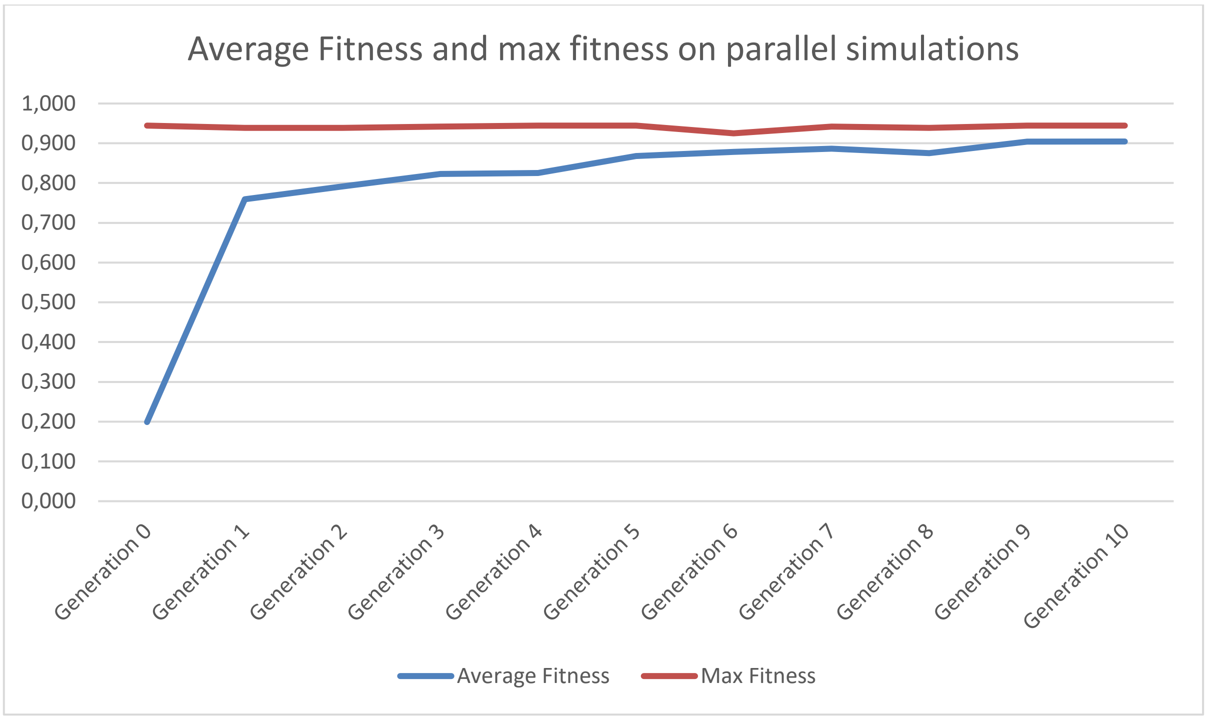


Figure 5.4 - Results of average fitness and max fitness values for parallel simulation tests, return to the base task

The results demonstrated a typical fitness graphic of an evolutionary algorithm. Because in this test all the parameters started with random values, the average fitness in the first generation was 0.199. In this generation, several individuals could not return to base, resulting in 0 fitness.

Results also demonstrated significant improvement during generations, and stagnation in the last two generations. This fact could indicate that the controller was already near the maximum fitness for this scenario. As it was expected, these results show better evolution when compared with the single simulation evolution, which can be explained by the higher number of individuals in this latter experiment.

Regarding the maximum fitness value, it was achieved in several generations. Again, as the tree did not had many nodes and the task was not so complex, the first generation found a maximum fitness value of 0.944. This factor also turned the evolution to become faster, as the following generations would be created based on the best individuals from the previous one.

By selecting the last best individual, which occurred in individual 529 from generation number 10, it is possible to observe the next parameters:

1. *Turn_left,* with 0.208 normalized value, which equals 10.48 degrees in scaled value;
2. *Turn_right* with 0.671 normalized value, which means 33.58 degrees in scaled value*;*

3. *ChangeAltitude,* with 0.384 normalized value, which means 31.34 meters in scaled value;
4. *Move_forward* with 1.028 normalized value, which means 15.42 meters in scaled value.

In this case, it is also important to verify the evolution of each parameter, to understand its meaning. Figure 5.5 demonstrates this evolution. Once again, the evolution and stabilization of each parameter are notorious. Also, this graphic can only be used in this analysis because all the individuals have the same tree structure.

As mentioned before, as the base is far away from the initial drone position, the *move_forward* action must be high, so the drone can move longer distances at each behavior tree *tick.* Also, the *turn_left* action finalized with an average value of 0,145, which is 7,33 degrees in scaled value. These results are consistent with the created scenario, as the drone's initial position is 15 degrees away on the left side from the base, resulting in the drone will perform two *turn lefts* before it starts to *move forward*.

Regarding the altitude order, it should only have a significant impact on the results when it is below 5 meters, as the drone will collide with the scenario obstacles. If the altitude order is approximately above 5 meters, the drone will go up and pass the obstacles.

The *turn_right* action has a significantly high normalized value, having an average of 0.805 in generation number 10. This means the scaled value is 40.269 degrees, which is a very high turn. However, by reading the results from the parameters' average results, it is possible to see that the drone aligns with the base by turning left 2 times, and does not need to turn right anymore. Instead, it changes the altitude for the specific order and moves forward as long as possible.

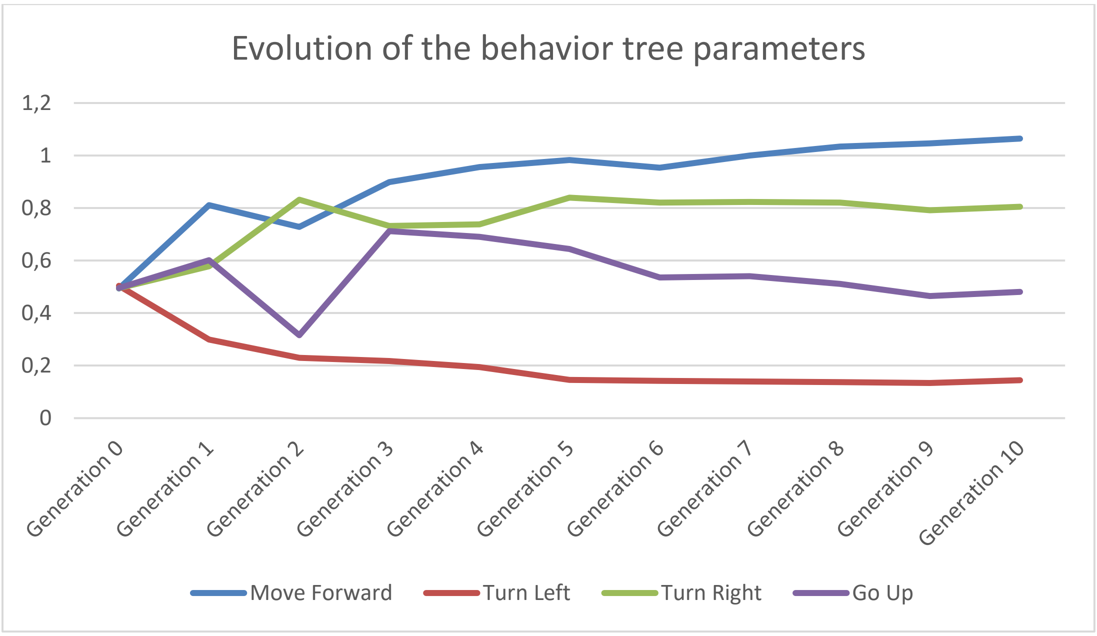


Figure 5.5 - Evolution of behavior tree parameters for the parallel simulation test, return to the base mission

The results from the parallel simulations demonstrated good fitness growth. They also showed a high coherence with the gazebo world and the base position. Hence, it is possible to conclude that the behavior tree controller was significantly evolved by the designed evolutionary algorithm.

5.3 Search and Rescue task – one single simulation

After testing the return to the base mission, it is important to test a more complex task. As mentioned earlier, drones can have an important role in search and rescue missions. In this scenario, a world model will be designed to simulate a scenario with destroyed buildings and four missing persons, two females and two males.

The drone's objective is to go up, perform a search and rescue movement pattern and look for the missing persons. If they are found, the drone lands and calculates fitness. A second drone is in the world model, and a second objective could also be the takeoff of the second drone and landing near the persons, where the first drone would be the "search one", with a high precision camera, and the second would be the "rescue one", with tools to help the missing persons. Because this would take more time than the one available for this experiment, the second drone is in the world model but will not be used in evolution. Therefore, the task to evolve is the search and rescue with one drone model.

Figures 5.6 and 5.7 represent the drones and the world model for this scenario.

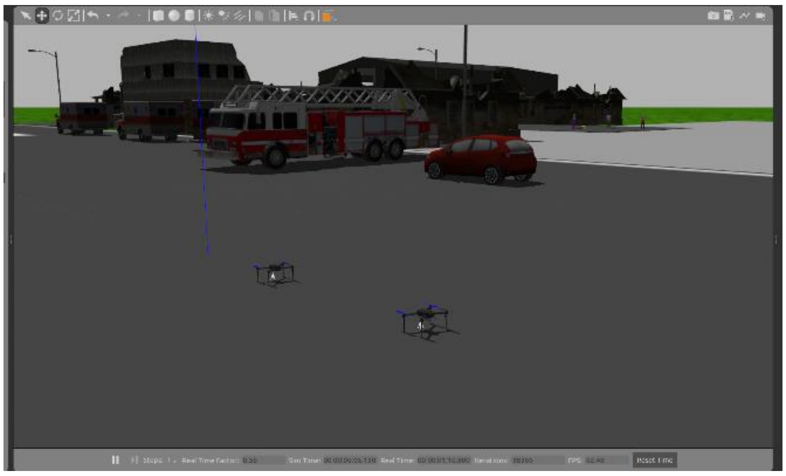


Figure 5.6 - Iris drones in search and rescue world

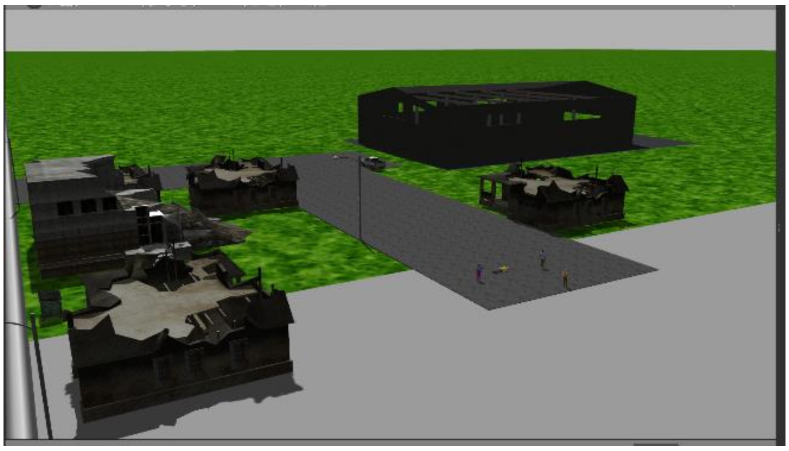


Figure 5.7 - Search and Rescue scenario with 4 missing persons

Also, as mentioned before, the image detection package is the *darknet_ros*, which is a machine learning package that subscribes to a ROS camera node and publishes the image classification. A callback function will listen to the image classification node and will activate the *FindMan* node in the behavior tree.

The behavior tree is the one shown in Figure 5.8, and it has the following nodes:

1. *FindMan*, if *darknet_ros* identifies a person, the drone will land and save the coordinates. No parameters to be evolved;
2. *ChangeAltitude,* same as in previous scenarios, but this time, minimum and maximum values are defined as 1 and 15 meters, respectively. This parameter is evolved;

3. *Forward_back_pattern,* where the drone will move forward 10x the *forward_order* value. This means that if the value is 5 meters, the drone will move 5 meters in front, slow and move again, doing 10 iterations. It is done to let the drone perform a search and rescue pattern where one leg is bigger than the other, and also to perform image classification. Minimum and maximum values are defined as 1 and 10 meters, respectively;
4. *Lateral_Pattern,* where the drone will do the lateral pattern in 5 times the *lateral_order* value. Minimum and maximum values are defined as 1 and 10 meters, respectively.

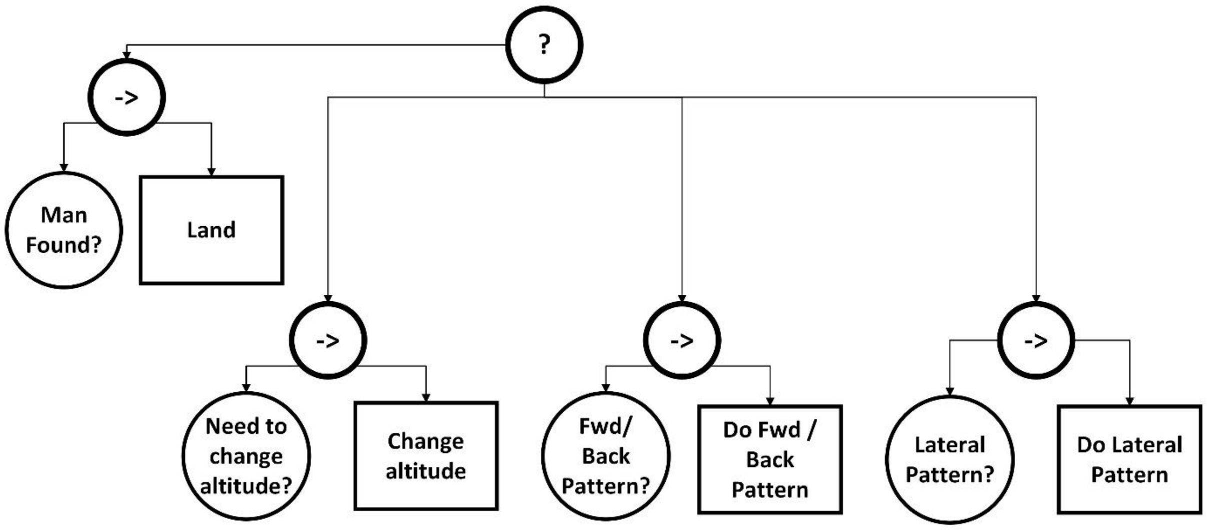


Figure 5.8 - Behavior tree for Search And Rescue scenario

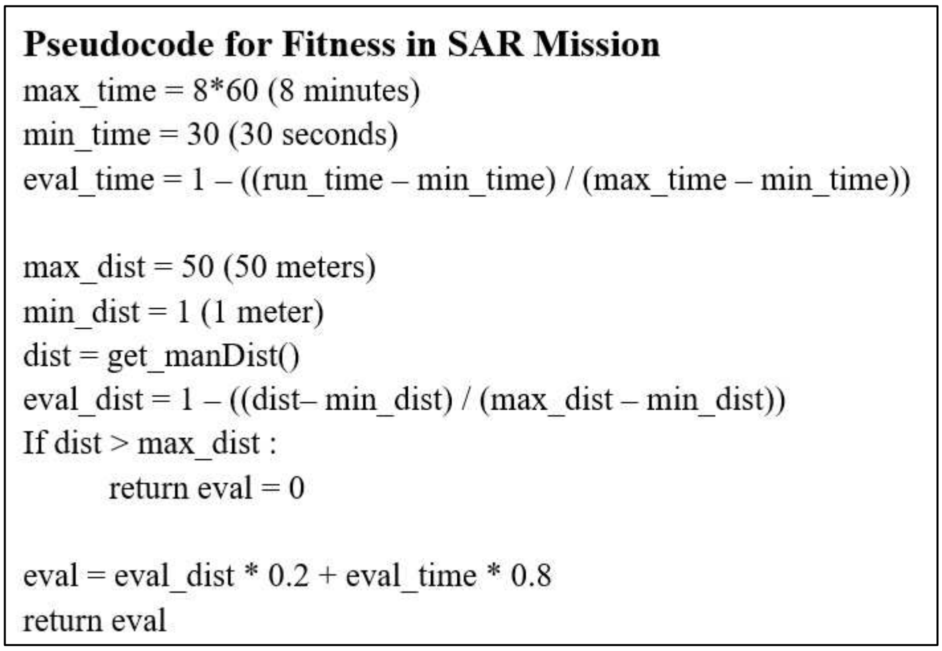
The controller and evolutionary algorithm parametrization are defined in Table 5.5.

Table 5.5 - Controller and evolutionary algorithm parametrization for Search and Rescue scenario

Controller / ER algorithm setup	**Values / Other commentaries**
Parameters to be evolved	ChangeAltitude (min. 1, max. 15), Forward_back_pattern (min. 1, max. 10, 10x), Lateral_pattern (min 1, max 10, 5x)
Starting individuals	All four individual parameters are started with random values between 0 and 1
Nr. of individuals/generations	1 simulation, 50 individuals per generation, 10 generations, total of 500 individuals

Fitness	Fitness is calculated according to the distance to the missing person, and the time it took to reach her. The pseudocode is shown in Figure 5.9.
Best individuals' percentage	10% of individuals are chosen for the next generation
Mutation	Generates 90% of mutated individuals for the next generation. Mutation occurs by choosing one random parameter and randomly adding/subtracting a value between 0 and 0.1.

As shown in Table 5.5, the fitness function is calculated based on the distance to the missing person, which is the distance between the position where the drone landed and the person's position. The time the robot takes to reach this position is also important in and addressed in the fitness function. If the drone lands in a distance higher than 50 meters, the evaluation is 0. The pseudocode is explained in Figure 5.9.



Pseudocode for Fitness in SAR Mission

```
max_time = 8*60 (8 minutes)
min_time = 30 (30 seconds)
eval_time = 1 – ((run_time – min_time) / (max_time – min_time))

max_dist = 50 (50 meters)
min_dist = 1 (1 meter)
dist = get_manDist()
eval_dist = 1 – ((dist– min_dist) / (max_dist – min_dist))
If dist > max_dist :
        return eval = 0

eval = eval_dist * 0.2 + eval_time * 0.8
return eval
```

Figure 5.9 - Pseudocode for fitness function in Search and Rescue Mission

The results are presented in Table 5.6 and Figure 5.10.

Table 5.6 - Results of average fitness and max value per generation, on Search and Rescue mission

Generation	0	1	2	3	4	5	6	7	8	9
Av. Fitness	0.596	0.692	0.741	0.789	0.792	0.777	0.807	0.801	0.818	0.824
Max. Value	0.782	0.794	0.804	0.811	0.827	0.834	0.844	0.844	0.844	0.845

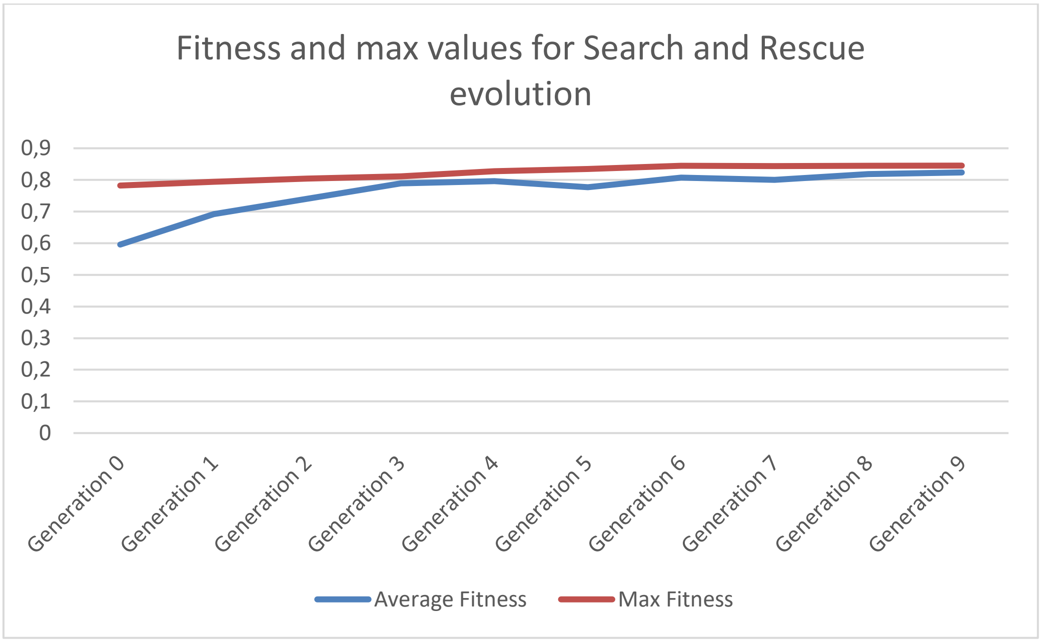


Figure 5.10 - Average fitness and max values per generation, on Search and Rescue evolution mission

Both the average fitness and max fitness value increased in every single generation, which is extremely important and demonstrates evolution. It is also very interesting to see that the drone started with the normal search and rescue pattern but inverted it, adapting it to the scenario circumstances. This is demonstrated in Figures 5.11 and 5.12.

By selecting one of the best individuals from the last generation, which occurred in individual 487 from generation number 9, it is possible to observe the next parameters:

1. *ChangeAltitude,* with 0.312 normalized value, which equals 5.4 meters in scaled value. It is a high enough value to avoid collision and to allow enough camera range to detect the missing persons;
2. *Forward_back_pattern* with 0.219 normalized value, which means 2,9 meters in the forward and back search and rescue pattern*;*

3. *Lateral_pattern,* with 0.816 normalized value, which means 8,344 meters in the lateral leg of the search and rescue pattern.

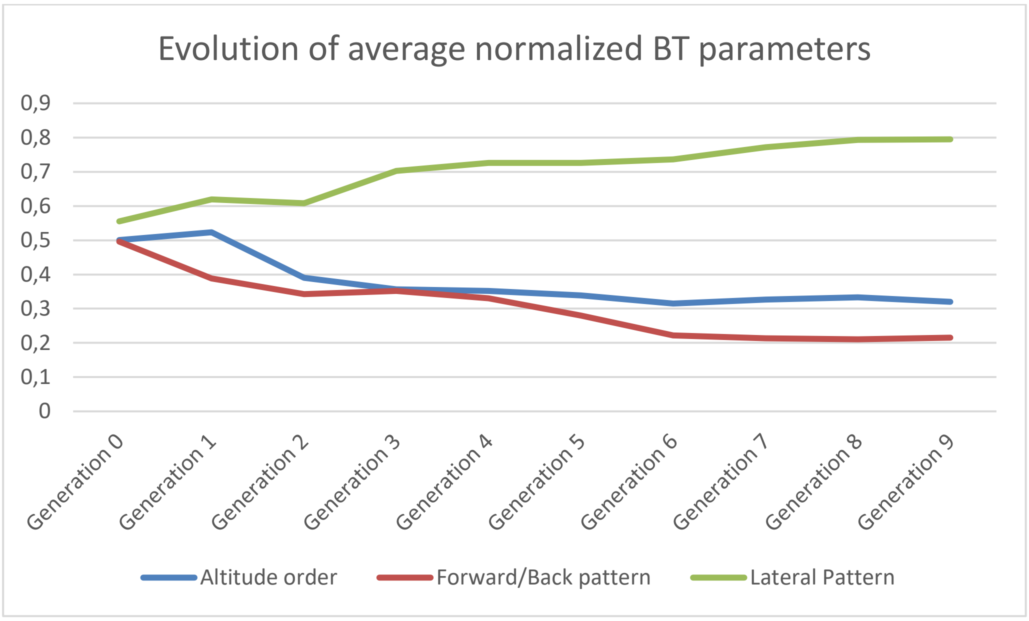


Figure 5.11 - Evolution of average normalized behavior tree node parameters

Figure 5.12 demonstrates the initial search and rescue pattern and the final pattern, which was evolved, resulting in an inversion of the pattern that is explained bellow. This type of pattern is a simple and typical search and rescue pattern, used in real missions. In this case, the drone started with longer distances in the Figure Y-axis, and shorter distances in the X-axis. The reason for this was explained previously, in the behavior tree nodes. However, during the evolution, it was possible to observe that the drone adapted to the scenario, and inverted the pattern, started to do longer X-axis distances and shorter Y-axis. This leads the drone to find the persons in a much faster way and presents very relevant results for this architecture and setup.

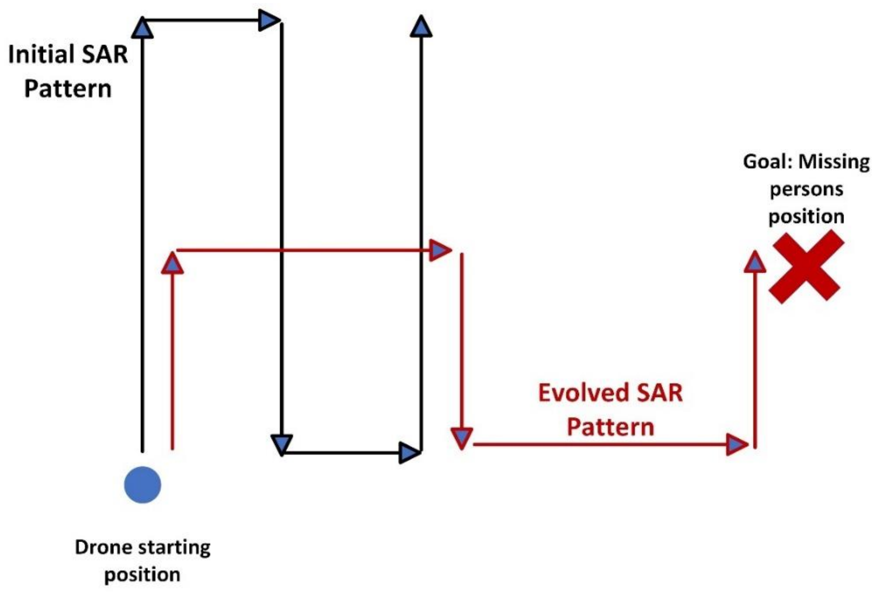


Figure 5.12 - Pattern evolution, initial Search and Rescue pattern and final evolved pattern

5.4 Summary

In summary, three relevant scenarios were demonstraded, shuch as the results and analysis. By analyzing the results, it is clear that evolution occurred and was very relevant for the tasks' success. Also, it was possible to observe the phenomena associated with the best individuals on each experiment and setup, and present relevant conclusions about each one.

CHAPTER 6

Conclusion

6.1 Conclusion

The application of evolutionary strategies in robotic environments has still proven to be complex, and there are still many research gaps in this field. Online evolution has its benefits, such as the ability to modify behavior according to environmental or task changes, however, it requires great amounts of computational power and time. Hence, the evolution of a controller in simulation, and its transfer to reality, may be the key to solve this issue.

This thesis proposed to study the use of realistic simulation environments in the evolution of robotic controllers. Specifically, it proposed to evolve controllers in simulation, where the simulated drone model conditions would be similar to the ones used in reality. Also, it proposed to use an approach that would be easily scaled for multi-robot systems or swarm robots.

Firstly, extensive research regarding the state of the art was conducted, where the topics of interest were addressed. Then, the methodology was defined, which included the simulator environment, drone operating system, flight control unit, and also controller model. After that, the experiments were presented, starting with the simple tests with no evolution, and finishing with a designed parallel simulation engine. Finally, results were presented in different scenarios, and analysis was performed on the most relevant results, including an inversion of a search and rescue pattern in a mission.

The proposed objectives of the thesis were accomplished, as it was possible to train robotic controllers, using evolutionary algorithms to train behavior trees, in realistic simulation environments. Although the work did not have multi-robot or SR evolution, this architecture and the chosen software make the scalability of these systems easy and simple.

In regards to the first RQ, it was possible to evolve robotic controllers using close to reality simulators and evolutionary algorithms, as it was demonstrated in chapter 5. Regarding RQ number two, the chosen software to run in the architecture and experiments was the same that is used in most of the real drones, i.e. ROS and ardupilot. Finally, this solution is also scalable for multi-robot systems and swarm robots, because the software was chosen based on this premise. Tests with no evolution were conducted to conclude this point.

Additionally, the drone model utilized in this thesis, the Iris Quadcopter drone, is a real drone, available in the market. This drone also uses ardupilot in real applications. Moreover, the architecture and the developed controllers use high-level flight control primitives, which guarantees the drone flight stabilization, as the drone receives orders that were already extensively tested by the robotic community. Also, even though the experiments used a SITL drone simulator, the usage of HITL would be just a simple improvement in the transition to reality, where a real drone hardware controller, such as a *Raspberry Py*, UP controller, or other similar companion computers, could be used with this same simulation environment and architecture.

To conclude, this thesis provides several advantages and significant gains to the evolutionary robotic field. By using realistic simulation environments, and the same software and drone models that are used in the real world, with high-level flight control primitives, it was possible to train controllers using evolutionary algorithms.

6.2 Limitations

Regarding limitations, and because this thesis marks a starting point in this architecture and in the evolutionary setup in highly realistic simulations, it was not possible to evolve multi-robot systems or swarm robots, as several engines and scripts had to be created and experimented with in the first place. The simulator was tested, as well as the evolutionary setup was built, and several problems were addressed and solved along the way. Also, a parallel simulation engine was developed, and the evolutionary algorithm, so as the integration of all these processes with a behavior tree controller.

6.3 Future work

The conducted work has the potential to be followed in several ways.

1. The architecture and experiments may be continued to train multi-robot or SR systems. Integration of these systems would be simple, but the tasks would probably be more complex to perform and evolve.
2. As previously mentioned, HITL usage in the simulation would be relevant work to cross the reality gap. This can also be done in future work, as a transition phase before experiments in the real world.
3. The same architecture and experiments could be done, but using the latest Gazebo Garden version that was released in September 2022. Initial feedback from the

community mentioned that the gazebo-ardupilot SITL plugin is significantly faster, increasing the RTF by at least 2x. Also, in these tests, the gazebo random noise functions could be explored, to increase controllers' robustness to failure, contributing to a minimization of the reality gap.

Bibliography

Albani, D., IJsselmuiden, J., Haken, R., & Trianni, V. (2017). Monitoring and mapping with robot swarms for agricultural applications. *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 1–6. https://doi.org/10.1109/AVSS.2017.8078478

Alkilabi, M. H. M., Lu, C., & Tuci, E. (2015). Cooperative Object Transport Using Evolutionary Swarm Robotics Methods. *07/20/2015-07/24/2015*, 464–471. https://doi.org/10.7551/978-0-262-33027-5-ch083

ArduCopter UK. (n.d.). *Iris QuadCopter model*. Http://Www.Arducopter.Co.Uk/Iris-Quadcopter-Uav.Html.

Ardupilot Dev Team. (n.d.). *Ardupilot Software In The Loop*. Https://Ardupilot.Org/Dev/Docs/Sitl-Simulator-Software-in-the-Loop.Html.

Bjelonic, M. (2018, January 7). *Darknet_ros*. Http://Wiki.Ros.Org/Darknet_ros.

Colledanchise, M., & Ögren, P. (2017). *Behavior Trees in Robotics and AI: An Introduction*. https://doi.org/10.1201/9780429489105

Collins, J., Chand, S., Vanderkop, A., & Howard, D. (2021). A review of physics simulators for robotic applications. *IEEE Access*, *9*, 51416–51431. https://doi.org/10.1109/ACCESS.2021.3068769

Costa, V. (2018). *Synthesis of formation control for an aquatic swarm robotics system*.

Duarte, M. (2015). *Engineering Evolutionary Control for Real-world Robotic Systems*.

Duarte, M., Oliveira, S., & Christensen, A. (2014). Hybrid Control for Large Swarms of Aquatic Drones. *Artificial Life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems*, 785–792. https://doi.org/10.7551/978-0-262-32621-6-ch127

Floreano, D., & Mondada, F. (1999). Automatic creation of an autonomous agent: genetic evolution of a neural-network driven robot. *Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior (SAB)*. https://www.researchgate.net/publication/2749362

Floreano, D., & Nolfi, S. (2000). *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines* (MIT Press).

Francesca, G., Brambilla, M., Brutschy, A., Garattoni, L., Miletitch, R., Podevijn, G., Reina, A., Soleymani, T., Salvaro, M., Pinciroli, C., Trianni, V., & Birattari, M. (2014). *An Experiment in Automatic Design of Robot Swarms* (pp. 25–37). https://doi.org/10.1007/978-3-319-09952-1_3

Grocholsky, B., Keller, J., Kumar, V., & Pappas, G. (2006). Cooperative air and ground surveillance. *IEEE Robotics and Automation Magazine*, *13*(3), 16–26. https://doi.org/10.1109/MRA.2006.1678135

Jakobi, N. (1997). Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis. *Adaptive Behavior*, *6*(2), 325–368. https://doi.org/10.1177/105971239700600205

Jhang, J.-Y., Lin, C.-J., & Young, K.-Y. (2019). Cooperative Carrying Control for Multi-Evolutionary Mobile Robots in Unknown Environments. *Electronics*, *8*(3), 298. https://doi.org/10.3390/electronics8030298

Johnson, E. A. S. (n.d.). *Intelligent Quads packages*. Https://Github.Com/Intelligent-Quads.

Juang, C.-F., Lu, C.-H., & Huang, C.-A. (2020). Navigation of Three Cooperative Object-Transportation Robots Using a Multistage Evolutionary Fuzzy Control Approach. *IEEE Transactions on Cybernetics*, 1–14. https://doi.org/10.1109/TCYB.2020.3015960

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering – A systematic literature review. *Information and Software Technology*, *51*(1), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009

Koenig, N., & Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 2149–2154. https://doi.org/10.1109/IROS.2004.1389727

Koos, S., Mouret, J. B., & Doncieux, S. (2013). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, *17*(1), 122–145. https://doi.org/10.1109/TEVC.2012.2185849

Lewis, M. A., Fagg, A. H., & Solidum, A. (1992). Genetic programming approach to the construction of a neural network for control of a walking robot. *Proceedings - IEEE International Conference on Robotics and Automation*, *3*, 2618–2623. https://doi.org/10.1109/robot.1992.220047

Markets and Markets. (2021, June). *Unmanned Aerial Vehicle (UAV) Market*. Market Research Report. https://www.marketsandmarkets.com/Market-Reports/unmanned-aerial-vehicles-uav-market-662.html

Nordmoen, J., Ellefsen, K. O., & Glette, K. (2018). Combining MAP-Elites and Incremental Evolution to Generate Gaits for a Mammalian Quadruped Robot. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10784 LNCS*, 719–733. https://doi.org/10.1007/978-3-319-77538-8_48

O'Dowd, P. J., Winfield, A. F. T., & Studley, M. (2011). The distributed co-evolution of an embodied simulator and controller for swarm robot behaviours. *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4995–5000. https://doi.org/10.1109/IROS.2011.6094600

Oh, S. H., & Suk, J. (2013). Evolutionary controller design for area search using multiple UAVs with minimum altitude maneuver. *Journal of Mechanical Science and Technology*, *27*(2), 541–548. https://doi.org/10.1007/s12206-012-1238-1

Open Robotics. (2018, March 3). *MAVROS*. Http://Wiki.Ros.Org/Mavros.

Open Robotics. (2022). *About Gazebo*. Https://Gazebosim.Org/About.

Peffers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., & Bragge, J. (2006). *Design Science Research Process: A Model for Producing and Presenting Information Systems Research*. http://arxiv.org/abs/2006.02763

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

PX4. (2022, August 9). *PX4 Hardware In The Loop*. Https://Docs.Px4.Io/Main/En/Simulation/Hitl.Html.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (2009). ROS: an open-source Robot Operating System. *ICRA Workshop on Open Source Software*, *3*. http://stair.stanford.edu

Ribeiro, M. I. (2005). Obstacle Avoidance. *The Robotics WEBook, Institute for Systems and Robotics*.

Romano, P. S. (2018). *A Cooperative Active Perception approach for Swarm Robotics*.

Rossi, C., Russo, F., & Russo Ferruccio. (2009). *Ancient Engineers' Inventions* (Vol. 8). Springer.

Roy, D., Maitra, M., & Bhattacharya, S. (2016). Study of formation control and obstacle avoidance of swarm robots using evolutionary algorithms. *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 003154–003159. https://doi.org/10.1109/SMC.2016.7844719

Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L. A., & Zurada, J. M. (Eds.). (2012). *Swarm and Evolutionary Computation* (Vol. 7269). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-29353-5

Sabino, S., & Grilo, A. (2018). Topology Control of Unmanned Aerial Vehicle (UAV) Mesh Networks. *Proceedings of the 4th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, 45–50. https://doi.org/10.1145/3213526.3213535

Silva, F., Duarte, M., Oliveira, S. M., & Christensen, A. L. (2015). *Open Issues in Evolutionary Robotics*. https://doi.org/10.1162/EVCO_a_00172#.Vk2jdoT70RV

Song, Y., Xi, Q., Xing, X., & Yang, B. (2020). Multi-UAV Cooperative Multi-Target Allocation Method based on Differential Evolutionary Algorithm. *2020 39th Chinese Control Conference (CCC)*, 1655–1660. https://doi.org/10.23919/CCC50068.2020.9189332

Syed Ahamed, M. F., Tewolde, G., & Kwon, J. (2018). Software-in-the-Loop Modeling and Simulation Framework for Autonomous Vehicles. *2018 IEEE International Conference on Electro/Information Technology (EIT)*, 0305–0310. https://doi.org/10.1109/EIT.2018.8500101

Thomas, A. (2016, July 26). *Iris 3dr drone follow me technology*. Https://Trackimo.Com/Iris-3dr-Drone-Follow-Me-Technology/.

Thomas, D. (2017, July 7). *ROS catkin workspaces*. Http://Wiki.Ros.Org/Catkin/Workspaces.

Toyota Research Institute. (n.d.). *Task_behavior_engine*. Https://Github.Com/ToyotaResearchInstitute/Task_behavior_engine.

Trueba, P., Prieto, A., & Bellas, F. (2017). Embodied evolution versus cooperative coevolution in multi-robot optimization. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 79–80. https://doi.org/10.1145/3067695.3076083

UP Shop. (2022). *UP companion computer for robotic control*. Https://Up-Shop.Org/up-Core-Series.Html.

Vaughan, N. (2018). *Evolutionary Robot Swarm Cooperative Retrieval* (pp. 517–521). https://doi.org/10.1007/978-3-319-95972-6_55

Weel, B., Hoogendoorn, M., & Eiben, A. E. (2012). On-Line Evolution of Controllers for Aggregating Swarm Robots in Changing Environments. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 7492 LNCS* (Issue PART 2, pp. 245–254). https://doi.org/10.1007/978-3-642-32964-7_25

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*, 1–10. https://doi.org/10.1145/2601248.2601268

Xiang-Yin, Z., & Hai-Bin, D. (2012). Differential evolution-based receding horizon control design for multi-UAVs formation reconfiguration. *Transactions of the Institute of Measurement and Control*, *34*(2–3), 165–183. https://doi.org/10.1177/0142331210366643

Yang, J., Liu, Y., Wu, Z., & Yao, M. (2012). The Evolution of Cooperative Behaviours in Physically Heterogeneous Multi-Robot Systems. *International Journal of Advanced Robotic Systems*, *9*(6), 253. https://doi.org/10.5772/53089

Yang, T., & Jiang, Y. (2015). Path planning for multiple robotic fish based on multi-objective cooperative co-evolution algorithm. *2015 10th International Conference on Computer Science & Education (ICCSE)*, 532–535. https://doi.org/10.1109/ICCSE.2015.7250304

Yao, X., & Liu, Y. (1998). Towards designing artificial neural networks by evolution. *Applied Mathematics and Computation*, *91*(1), 83–90. https://doi.org/10.1016/S0096-3003(97)10005-4

Yu, T., Yasuda, T., Ohkura, K., Matsumura, Y., & Goka, M. (2014). Apply incremental evolution with CMA-NeuroES controller for a robust swarm robotics system. *2014 Proceedings of the SICE Annual Conference (SICE)*, 295–300. https://doi.org/10.1109/SICE.2014.6935195

Zhao, Z., & Lu, G. (2012). Receding horizon control for cooperative search of multi-UAVs based on differential evolution. *International Journal of Intelligent Computing and Cybernetics*, *5*(1), 145–158. https://doi.org/10.1108/17563781211208260

Zheng, Q., Simon, D., Richter, H., & Gao, Z. (2014). Differential particle swarm evolution for robot control tuning. *2014 American Control Conference*, 5276–5281. https://doi.org/10.1109/ACC.2014.6858721